**Beschreibung und Begründung der Implementierungsenscheidungen**

**Projekt Aufbau :**

Auftragscontroller (Auftragscontroller): Im Auftragscontroller-Ordner befindet sich der Code, der für die Verarbeitung von Anfragen im Zusammenhang mit Aufträgen verantwortlich ist. Hier werden Funktionen implementiert, um Aufträge zu erstellen, zu bearbeiten oder zu löschen. Die Verwendung eines Controllers ermöglicht eine klare Trennung der Logik zur Auftragsverwaltung von anderen Teilen des Codes und verbessert die Wartbarkeit und Lesbarkeit. Durch die Konzentration aller auftragsbezogenen Funktionen an einem Ort ist es einfacher, sie zu finden und zu warten.

Db (Datenbankcode): Der Db-Ordner enthält den Code für die Datenbankverbindung und die Interaktion mit der Datenbank. Hier werden Funktionen implementiert, um Daten abzurufen, zu speichern, zu aktualisieren oder zu löschen. Durch die Organisation des Datenbankcodes in einem separaten Ordner wird die Wiederverwendbarkeit und Lesbarkeit verbessert. Außerdem bleibt die Datenbanklogik isoliert, was die Wartung und Erweiterung der Datenbankfunktionalität erleichtert.

Models (Modelle): Im Models-Ordner werden die Datenmodelle definiert, die die Struktur der Daten in der Datenbank repräsentieren. Jedes Datenmodell enthält Eigenschaften und Methoden, um auf die Daten zuzugreifen und sie zu verarbeiten. Durch die Verwendung von Modellen wird eine klare Struktur für die Daten definiert und die Interaktion mit der Datenbank vereinfacht. Dadurch wird auch die Codelesbarkeit erhöht, da die Modelle die Datenstrukturen klar definieren.

Public: Der Public-Ordner enthält Dateien, die öffentlich zugänglich sind, wie CSS-Dateien und javascript Datein . Öffentlich zugängliche Dateien können direkt über die URL aufgerufen werden und sind für die Benutzer sichtbar. Durch die Trennung dieser Dateien von anderen Teilen des Codes wird die Organisation verbessert und es ist einfacher, auf diese Ressourcen zuzugreifen und sie zu verwalten.

Routes: Im Routes-Ordner werden die Routen definiert, die den Benutzern ermöglichen, verschiedene Teile der Webseite aufzurufen. Hier werden die Endpunkte definiert, auf die die Benutzeranfragen abzielen, und die zugehörige Logik wird implementiert, um die Anfragen zu verarbeiten und die entsprechenden Daten zurückzugeben. Durch die Trennung der Routenlogik von anderen Teilen des Codes wird die Lesbarkeit und Wartbarkeit verbessert, da die Routenlogik an einem zentralen Ort konzentriert ist.

Views: Der Views-Ordner enthält die EJS-Dateien (Embedded JavaScript) für die Ansichten der Webseite. EJS ist eine Template-Engine, die es ermöglicht, dynamische Inhalte in HTML-Dateien zu generieren. Hier werden die HTML-Vorlagen mit den EJS-Tags erstellt, um Daten aus der Datenbank oder dem Controller in die Ansichten einzufügen. Die Trennung der Ansichten von anderen Teilen des Codes erleichtert das Verständnis der Benutzeroberfläche und ermöglicht eine

**Die Begründung:**

Eine strukturierte Ordnerstruktur bietet viele Vorteile für die Entwicklung eines Projekts. Durch die Aufteilung des Codes in separate Ordner und Dateien nach ihrer Funktion wird der Code modularer, wiederverwendbarer und besser organisiert. Dies erleichtert die Zusammenarbeit in größeren Entwicklerteams. Die klare Trennung der Verantwortlichkeiten ermöglicht eine verbesserte Lesbarkeit und Wartbarkeit des Codes, da Probleme leichter identifiziert und behoben werden können, ohne den gesamten Code durchsuchen zu müssen. Die strukturierte Ordnerstruktur erleichtert auch die Skalierbarkeit des Projekts und unterstützt die Versionskontrolle und Zusammenarbeit in Entwicklerteams. Insgesamt führt sie zu einem effizienteren Entwicklungsprozess und verbessert die Qualität des Codes.

**1.Client**

**Die Strukturierung der HTML-Seiten:**

Alle HTML-Seiten verwenden denselben Header und Footer, die als EJS-Dateien implementiert sind. Dieser Ansatz sorgt für eine einheitliche Gestaltung und Navigation auf der gesamten Website.

Der Header wird in einer separaten EJS-Datei erstellt .

Der Header besteht aus einem <header>-Element mit der Klasse "myheader". Innerhalb des Headers befindet sich ein <h1>-Element, das den Titel "Work\_Flow" enthält. Der Titel ist ein anklickbarer Link, der zur Startseite ("/") führt.

Darunter befindet sich eine Navigationsleiste, die in einem <nav>-Element gruppiert ist. Die Navigationsleiste enthält eine ungeordnete Liste <ul> mit der Klasse "parent". Innerhalb der Liste befinden sich drei Listenelemente <li>, die jeweils einen Link <a> enthalten. Die Links verweisen auf die folgenden Seiten:

1."Alle Aufträge" mit dem Link "/alle"

2."Neuer Auftrag" mit dem Link "/add-new-article"

3."Suche" mit dem Link "/suche"

Dieser Header-Code wird in jede der sechs HTML-Seiten eingefügt, um eine konsistente Darstellung und Navigation über die gesamte Website sicherzustellen. , die auf jeder Seite angezeigt werden sollen. Durch die EJS-Include-Anweisung <%- include("./partials/header.ejs") %> wird der Header in jede HTML-Seite eingebunden.

Ebenso wird der Footer als eigenständige EJS-Datei erstellt. Der Footer befindet sich am Ende jeder HTML-Seite und enthält Informationen über den Person, der das Projekt entwickelt hat. Durch die EJS-Include-Anweisung <%- include("./partials/footer.ejs") %> wird der Footer in jede HTML-Seite eingefügt.

Durch die Verwendung der gleichen Header- und Footer-EJS-Dateien wird eine einheitliche Benutzererfahrung auf der gesamten Website gewährleistet. Änderungen am Design oder an der Navigation müssen nur an einer Stelle vorgenommen werden und werden automatisch auf alle Seiten übernommen. Dadurch wird die Wartung der Website vereinfacht und eine konsistente Darstellung gewährleistet.

Die Hauptinhalt ist für jede Seite individuell, besteht aber immer aus einer Unterüberschrift, die den Inhalt grob beschreibt, was in der Seite erwartet wird.

Hauptseite (HOME):

**1. Tabelle "Neue Aufträge" :**

Diese Tabelle zeigt die neu eingegangenen Aufträge an, die vom Kundenservice erfasst wurden und noch nicht von anderen Abteilungen bearbeitet wurden.

Die Spalten "Order Date" und "Anzahl" enthalten Informationen zum Bestelldatum des Auftrags und zur Anzahl der zu bearbeitenden Artikel.

Jede Zeile in der Tabelle repräsentiert einen Auftrag mit dem entsprechenden Bestelldatum und der Anzahl.

Die Daten in dieser Tabelle stammen aus den Erfassungen des Kundenservice-Teams, die die Aufträge im System registrieren.

**Tabelle "In Bearbeitung" :**

Diese Tabelle zeigt die Aufträge an, die sich derzeit in Bearbeitung befinden und von verschiedenen Abteilungen übernommen wurden.

Auch hier enthält sie die Spalten "Order Date" und "Anzahl" mit Informationen zum Bestelldatum und zur Anzahl der bearbeiteten Artikel.

Die Daten in dieser Tabelle werden eingetragen solange es noch von den Abteilungen was zu tun gibt, somit kann der Custmer service auf den ersten Blick herausfinden, wie viele Auftäge noch in der Bearbeitung sind.

**Tabelle "Fertig" :**

In dieser Tabelle werden die abgeschlossenen Aufträge angezeigt, die von den verschiedenen Abteilungen als "fertig" markiert wurden.

Wie zuvor enthält sie die Spalten "Order Date" und "Anzahl" mit Informationen zum Bestelldatum und zur Anzahl der fertiggestellten Artikel.

Die Daten in dieser Tabelle werden von den Abteilungen erfasst, sobald sie die Bearbeitung eines Auftrags abgeschlossen haben.

Durch diese Tabellen erhalten die Benutzer einen Überblick über den Status der Aufträge, beginnend mit der Erfassung durch den Kundenservice bis zur Bearbeitung und abschließenden Fertigstellung durch die verschiedenen Abteilungen. Dies ermöglicht eine effiziente Kommunikation und Zusammenarbeit zwischen den Abteilungen, um die Aufträge erfolgreich abzuwickeln.

**Alle Aufträge:**

Die Hauptfunktionalität der Seite besteht darin, eine Übersicht aller Aufträge bereitzustellen.

Die Auftragsliste wird durch eine Schleife erstellt, die durch das Array "arrAuftrag" iteriert.

Jeder Auftrag wird in einem eigenen Abschnitt mit dem HTML-Tag "article" angezeigt.

Innerhalb jedes Auftrags wird eine Tabelle erstellt, die wichtige Informationen wie Auftrags-ID, Kunden-ID und Lieferdatum enthält.

Diese Informationen werden dynamisch aus den Eigenschaften des jeweiligen Auftragsobjekts generiert.

Darüber hinaus wird ein "Mehr" -Link angezeigt, der es dem Benutzer ermöglicht, detailliertere Informationen über den Auftrag aufzurufen. Dieser Link enthält eine eindeutige ID des Auftrags als Teil der URL.

Information:

Die Seite zeigt Auftragsinformationen an, die in einer "card" strukturiert sind. Diese Informationen umfassen die Auftrags-ID, die Kunde-ID und das gewünschte Lieferdatum. Der Header mit dem Titel "AuftragsInfo" gibt an, dass es sich um spezifische Auftragsdetails handelt. Die Daten für diese Informationen werden dynamisch aus den Auftragsobjekteigenschaften generiert.

Die Seite enthält Abschnitte für verschiedene Abteilungen wie "Service Disk", "System Configuration", "Professional Services" und "Logistik".

In diesem Szenario wird der Status für jede Abteilung zu Beginn auf "Open" gesetzt und ändert sich dann, sobald die Aufgabe abgeschlossen ist. Hier ist eine Beschreibung des Ablaufs:

Zu Beginn: Wenn ein neuer Auftrag erstellt wird, werden alle Abteilungen mit dem Status "Open" versehen. Dies bedeutet, dass alle Aufgaben noch ausstehen und bearbeitet werden müssen.

Abteilungsaufgaben: Jede Abteilung hat spezifische Aufgaben im Zusammenhang mit dem Auftrag. Sobald eine Abteilung ihre Aufgabe abgeschlossen hat, kann ein Mitarbeiter den Status der Aufgabe auf "Fertig" setzen.

Aktualisierung des Status: Sobald eine Aufgabe als erledigt markiert wird, erfolgt eine Aktualisierung des Status. Der Status ändert sich von "Open" auf "Fertig", um anzuzeigen, dass die Aufgabe erfolgreich abgeschlossen wurde.

Anzeige des aktualisierten Status: Die Seite wird automatisch aktualisiert, um den geänderten Status anzuzeigen. Dadurch können andere Mitarbeiter und Benutzer den aktuellen Status der Aufgabe einsehen.

Verfolgung des Fortschritts: Während die verschiedenen Abteilungen ihre Aufgaben erledigen und den Status auf "Fertig" setzen, können die Benutzer den Fortschritt des Auftrags anhand der Statusänderungen verfolgen. Dadurch wird eine transparente Darstellung des aktuellen Standes des Auftrags ermöglicht.

Diese Vorgehensweise ermöglicht eine effektive Verwaltung der Aufgaben und bietet den Benutzern eine klare Übersicht über den Fortschritt der Auftragsbearbeitung.

To Do: Die "To Do"-Liste enthält eine Aufstellung der Aufgaben, die von den einzelnen Abteilungen erledigt werden müssen. Sie fungiert als Leitfaden für die Abteilungen, um ihre spezifischen Aufgaben im Zusammenhang mit dem Auftrag zu identifizieren und erfolgreich abzuschließen. Die Liste stellt sicher, dass jede Abteilung über ihre Zuständigkeiten informiert ist und keine Aufgaben übersehen werden.

Anzahl: Der Begriff "Anzahl" bezieht sich auf die Anzahl der Geräte, die von einer Abteilung für ihre Aufgabe benötigt werden. Je nach Art der Aufgabe können dies Laptops oder PCs sein. Die Anzahl gibt an, wie viele Geräte von der Abteilung bearbeitet werden müssen, um die Aufgabe erfolgreich abzuschließen. Dies hilft den Abteilungen, den Umfang ihrer Arbeit zu verstehen und sicherzustellen, dass sie über ausreichend Ressourcen verfügen, um die Aufgabe zu bewältigen.

Ankommen Teile: "Ankommen Teile" beschreibt den Zeitpunkt, an dem die für die Geräte bestellten Teile vom Customer Service erwartet werden. Diese Teile sind notwendig, um die Aufgaben der Abteilungen abzuschließen. Indem die Abteilungen über das voraussichtliche Ankunftsdatum informiert sind, können sie ihre Zeitplanung entsprechend anpassen und sicherstellen, dass sie die benötigten Teile rechtzeitig erhalten. Dies ermöglicht eine effiziente Durchführung der Aufgaben und verhindert Verzögerungen aufgrund fehlender Teile.

Vormerk: Ein "Vormerk" ist eine Notiz oder Anmerkung, die der Customer Service für jede Abteilung hinterlassen kann. Diese Notiz enthält wichtige Informationen oder Anweisungen, die für die erfolgreiche Durchführung der Aufgaben der Abteilung relevant sind. Der Vormerk ermöglicht es dem Customer Service, spezifische Anforderungen, Prioritäten oder besondere Hinweise mitzuteilen. Dies stellt sicher, dass die Abteilungen die Aufgaben entsprechend den Vorgaben erledigen und potenzielle Probleme oder Besonderheiten berücksichtigen können. Der Vormerk fördert die Kommunikation und Zusammenarbeit zwischen dem Customer Service und den Abteilungen, um den reibungslosen Ablauf der Auftragsabwicklung sicherzustellen.

Die Funktion arbeitet wie folgt: Sobald das Fenster geladen wird, werden alle Elemente mit der Klasse "select" mithilfe der Methode querySelectorAll ausgewählt. Diese Elemente repräsentieren Dropdown-Listen, aus denen der Benutzer eine Option auswählen kann. Für jedes ausgewählte Element wird die Funktion myfunktion aufgerufen.

In der myfunktion wird zunächst die Auftrags-ID extrahiert, indem auf den Button mit der Klasse "Delete" zugegriffen und die Auftrags-ID aus dem Attribut "data-linkid" gelesen wird. Diese Auftrags-ID wird später für den API-Aufruf verwendet, um die entsprechenden Auftragsinformationen abzurufen.

Anschließend wird ein API-Aufruf an den Server gesendet, um die Auftragsinformationen für die angegebene Auftrags-ID abzurufen. Die erhaltene Antwort wird als JSON erwartet. Sobald die Antwort empfangen wird, werden die Daten verarbeitet.

Die Funktion durchläuft die Auftragsinformationen, die in der empfangenen JSON-Antwort enthalten sind. Für jedes Element wird überprüft, ob die Teil-ID den ausgewählten Wert enthält. Zusätzlich wird das erste Zeichen der Teil-ID extrahiert und mit bestimmten Präfixen (wie "DSK", "DPS", "DLG", "DSD") verglichen, um die Art der Aufgabe zu bestimmen.

Wenn eine Übereinstimmung gefunden wird, werden bestimmte HTML-Elemente anhand ihrer IDs aktualisiert. Dies geschieht mithilfe der Methode getElementById. Die Werte wie Anzahl, Ankunftsdatum und Vormerkung werden in den entsprechenden HTML-Elementen aktualisiert, um sie anzuzeigen.

Die Funktion beginnt damit, das HTML-Element mit der Klasse "Delete" mithilfe der Methode querySelector auszuwählen. Dieses Element repräsentiert einen Button, der zum Löschen eines Auftrags verwendet wird.

Anschließend wird das Attribut "data-linkid" des Buttons ausgelesen und in der Variablen auftragID gespeichert. Diese ID wird benötigt, um den entsprechenden Auftrag zu identifizieren, der gelöscht werden soll.

Ein Event-Listener wird dem Button hinzugefügt, der auf das Klicken des Buttons reagiert. Sobald der Benutzer den Button klickt, wird eine Bestätigungsmeldung angezeigt, in der er gefragt wird, ob er den Auftrag wirklich löschen möchte.

Wenn der Benutzer die Löschung bestätigt, wird ein API-Aufruf an den Server gesendet, um den Auftrag zu löschen. Dabei wird die DELETE-Methode verwendet und die auftragID wird in der URL /auftrag-informationen/{auftragID} verwendet, um den spezifischen Auftrag zu identifizieren.

Sobald die Antwort des Servers empfangen wird, werden die Daten als JSON verarbeitet. Die Methode then wird verwendet, um die Antwortdaten zu verarbeiten. Anschließend wird die window.location.href auf den in den Daten enthaltenen Link gesetzt, um den Benutzer auf eine andere Seite weiterzuleiten. Dies kann beispielsweise die Liste der verbleibenden Aufträge sein.

Falls während des API-Aufrufs ein Fehler auftritt, wird die catch-Methode verwendet, um den Fehler in der Konsole anzuzeigen und potenzielle Probleme bei der Verarbeitung des Löschvorgangs zu identifizieren und zu beheben.

Am Ende der Seite befindet sich ein Abschnitt mit Buttons. Der "Update"-Button verlinkt auf eine andere Seite, auf der der Customer Service Änderungen an den Auftragsdetails vornehmen kann. Die eindeutige Auftrags-ID wird als Teil der URL übergeben.

Update:

wennd das Update Button bei der Information Seite geklickt wird, dann kommt man auf Update Seite.

Der vorliegende Code ermöglicht die interaktive Bearbeitung von Auftragsinformationen auf einer Webseite. Zunächst wird der Button mit der Klasse "create" ausgewählt, um die Auftrags-ID aus seinem Attribut "data-linkid" abzurufen. Diese Auftrags-ID dient zur eindeutigen Identifikation des Auftrags, mit dem später kommuniziert wird.

Anschließend werden verschiedene Variablen deklariert, um auf die relevanten HTML-Elemente zuzugreifen und Informationen zu speichern. Unter anderem werden die Elemente mit den IDs "Auftrag\_id", "Kunde\_ID" und "Lieferdatum" referenziert, um später ihre Werte zu aktualisieren. Das leere Array "auftrag\_Info" dient dazu, Informationen über die Auftragsdetails zu speichern, die später aktualisiert werden sollen.

Mittels einer HTTP-Anfrage (Fetch) an den Server mit der Auftrags-ID als Pfadparameter wird versucht, die entsprechenden Auftragsdaten abzurufen. Die empfangenen Daten werden als JSON-Objekt zurückgegeben.

Nachdem die Antwort vom Server erhalten wurde, werden die Daten verarbeitet. Die Auftrags-ID, die Kunde-ID und das Lieferdatum werden in die entsprechenden HTML-Formularfelder eingefügt, indem ihre "value"-Attribute aktualisiert werden. Die Länge des Arrays "auftrag\_Info" wird anhand der empfangenen Daten festgelegt.

Falls die Länge des Arrays "auftrag\_Info" größer als 0 ist, gibt es bereits Auftragsdetails, und es werden zusätzliche Eingabefelder generiert und mit den entsprechenden Werten befüllt. Dafür wird ein vorhandenes Div-Element mit der Klasse "auftragInput" geklont und die Klone dem HTML-Container mit der ID "container" hinzugefügt. Dabei werden die Labels und Input-Felder in den Klonen anhand ihrer Namen und IDs aktualisiert, indem die Ziffer "0" durch den aktuellen Index ersetzt wird.

Anschließend werden die Werte der Auftragsdetails (Anzahl, Teil-ID, ankommendes Datum) aus den empfangenen Daten in die entsprechenden Eingabefelder eingefügt. Die IDs der Eingabefelder werden basierend auf dem Index des Schleifen-Durchlaufs dynamisch aktualisiert.

Für jede Auftragsdetailzeile wird ein JavaScript-Objekt erstellt, das die Referenzen auf die Eingabefelder (Anzahl, Teil-ID, ankommendes Datum) enthält. Diese Objekte werden dem Array "auftrag\_Info" hinzugefügt. Zusätzlich wird eine Ausgabe und Überprüfung der einzelnen Elemente im "auftrag\_Info"-Array durchgeführt.

Nachdem die Auftragsdaten erfolgreich abgerufen und verarbeitet wurden, wird ein Eventlistener zum Formular hinzugefügt, der auf das Abschicken des Formulars reagiert.

Beim Abschicken des Formulars wird überprüft, ob die eingegebenen Werte gültig sind. Dabei wird überprüft, ob die Anzahl, Teil-ID, Auftrags-ID und Kunde-ID größer oder gleich Null sind und ob die Anzahl eine Zahl ist. Bei ungültigen Eingaben wird ein Alert mit einer Warnung angezeigt und die Aktion des Formulars wird verhindert.

Sind die eingegebenen Werte gültig, wird eine Bestätigungsmeldung angezeigt und die Daten werden an den Server gesendet. Dabei werden die Daten in ein JSON-Objekt verpackt und über eine PUT-Anfrage an den Server gesendet. Die empfangene Antwort wird im JSON-Format zurückgegeben.

Nach Erhalt der Antwort vom Server wird die Webseite auf die in der Antwort enthaltene Link-URL weitergeleitet. Dies deutet darauf hin, dass die Seite nach der Aktualisierung der Auftragsdetails auf eine neue Seite oder auf eine aktualisierte Ansicht des Auftrags weitergeleitet wird.

Neue Auftrag:

Das HTML-Formular auf der Seite ermöglicht es den Benutzern, einen neuen Auftrag zu erstellen, indem sie relevante Informationen eingeben. Dieses Formular ist der Hauptbestandteil der Seite und ermöglicht die Interaktion zwischen dem Benutzer und dem Server.

Das Formular verwendet die POST-Methode, um die eingegebenen Daten an den Server zu senden. Wenn der Benutzer auf den "create"-Button klickt, wird das Formular abgeschickt und die eingegebenen Daten werden an den Server gesendet, um weiterverarbeitet zu werden.

Das Formular besteht aus verschiedenen Eingabefeldern, die mit Hilfe von <input>-Tags definiert sind. Jedes Eingabefeld hat einen Namen, der dazu dient, die eingegebenen Werte im Servercode zu identifizieren und korrekt zuzuordnen. Zum Beispiel gibt es Eingabefelder wie "Auftrag\_id", "Kunde\_ID" und "Lieferdatum", die spezifische Informationen zum Auftrag erfassen.

as Eingabefeld "Teil\_id" erfasst eine eindeutige Kennung für das Teil im Auftrag. Die Teil-ID besteht aus insgesamt 10 Zeichen. Die ersten drei Buchstaben in der Teil-ID repräsentieren die Abteilung, für die die Aufgabe bestimmt ist. Es gibt vier spezifische Abteilungen mit ihren entsprechenden Buchstabenkombinationen:

1. "DSK" steht für DSK System Konfigration.

2."DPS" steht für DPS Professional Services.

3."DLG" steht für DLG Logistik.

4."DSD" steht für DSD Service Disk.

Die verbleibenden sieben Zahlen in der Teil-ID geben zusätzliche Informationen über die spezifischen Aufgaben oder Aktionen an, die von der jeweiligen Abteilung durchgeführt werden sollen. Es ist wichtig zu beachten, dass diese Zahlen intern verwendet werden und von Abteilung zu Abteilung unterschiedlich sein können.

Ein Beispiel für eine gültige Teil-ID wäre beispielsweise "DPS1234567". In diesem Fall gehört das Teil zur DPS-Abteilung und die Zahlenkombination "1234567" gibt weitere spezifische Details für diese Aufgabe an.

Diese Teil-ID wird verwendet, um das Teil eindeutig zu identifizieren und es ermöglicht eine bessere Zuordnung zu den entsprechenden Abteilungen und Aufgaben im Auftragssystem.

Einige der Eingabefelder sind als "required" markiert, was bedeutet, dass diese Felder zwingend ausgefüllt werden müssen, bevor das Formular abgeschickt werden kann. Wenn ein Benutzer versucht, das Formular abzuschicken, ohne diese Pflichtfelder auszufüllen, wird eine Validierungsfehlermeldung angezeigt, die den Benutzer auffordert, die fehlenden Informationen einzugeben.

Die verschiedenen Eingabefelder können unterschiedliche Typen haben, die den Benutzern bei der Eingabe bestimmter Arten von Informationen helfen. Beispielsweise gibt es Eingabefelder vom Typ "text", "number" und "date", die je nach Anforderungen des Auftrags verschiedene Arten von Daten erfassen können.

Neben dem Formular gibt es auch andere Elemente auf der Seite, wie zum Beispiel das <h1>-Element mit der Überschrift "Neue Auftrag erstellen". Dieses Element dient dazu, den Benutzern den Zweck der Seite deutlich zu kommunizieren und ihnen eine klare Orientierung zu geben.

Der JavaScript-Code auf der Seite fügt zusätzliche Funktionalität hinzu und reagiert auf bestimmte Benutzerinteraktionen. Zum Beispiel wird eine JavaScript-Funktion ausgeführt, wenn der Benutzer auf das "add"-Symbol klickt. Diese Funktion klonen den ersten Abschnitt für Teileinformationen und fügt ihn dem Formular hinzu. Dadurch kann der Benutzer weitere Informationen zu den Teilen hinzufügen, die für den Auftrag benötigt werden.

Eine weitere JavaScript-Funktion wird ausgeführt, wenn das Formular abgeschickt wird. Diese Funktion überprüft die eingegebenen Daten, insbesondere ob negative Zahlen oder ungültige Zeichen in den Eingabefeldern vorhanden sind. Wenn solche Fehler auftreten, wird dem Benutzer eine Warnmeldung angezeigt und das Formular wird nicht abgeschickt.

Wenn jedoch alle eingegebenen Daten korrekt sind, wird dem Benutzer eine Bestätigungsmeldung angezeigt, in der er gefragt wird, ob er den Vorgang fortsetzen möchte. Je nach Antwort des Benutzers wird das Formular entweder abgeschickt und die Daten werden an den Server gesendet, oder der Vorgang wird abgebrochen.

Nach dem Abschicken des Formulars werden die eingegebenen Daten an den Server gesendet und dort weiterverarbeitet. Die genaue Verarbeitung der Daten hängt vom Servercode ab, der auf die gesendeten Daten zugreift und entsprechende Aktionen durchführt, wie beispielsweise das Speichern des Auftrags in einer Datenbank oder das Ausführen weiterer Geschäftslogik.

suche:

Die Webseite implementiert eine Suchfunktion für Aufträge, die den Benutzern ermöglicht, gezielt nach Aufträgen zu suchen und detaillierte Informationen abzurufen. Die Seite folgt einem einheitlichen Design, indem sie den Header und Footer aus anderen Dateien importiert.

Das Suchformular besteht aus drei Eingabefeldern für die Auftrags-ID, die Kunde-ID und das Lieferdatum, sowie einem "Suchen"-Button. Nachdem der Benutzer den Button betätigt hat, wird die JavaScript-Funktionalität aktiviert.

Der JavaScript-Code ist in das HTML-Dokument eingebettet und wird nach dem Laden der Seite ausgeführt. Er enthält die Logik zur Suche und Verarbeitung der Daten. Zunächst werden die relevanten HTML-Elemente den JavaScript-Variablen zugewiesen.

Ein Eventlistener wird dem "Suchen"-Button hinzugefügt, um auf das Klick-Ereignis zu reagieren. Sobald der Button geklickt wird, werden die eingegebenen Werte der Auftrags-ID und der Kunde-ID abgerufen.

Daraufhin wird eine GET-Anfrage mittels Fetch an den Server gesendet, um die Auftragsdaten abzurufen. Die empfangenen Daten werden im JSON-Format interpretiert und durchlaufen. Dabei werden die eingegebenen Werte mit den entsprechenden Werten in den Auftragsdaten verglichen.

Wenn eine Übereinstimmung für die Auftrags-ID gefunden wird, erfolgt eine Weiterleitung zur Auftragsinformationen-Seite, auf der detaillierte Informationen zu dem entsprechenden Auftrag angezeigt werden.

Falls keine Übereinstimmung für die Auftrags-ID gefunden wird, wird die Kunde-ID überprüft. Bei einer Übereinstimmung werden die relevanten Auftragsobjekte in das JSON-Format umgewandelt. Anschließend wird die Seite zur Suchergebnisseite weitergeleitet, auf der die gefundenen Aufträge angezeigt werden.

Die Funktionalität der Webseite bietet den Benutzern eine benutzerfreundliche Suchfunktion für Aufträge. Durch die Verwendung von JavaScript und Fetch-Anfragen werden die Suchanfragen effizient an den Server gesendet und die Ergebnisse werden dynamisch auf der Webseite angezeigt.

Server:

**Express und Nod js:**

Node.js und Express.js sind hervorragende Optionen für ein Workflow-Management-Projekt aus Sicht eines Informatikers. Die Kombination dieser beiden Technologien bietet eine Vielzahl von Vorteilen, die speziell auf die Anforderungen eines Workflow-Management-Systems zugeschnitten sind.

Zunächst einmal ermöglicht Node.js durch seine ereignisgesteuerte, nicht blockierende Architektur eine außergewöhnliche Skalierbarkeit. Workflow-Management-Systeme müssen in der Lage sein, eine große Anzahl von Benutzern und gleichzeitigen Aktivitäten effizient zu verwalten. Node.js kann problemlos eine hohe Anzahl gleichzeitiger Verbindungen verarbeiten, was für ein reibungsloses Funktionieren des Workflow-Management-Systems unter Belastung unerlässlich ist.

Ein weiterer Vorteil von Node.js ist seine effiziente Verarbeitung von Ein- und Ausgaben (I/O). Workflow-Management-Systeme beinhalten oft umfangreiche Datenverarbeitungsaufgaben, wie den Zugriff auf Datenbanken, das Lesen und Schreiben von Dateien und die Kommunikation mit externen APIs. Node.js ermöglicht eine schnelle und reibungslose Ausführung dieser Aufgaben, was die Leistung des Systems optimiert.

Darüber hinaus bietet Node.js native Unterstützung für Echtzeitkommunikation über WebSockets. In Workflow-Management-Systemen ist Echtzeitkommunikation oft entscheidend, um Benutzer über Aktualisierungen, Benachrichtigungen oder Aufgabenzuweisungen zu informieren. Node.js ermöglicht eine schnelle und zuverlässige Kommunikation zwischen dem Server und den Benutzern, was die Effektivität des Workflow-Managements erheblich verbessert.

Ein weiterer Grund für die Wahl von Node.js und Express.js ist die Verwendung von JavaScript als gemeinsame Sprache. Da JavaScript sowohl auf der Server- als auch auf der Clientseite verwendet wird, ermöglicht dies einen nahtlosen Austausch von Code zwischen den verschiedenen Komponenten des Workflow-Management-Systems. Entwickler, die bereits mit JavaScript vertraut sind, können ihre vorhandenen Kenntnisse und Fähigkeiten nutzen, was die Entwicklung, Wartung und Skalierung des Projekts vereinfacht.

Darüber hinaus profitieren Node.js und Express.js von einem umfangreichen Ökosystem und einer aktiven Entwicklergemeinschaft. Es gibt eine Fülle von Modulen, Bibliotheken und Tools, die bei der Entwicklung eines Workflow-Management-Systems helfen. Die Verfügbarkeit von Ressourcen, Dokumentation, Tutorials und Supportkanälen erleichtert Entwicklern die Lösung von Problemen und den Austausch bewährter Methoden.

Schließlich fördert Express.js, das Framework für Node.js, eine modulare Architektur und die Wiederverwendbarkeit von Code. Dies ermöglicht eine bessere Strukturierung des Codes in separate und wiederverwendbare Komponenten, was die Wartbarkeit, Skalierbarkeit und Lesbarkeit des Codes verbessert. Entwickler können effizienter arbeiten und die Zusammenarbeit in einem Team wird erleichtert.

**EJS:**

EJS, ein serverseitiges Rendering-Framework namens "Embedded JavaScript", bietet diverse Vorteile für Workflow-Management-Systeme. Es erlaubt die dynamische Generierung von Benutzeroberflächen mit besserer Kontrolle über die Darstellung. Durch die Verwendung von wiederverwendbarem Code in Form von Vorlagenkomponenten wird die Entwicklung und Wartung erleichtert. EJS unterstützt auch die Datenbindung, um Daten nahtlos in Vorlagen einzufügen und automatisch bei Aktualisierungen zu aktualisieren. Die Integration von JavaScript-Code ermöglicht die Implementierung komplexer Logik und die Interaktion mit dem System. Layouts und Partials werden unterstützt, um konsistente und modulare Benutzeroberflächen zu erstellen. EJS verfügt über eine umfangreiche Dokumentation und wird von einer aktiven Entwicklergemeinschaft unterstützt. Insgesamt erleichtert EJS die Entwicklung und Wartung von Workflow-Management-Systemen.

Contrller:

suche\_auftrag\_get :

Die Funktion suche\_auftrag\_get wird verwendet, um das "suche"-View-Template anzuzeigen, wenn eine GET-Anfrage für die "suche" Route empfangen wird. Das Template enthält wahrscheinlich eine Suchmaske oder ein Formular für die Auftragssuche. Das Datenobjekt { mytite: "Suche" } wird dem Template übergeben, um den Titel der Seite festzulegen.

auftrag\_json\_info\_get :

Die Funktion auftrag\_json\_info\_get wird aufgerufen, wenn eine GET-Anfrage für die "auftrag/:id" Route empfangen wird. Sie sucht den Auftrag mit der übergebenen ID in der Datenbank und gibt das Ergebnis als JSON-Antwort zurück. Zuerst wird der Inhaltstyp der Antwort auf "application/json" gesetzt. Das Suchergebnis wird dann in das JSON-Format konvertiert und an den Client gesendet.

alle\_auftraege\_json\_get :

Bei der Funktion alle\_auftraege\_json\_get handelt es sich um eine GET-Anfrage für die "alle" Route. Sie sucht alle Aufträge in der Datenbank und sendet das Ergebnis als JSON-Antwort zurück. Der Inhaltstyp der Antwort wird ebenfalls auf "application/json" gesetzt, und das Suchergebnis wird als JSON an den Client gesendet.

auftrag\_id\_information\_get :

Die Funktion auftrag\_id\_information\_get wird aufgerufen, wenn eine GET-Anfrage für die "auftrag/:id/information" Route empfangen wird. Sie sucht den Auftrag mit der übergebenen ID in der Datenbank und rendert das "information"-View-Template. Das Template erhält ein Datenobjekt mit dem Titel der Seite (mytite) und dem Auftragsobjekt (objekt\_Auftrag). Diese Daten werden verwendet, um Informationen über den Auftrag im Template anzuzeigen.

delete\_auftrag\_id\_delete:

Die Funktion delete\_auftrag\_id\_delete wird aufgerufen, wenn eine DELETE-Anfrage für die "auftrag/:id/delete" Route empfangen wird. Sie sucht den Auftrag mit der übergebenen ID in der Datenbank und löscht den Auftrag. Danach wird eine JSON-Antwort zurückgegeben, die einen Link zur Seite "/alle" enthält.

update\_auftrag\_id\_get :

Die Funktion update\_auftrag\_id\_get wird aufgerufen, wenn eine GET-Anfrage für die "auftrag/:id/update" Route empfangen wird. Sie sucht den Auftrag mit der übergebenen ID in der Datenbank und rendert das "update"-View-Template. Das Template erhält ein Datenobjekt mit dem Titel der Seite (mytite) und dem Auftragsobjekt (obAuftrag). Diese Daten können verwendet werden, um ein Formular zum Aktualisieren der Auftragsinformationen im Template anzuzeigen

auftrag\_update\_jsonId\_get:

Diese Funktion sucht in der Datenbank nach dem Auftrag, der mit der angegebenen ID verknüpft ist, und sendet das Suchergebnis als JSON-Antwort zurück. Zunächst wird der Inhaltstyp der Antwort auf "application/json" festgelegt. Anschließend wird das Suchergebnis in das JSON-Format umgewandelt und an den Client gesendet.

update\_auftrag\_id\_put:

Wenn eine PUT-Anfrage für die Route "auftrag/update/:id" empfangen wird, wird diese Funktion aufgerufen. Sie extrahiert die Informationen zum Auftrag, wie die Auftrags-ID, Kunden-ID, Lieferdatum und Auftragsdetails, aus dem Anfragekörper (req.body). Dann wird die angegebene Auftrags-ID verwendet, um den entsprechenden Auftrag in der Datenbank zu aktualisieren. Die Aktualisierung erfolgt über die Funktion Auftrag.updateOne(), bei der die Felder des Auftrags mit den neuen Daten aktualisiert werden. Wenn die Aktualisierung erfolgreich ist, wird eine JSON-Antwort zurückgegeben, die einen Link zur Seite "/alle" enthält. Im Falle eines Fehlers wird eine Fehlermeldung mit dem HTTP-Statuscode 500 zurückgegeben.

kundeid\_auftraege\_get:

Diese Funktion wird aufgerufen, wenn eine GET-Anfrage für die Route "kundeid/:json" empfangen wird. Der Parameter "json" enthält die Auftragsdaten im JSON-Format. Die Funktion analysiert dieses JSON und rendert das View-Template "alle". Dabei werden die Auftragsdaten als Array mit dem Namen "arrAuftrag" an das Template übergeben. Das Template kann dann diese Daten verwenden, um die Aufträge darzustellen.

alle\_auftraege\_get:

Diese Funktion sucht alle Aufträge in der Datenbank und rendert das View-Template "alle". Das Template erhält die Auftragsdaten als Array mit dem Namen "arrAuftrag". Dieses Array wird aus dem Suchergebnis (result) generiert. Das Template kann dann die Auftragsdaten verwenden, um die Aufträge anzuzeigen.

add\_auftrag\_post:

Wenn eine POST-Anfrage für die Route "auftrag/add" empfangen wird, wird diese Funktion aufgerufen. Sie erstellt einen neuen Auftrag anhand der Daten im Anfragekörper (req.body) und speichert den Auftrag in der Datenbank. Wenn die Speicherung erfolgreich ist, wird der Client zur Seite "/alle" weitergeleitet. Im Falle eines Fehlers wird eine Fehlermeldung in der Konsole ausgegeben.

Router:

Im "Router"-Ordner befinden sich drei JavaScript-Dateien: "alleR.js", "auftragsInfoR.js" und "updateR.js". Diese Dateien enthalten Router-Definitionen für verschiedene Routen in der Anwendung.

Die Datei "alleR.js" definiert einen Router für die Route "/", der den "auftragController" verwendet, um die entsprechenden Funktionen für GET- und POST-Anfragen zu verarbeiten. Eine GET-Anfrage an die Route "/" ruft die Funktion "alle\_auftraege\_get" des "auftragControllers" auf, während eine POST-Anfrage die Funktion "add\_auftrag\_post" aufruft.

Die Datei "auftragsInfoR.js" enthält einen Router für die Route "/auftrag-informationen/:id". Dieser Router verwendet den "auftragController" für GET- und DELETE-Anfragen. Eine GET-Anfrage an diese Route ruft die Funktion "auftrag\_id\_information\_get" auf, während eine DELETE-Anfrage die Funktion "delete\_auftrag\_id\_delete" aufruft.

In der Datei "updateR.js" wird ein Router für die Routen "/update/:id" erstellt. Dieser Router verwendet den "auftragController" für GET- und PUT-Anfragen. Eine GET-Anfrage an die Route "/update/:id" ruft die Funktion "update\_auftrag\_id\_get" auf, während eine PUT-Anfrage die Funktion "update\_auftrag\_id\_put" aufruft.

Jeder Router wird über "module.exports" exportiert, um ihn in anderen Modulen verwenden zu können. Dadurch können die Router in der Hauptanwendung eingebunden werden, um die entsprechenden Routen und Controller-Funktionen zu verarbeiten.

App.js:

Der Code beginnt damit, dass wichtige Module wie Express, Body Parser, CORS und Livereload importiert werden. Diese Module dienen dazu, den Server zu erstellen, Anfragedaten zu verarbeiten, CORS zu aktivieren und die automatische Aktualisierung der Seite zu ermöglichen.

Als nächstes wird eine Express-Anwendung initialisiert, indem eine Instanz von Express erstellt und der Port 5000 festgelegt wird. Der Port gibt an, auf welchem Port der Server eingehende Anfragen entgegennimmt.

Es werden verschiedene Middleware-Komponenten konfiguriert und verwendet. Dies umfasst das Festlegen der EJS-View Engine als Standard-View Engine, um dynamische HTML-Vorlagen zu rendern. Außerdem wird der öffentliche Ordner als statischer Dateiordner festgelegt, sodass Dateien wie CSS- oder JavaScript-Dateien statisch ausgeliefert werden können, ohne spezifische Routen dafür zu definieren. Der Body Parser wird verwendet, um Anfragedaten im JSON- und URL-codierten Format zu analysieren und im Anfrageobjekt (req.body) verfügbar zu machen. Durch die Aktivierung von CORS können Cross-Origin-Anfragen ermöglicht werden, indem die Beschränkungen der gleichen Herkunftsrichtlinie umgangen werden.

Der Auftrags-Controller wird importiert, der Funktionen enthält, um die Logik für die verschiedenen Auftragsrouten zu implementieren.

Ein Livereload-Server wird erstellt und konfiguriert, um die automatische Aktualisierung der Seite zu ermöglichen, wenn Änderungen am Quellcode vorgenommen werden. Der Server überwacht den öffentlichen Ordner und sendet eine Aktualisierungsanforderung an den Browser, wenn Änderungen erkannt werden.

Es wird eine Verbindung zur Datenbank hergestellt, wobei angenommen wird, dass der Code eine eigene Funktion (connectDB) enthält, um die Verbindung zur Datenbank aufzubauen.

Die Serverstart-Funktion wird definiert, die aufgerufen wird, um den Server zu starten. Zunächst wird eine Verbindung zur Datenbank hergestellt und dann wird auf eingehende Anfragen auf dem angegebenen Port gewartet. Wenn der Server erfolgreich gestartet wird, wird eine Konsolennachricht ausgegeben.

Es werden verschiedene Routen definiert, die auf spezifische URLs verweisen und die entsprechenden Controller-Funktionen aufrufen. Es gibt Routen wie "/", die auf "/all-articles" weiterleitet, "/all-articles", die die "index"-View-Vorlage rendert, und "/add-new-article", die die "add-new-article"-View-Vorlage rendert. Weitere Routen wie "/Suche", "/auftragInfo/:id", "/auftragArry", "/auftrag-informationen/:id", "/update/:id" und "/updateJson/:id" sind ebenfalls definiert und haben jeweils eine spezifische Controller-Funktion, die die entsprechende Logik ausführt.

Nach den Routen werden REST-API-Routen definiert, die den RESTful-Prinzipien entsprechen und auf bestimmte HTTP-Methoden und URLs reagieren. Diese Routen rufen die entsprechenden Controller-Funktionen auf, um die Logik für die API-Endpunkte zu implementieren.

Am Ende des Codes wird die Serverstart-Funktion aufgerufen, um den Server zu starten.

Es wird eine Fehlerbehandlungsroutine definiert, die auf alle nicht übereinstimmenden Routen reagiert und eine "404"-Fehlerseite rendert. Wenn ein Benutzer eine nicht definierte Route aufruft, wird ihm die "404"-Seite mit dem Titel "Page Not Found" angezeigt.

env:

.env ist eine wichtige Methode zur Konfiguration von Umgebungsvariablen in einer Anwendung. Sie bietet Sicherheit, indem sensible Informationen sicher in Umgebungsvariablen und nicht im Quellcode gespeichert werden. Dies reduziert das Risiko von Sicherheitsverletzungen. Darüber hinaus ermöglicht .env die Portabilität, da die Anwendung ohne Änderungen am Quellcode zwischen verschiedenen Umgebungen verschoben werden kann. Dies erleichtert die Entwicklung und Bereitstellung in unterschiedlichen Umgebungen.

Die Verwendung von .env bietet auch Flexibilität, da verschiedene Aspekte der Anwendung über Umgebungsvariablen konfiguriert werden können. Dies ermöglicht es Entwicklern, Einstellungen anzupassen, ohne den Code ändern zu müssen. Dadurch können Änderungen schnell vorgenommen und die Anwendung an unterschiedliche Anforderungen angepasst werden.

Ein weiterer Vorteil von .env ist die Verbesserung der Zusammenarbeit im Team. Jeder Entwickler kann seine eigenen Umgebungsvariablen in seiner lokalen .env-Datei konfigurieren, ohne die Einstellungen anderer zu beeinflussen. Dies fördert die Zusammenarbeit und ermöglicht es jedem Teammitglied, die Anwendung in seiner eigenen Umgebung auszuführen, ohne Konflikte zu verursachen.

Postman:

Postman ist ein vielseitiges Tool, das in der API-Entwicklung eine zentrale Rolle spielt. Es bietet Entwicklern eine benutzerfreundliche Oberfläche, um APIs schnell zu erstellen, zu testen und zu dokumentieren. Mit Postman können detaillierte API-Dokumentationen erstellt werden, um eine klare Kommunikation und ein besseres Verständnis der Funktionalität zu ermöglichen. Das Testen von APIs wird mit Postman einfach und effektiv, da Entwickler Anfragen mit verschiedenen Parametern senden und automatisierte Tests durchführen können. Die Zusammenarbeit wird durch die Möglichkeit, Arbeitsergebnisse zu teilen, gemeinsam an Sammlungen zu arbeiten und Änderungen zu verfolgen, verbessert. Postman bietet auch Funktionen zum Debuggen und zur Fehlerbehebung von APIs, indem Entwicklern Einblicke in den Request- und Response-Verlauf sowie Tools zur Überwachung von Fehlern geboten werden. Insgesamt ist Postman ein unverzichtbares Werkzeug, das die Effizienz, Qualität und Zusammenarbeit in der API-Entwicklung steigert.

datenbank:



Abbildung Relation Model



ERR Diagram

**Tabelle "Auftrag":**

Die Tabelle "Auftrag" enthält Informationen über Aufträge, wie die Auftrags-ID, Kunden-ID, Lieferdatum, Anzahl, Teil-ID, Ankommende\_Teile und Vormerkung.

Es besteht eine 1:n-Beziehung zwischen der Tabelle "Auftrag" und der Tabelle "Produkt". Das bedeutet, dass ein Auftrag mehrere Produkte enthalten kann, während ein Produkt immer genau einem Auftrag zugeordnet ist. Dies wird durch die Spalte "Auftrag\_Id" in der Tabelle "Produkt" erreicht, die als Fremdschlüssel auf die Auftrags-ID in der Tabelle "Auftrag" verweist.

**Tabelle "Mitarbeiter":**

Die Tabelle "Mitarbeiter" enthält Informationen über Mitarbeiter, wie die Mitarbeiter-ID, Vorname, Nachname, Abteilungs-ID, E-Mail und Passwort.

Es besteht eine n:1-Beziehung zwischen der Tabelle "Mitarbeiter" und der Tabelle "Abteilung". Das bedeutet, dass ein Mitarbeiter einer bestimmten Abteilung zugeordnet ist, während eine Abteilung mehrere Mitarbeiter haben kann. Dies wird durch die Spalte "Abteilung\_id" in der Tabelle "Mitarbeiter" erreicht, die als Fremdschlüssel auf die Abteilungs-ID in der Tabelle "Abteilung" verweist.

**Tabelle "Abteilung":**

Die Tabelle "Abteilung" enthält Informationen über Abteilungen, wie die Abteilungs-ID und den Abteilungsnamen.

Es besteht eine 1:n-Beziehung zwischen der Tabelle "Abteilung" und der Tabelle "Prozess". Das bedeutet, dass eine Abteilung mehrere Prozesse haben kann, während ein Prozess immer genau einer Abteilung zugeordnet ist. Dies wird durch die Spalte "AbteilungId" in der Tabelle "Prozess" erreicht, die als Fremdschlüssel auf die Abteilungs-ID in der Tabelle "Abteilung" verweist.

**Tabelle "Prozess":**

Die Tabelle "Prozess" enthält Informationen über Prozesse, wie die Prozess-ID, Aufgabenbeschreibung und Abteilungs-ID.

**Tabelle "produkt\_prozess" :**

Die Tabelle "produkt\_prozess" dient dazu, die Beziehung zwischen Produkten und Prozessen darzustellen. Hier sind die Details:

Spalte "ProduktId": Dies ist ein Primärschlüssel (PK) und dient als Fremdschlüssel zur Tabelle "Produkt". Jeder Eintrag in der Tabelle "produkt\_prozess" ist einem bestimmten Produkt zugeordnet.

Spalte "ProzessId": Dies ist ebenfalls ein Primärschlüssel (PK) und dient als Fremdschlüssel zur Tabelle "Prozess". Jeder Eintrag in der Tabelle "produkt\_prozess" ist einem bestimmten Prozess zugeordnet.

Spalte "Status": Dies ist eine Aufzählungsspalte (enum) mit den Werten 'fertig' und 'open'. Sie gibt den Status der Beziehung zwischen dem Produkt und dem Prozess an. Ein Wert von 'fertig' bedeutet, dass der Prozess für das Produkt abgeschlossen ist, während 'open' bedeutet, dass der Prozess noch nicht abgeschlossen ist.

Spalte "Enddatum": Dies ist eine Datumsangabe und gibt das Enddatum des Prozesses für das jeweilige Produkt an.

Mongodb:

MongoDB bietet viele Vorteile für das Workflow-Management:

Mit dem dokumentenorientierten Datenmodell von MongoDB kannst du komplexe Workflow-Strukturen flexibel abbilden. Du kannst hierarchische Datenstrukturen verwenden, um Workflow-Schritte, Zuständigkeiten, Benutzerinformationen und andere relevante Daten in einer einzigen Dokumentenstruktur zu speichern. Im Vergleich zu traditionellen relationalen Datenbanken mit starren Tabellenstrukturen bietet dies mehr Flexibilität bei der Definition und Anpassung von Workflows.

MongoDB ist für horizontale Skalierbarkeit ausgelegt, was bedeutet, dass du die Leistung und Kapazität deiner Datenbank durch das Hinzufügen von zusätzlichen Servern oder das Erstellen eines MongoDB-Clusters erhöhen kannst. Dies ist besonders wichtig, wenn du mit einer großen Anzahl von Workflows oder vielen gleichzeitigen Benutzern arbeitest, da MongoDB die Last auf mehrere Server verteilen kann.

MongoDB bietet schnelle Leistung durch Indexierung, Caching und die Möglichkeit, Daten im Arbeitsspeicher zu halten. Dadurch werden Lese- und Schreibzugriffe auf Workflow-Daten beschleunigt und die Reaktionszeit deiner Anwendung verbessert. Darüber hinaus bietet MongoDB Funktionen wie Sharding, um große Datenmengen effizient zu verarbeiten.

MongoDB verfügt über ein leistungsstarkes Aggregation Framework, das erweiterte Analyse- und Aggregationsoperationen auf Workflow-Daten ermöglicht. Du kannst komplexe Abfragen und Aggregationen durchführen, um beispielsweise den Fortschritt von Workflows zu analysieren, Metriken zu berechnen oder Daten für Berichte zu aggregieren. Dadurch kannst du wertvolle Erkenntnisse aus deinen Workflow-Daten gewinnen.

MongoDB bietet eine Vielzahl von Treibern und Integrationsoptionen für verschiedene Programmiersprachen und Frameworks. Dadurch wird die Integration von MongoDB in deine bestehende Workflow-Management-Anwendung oder in andere Tools und Systeme, die du möglicherweise verwendest, erleichtert. Du kannst MongoDB problemlos in deine Workflow-Management-Pipeline integrieren und von den Vorteilen einer dokumentenorientierten Datenbank profitieren.

db Folder :

im Code wird das Modul dotenv verwindet, um die Umgebungsvariablen aus der .env-Datei zu laden. Diese Datei enthält sensible Informationen wie den Benutzernamen und das Passwort für den Zugriff auf die MongoDB-Datenbank. Durch die Verwendung von dotenv können diese Informationen sicher gespeichert werden, ohne sie direkt im Quellcode zu hinterlegen.

Nach dem Laden der Umgebungsvariablen wird das Modul mongoose importiert. Mongoose ist ein ODM (Object-Document Mapping) für MongoDB in Node.js und ermöglicht die Interaktion mit der Datenbank.

Der Code deklariert dann Variablen für den Benutzernamen, das Passwort und den Datenbanknamen. Dabei werden die Umgebungsvariablen verwendet, die zuvor aus der .env-Datei geladen wurden. Die Variablen werden mit encodeURIComponent codiert, um eventuelle Sonderzeichen zu berücksichtigen.

Anschließend wird die Verbindungszeichenfolge zur MongoDB-Datenbank erstellt. Dabei werden der codierte Benutzername, das codierte Passwort, der Datenbankname und weitere Verbindungsoptionen in die Zeichenfolge eingefügt. Die Verbindungszeichenfolge basiert auf dem MongoDB-Verbindungsmuster und enthält Informationen wie den Hostnamen und die Optionen für das Schreiben von Daten.

Die Funktion connectDB wird definiert, die eine Verbindung zur MongoDB-Datenbank herstellt. Sie verwendet die mongoose.connect()-Methode und übergibt die zuvor erstellte Verbindungszeichenfolge als Argument. Die Funktion gibt ein Promise-Objekt zurück, das den Verbindungsstatus repräsentiert. Durch den Rückgabewert des Promises können andere Teile der Anwendung auf den Verbindungsstatus zugreifen und entsprechende Aktionen durchführen, wenn die Verbindung erfolgreich hergestellt wurde oder ein Fehler aufgetreten ist.

Schließlich wird die Funktion connectDB über module.exports exportiert, um sie in anderen Modulen verwenden zu können. Dadurch kann die Funktion an anderer Stelle in der Anwendung aufgerufen werden, um eine Verbindung zur MongoDB-Datenbank herzustellen und mit der Datenbank zu interagieren.

ressourcen:

<https://expressjs.com/en/starter/static-files.html>

<https://bytearcher.com/articles/refresh-changes-browser-express-livereload-nodemon/>

<https://www.w3schools.com/jsref/default.asp>

<https://www.npmjs.com/package/mongoose>

<https://www.w3schools.com/html/html_forms_attributes.asp>

<https://www.w3schools.com/tags/ref_httpmethods.asp>

<https://expressjs.com/en/starter/basic-routing.html>

<https://mongoosejs.com/docs/documents.html#updating-using-save>

<https://mongoosejs.com/docs/api/model.html#model_Model.find>

<https://mongoosejs.com/docs/api/model.html#model_Model.findById>

<https://expressjs.com/en/4x/api.html#req.params>

<https://mongoosejs.com/docs/api/model.html#model_Model.findByIdAndDelete>

<https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch>

<https://expressjs.com/en/starter/basic-routing.html>

<https://expressjs.com/en/4x/api.html#req.params>

<https://expressjs.com/en/guide/routing.html>

<https://developer.mozilla.org/en-US/docs/Glossary/MVC>

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes>