

# DB2 Projekt – Hochschule Esslingen

Erstellt von Selim Cetin

17.01.2024

## Inhaltsverzeichnis

1 DB2 Projekt Einleitung.....	2
2 Datenbank Struktur.....	2
2.1 Tabellen.....	2
2.1.1 Veterinary Practices.....	2
2.1.2 Customers.....	2
2.1.3 Pets.....	2
2.2 Stored Procedures.....	3
3 Database Triggers.....	4
4 Functions.....	4
5 Backend.....	5
5.1 server.js.....	5
6 Frontend.....	6
6.1 main.jsx.....	6

# 1 DB2 Projekt Einleitung

Diese Dokumentation dient dazu, einen Überblick über das DB2 Software-Projekt zu geben. Das Ziel des Projekts besteht darin, das erlangte Wissen über Datenbanken in Code umzusetzen. Die Programmiersprache für sowohl Backend als auch Frontend ist JavaScript. Das Backend wurde mithilfe von Node.js und Express.js umgesetzt. Das Frontend wurde unter Verwendung von React.js und der React-Router-Dom-Komponente realisiert. Als Datenbankserver wurde ein Microsoft SQL Server verwendet, der T-SQL als Dialekt verwendet. Als Beispielkontext werden in der Webapplikation mehrere Arztpraxen mit ihren Kunden und Haustieren dargestellt.

## 2 Datenbank Struktur

Der SQL Server hat zwei Datenbanken, AuditTrail und veterinary. Die AuditTrail hat die gleiche Anzahl an Tabellen, die die veterinary Datenbank auch hat. Sie hat die Aufgabe, alle Änderungen, dass heißt Hinzufügen, Ändern oder Löschen, der Tabellen wie ein Log zu speichern.

Die veterinary Datenbank hat drei Tabellen: Praxen, Kunden und Haustiere.

### 2.1 Tabellen

#### 2.1.1 Veterinary Practices

Column Name	Data Type	Allow Nulls
PracticeID	int	<input type="checkbox"/>
PracticeName	varchar(255)	<input type="checkbox"/>
Location	varchar(255)	<input type="checkbox"/>
Street	varchar(255)	<input type="checkbox"/>

#### 2.1.2 Customers

Column Name	Data Type	Allow Nulls
CustomerID	int	<input type="checkbox"/>
PracticeID	int	<input checked="" type="checkbox"/>
CustomerName	varchar(255)	<input type="checkbox"/>
Email	varchar(255)	<input checked="" type="checkbox"/>
Phone	varchar(15)	<input checked="" type="checkbox"/>

### 2.1.3 Pets

Column Name	Data Type	Allow Nulls
PetID	int	<input type="checkbox"/>
CustomerID	int	<input checked="" type="checkbox"/>
PetName	varchar(255)	<input type="checkbox"/>
Species	varchar(50)	<input checked="" type="checkbox"/>
BirthDate	date	<input checked="" type="checkbox"/>
IsAlive	bit	<input type="checkbox"/>

## 2.2 Stored Procedures

Die Stored Procedures wurden zum Erstellen von Tabellen Einträgen benutzt. Hierzu gibt es drei Stück die im Code ausgeführt werden:

- dbo.sp\_CreateVeterinaryPractice
- dbo.sp\_CreateCustomer
- dbo.sp\_CreatePet

Hier ist ein Auszug eines Stored Procedures als Beispiel:

Die SCOPE\_IDENTITY Funktion wurde dafür genutzt, eine fortlaufende ID zu generieren.

```
ALTER PROCEDURE [dbo].[sp_CreatePet]
    @CustomerID INT,
    @PetName VARCHAR(255),
    @Species VARCHAR(50),
    @BirthDate DATE,
    @IsAlive BIT
AS
BEGIN
    -- Insert into Pets table
    INSERT INTO dbo.Pets (CustomerID, PetName, Species, BirthDate, IsAlive)
    VALUES (@CustomerID, @PetName, @Species, @BirthDate, @IsAlive);

    -- Get the newly generated PetID
    DECLARE @NewPetID INT;
    SET @NewPetID = SCOPE_IDENTITY();

    -- Output the generated PetID if needed
    SELECT @NewPetID AS NewPetID;
END
```

## 3 Database Triggers

Trigger wurden für jede Tabelle genutzt, um die AuditTrail Datenbank bei Änderungen der Tabelle mit Einträgen zu füllen. Hier ist ein Auszug eines Triggers:

```
ALTER TRIGGER [dbo].[trg_AuditTrailVeterinary]
ON [dbo].[VeterinaryPractices]
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- Perform auditing actions here, logging changes to the AuditTrail table
    INSERT INTO AuditTrail.dbo.AuditTrail (TableName, Action, [Timestamp], UserId, OldValues, NewValues)
    SELECT 'dbo.VeterinaryPractice',
        CASE
            WHEN EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted) THEN 'UPDATE'
            WHEN EXISTS (SELECT * FROM inserted) THEN 'INSERT'
            WHEN EXISTS (SELECT * FROM deleted) THEN 'DELETE'
        END,
        GETDATE(),
        SYSTEM_USER,
        (SELECT * FROM deleted FOR JSON AUTO),
        (SELECT * FROM inserted FOR JSON AUTO);
END
```

## 4 Functions

Als Function wurde eine Altersermittlung der Haustiere umgesetzt. Der Code sieht wie folgt aus:

```
ALTER FUNCTION [dbo].[CalculatePetAge](@Birthdate DATE)
RETURNS INT
AS
BEGIN
    DECLARE @Age INT

    -- Calculate the age based on the difference between the current date and the birthdate
    SET @Age = DATEDIFF(YEAR, @Birthdate, GETDATE()) -
        CASE
            WHEN MONTH(@Birthdate) > MONTH(GETDATE()) OR
                (MONTH(@Birthdate) = MONTH(GETDATE()) AND DAY(@Birthdate) > DAY(GETDATE()))
            THEN 1
            ELSE 0
        END

    RETURN @Age
END
```

Hier eine Beispielrückgabe:

```
SELECT BirthDate, dbo.CalculatePetAge(BirthDate) AS Age
FROM dbo.Pets WHERE PetID=3;
```

	BirthDate	Age
1	2018-05-10	5

## 5 Backend


Im Root-Verzeichnis sind folgende Komponente zu finden:

- .env: hier sind Umgebungsvariablen definiert die nicht auf Github committed werden. Zum Beispiel sind hier die Login-Daten für das SQL-Server hinterlegt
- package.json: hier findet man generelle Infos zum Projekt. Zum Beispiel welche Datei als ausgeführt wird oder welche Pakete installiert sind.

Im src-Ordner sind folgende Komponente zu finden:

- Main-/Hauptdatei: die als erstes ausgeführte Datei ist die server.js Datei
- Controllers: sind dafür da Daten vom SQL-Server abzufragen
- Middlewares: sind für das Abarbeitung der Anfragen auf dem Backend zuständig. Je nach Pfad werden unterschiedliche Middlewares ausgeführt. Die Middlewares beinhalten die Logik, die bei einer Anfrage bearbeitet wird.
- Routes: sind für die modularisierte Darstellung von Pfaden da. Man kann die abrufbaren Pfade des Backends in Route-Dateien auslagern, damit die Hauptdatei server.js übersichtlich bleibt.
- Schemas: sind für die Validierung der Daten bei den Anfragen zuständig. Beispielsweise wird überprüft, dass keine PracticelD bei einer CreatePet Anfrage dabei ist.
- Utils: ist als Unterstützung der Codebasis gedacht. Hilft den Code übersichtlich zu behalten.

### 5.1 server.js

```
db2_project > backend > src >  server.js > ...
1  const express = require("express");
2  const { connectToMysqlDatabase, pool, sql } = require("../utils/sqlConnection");
3  const { errorHandler } = require("../middlewares/errorHandler");
4  const { logger } = require("../middlewares/logger");
5
6  const app = express();
7  app.use(express.json()); // needed for using req.body in middlewares
8
9  const customerRouter = require("../routes/customers");
10 const petRouter = require("../routes/pets");
11 const veterinaryPracticeRouter = require("../routes/veterinaryPractices");
12
13 connectToMysqlDatabase();
14
15 app.use(logger);
16 app.use("/api/customer", customerRouter.router);
17 app.use("/api/pet", petRouter.router);
18 app.use("/api/veterinary_practice", veterinaryPracticeRouter.router);
19 app.use(errorHandler);
20
21 app.listen(process.env.PORT, () => {
22   console.log(`Server is listening at http://localhost:${process.env.PORT}`);
23 });
24
```

## 6 Frontend

Im Root-Verzeichnis sind folgende Komponente zu finden:

- Main-/Hauptdatei: die als erstes ausgeführte Datei ist die main.jsx
- package.json: hier findet man generelle Infos zum Projekt. Zum Beispiel welche Datei als ausgeführt wird oder welche Pakete installiert sind.

Im src-Verzeichnis sind folgende Komponente zu finden:

- Components: beinhaltet alle React komponente. Diese dienen zur Modularität des Codes.
- Constants: hier sind konstante Werte die sich nicht ändern. Es dient als zentrale Stelle für Werte wie z.B. API-Pfade oder ID-Spaltennamen der Tabellen. Bei zukünftigen Änderungen macht dies die Anpassung des Codebases einfacher.
- Context: hier sind die React-Contexts definiert, die verwendet werden. Contexts sind wie globale Variablen zu denken, sie sind überall da verfügbar, wo die Provider bereitgestellt sind.
- Controllers: noch nicht viel benutzt, sollte zukünftig aber als zentrale Stelle für die Datenbearbeitung genutzt werden.
- Hooks: aktuell gibt es nur Hooks für die Contexts. Die Hooks geben ein Fehler, wenn im JSX-Code die Provider falsch oder gar nicht platziert sind.
- Layouts: hier gibt es aktuell nur das RootLayout. Wird im Zusammenhang mit React-Router-Dom verwendet.
- Pages: Die einzelnen Seiten die gerendert werden. Wird ebenfalls in Zusammenhang mit React-Router-Dom verwendet.
- Reducers: ist ein Hook von React für komplexes State-Management mit Dispatch-Funktion. Man kann unterschiedliche Actions für das State-Management definieren und darüber unterschiedliche State-Manipulationen durchführen.
- Utils: Hilfsfunktionen für die Codebase.

### 6.1 main.jsx

```
db2_project > react-db2 > src > main.jsx
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import App from "../App.jsx";
4  import "@mantine/core/styles.css";
5  import { MantineProvider } from "@mantine/core";
6  import DataContextProvider from "../context/DataContextProvider.jsx";
7
8  ReactDOM.createRoot(document.getElementById("root")).render(
9    <React.StrictMode>
10     <MantineProvider>
11       <DataContextProvider>
12         <App />
13       </DataContextProvider>
14     </MantineProvider>
15   </React.StrictMode>
16 );
```