FH JOANNEUM
University of Applied Sciences

Journaling Desktop-App und gRPC-Server

Projektarbeit

FH JOANNEUM (University of Applied Sciences), Kapfenberg

Betreuer: DI (FH) Michael Ulm, MA Eingereicht von:
Michael Miihlberger Personenkennzahl:52204344

28.01.2025

Inhaltsverzeichnis

Ausgangssituation / Problemstellung:

Ziel der Projektarbeit ist es im Front-End eine Electron Desktopanwendung mit React und im Back-End eine .NET Server Anwendung zu erstellen, welche vom Benutzer eingegebenen Daten speichert.

Bei der Desktopanwendung handelt sich um ein Journaling Programm.
Der Benutzer soll seine persönlichen Gedanken, Erlebnissen, Reflexionen, ... in Tagebuchform festhalten können. Außerdem soll es möglich sein Reflexionsfragen festzulegen die dem Benutzer in wiederholenden Abständen abgefragt werden. Der Benutzer kann diese Fragen an den eingestellten Tagen beantworten.

Sowohl die Antworten der Reflexionsfragen auch als auch die Journalingbeiträge sollen mithilfe einer Serveranwendung in einer Datenbank abgespeichert werden können und beim erneuten Aufrufen der Desktopanwendung an den jeweils gespeicherten Tagen angezeigt werden.

Motivation:

Die Hauptmotivation der Projektarbeit ist es Erfahrungen mit den Programmiersprachen JavaScript / TypeScript zu sammeln und das Einarbeiten in das Framework React.
Da mit diesen Technologien noch nicht viel gearbeitet wurde, aber feste Bestandteile im IT – Bereich sind, besteht die Motivation sich in diese Themen einzuarbeiten.

Außerdem wurde die Projektarbeit genutzt um sich mehr in das Thema Client / Server Kommunikation und der gRPC Technologie und deren Umsetzung zu befassen.

Mögliche Lösungsansätze:

1. Verwendung einer RESTful API für die Client/Server-Kommunikation
Vorteile: straight-foward, einfache Implementierung, etablierter Standard mit viel Dokumentation und Beispielen, Vorwissen
Nachteil: Overhead durch JSON, nicht typsicher
2. Verwendung von GraphQL für die Client/Server-Kommunikation
Vorteil: Effizienter als REST, nur Daten die angefordert werden, werden gesendet, flexibel, typisiert
Nachteil: Komplexer als REST, wenig Vorwissen
3. Verwendung von gRPC für die Client/Server-Kommunikation
Vorteil: sehr effizient, typisiert, Interesse mehr über Technologie zu lernen
Nachteil: weniger Vorwissen, eventuell Probleme mit Electron Umgebung

Implementierung:

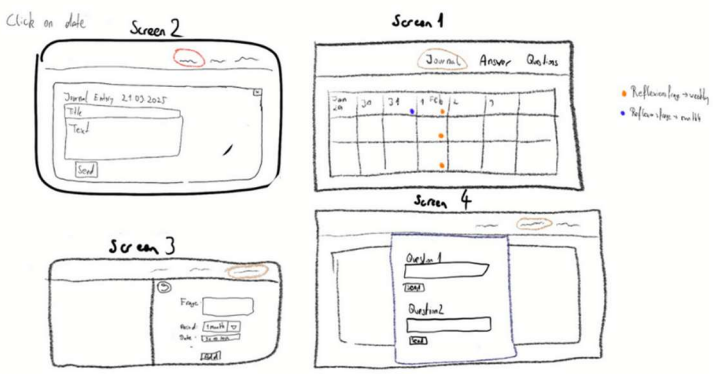
Repository der Projektarbeit: https://github.com/michaawl/Journal

Planung:

Das Projekt wurde damit begonnen, in dem eine grobe Einteilung der Vorgehensweise der Implementierung geplant wurde:

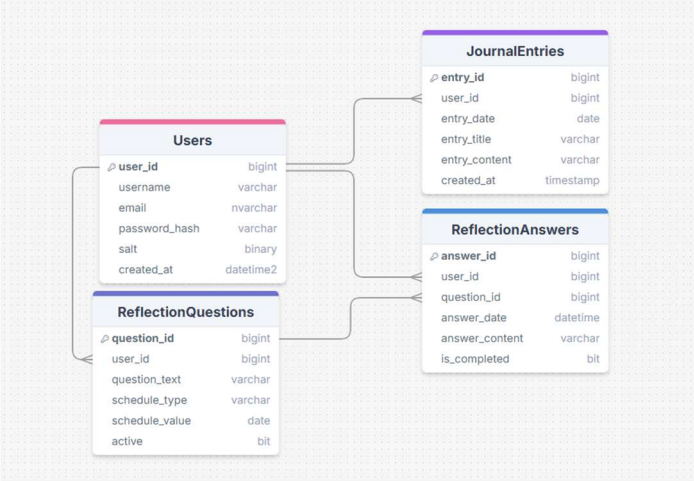
1. Konzipieren des User Interfaces des Clients und der API – des Servers
2. Programmieren des Clients und einarbeiten in die Technologie
3. Konzipieren des Servers, der Datenbank und Auswahl der Technologien
4. Programmieren des Servers und der Schnittstelle im Client

Das User Interface wurde mithilfe eines Pen & Paper Prototyps grob konzipiert, damit die grundlegende Struktur und Benutzerführung vor der eigentlichen Implementierung evaluiert und gegebenenfalls angepasst werden kann.



Datenbankschema:

Folgendes Datenbankschema wurde definiert und in SQL Server umgesetzt. Das Create Skript kann im Repository gefunden werden.



Client:

Recherche der Technologien:

Electron:
Um die Desktopanwendung zu erstellen wurde Electron verwendet. Electorn ist ein kostenloses open source Softwareframework, das dazu genutzt wird Desktopanwendungen mit Web Technologien (JS, HTML, CSS) zu erstellen.
Das Framework wurde für die Projektarbeit gewählt, da es sehr etabliert ist und somit viel Dokumentation und Foreneinträge online zur Verfügung stellt.
Electron packt die Anwendung mit einem Chromium-Browser und Node.js in eine ausführbare Datei. Dies macht sie eigenständig und einfach zu verteilen.

Funktion von Electron:
Electorn startet zwei Prozesse, einen Main Prozess und Render Prozesse. Die beiden Prozesse können mittels IPC (Inter Process Communication) miteinander kommunizieren.

Main Prozess: Der Main Prozess läuft als eigenständer node.js Prozess und startet z.B. das index.js. Dieser Prozess ist für die Logik verantwortlich. Nur der Main Prozess kann auf die Systemressourcen zugreifen und mit BrowserWindow neue Render Prozesse erstellen.

Render Prozess: Der Render Prozess läuft in einem Chromium Browser. Jedes Fenster läuft in einem eigenen Renderprozess, indem die Benutzeroberfläche mittels Web Technologien in der Chromium Browser Engine gerendert werden.

React:
React ist eine der meist verwandtesten Open Source JavaScript Library die verwendet wird um Benutzeroberflächen zu erstellen. Sie wird häufig für den Aufbau von Single-Page-Applications verwendet, bei der die Benutzeroberfläche dynamisch und effizient aktualisiert wird, ohne dass die gesamte Seite neu geladen werden muss.

Hauptbestandteile von React:

Komponenten: Eine React Anwendung besteht aus Components, die wie zusammengefügt werden können. Components sind wiederverwendbare UI-Elemente die mithilfe von JS, HTML und CSS definiert werden.

Virtuelles DOM: React benutzt ein virtuelles DOM (Document Object Model), das nicht bei jeder Änderung das komplette DOM neu lädt sondern immer nur die geänderten

Elemente, was die Performance steigert. Das virtuelle DOM ist eine Kopie des echten DOMs, die vollständig in JavaScript verwaltet wird.

Zustandsbasiertes Rendering:

Bei React kann die Benutzeroberfläche basierend auf dem Zustand der Anwendung beschrieben werden. Ändert sich der Zustand, aktualisiert React automatisch die Ansicht.

Beispiel:
Counter ist die Komponente.
count ist der State und setCount ist die Funktion, welche den State ändern kann, danach wird der jeweilige Teil des DOMs neu gerendert. useState mit Initialwert 0 ist der State Hook der verwendet wird um Zustände zu definieren. Dieses Konzept ist in React sehr verbeitet.

```
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
      <div>
          <p>Count: {count}</p>
          <button onClick={() => setCount(count + 1)}>Increase</button>
      </div>
  );
}
```

Bootstrap und Tailwind.css:
Bootstrap und Tailwind.css mit DaisyUI sind CSS Bibliotheken die für ein einfacheres Styling verwendet werden können. Im Rahmen der Projektarbeit wurden sich die Bibliotheken genauer angeschaut und in das Projekt integriert.

Implementierung:

Zustandsdiagramm und Funktion des Clients:

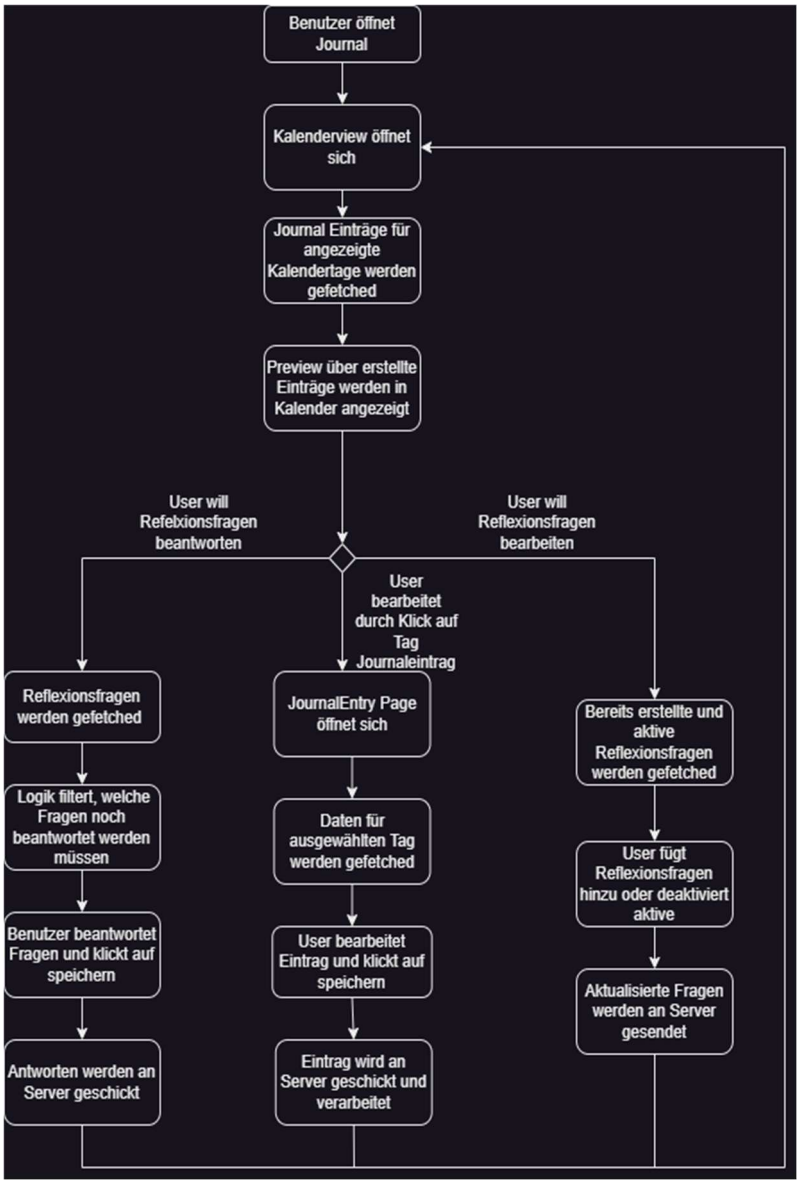
Wird die Desktopanwendung geöffnet werden Daten vom Server gefetched und die Ansicht der jeweiligen Kalendertage wird angepasst.

Der Benutzer hat grundsätzlich 3 Optionen:

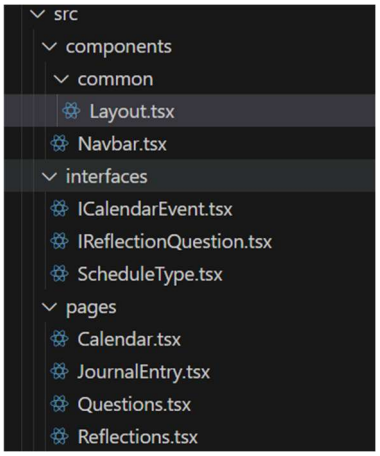
1. es wird auf einen Kalendertag geklickt, nun kann entweder ein neuer Journaleintrag erstellt oder ein bereits erstellter bearbeitet werden. Sollten für den jeweiligen Tag bereits Reflexionsfragen beantwortet worden sein, werden diese ebenfalls hier angezeigt

2. Es wird auf den „Reflections“ Button geklickt. Hier werden alle Reflection Questions geladen und eine Logik ermittelt, welche für den jeweiligen Tag vorgesehen sind, und noch nicht beantwortet wurden. Der Benutzer kann diese hier beantworten.

3. Es wird auf den „Manage Questions“ geklickt. Hier können neue Reflexionsfragen erstellt oder bestehende deaktiviert werden.



Die Componenten der React Anwendunge haben an sich folgende Struktur. Der Code wird genauer im Repository beschrieben. Anschließend folgt eine kurze Übersicht der Hauptkomponenten.



Layout:

Die Layout Komponente ist ein übergeordnetes Wrapper Layout. Es bindet die NavBar und die dynamischen Componenten (pages) ein. Das Routing der dynmischen pages ist im index.tsx definiert.

Calender:
Das Calender Component ist die Startseite. Es bindet die FullCalender Library ein, welche für das Projekt genutzt wurde um Kalender und Kalenderevents einzubinden. Ein Klick auf ein Datum führt in die JournalEntry Component.

JournalEntryComponent:
Hier können Journal posts bearbeitet oder gepostet werden, bzw. beantwortete Relexionsfragen angezeigt werden.

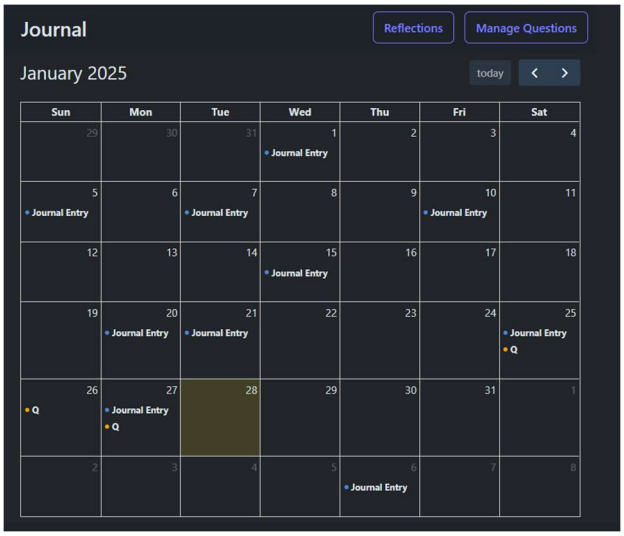
Questions:
Hier können Reflexionsfragen hinzugefügt bzw. deaktiviert werden.

Reflections:
Hier können Reflexionsfragen, welche für den heutigen Tag vorhergesehen sind beantwortet werden.

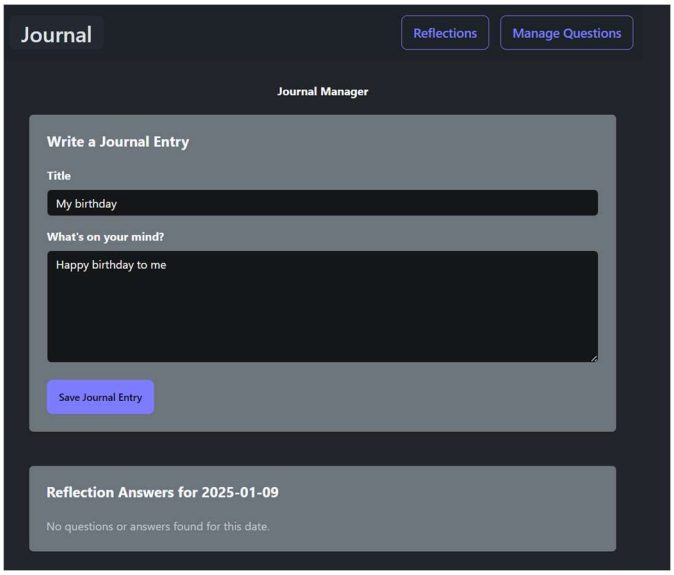
Services:
Es wurden jeweils ein journal service und ein reflection service definiert. Welche die jeweiligen Funktionalitäten für die jeweiligen Anwendungsfälle bereitstellen.

Fertige Desktopanwendung:

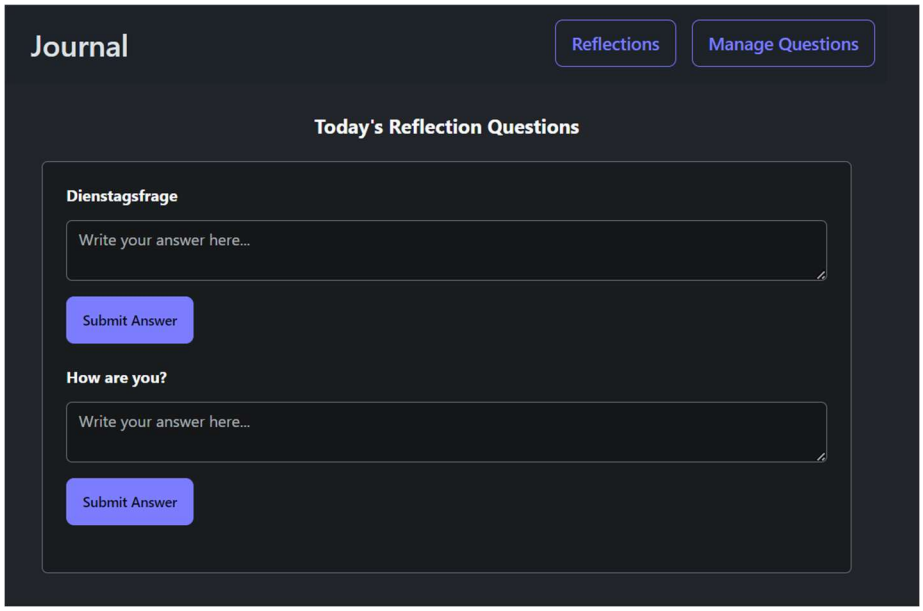
Calender:



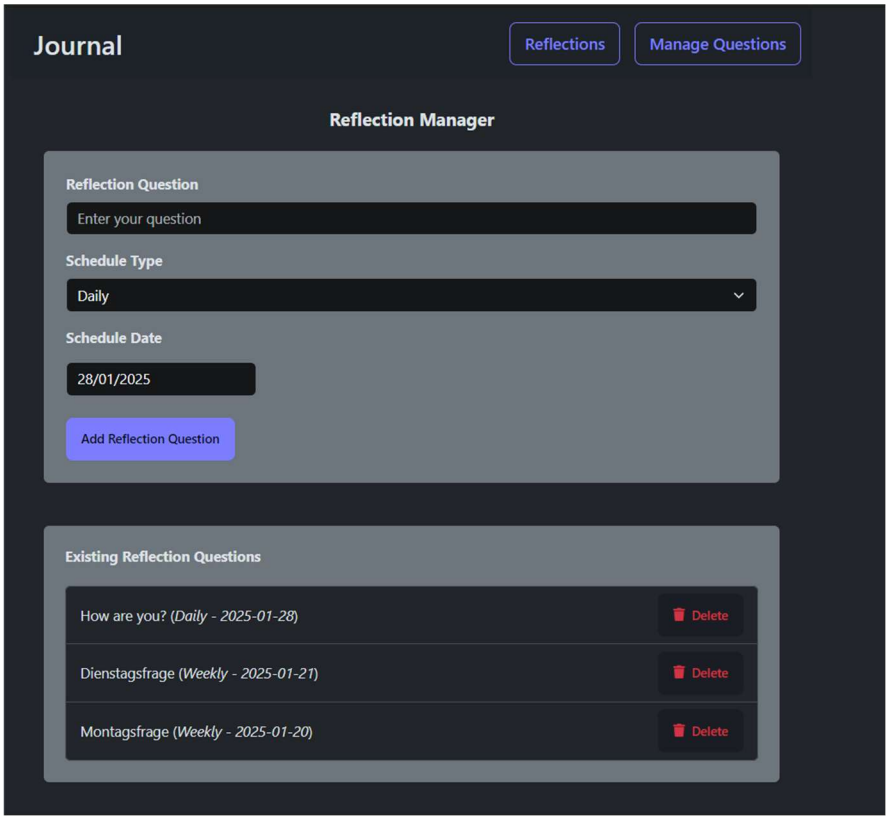
Journal Manager:



Today’s Reflection Questions:



Reflection Manager:



Server:

Der Server wurde mittels einer .NET Anwendung umgesetzt welcher mit gRPC mit dem Client kommuniziert.
Hauptaufgabe des Servers ist es, Informationen in der Datenbank zu speichern und bereits gespeicherte Daten aufzubereiten und dem Client zur Verfügung zu stellen.

Recherche (gRPC):

gRPC (google Remote Procedure Call) ist eine von Google entwickelte Technologie, welche Effizient RPC durchführt. gRPC wird vorallem für MicroService Anwendungen verwendet.
Vorteile von gRPC gegenüber REST basierten Technologien ist die höhere Performance und niedrigere Latenz, streaming support und dass die übertragenen Daten strongly typed sind.
Das gRPC Framework unterstützt die Mehrheit der etablierten Programmiersprachen.

Protopuf File:
Die Übertragung findet bei gRPC mittels Protopuf Files statt. Diese können als Äquivalent von JSON Files bei REST Anwendungen gesehen werden.
Die zu übertragenen Daten werden dabei binär kodiert gesendet, sodass so wenig Overhead wie möglich zustande kommt.
In den Protopuf files werden zum einen die zu übertragenden Nachrichten als auch die zur Verfügung stehenden Services definiert. Aus diesen Daten werden anschließend Objekte und Interfaces generiert. Die Interfaces müssen dann jeweils am Server implementiert werden.
Durch die Protopuf files wird somit auch eine Art Dokumentation zur Verfügung gestellt, aus welcher ersichtlich ist, welche Services und Objekte die API enthält.
Das Protopuf files muss sich sowohl auf dem Server als auch auf dem Client befinden, und auf beiden System müssen die sich darin befindenden Klassen mittels einem command (siehe README.md) generiert werden.

Da die Übertragung mittels HTTP/2 Funktionen stattfindet, welche von vielen Browsern nicht unterstützt werden muss bei der Kommunikation zwischen Browsern und gRPC ein API Gateway bzw. Proxy verwendet werden, welcher auf das gRPC Web Protokoll übersetzt (HTTP1.1 bzw. eingeschränktes http/2), damit die Daten vom Browser interpretiert werden können.

Implementierung:

Folgende Services wurden im .proto definiert und anschließend in der jeweiligen Service Klasse implementiert (die genaue Implementierung und Details der Messages befinden sich im Repository)

journal.proto:

Services die Funktionen für Journal-Funktionen bereitstellen

```
service Journal {

  rpc GetJournalEntries(GetJournalEntriesRequest) returns (GetJournalEntriesReply);

  rpc PostJournalEntry(PostJournalEntryRequest) returns (PostJournalEntryReply);

  rpc GetUsers(GetUsersRequest) returns (GetUsersReply);
}
```

GetJournalEntries: liefert alle Journal Einträge zurück
PostJournalEntry: postet Journal Entry in Datenbank.
GetUsers: lifert alle Benutzer (wurde im Rahmen der Projektarbeit nicht verwendet)

reflection.proto

Services die Funktionen für Reflection-Funktionen bereitstellen

```
service Reflection {
  rpc GetReflectionQuestions(GetReflectionQuestionsRequest) returns (GetReflectionQuestionsReply);

  rpc GetReflectionAnswerByQuestionId(GetReflectionAnswerByQuestionIdRequest) returns (GetReflectionAnswerByQuestionIdReply);

  rpc PostReflectionQuestion(PostReflectionQuestionRequest) returns (PostReflectionQuestionReply);

  rpc PostReflectionAnswer(PostReflectionAnswerRequest) returns (PostReflectionAnswerReply);

  rpc UpdateReflectionQuestionActiveStatus(UpdateReflectionQuestionActiveRequest) returns (UpdateReflectionQuestionActiveReply);

 rpc GetReflectionAnswersByDate(GetReflectionAnswersByDateRequest) returns (GetReflectionAnswersByDateReply);}
```

GetReflectionAnswerByQuestionId: Liefert alle Antworten einer bestimmten QuestionId zurück.
PostReflectionQuestion: Speichert eine Reflexionsfrage.
PostReflectionAnswer: Speichert eine Antwort auf eine Frage.
UpdateReflectionQuestionActiveStatus: Für die Aktivierungs bzw. Deaktivierung einer Reflektionsfrage
GetRelctionAnswerByDate: Liefert alle Antworten auf Reflectionsfragen eines bestimmten Datums zurück.

Known Problems:

Keine Vorkenntnisse zu React und wenige Vorkenntnisse zu gRPC
Problem: es wurden nur wenige kleinere Projekte mit Webtechnologien umgesetzt und noch keine mit React. Es wurde bereits ein kleineres Projekt mit gRPC umgesetzt, jedoch noch keines in dem Web Technologien beinhaltet waren.
Lösung: Recherche bevor Implementierung

Unerwartetes Verhalten in Electron-Umgebung
Problem: gRPC funktioniert nativ nicht in den meisten Browsern, da http/2 Funktionen verwendet werden die von Browsern nicht unterstützt werden. Da in der Projektarbeit Webtechnologien verwendet werden, ist das Verhalten an sich noch unerwartet.
Lösung: Recherche und Suche nach ähnlichen Projekten mit gRPC und Electron.

Ausblick und Fragestellung der Bachelorarbeit :

Einleitung
Bei Echtzeitanwendungen wie Chat-Applikationen oder Streaming Diensten werden oft große Datenmengen ausgetauscht. Dabei ist eine effiziente Übertragung der Daten essenziell, damit die Latenz so klein wie möglich bleibt.
Eine Methode um dies in einem Softwareprojekt zu implementieren ist das gRPC Framework, indem die Daten binär kodiert mittels dem http/2 Protokoll übertragen werden.
Der Hauptanwendungsbereich von gRPC ist die Kommunikation bei sogenannten Microservices, während bei der Kommunikation zwischen Frontend und Backend meist aufgrund von Kompatibilitäts- und Komplexitätsgründen REST oder GraphQL Technologien dominieren.
Eine gRPC-Übertragung wird in der Regel nur verwendet, wenn eine effiziente und schnelle Kommunikation besonders wichtig ist.

Ziel der Bachelorarbeit soll es sein verschieden API - Übertragungstechnologien vor allem mit dem Fokus auf Latenz, Effizienz und Skalierbarkeit zu überprüfen und zu vergleichen.

Fragestellung und Problemstellung:

<u>Optimierung von Datenströmen zwischen Frontend und Backend</u>

- Wie kann die Nutzung von gRPC in Kombination mit React zur Optimierung von Datenströmen in Echtzeitanwendungen beitragen?
- Untersuchung der **Latenz, Effizienz und Skalierbarkeit** im Vergleich zu REST-basierten Architekturen.

- Welche Herausforderungen entstehen bei der Integration von gRPC in React-Frontends, insbesondere in Bezug auf gRPC-Web und Browser-Kompatibilität?

- Wie unterscheiden sich der Entwicklungsaufwand und die Wartbarkeit von gRPC und REST in der Kommunikation zwischen Frontend und Backend?

Forschungsfragen und Hypothesen:

Hypothesen:
Die Nutzung von gRPC reduziert die Latenz und erhöht die Effizienz im Vergleich zu REST, vor allem bei Echtzeitanwendungen mit hohen Datenmengen.

Die Implementierung von gRPC in React ist komplexer, da dadurch zusätzliche Toos verwendet werden müssen damit eine Browser Kompatibilität garantiert werden kann.

REST ist geeigneter für einfache Anwendungen mit niedrigeren Anforderungen an Echtzeitkommunikation.

Methodik:
Theoretische Analyse:

- Untersuchung und Vergleich der Kommunikationsprotokolle gRPC und REST in Bezug auf Latenz, Effizienz und Skalierbarkeit.
- Analyse von React und gRPC-Web zur Integration in Frontend-Architekturen.

Erstellen von Prototypen:
Im Zuge der Bachelorarbeit werden Prototypen, wie zB verschiedene Anwendungen mittels den jeweiligen Kommunikationsprotokollen oder API Gateway erstellt und verglichen die Eigenschaften der verschiedenen Übertragungstechniken verglichen.

Dabei werden Performance Tests, Benchmarking, Skalierbarkeitstests durchgeführt

Verwandte Arbeiten und Stand der Forschung:
Zu den oben genannten Fragestellungen und Technologien sollen verwandte Arbeiten analysiert und der aktuelle Stand der Forschung erfasst werden.