Documentation

Einleitung M165

Im Rahmen dieses Projektauftrags werden wir eine Applikation entwickeln, die auf eine MongoDB-Datenbank zugreift und sowohl Lese- als auch Schreibzugriffe ermöglicht. Wir haben die Möglichkeit, das Projekt entweder alleine oder in Partnerschaft mit einer anderen Person umzusetzen, dabei haben wir uns für eine Partnerarbeit entschieden

Ein wichtiger Bestandteil der Dokumentation ist das Arbeitsjournal, in dem wir ausführlich zu jedem Unterrichtsabschnitt reflektieren. Dabei beantworten wir Fragen wie: Was haben wir erledigt? Wie sind wir vorgegangen? Welche Schwierigkeiten sind uns begegnet und wie konnten wir sie lösen? Welche Aufgaben stehen als nächstes an? Was haben wir aus dem Unterricht gelernt? In unserem Falle haben wir uns für ein Einfaches „Ziel - Lösungsweg – Reflexion“ Schema entschieden.

Die Applikation, die im Projekt entwickelt wird, greift auf eine MongoDB-Datenbank zu und ermöglicht das Anzeigen, Erstellen, Bearbeiten und Löschen von Dokumenten in der MongoDB-Datenbank Dabei werden zwei Collections mit einer 1:m-Beziehung oder Sub-Dokumente verwendet. Die Applikation soll benutzerfreundlich sein, Fehlererkennung bieten und Testdaten enthalten.

Arbeitsjournal

08.06.2023

Erster Tag des Projektes

15.06.2023

Der 2. Tag

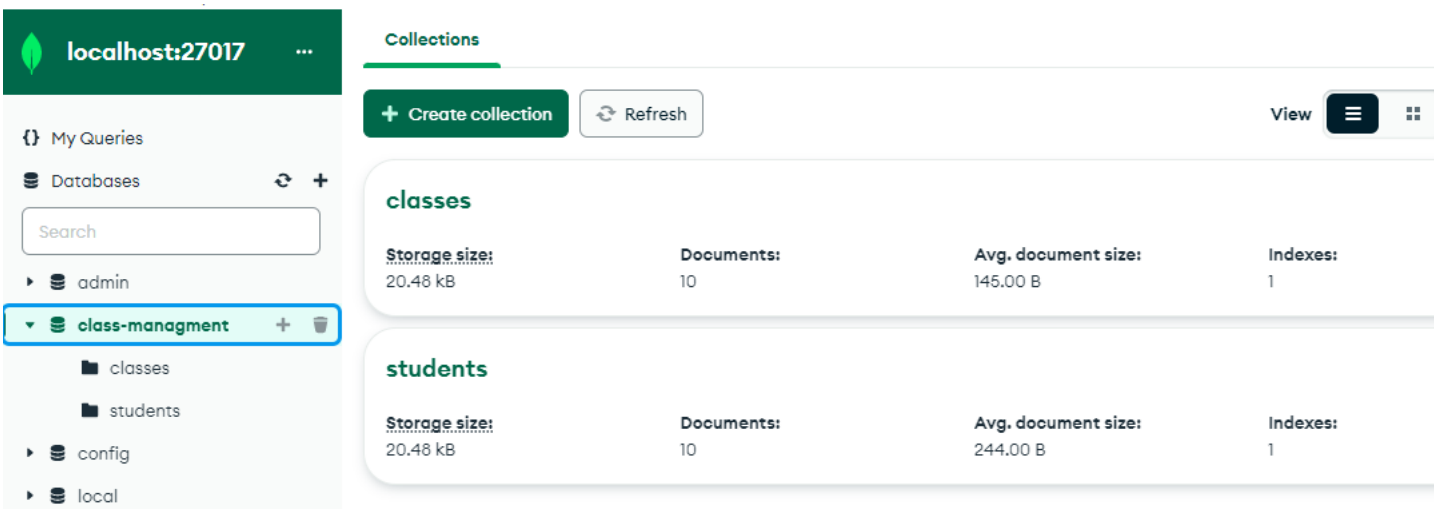
22.06.2023

Letzer Tag in der Schule

23.06.2023

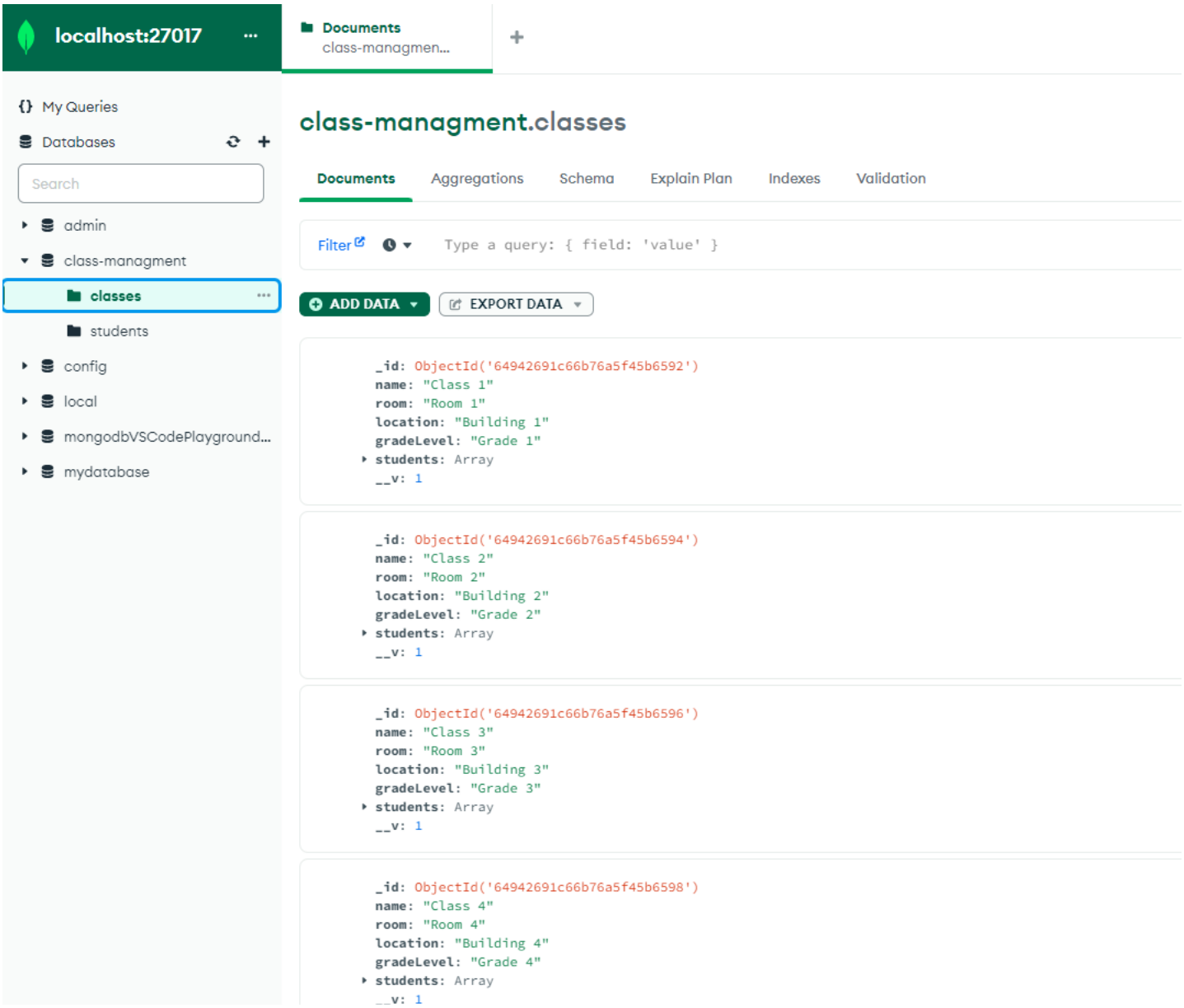
Der Tag der Abgabe

Zusammenfassung der Datenbankstruktur



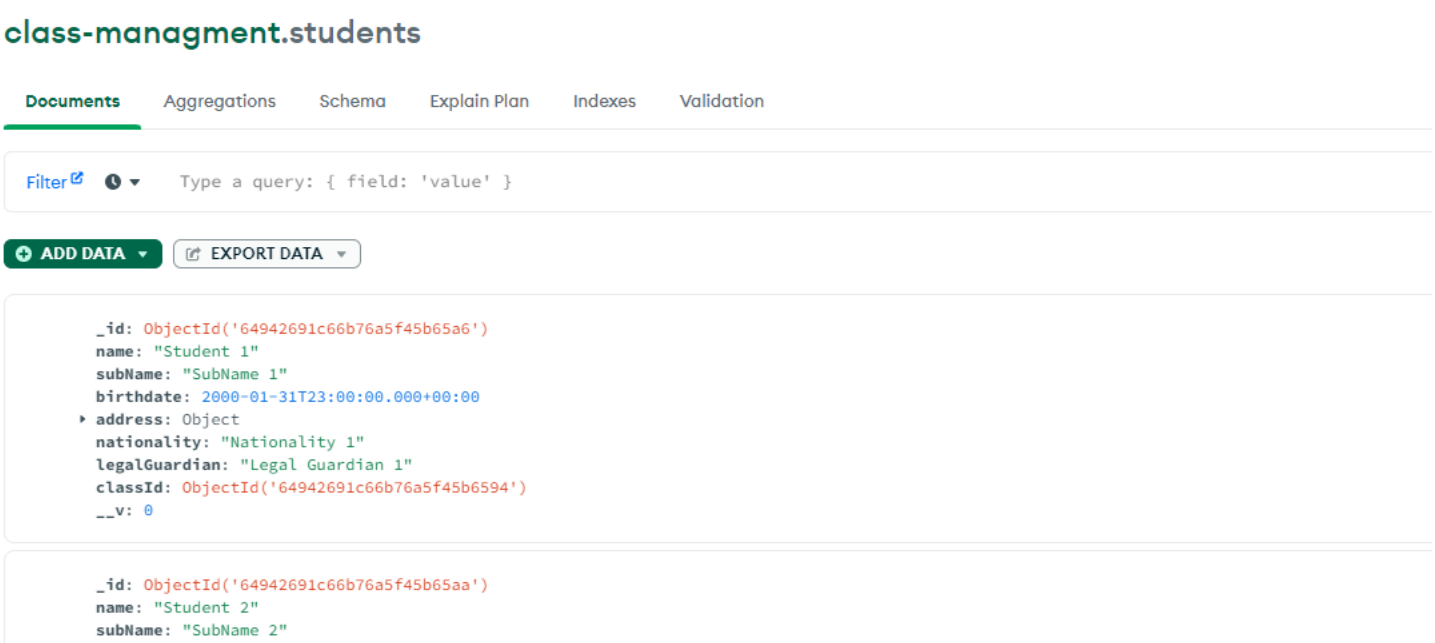
Die Datenbank des Klassenmanagementsystems besteht aus zwei Sammlungen: "Class" und "Student". Hier ist eine Zusammenfassung der Felder in beiden Sammlungen:

Class-Sammlung



- `_id`: Eindeutiger Identifikator des Dokuments.
- `name`: Name der Klasse.
- `room`: Raum, in dem die Klasse stattfindet.
- `location`: Standort des Klassenraums.
- `gradeLevel`: Klassenstufeniveau.
- `students`: Array von Referenzen auf Schüler-Dokumente, die zur Klasse gehören.

Student-Sammlung



- `_id`: Eindeutiger Identifikator des Dokuments.
- `name`: Vorname des Schülers.
- `subName`: Nachname des Schülers.
- `birthdate`: Geburtsdatum des Schülers.
- `address`: Objekt mit den Adressdetails des Schülers (PLZ, Straße, Stadt).
- `nationality`: Nationalität des Schülers.
- `legalGuardian`: Name des rechtlichen Vormunds des Schülers.
- `classId`: Referenz auf das "Class" Dokument, zu dem der Schüler gehört.

Die "Class"-Sammlung enthält Informationen über die Klassen, einschließlich ihrer Namen, Raumdetails, Standort und Klassenstufen. Die "Student"-Sammlung enthält Informationen über die Schüler, einschließlich ihrer Namen, Geburtsdaten, Adressen, Nationalität und rechtlichen Vormunde. Die "students" Feld in der "Class"-Sammlung enthält Referenzen auf die Schüler, die zur entsprechenden Klasse gehören.

Diese Datenbankstruktur ermöglicht es, die Beziehungen zwischen Klassen und Schülern effizient zu verwalten und abzufragen, indem Referenzen verwendet werden, um auf die relevanten Dokumente in den jeweiligen Sammlungen zu verweisen.

DB bearbeiten

Class

find class

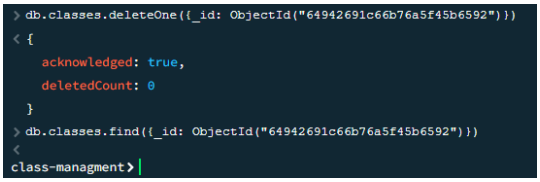
update class

```
> db.classes.updateOne({_id: ObjectId("64942691c66b76a5f45b6592")}, {$set: {name: "Neuer Klassenname"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.classes.find({_id: ObjectId("64942691c66b76a5f45b6592")})
< {
    _id: ObjectId("64942691c66b76a5f45b6592"),
    name: 'Neuer Klassenname',
    room: 'Room 1',
    location: 'Building 1',
    gradeLevel: 'Grade 1',
    students: [
      ObjectId("64942691c66b76a5f45b65ca")
    ],
    __v: 1
  }
class-managment>
```

create class

```
> db.classes.insertOne({
    name: "Neue Klasse",
    room: "Neuer Raum",
    location: "Neues Gebäude",
    gradeLevel: "Neue Stufe",
    students: [
      ObjectId("64942691c66b76a5f45b65ca"),
      ObjectId("64942691c66b76a5f45b65a6")
    ],
    __v: 1
  })
< {
    acknowledged: true,
    insertedId: ObjectId("64944009c6443d3633be5c88")
  }
> db.classes.find({name: "Neue Klasse"})
< {
    _id: ObjectId("64944009c6443d3633be5c88"),
    name: 'Neue Klasse',
    room: 'Neuer Raum',
    location: 'Neues Gebäude',
    gradeLevel: 'Neue Stufe',
    students: [
      ObjectId("64942691c66b76a5f45b65ca"),
      ObjectId("64942691c66b76a5f45b65a6")
    ],
    __v: 1
  }
class-managment>
```

delete class



```
> db.classes.deleteOne({_id: ObjectId("64942691c66b76a5f45b6592")})
< {
    acknowledged: true,
    deletedCount: 0
  }
> db.classes.find({_id: ObjectId("64942691c66b76a5f45b6592")})
<
class-managment>
```

Student

find student

```
> db.students.find({ name: "Student 1" })
< {
    _id: ObjectId("64942691c66b76a5f45b65a6"),
    name: 'Student 1',
    subName: 'SubName 1',
    birthdate: 2000-01-31T23:00:00.000Z,
    address: {
      plz: '1',
      street: 'Street 1',
      city: 'City 1'
    },
    nationality: 'Nationality 1',
    legalGuardian: 'Legal Guardian 1',
    classId: ObjectId("64942691c66b76a5f45b6594"),
    __v: 0
  }
class-managment>
```

update student

```
> db.students.updateOne(
    { _id: ObjectId("64942691c66b76a5f45b65a6") },
    { $set: { name: "Updated Student", subName: "Updated SubName" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.students.find({ name: "Updated Student" })
< {
    _id: ObjectId("64942691c66b76a5f45b65a6"),
    name: 'Updated Student',
    subName: 'Updated SubName',
    birthdate: 2000-01-31T23:00:00.000Z,
    address: {
      plz: '1',
      street: 'Street 1',
      city: 'City 1'
    },
    nationality: 'Nationality 1',
    legalGuardian: 'Legal Guardian 1',
    classId: ObjectId("64942691c66b76a5f45b6594"),
    __v: 0
  }
class-managment>
```

create student

```
> db.students.insertOne({
    name: "New Student",
    subName: "New SubName",
    birthdate: new Date("2000-01-01"),
    address: {
      plz: "12345",
      street: "New Street",
      city: "New City"
    },
    nationality: "New Nationality",
    legalGuardian: "New Legal Guardian",
    classId: ObjectId("64942691c66b76a5f45b6592"),
    __v: 0
  })
< {
    acknowledged: true,
    insertedId: ObjectId("64944224c6443d3633be5c89")
  }
> db.students.find({ name: "New Student" })
< {
    _id: ObjectId("64944224c6443d3633be5c89"),
    name: 'New Student',
    subName: 'New SubName',
    birthdate: 2000-01-01T00:00:00.000Z,
    address: {
      plz: '12345',
      street: 'New Street',
      city: 'New City'
    },
    nationality: 'New Nationality',
    legalGuardian: 'New Legal Guardian',
    classId: ObjectId("64942691c66b76a5f45b6592"),
    __v: 0
  }
class-managment>
```

delete student

```
> db.students.deleteOne({ _id: ObjectId("64942691c66b76a5f45b65a6") })
< {
    acknowledged: true,
    deletedCount: 1
  }
> db.students.find({ _id: ObjectId("64942691c66b76a5f45b65a6") })
<
class-managment>
```

Zusammenfassung des Backends

Das Backend des Klassenmanagementsystems besteht aus Node.js-Dateien, die Express verwenden und mit einer MongoDB-Datenbank interagieren. Die wichtigsten Dateien und Funktionen sind:

- `server.js`: Erstellt den Express-Server, verbindet sich mit der MongoDB und definiert Routing-Endpunkte.
- `db.js`: Stellt die Verbindung zur MongoDB-Datenbank her.
- `models/class.js`: Definiert das Mongoose-Modell für Klassen.
- `models/student.js`: Definiert das Mongoose-Modell für Schüler.
- `controllers/classController.js`: Enthält Funktionen zur Verwaltung von Klassen.
- `controllers/studentController.js`: Enthält Funktionen zur Verwaltung von Schülern.
- `routes.js`: Definiert die Routing-Endpunkte für die API.

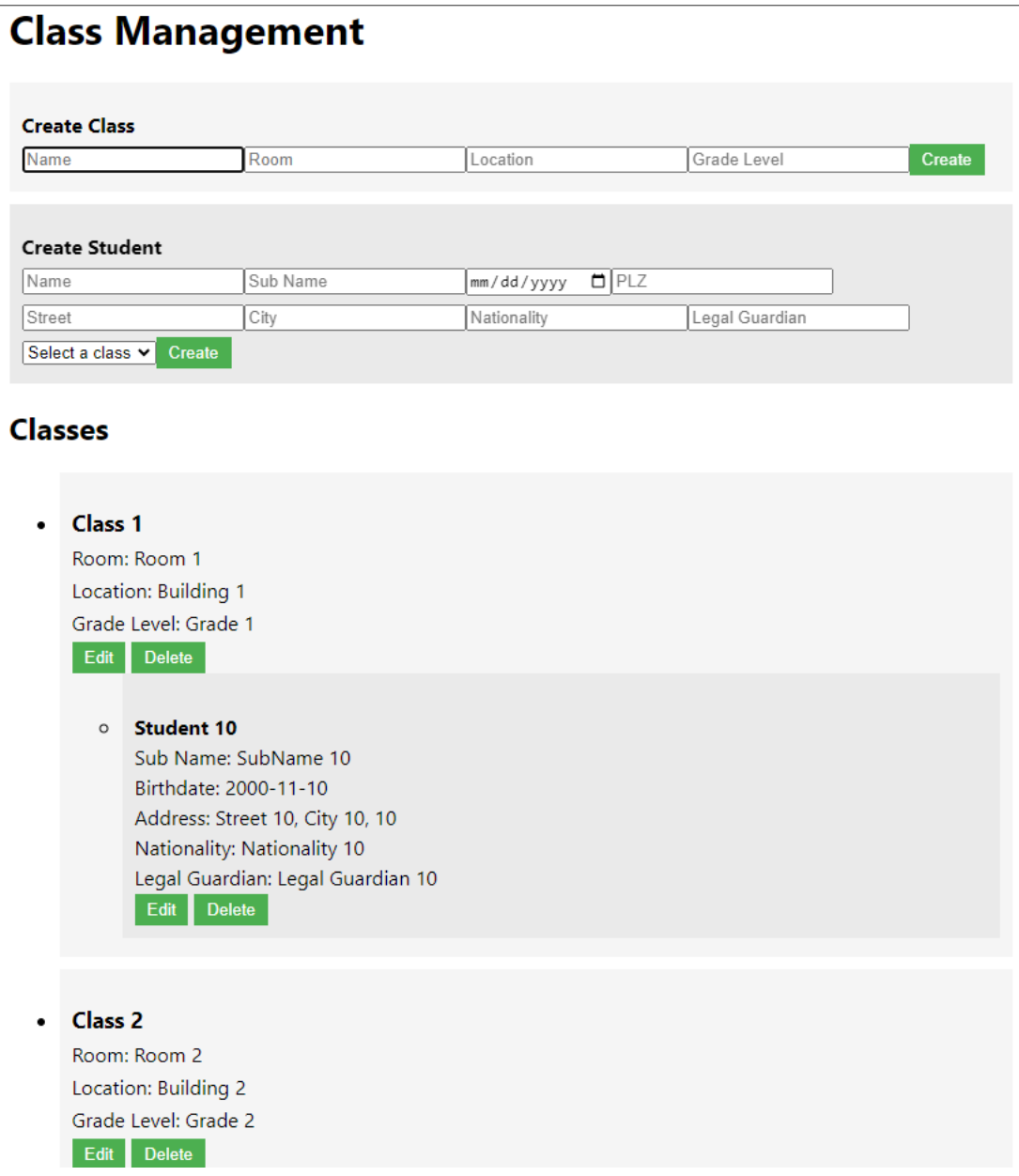
Das Backend ermöglicht das Erstellen, Aktualisieren, Löschen und Abrufen von Klassen und Schülern über die bereitgestellten API-Endpunkte. Die Datenbankinteraktion erfolgt über die Mongoose-Modelle, die eine Abstraktionsschicht über der MongoDB-Datenbank bieten.

Zusammenfassung des GUI-Codes

Der gegebene Code implementiert ein GUI für das Klassenmanagement-System. Es basiert auf React und verwendet verschiedene Komponenten und API-Aufrufe, um mit dem Backend zu interagieren.

- Die `App`-Komponente ist die Hauptkomponente, die den gesamten Inhalt der Anwendung enthält. Sie verwaltet den Zustand der Klassen, Schüler und der bearbeiteten Klasse/Schüler.
- Es gibt API-Funktionen (`classApi.js` und `studentApi.js`), die HTTP-Anfragen an das Backend senden, um Klassen und Schüler zu erstellen, zu aktualisieren, zu löschen und abzurufen.
- Die `CreateClass`-Komponente enthält ein Formular zum Erstellen einer neuen Klasse.
- Die `CreateStudent`-Komponente enthält ein Formular zum Erstellen eines neuen Schülers.
- Die restlichen Komponenten und Funktionen ermöglichen die Darstellung und Interaktion mit den Daten im GUI.

Die wichtigsten Funktionen sind das Abrufen von Klassen und Schülern vom Server, das Aktualisieren und Löschen von Klassen und Schülern, sowie das Erstellen neuer Klassen und Schüler über die API-Funktionen.



Auf dieser Webseite besteht die Möglichkeit, Klassen und Schüler zu erstellen. Während des Erstellungsprozesses können Schüler bestimmten Klassen zugewiesen werden. Darüber hinaus können alle erstellten Klassen und Schüler aktualisiert und gelöscht werden. Die Webseite zeigt zudem eine Liste aller Klassen an, unter denen die jeweilig zugehörigen Schüler aufgeführt sind.