Entwicklung einer Notizanwendung mit einer NoSQL-Datenbank

Neue Datenbankkonzepte

An der Dualen Hochschule Baden-Württemberg Heidenheim

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Git Repository:

https://github.com/Alexander-Friz/NeueDatenbankKonzepte

Projektteam: Alexander Friz, Marina Schwärzle, Max Sontheim

Kurs: WWI2022/A

Projektnummer: A10

Abgabedatum: 31.03.2024

Projektvorstellung: Notizanwendung mit MongoDB

Unser Team hat sich entschieden, eine API für eine Notizanwendung zu entwickeln, die auf der leistungsfähigen NoSQL-Datenbank MongoDB basiert. Unsere Vision ist es, eine flexible und benutzerfreundliche Plattform zu schaffen, die es den Nutzern ermöglicht, ihre Notizen effizient zu verwalten und zu organisieren.

Die Notizanwendung bietet eine Vielzahl von Funktionen, darunter:

- Accounterstellung: Nutzer können Konten erstellen, um Zugang zur Anwendung zu erhalten.
- Notizen erstellen, bearbeiten und löschen: Benutzer können ihre Gedanken und Aufgaben festhalten und Informationen bei Bedarf aktualisieren oder entfernen.
- Notizen kategorisieren: Nutzer können ihre Notizen thematisch organisieren, indem sie ihnen Kategorien zuordnen.
- Bilder hochladen und anhängen: Die Anwendung ermöglicht es den Benutzern, Bilder zu ihren Notizen hochzuladen und anzuhängen, um visuelle Informationen zusammen mit Textnotizen zu speichern.
- Optionales Fälligkeitsdatum setzen: Nutzer können optional Fälligkeitsdaten für Notizen festlegen, um an wichtige Aufgaben erinnert zu werden.
- Notizen mit PIN versehen: Sensible oder private Notizen können mit einem geheimen PIN geschützt werden, um den Zugriff darauf zu beschränken.

Wie im Anforderungskatalog beschrieben, konzentrieren wir uns auf die grundlegenden Funktionen der Anwendung, um diese schnell und effizient umzusetzen und unseren Nutzern eine sofortige Nutzererfahrung zu bieten.

Beschreibung der Funktionsweise der gewählten NoSQL-Datenbank (MongoDB):

MongoDB ist eine dokumentenorientierte NoSQL-Datenbank, die auf dem Konzept von Collections und Dokumenten basiert. Hier sind einige Schlüsselkonzepte der Funktionsweise von MongoDB:

1. **Dokumente**: In MongoDB werden Daten in Form von Dokumenten gespeichert, die im JSON-ähnlichen BSON-Format vorliegen. Diese Dokumente können komplexe Strukturen haben und Felder unterschiedlicher Typen enthalten.
2. **Collections**: Dokumente werden in MongoDB in Collections organisiert, ähnlich wie Tabellen in relationalen Datenbanken. Eine Collection enthält eine Gruppe von Dokumenten, die gemeinsame Eigenschaften teilen.
3. **Flexibles Schema**: Anders als bei relationalen Datenbanken erfordert MongoDB kein festes Schema für die Daten. Dokumente in einer Collection können unterschiedliche Felder und Strukturen haben, was eine einfache Skalierung und Anpassung ermöglicht.
4. **Skalierbarkeit**: MongoDB ist für horizontale Skalierbarkeit ausgelegt, was bedeutet, dass es einfach ist, zusätzliche Server hinzuzufügen, um die Datenbankleistung zu erhöhen, wenn die Anwendungsanforderungen wachsen.
5. **Replikation und Ausfallsicherheit**: MongoDB unterstützt Replikation, wodurch Kopien der Daten auf mehreren Servern erstellt werden können, um Ausfallsicherheit und Datenintegrität zu gewährleisten.

Begründung der Wahl von MongoDB:

Wir haben uns für MongoDB als Datenbank für unsere Notizanwendung ausfolgenden Gründen entschieden:

1. **Dokumentenorientiertes Modell**: MongoDB's dokumentenorientiertes Modell passt gut zu unserer Anwendungsstruktur, da Notizen in Form von Dokumenten mit variabler Länge gespeichert werden können, ohne ein festes Schema definieren zu müssen.
2. **Skalierbarkeit**: Da wir erwarten, dass unsere Nutzerbasis mit der Zeit wächst, ist es wichtig, eine Datenbank zu wählen, die einfach horizontal skalierbar ist. MongoDB

ermöglicht uns, zusätzliche Server hinzuzufügen, um mit dem Anstieg der Datenmenge und Benutzerzahl umzugehen.

3. **Flexibles Schema**: Die flexiblen Schemata von MongoDB erleichtern es uns, Änderungen an unserer Datenstruktur vorzunehmen, ohne das gesamte Schema zu ändern. Dies ist besonders wichtig in einem agilen Entwicklungsumfeld, in dem sich Anforderungen schnell ändern können.
4. **Leistung**: MongoDB bietet eine hohe Leistung für Lese- und Schreiboperationen, was für eine reaktionsschnelle Benutzererfahrung in unserer Notizanwendung wichtig ist.

Argumentative Abgrenzung zum Einsatz einer relationalen Datenbank:

Eine relationale Datenbank könnte theoretisch auch für unsere Notizanwendung verwendet werden. Jedoch gibt es einige Einschränkungen und Herausforderungen, die uns davon abhalten, eine relationale Datenbank zu wählen:

1. **Begrenzte Skalierbarkeit**: Relationale Datenbanken sind typischerweise vertikal skalierbar, was bedeutet, dass die Leistung durch die Erhöhung der Ressourcen auf einem einzelnen Server verbessert wird. Dies kann teuer und nicht so effizient sein wie die horizontale Skalierung, die NoSQL-Datenbanken wie MongoDB bieten.
2. **Komplexität beim Schema-Design**: In relationalen Datenbanken müssen wir im Voraus ein starres Schema definieren, das möglicherweise nicht leicht anzupassen ist, wenn sich die Anwendungsanforderungen ändern. Bei einer Notizanwendung, in der die Länge der Notizen variabel ist, kann dies zu Problemen führen.
3. **Leistung bei unstrukturierten Daten**: Relationale Datenbanken sind optimal für strukturierte Daten geeignet, wie sie in tabellarischen Formaten vorliegen. Für unstrukturierte Daten wie lange Texte, wie sie in unseren Notizen vorkommen, könnten NoSQL-Datenbanken wie MongoDB besser geeignet sein, da sie weniger Overhead haben.

Insgesamt bietet MongoDB eine passende Lösung für unsere Notizanwendung, da sie flexibel, skalierbar und leistungsstark ist und gut zu den Anforderungen unseres Projekts passt.

Erklärung der Schnittstelle

Nachstehend wird die API-Schnittstelle erklärt, außerdem gibt es eine Postman-Collection, in der sich alle Routen befinden. Diese API-Schnittstellen eignen sich für eine spätere Implementierung eines Frontend für die Notizanwendung. Die Jeweilige Nutzer-ID, Kategorie-ID und die Notiz-ID können mittels der GET-Methoden abgerufen werden.

1. **Neuen Nutzer anlegen**
    - Methode: POST
    - Beschreibung: Diese Route ermöglicht das Anlegen eines neuen Nutzers.
    - Endpoint: http://localhost:3000/users/create
    - Body-Parameter:
        - email (String): Die E-Mail-Adresse des neuen Nutzers.
        - password (String): Das Passwort des neuen Nutzers.
2. **Alle Nutzer abfragen**
    - Methode: GET
    - Beschreibung: Diese Route ermöglicht das Abrufen aller Nutzer.
    - Endpoint: http://localhost:3000/users/all
3. **Notiz erstellen**
    - Methode: POST
    - Beschreibung: Diese Route ermöglicht das Erstellen einer neuen Notiz.
    - Endpoint: http://localhost:3000/notes/create
    - Body-Parameter:
        - title (String): Der Titel der Notiz.
        - content (String): Der Inhalt der Notiz.
        - userId (String): Die ID des Nutzers, dem die Notiz zugeordnet sein soll.

4. **Nutzer fragt alle seine Notizen ab**
    - Methode: GET
    - Beschreibung: Diese Route ermöglicht einem Nutzer das Abrufen aller seiner Notizen.
    - Endpoint: http://localhost:3000/notes/user/:userId
    - URL-Parameter:
        - userId (String): Die ID des Nutzers, dessen Notizen abgerufen werden sollen.
5. **Eine Notiz löschen**
    - Methode: DELETE
    - Beschreibung: Diese Route ermöglicht das Löschen einer bestimmten Notiz.
    - Endpoint: http://localhost:3000/notes/:noteId
    - URL-Parameter:
        - noteId (String): Die ID der zu löschenden Notiz.
6. **Kategorie anlegen**
    - Methode: POST
    - Beschreibung: Diese Route ermöglicht das Anlegen einer neuen Kategorie.
    - Endpoint: http://localhost:3000/categories/create
    - Body-Parameter:
        - name (String): Der Name der Kategorie.
        - userId (String): Die ID des Nutzers, dem die Kategorie zugeordnet ist.
7. **Kategorie löschen**
    - Methode: DELETE
    - Beschreibung: Diese Route ermöglicht das Löschen einer bestimmten Kategorie.
    - Endpoint: http://localhost:3000/categories/:categoryId
    - URL-Parameter:
        - categoryId (String): Die ID der zu löschenden Kategorie.

8. **Nutzer fragt Kategorien ab**
    - Methode: GET
    - Beschreibung: Diese Route ermöglicht einem Nutzer das Abrufen aller seiner Kategorien.
    - Endpoint: http://localhost:3000/categories/user/:userId
    - URL-Parameter:
        - userId (String): Die ID des Nutzers, dessen Kategorien abgerufen werden sollen.
9. **Kategorie bearbeiten**
    - Methode: PUT
    - Beschreibung: Diese Route ermöglicht das Bearbeiten einer bestimmten Kategorie.
    - Endpoint: http://localhost:3000/categories/:categoryId
    - URL-Parameter:
        - categoryId (String): Die ID der zu bearbeitenden Kategorie.
    - Body-Parameter:
        - name (String): Der neue Name der Kategorie.
10. **Notiz bearbeiten**
    - Methode: PUT
    - Beschreibung: Diese Route ermöglicht das Bearbeiten einer bestimmten Notiz.
    - Endpoint: http://localhost:3000/notes/:noteId
    - URL-Parameter:
        - noteId (String): Die ID der zu bearbeitenden Notiz.
    - Body-Parameter:
        - title (String): Der neue Titel der Notiz.
        - content (String): Der neue Inhalt der Notiz.
        - categories (Array): Ein Array von Kategorie-IDs, die der Notiz zugeordnet sind oder zugeordnet werden sollen.

11. **Nutzer ändern**
    - Methode: PUT
    - Beschreibung: Diese Route ermöglicht das Bearbeiten eines bestimmten Nutzers.
    - Endpoint: http://localhost:3000/users/:userId
    - URL-Parameter:
        - userId (String): Die ID des zu bearbeitenden Nutzers.
    - Body-Parameter:
        - email (String): Die neue E-Mail-Adresse des Nutzers.
        - password (String): Das neue Passwort des Nutzers.
12. **Nutzer und Nutzerinhalte löschen**
    - Methode: DELETE
    - Beschreibung: Diese Route ermöglicht das Löschen eines bestimmten Nutzers und aller mit ihm verbundenen Notizen und Kategorien.
    - Endpoint: http://localhost:3000/users/:userId
    - URL-Parameter:
        - userId (String): Die ID des zu löschenden Nutzers.

Zusätzliche Implementierung

Das beim Anlegen eines Users vergebene Passwort wird mittels Bcrypt verschlüsselt und so nicht im Klartext in der Datenbank gespeichert.

Außerdem wird nach einer erfolgreichen Anmeldung vom Backend eine 30 Minuten gültiger Token mitgesendet. Dieser bietet das Potential für ein zukünftiges Frontend zu verifizieren, dass der Nutzer angemeldet ist.

Installation und Ausführung

Für die Installation und Ausführung der Anwendung sind Node.js, der Node Package Manager sowie Docker und Postman erforderlich. Weitere Anweisungen zur Installation und Ausführung der Anwendung sind in der README-Datei enthalten.