SPENGERGASSE
ausbildung mit zukunft

FindME
DAS DIGITALE FUNDBÜRO

Relationales vs. NoSQL-Datenbanken

Verfasser: Maximilian A. Fenusz
Klasse: 6CAIF
Betreuer: Wenz Rene
Jahrgang: 2024/25

HTBLVA, Wien V, Spengergasse
Höhere Lehranstalt für Informatik
Ausbildungsschwerpunkt Informatik für Erwachsene
Abgabedatum: 27.06.2025

Inhalt

1. Einleitung

1.1. Projektüberblick

Das Projekt „FindMe“ ist eine digitale Webapplikation, die als modernes Fundbüro für den Gebrauch in der Schule konzipiert wurde. Nutzer können verlorene Gegenstände melden oder gefundene Objekte erfassen. Ziel ist es, eine zentrale Plattform zur Verwaltung und Rückgabe von Fundsachen bereitzustellen, um den Prozess für Schüler*innen und Schulpersonal effizienter und transparenter zu gestalten.

Die Applikation wurde als Fullstack-Webprojekt umgesetzt:

- Frontend: React in Kombination mit Material UI für ein modernes und responsives Design.
- Backend: ASP.NET Core für die API-Logik und Serverkommunikation.
- Datenbank: MongoDB als NoSQL-Datenbank zur flexiblen Speicherung von Benutzer- und Objektdaten.

Ein zentraler Bestandteil de Arbeit ist der Vergleich von NoSQL- und relationalen Datenbanken, um zu begründen, warum für dieses Projekt auf eine dokumentenbasierte Lösung (MongoDB) gesetzt wurde.

1.2. Motivation und Zielsetzung

In vielen Bildungseinrichtungen geht täglich eine Vielzahl von Gegenständen verloren – sei es Kleidung, Schlüssel, USB-Sticks oder technische Geräte. Häufig

werden diese Funde zwar abgegeben, aber selten erfolgreich ihren Besitzer*innen zugeordnet, da ein strukturierter digitaler Prozess fehlt.

Die Motivation hinter dem Projekt „FindMe“ war es daher, eine benutzerfreundliche Webanwendung zu entwickeln, die das Verwalten und Wiederfinden von verlorenen Gegenständen einfach und zugänglich macht.

Zielsetzung der Arbeit:

- Entwicklung einer funktionalen Webapplikation zur Verwaltung von Fund- und Verlustmeldungen.
- Einsatz moderner Technologien (React, ASP.NET, MongoDB) zur praxisnahen Umsetzung.
- Vergleich von NoSQL- und relationalen Datenbanksystemen, um deren Vor- und Nachteile im Kontext des Projekts zu analysieren.
- Auswahl des optimalen Datenbankmodells für das Szenario anhand realer Anforderungen: Flexibilität, Skalierbarkeit und einfache Integration in ein agiles Frontend.

Diese Ziele spiegeln sowohl die technische Herausforderung als auch den wissenschaftlichen Vergleich der Datenbankmodelle wider.

1.3. Technologien im Projekt „FindMe“

Für die Umsetzung des Projekts „FindMe“ wurde auf moderne und bewährte Technologien gesetzt, um eine performante und skalierbare Webapplikation zu entwickeln. Die gewählte Architektur folgt dem klassischen Three-Tier-Modell (Frontend – Backend – Datenbank), was eine klare Trennung von Zuständigkeiten ermöglicht.

Frontend

- React: JavaScript-Bibliothek für die Erstellung dynamischer Benutzeroberflächen.
- Material UI: Design-Framework für ein konsistentes und modernes UI mit Fokus auf Benutzerfreundlichkeit und Barrierefreiheit.
- Axios: Für die Kommunikation mit der REST-API im Backend.

Backend

- ASP.NET Core: Moderne, plattformunabhängige Web-API-Technologie von Microsoft.

- Verwendung des MVC-Prinzips zur Trennung von Logik, Daten und Darstellung.
- Implementierung von REST-Endpunkten für CRUD-Operationen.

Datenbank

- MongoDB: Dokumentenbasierte NoSQL-Datenbank, die JSON-ähnliche Datenstrukturen (BSON) speichert.
- Besonders geeignet für Anwendungen mit sich dynamisch verändernden Datenstrukturen (z. B. Benutzerprofile, Objekte mit optionalen Eigenschaften).

Diese Technologieauswahl ermöglicht eine flexible, wartbare und zukunftssichere Lösung, die den Anforderungen eines modernen Schulprojekts gerecht wird.

2. Theorie

2.1. Grundlagen relationaler Datenbanken

Relationale Datenbanken sind das klassische Modell zur strukturierten Datenspeicherung und werden seit den 1970er-Jahren in der IT breit eingesetzt. Sie basieren auf der Idee, Daten in Tabellen (Relationen) zu organisieren, wobei jede Tabelle eine definierte Struktur besitzt.

Wesentliche Merkmale:

- Tabellenstruktur: Daten werden in Zeilen und Spalten gespeichert.
- Feste Schemata: Jede Tabelle hat ein vordefiniertes Datenmodell – z. B. „Benutzer“ mit den Spalten „ID“, „Vorname“, „Nachname“, „Email“.
- Primärschlüssel: Jede Tabelle hat einen eindeutigen Identifikator (z. B. ID), um Datensätze zu unterscheiden.
- Fremdschlüssel: Relationen zwischen Tabellen werden über Verweise hergestellt (z. B. ein Benutzer hat viele Bestellungen).

Vorteile:

- Datenintegrität durch klare Schemata und Schlüssel.
- Transaktionssicherheit (ACID): garantierte Konsistenz bei parallelen Operationen.
- Ausgereifte Tools und Community (z.B. MySQL, PostgreSQL, Oracle)

Nachteile:

- Geringe Flexibilität: Änderungen am Schema sind aufwändig.
- Schwierige horizontale Skalierung.
- Nicht ideal für unstrukturierte oder sich stark ändernde Datenformate.

2.2. Grundlagen von NoSql-Datenbanken

NoSQL-Datenbanken („Not only SQL“) sind ein moderner Ansatz zur Datenspeicherung, der sich von den starren Tabellenstrukturen relationaler Systeme löst. Sie wurden entwickelt, um die Anforderungen an Skalierbarkeit, Flexibilität und Performance in modernen Web- und Cloud-Anwendungen zu erfüllen.

Charakteristische Merkmale:

- Schemafrei: Keine feste Tabellenstruktur – jedes „Dokument“ kann andere Felder besitzen.
- Datenformate: Meistens JSON, BSON oder Schlüssel-Wert-Paare.
- Horizontale Skalierbarkeit: Daten werden auf mehrere Server verteilt (Sharding).
- Hohe Verfügbarkeit und Geschwindigkeit, auch bei großen Datenmengen.

Arten von NoSQL-Datenbanken:

- Dokumentbasiert: Speicherung von JSON-ähnlichen Dokumenten (z.B. MongoDB)
- Key-Value-Stores: Einfache Zuordnung von Schlüssel zu Wert (z.B. Redis)
- Spaltenorientiert: Optimiert für Lesezugriffe auf viele Spalten (z.B. Cassandra)
- Graphdatenbanken: Fokus auf Beziehungen zwischen Datenpunkten (z.B. Neo4j)

Vorteile:

- Hohe Flexibilität bei Datenmodellierung.
- Ideal für unstrukturierte oder sich ändernde Daten.
- Leicht skalierbar für große Nutzerzahlen und Datenmengen.
- Schnelle Entwicklungszyklen, da keine Migrationen bei Feldänderungen notwendig sind.

Nachteile:

- Weniger standardisiert als SQL.
- Weniger geeignet für komplexe Transaktionen.
- Eventual Consistency statt starker ACID-Garantien (bei verteilten Systemen).

2.3. Relationale vs. NoSql-Datenbanken

Die Wahl zwischen relationalen und NoSQL-Datenbanken hängt stark vom Anwendungsszenario und den Anforderungen der Applikation ab. In diesem Abschnitt werden die wichtigsten Unterschiede anhand verschiedener Kriterien dargestellt.

2.3.1. Struktur und Datenmodell

Kriterium	**Relationale DB**	**NoSQL DB**
Datenmodell	Tabellen mit festen Schemata	Dokumente, flexibel
Beziehungen	Explizit mit Fremdschlüssel	Implizit oder eingebettet
Änderung am Modell	Aufwendig (Migration Notwendig)	Flexibel, on-the-fly

2.3.2. Konsistenz- und Transaktionsmodelle

Kriterium	**Relationale DB**	**NoSQL DB**
Konsistenzmodell	ACID	BASE / Eventual Consistency
Transaktionen	Mehrstufige, garantierte	Eingeschränkt oder optional
Datenintegrität	Sehr hoch (Constraints)	Muss manuell umgesetzt werden

2.3.3. Skalierbarkeit und Performance

Kriterium	**Relationale DB**	**NoSQL DB**
Skalierbarkeit	Vertikal (stärkere Server)	Horizontal (mehr Server)
Schreibzugriffe	Langsamer bei viel Last	Sehr schnell
Lesegeschwindigkeit	Gut mit Indexen	Sehr gut bei flachen Datenstrukturen

2.3.4. Flexibilität und Einsatzbereiche

Kriterium	Relationale DB	NoSQL DB
Flexibilität	Niedrig – starres Schema	Hoch – strukturlose Speicherung
Einsatzbereiche	Finanzwesen, ERP, klassische Business-Apps	Web-Apps, IoT, Big Data, Startups
Beispielsysteme	MySQL, PostgreSQL, SQL Server	MongoDB, Redis, Firebase, Cassandra

3. Anwendungsfälle und Einsatzszenarien

Die Wahl der richtigen Datenbank hängt nicht nur von theoretischen Vor- und Nachteilen ab, sondern vor allem davon, wofür die Datenbank eingesetzt wird. In diesem Kapitel werden typische Anwendungsfälle beschrieben, bei denen sich relationale oder NoSQL-Datenbanken jeweils besser eignen.

3.1. Wann sind relationale Datenbanken besser geeignet?

Relationale Datenbanken sind ideal, wenn:

- Starke Datenkonsistenz und Validierung entscheidend sind (z. B. bei Geldbeträgen, Bestellungen).
- Die Datenstruktur statisch und vorhersehbar ist.
- Komplexe Abfragen (Joins, Aggregationen) benötigt werden.
- Es viele Beziehungen zwischen Entitäten gibt (z. B. Kunden, Bestellungen, Rechnungen).

Typische Einsatzgebiete:

- Banken und Finanzdienstleister
- Buchhaltungssysteme
- Behörden, Registrierungsdatenbanken
- Lagerverwaltungssysteme

3.2. Wann sind NoSql-Datenbanken im Vorteil?

NoSQL-Datenbanken glänzen bei:

- Dynamischen Datenstrukturen (z. B. Benutzer mit optionalen Feldern, Objekte mit Bildern, Tags etc.)
- Hohen Datenmengen mit schnellen Schreib-/Lesezugriffen.
- Verteilten Systemen (Cloud, Microservices).
- Agilen Entwicklungsprozessen, bei denen das Datenmodell oft geändert wird.

Typische Einsatzgebiete:

- Webapplikationen mit vielen gleichzeitigen Nutzern
- Social Media Plattformen
- Mobile Apps
- IoT-Systeme (Sensoren, Logs etc.)
- Content-Management-Systeme (CMS)

3.3. Typische Praxisbeispiele und bekannte Anwendungen

Szenario	**Empfohlene Datenbank**	**Grund**
E-Commerce-System mit Warenkorb	Relationale DB (SQL)	Transaktionen, Lagerstand, Rechnungen
Messaging-App	NoSQL (z. B. MongoDB)	Flexible Nachrichtenstruktur, hohe Performance
Schüler-Fundbüro (FindMe)	NoSQL (MongoDB)	Unterschiedliche Objekttypen, einfache Erweiterung
Personalverwaltung	Relationale DB	Strukturierte Stammdaten, viele Relationen

4. Praktische Umsetzung

Im Rahmen der praktischen Arbeit wurde die Webapplikation „FindMe“ als digitales Fundbüro umgesetzt. Das Projekt verfolgt das Ziel, verlorene Gegenstände effizient zu erfassen, sichtbar zu machen und zurückzugeben. Der Fokus lag nicht nur auf der technischen Umsetzung, sondern auch auf der bewussten Auswahl der passenden Datenbanktechnologie – in diesem Fall MongoDB als NoSQL-Datenbank.

4.1. Architekturüberblick

Die Applikation basiert auf einer klassischen Client-Server-Architektur:

- Frontend: React + Material UI
- Backend: ASP.NET Core (C#)
- Datenbank: MongoDB

Die Kommunikation erfolgt über eine REST-API, die JSON-Daten zwischen Frontend und Backend austauscht

4.2. Verwendung von MongoDB im Backend

- MongoDB wurde verwendet, um die Daten wie Fundobjekte, Benutzer, Nachrichten und Gruppen flexibel zu speichern.
- Es wurden keine festen Tabellen definiert – jede Entität (z. B. Task, User, WorkSlot) ist ein eigenes JSON-Dokument.
- Dank der schemalosen Struktur konnten Felder je nach Objekt erweitert oder weggelassen werden – perfekt für Fundobjekte mit optionalen Eigenschaften (z. B. Farbe, Fundort, Bild, Beschreibung).

4.3. Integration mit ASP.NET Core

- Das Backend wurde mithilfe von ASP.NET Core Web API erstellt.
- Die Anbindung an MongoDB erfolgte über den offiziellen MongoDB C# Driver.
- CRUD-Operationen wurden über Services, Controller und Repository-Schichten implementiert

4.4. Datenmodell und Datenzugriff

- Es wurden mehrere Entitätsklassen erstellt, z. B. „User“, „Group“, „Task“, „Message“, „Profile“.
- Die meisten Klassen erben von einer BaseEntity, die automatisch eine ID, Erstellungs- und Änderungszeit speichert.
- Die Daten werden als BSON-Dokumente in MongoDB gespeichert, was schnelles Abfragen, Erstellen und Bearbeiten ermöglicht.

4.5. Kommunikation zwischen Frontend und Backend (REST API)

- Über HTTP-Endpunkte wie „/api/user“, „/api/task“, „/api/message“ konnten Daten abgerufen und bearbeitet werden.
- Authentifizierung und Autorisierung wurden ebenfalls im Backend umgesetzt.
- Das Frontend verwendet Axios für die Kommunikation mit dem Backend.

4.6. Vor- und Nachteile der NoSQL-Nutzung im Projekt

Vorteile:

- Flexibles Datenmodell für heterogene Fundobjekte.
- Schnellere Entwicklung durch fehlende Migrationen.
- Gute Performance auch bei vielen Einträgen.

Nachteile:

- Datenintegrität (z. B. eindeutige Referenzen) muss manuell geprüft werden.
- Keine Transaktionen über mehrere Sammlungen hinweg.
- Weniger strukturierter Überblick bei großen Datenmengen ohne zusätzliches Management-Tool.

5. Lessons Learned

Während der Entwicklung von „FindMe“ konnten viele praktische Erfahrungen gesammelt werden – sowohl im Umgang mit modernen Webtechnologien als auch beim direkten Vergleich relationaler und NoSQL-Datenbanksysteme.

5.1. Herausforderungen währen der Entwicklung

- Die schemalose Struktur von MongoDB war anfangs ungewohnt. Ohne vorgegebenes Datenmodell mussten Validierungen im Code selbst durchgeführt werden.
- Die Trennung von Geschäftslogik und Datenzugriff in ASP.NET erforderte eine saubere Strukturierung (Controller → Services → Repositories).
- Einige Features wie Referenzen zwischen Dokumenten (z. B. User ↔ Tasks) mussten manuell modelliert werden, da MongoDB keine klassischen Fremdschlüssel kennt.

5.2. Erkenntnisse aus der Arbeit mit MongoDB

- MongoDB eignet sich hervorragend für Anwendungen mit dynamischen Daten, wie es bei Fundobjekten der Fall ist.
- Die Arbeit mit JSON-Dokumenten war besonders angenehm in Kombination mit React, da beide Seiten (Client/Server) dieselbe Datenstruktur nutzen.
- Die Geschwindigkeit beim Lesen/Schreiben war deutlich höher als in vergleichbaren relationalen Setups – besonders bei Listenoperationen und einfachem Filtern.

5.3. Reflexion: Wäre eine relationale Lösung sinnvoll gewesen?

In diesem konkreten Fall: Nein.

Ein relationales System hätte:

- Ein komplexes und starres Datenmodell erfordert.
- Häufige Migrationen bei Änderung von Feldern verursacht.
- Die agile Entwicklung unnötig verlangsamt.

MongoDB war die richtige Wahl für dieses Projekt, weil es maximale Flexibilität, schnelle Entwicklung und gute Performance bei gleichzeitig überschaubarem Datenvolumen ermöglichte.

6. Fachbegriffe

6.1. NoSQL

Sammelbegriff für Datenbanksysteme, die nicht auf dem relationalen Modell basieren. NoSQL-Datenbanken sind in der Regel schemafrei und besonders gut für unstrukturierte oder sich häufig ändernde Daten geeignet.

6.2. MongoDB

Eine weit verbreitete dokumentenorientierte NoSQL-Datenbank, bei der Daten als BSON (Binary JSON) gespeichert werden. Sie bietet hohe Flexibilität und einfache Skalierbarkeit.

6.3. Relationale Datenbank

Ein Datenbanksystem, das auf dem Tabellenmodell mit festem Schema basiert. Daten werden in Relationen (Tabellen) gespeichert, die über Schlüssel miteinander verknüpft sind. Bekannte Vertreter: MySQL, PostgreSQL, Oracle.

6.4. ACID

Abkürzung für vier Eigenschaften von Transaktionen in relationalen Datenbanken:

- Atomicity (Unteilbarkeit)
- Consistency (Konsistenz)
- Isolation (Isolation paralleler Vorgänge)
- Durability (Dauerhaftigkeit)

6.5. BASE

Gegenmodell zu ACID, das oft bei NoSQL-Systemen verwendet wird:

- Basically available
- Asoft state
- Eventual consistency

6.6. CRUD

Steht für die vier grundlegenden Operationen auf Daten:

- Create
- Read

- Update
- Delete

6.7. REST API

Ein Kommunikationsmodell zwischen Client und Server, bei dem HTTP-Methoden (GET, POST, PUT, DELETE) verwendet werden, um Daten zwischen Frontend und Backend auszutauschen

7. Quellen und Literaturverzeichnis

7.1. Fachartikel und Online-Ressourcen

1. MongoDB – Offizielle Dokumentation (22.06.2025)
   https://www.mongodb.com/docs
2. PostgreSQL – Dokumentation (22.06.2025)
   https://www.postgresql.org/docs/
3. Microsoft ASP.NET Core API Doku (22.06.2025)
   https://learn.microsoft.com/en-us/aspnet/core/
4. MongoDB vs SQL: Key Differences (22.06.2025)
   https://www.mongodb.com/nosql-explained/nosql-vs-sql
5. REST API Grundlagen – Mozilla Developer Network (22.06.2025)
   https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

7.2. Projektspezifische Quellen

6. Eigene Implementierung und Praxiserfahrung im Projekt „FindMe“
   (HTL Spengergasse, Jahrgang 2024/25)
7. Unterrichtsskripten der HTL Spengergasse (Fächer: DBI, POS, WMC, DSAI)