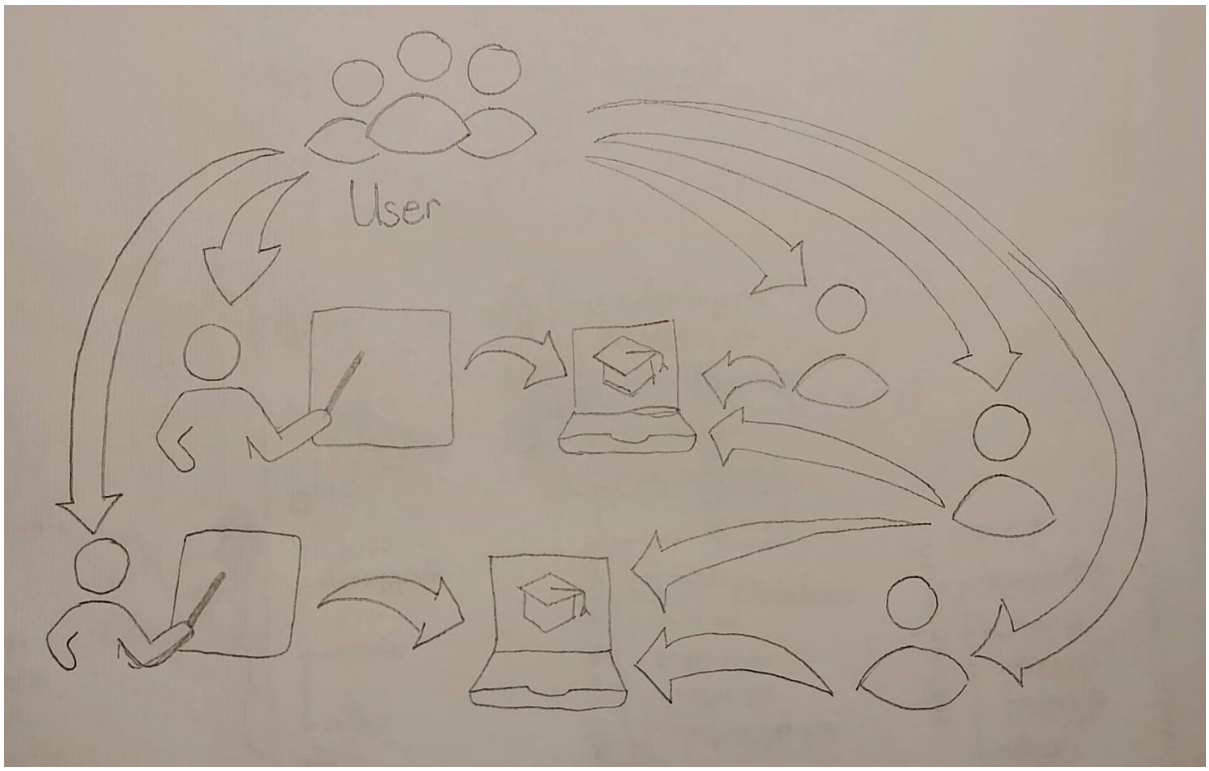
Dokumentation

Idee: Eine Website, in welcher Nutzer Kurse erstellen, sowie anderen Kursen beitreten können, um sich gegenseitig Dinge beizubringen. -> Lerngruppen online bündeln

Idee

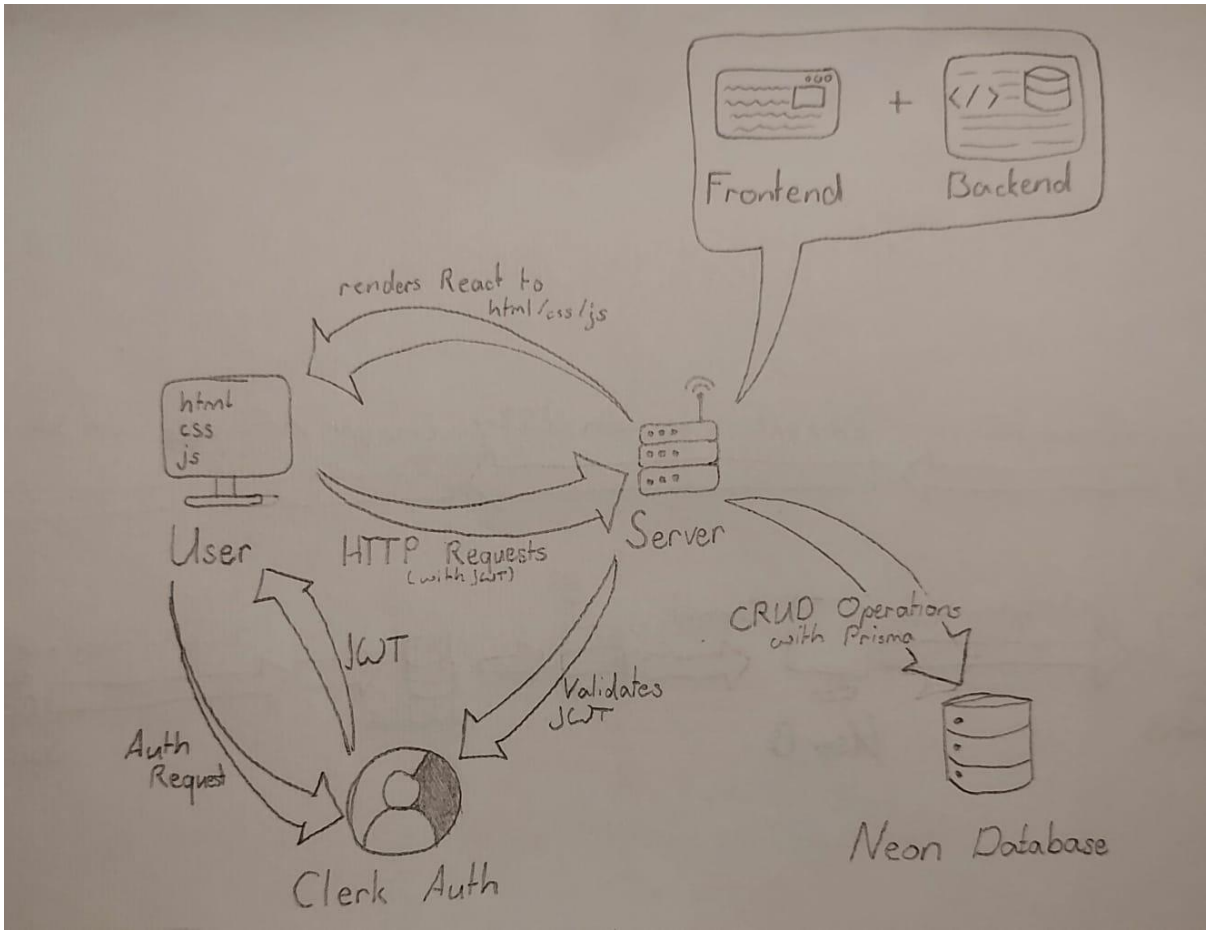


Meine Idee war es, eine Website zu bauen, in der Nutzer Lernkurse erstellen können, sowie anderen Kursen beitreten können, um anderen Nutzern etwas beizubringen (beispielsweise Trigonometrie, Ökologie, Satzglieder bestimmen, etc.). Dazu sind folgende Kriterien zu erfüllen:

- Nutzer müssen sich anmelden können
- Nutzer müssen Kurse posten können
- Nutzer müssen Kursen folgen können
- Jeder Kurs braucht einen eigenen Chat
- Jeder Chat muss verschlüsselt übermittelt werden
- Die Website braucht eine Homepage
- Der Nutzer kann neue Kurse finden und suchen
- Der Nutzer kann sein Profil bearbeiten
- Man kann Kursen beim Erstellen Bilder hinzufügen
- Kurse haben einen Kommentarabschnitt, um sich über Erfahrungen mit dem Kurs austauschen zu können

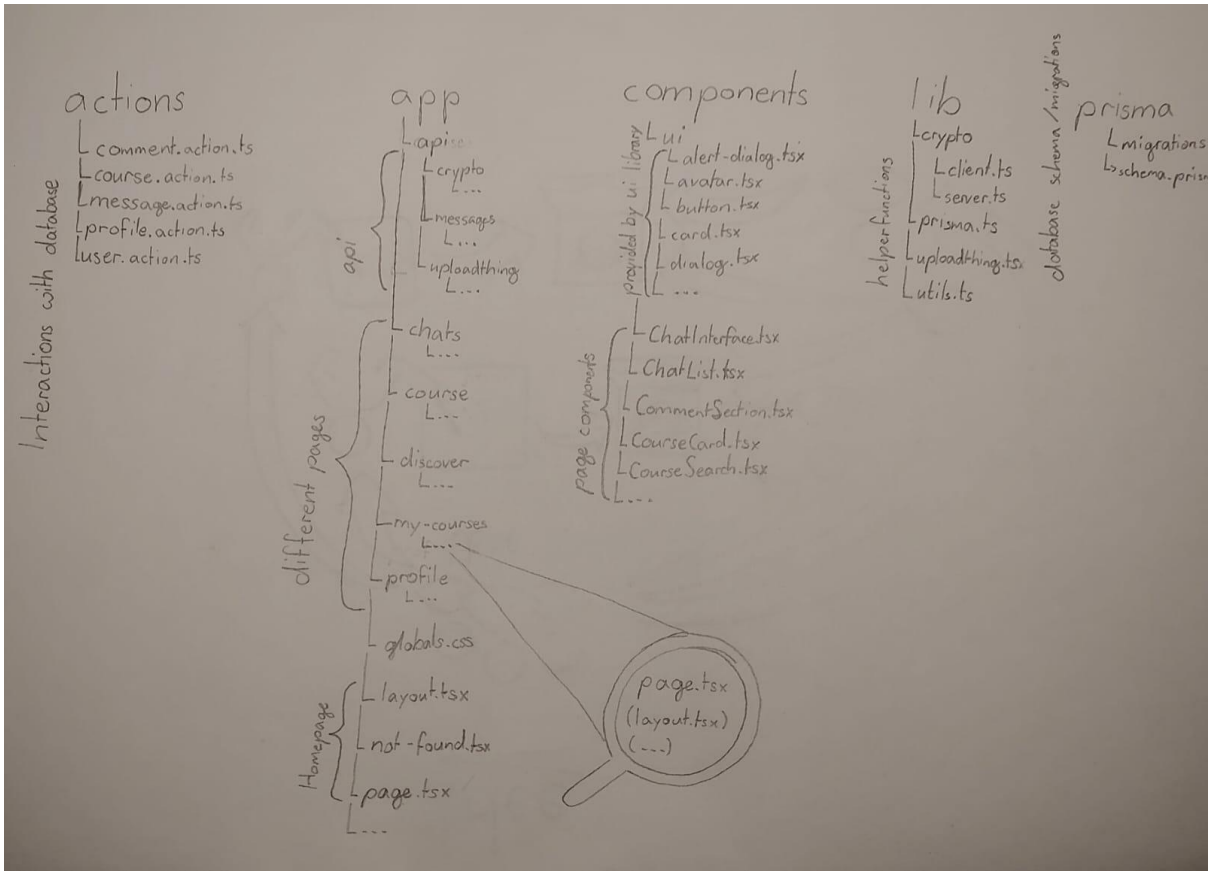
Produktwahl

- Clerk:
  - Einfache, sichere Authentifizierung
  - verschiedene Login-Möglichkeiten
- Next.js:
  - Server-Side-Rendering (Seite wird auf Server gerendert) -> Performance
  - Static-Site-Rendering (Seiten werden beim Build vorgerendert) -> Performance
  - Ordner-basiertes Routing
  - Backend / Frontend Logik an einem Ort
- Shadcn/ui / Tailwind CSS:
  - Dynamische UI Bibliothek: nur benötigte Teile werden dem Projekt hinzugefügt
  - Kohärent im Style
- PostgreSQL:
  - Relationale Datenbank
  - Kann mit JSON umgehen
  - Kompatibel mit Neon und Prisma
- Neon DB:
  - Serverless
    - Kein Server-Setup
    - Kein manuelles Scaling
  - Branches für Datenbank (alte Versionen bleiben gespeichert -> Rückmachbar)
  - Kompatibel mit PostgreSQL
- Prisma:
  - Type-Safety mit TypeScript
  - Sehr sauberes Schema-System
  - Einfache Migrationen
  - Gut lesbarer Query-Style (Im Vergleich zu SQL!)



Architektur

In Nextjs lebt sowohl das Backend, als auch das Frontend auf dem Server. Wenn von einem Nutzer die Website aufgerufen wird, konvertiert der Server die React Components in eine html/css/javascript Website. Das Login verläuft über einen externen Service (Clerk), zurückgegeben wird ein JSON WEB TOKEN (JWT). Dies ist ein mit RSA verschlüsselter Cookie, welcher bei jeder http-Anfrage an den Server weitergegeben wird. Dieser verifiziert den JWT mit Clerk, bevor er Aktionen mit durchführt, welche ein Login benötigen. Muss etwas von der Datenbank abgerufen werden, greift der Server über Prisma auf die Datenbank zu. Dadurch lassen sich JSON Objekte zurückgeben, Migrationen verwalten und der Datenbank-Code lässt sich in TypeScript statt SQL schreiben.



<u>Projektstruktur</u>

Für den Entwickler gibt es 4 wichtige Ordner: actions, app, components, lib, prisma

Actions

Beinhaltet Funktionen für Interaktionen von Server zu Datenbank.

App

Im Grunde genommen ist dieser Ordner für die eigentlichen Seiten der Website verantwortlich. Will man eine weitere Seite hinzufügen, fügt man einfach einen Ordner mit dem Namen der gewollten Subdomain hinzu. Diese Ordner sind dann von folgender Struktur:

- page.tsx
    - die eigentliche Seite, die gerendert an den Client geschickt wird
- layout.tsx
    - dient dazu, Komponenten auf der Seite hinzuzufügen, welche bei Subdomains der eigenen Domain bestehen bleiben. Beispielsweise eine Navbar.

- Ordner
    - Optional, dadurch werden Subdomains dieser Domain erstellt.

Zusätzlich gibt es weitere Dateien, welche dieser Struktur hinzugefügt werden können, ein Beispiel dafür ist not-found.tsx . Diese Datei lässt den Entwickler eine eigene 404 page not found für alle Subdomains darstellen.

Components

Dieser Ordner ist in zwei Teile gegliedert:

- Unterordner ui:
    - Enthält alle von shadcn/ui importierten ui Komponenten. Beinhalten keine zusätzliche Funktion, rein graphisch.
- Komponenten:
    - Alle selbsterstellten Komponenten, aus denen die Website aufgebaut ist. Beispielsweise die Navbar oder der Chat-Komponente.

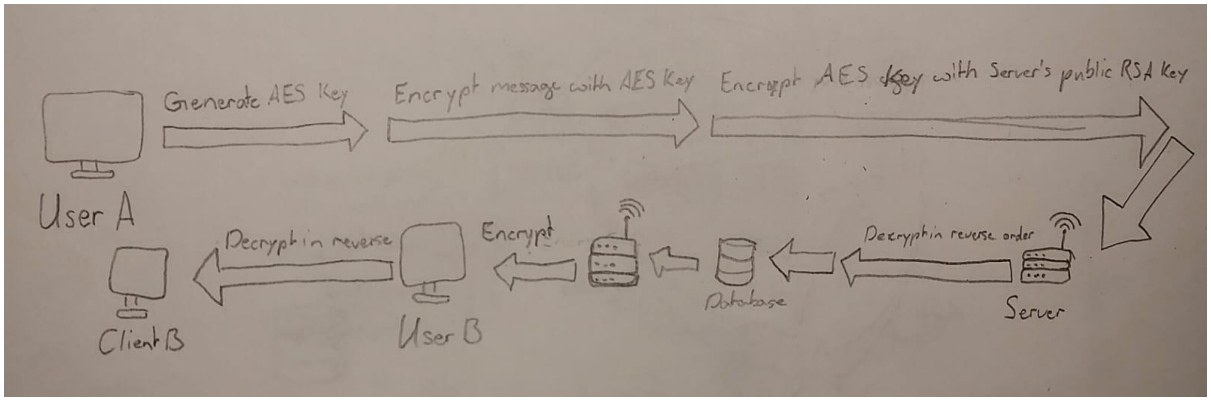
Lib

Hier sind Helferfunktionen definiert, welche nicht mit der Datenbank interagieren (actions!) und projektweit verwendet werden.

Prisma

Migrations: Enthält alle vorherigen Migrationen, sodass man zu einem früheren Aufbau der Datenbank gehen kann, falls etwas schief läuft.

Schema.prisma : enthält den Aufbau der Datenbank



Verschlüsselung (bis jetzt nur auf Chats angewendet)

In diesem Projekt wird eine Hybrid-Verschlüsselung mit AES und RSA verwendet. AES ist eine symmetrische Verschlüsselung, das heisst, dass man zum verschlüsseln und entschlüsseln den gleichen Schlüssel braucht. Dafür muss man aber einen Schlüssel austauschen. Hier kommt RSA ins Spiel: RSA (asymmetrisch) verschlüsselt nur den Schlüssel der AES Verschlüsselung mit dem öffentlichen Schlüssel des Rezipienten und schickt diesen zusammen mit der in AES verschlüsselten Nachricht an den Rezipienten.