

SSE NOTIZANWENDUNG

Moritz Foglia

Alexander Afanasjew



Inhaltsverzeichnis

- Anwendung
- Verwendete Technologien
- Verwendete Bibliotheken
- Registrierung
- Anmeldung
- Autorisierung
- Notizerstellung

Inhaltsverzeichnis

- Linkstruktur der Notiz
- Notiz suchen
- Passwort Zurücksetzen
- CI/CD
- Weitere Sicherheitsrisiken

Die Anwendung

- Nutzer registrieren
- Nutzer können sich mit Credentials und Discord einloggen
- Nutzer können Notizen mittels Markdown und HTML-Syntax erstellen
- Nutzer können ihre Notizen suchen und anzeigen lassen
- Nutzer können ihr Passwort zurücksetzen

Verwendete Technologien

- Next.js -> Ein React-Framework, bietet Serverseitiges Rendering und statische Seitenerzeugung
- Auth.js -> Authentifizierungs Bibliothek
- TypeScript
- tRPC -> Erstellung von Endpunkten und APIs
- Prisma -> ORM. Schützt durch die Nutzung von Prepared Statements vor SQL-Injections
- Tailwind CSS -> Styling

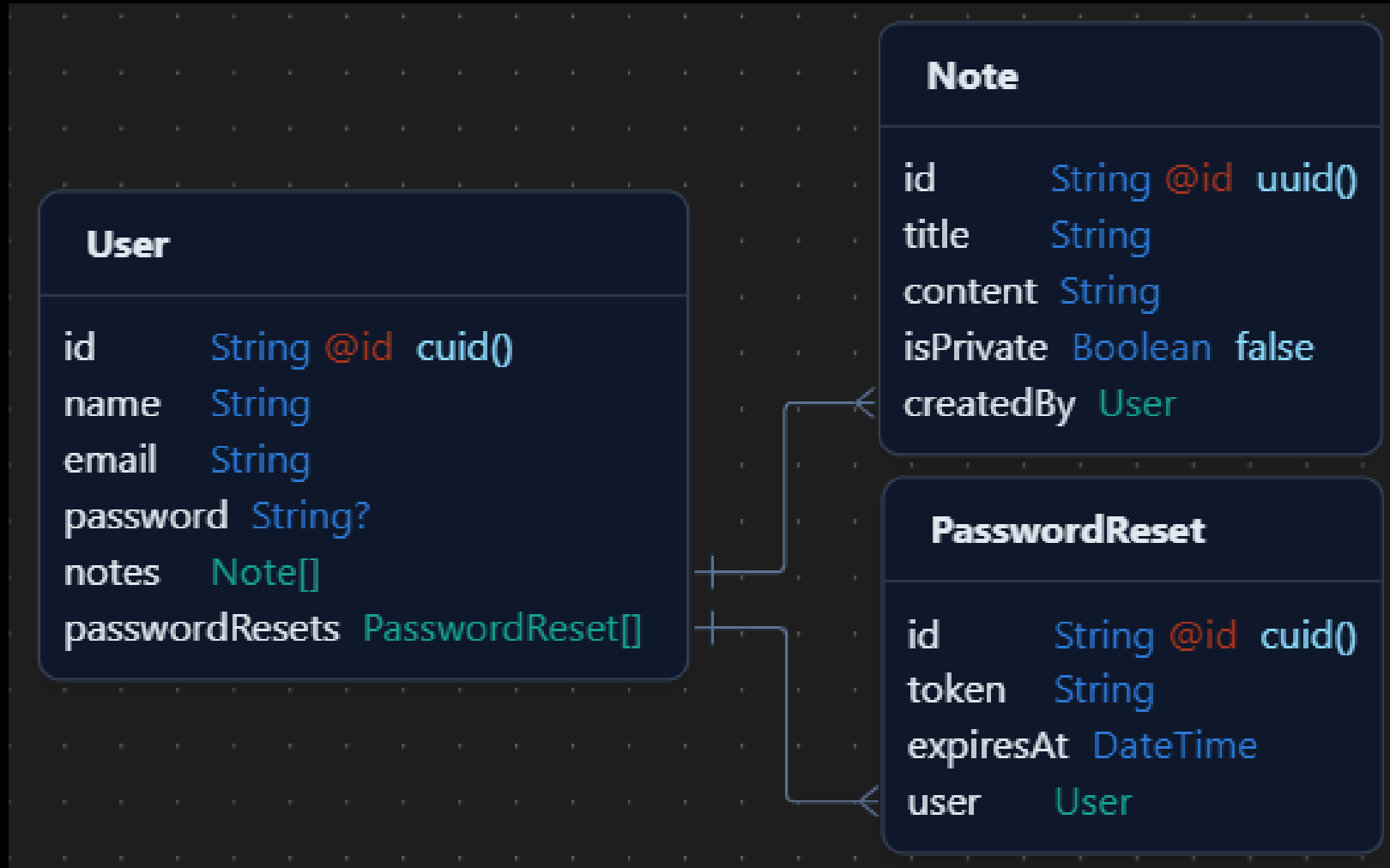
Verwendete Bibliotheken

- React-Markdown: Zum Rendern der Markdown/HTML-Syntax
- Rehype-Raw: React-Markdown-Plugin zum verarbeiten von HTML innerhalb von Markdown
- Remark-GFM: React-Markdown Plugin zur Unterstützung von GitHub Flavored Markdown
- Dompurify: Zur Sanitisierung der Markdown und HTML-Eingabe zum Schutz vor XSS
- Bcrypt: Zum Hashen und Salten von Passwörtern, Passwörter abgleichen

Verwendete Bibliotheken

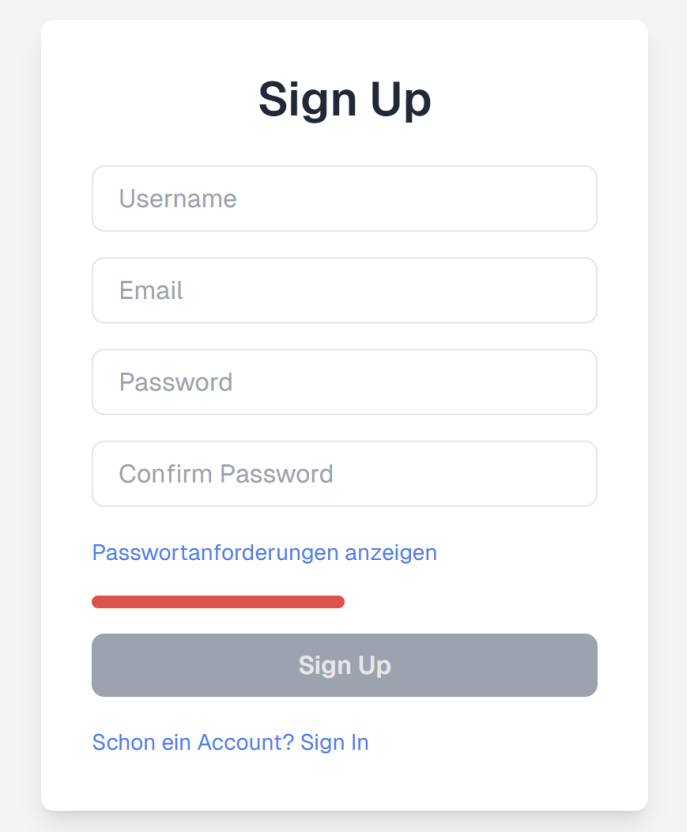
- Zod: Zur Validierung von Eingaben
- tailwind typography plugin: Dafür da, um die gerenderte Notiz zu stylen.
- zxcvbn-ts: Bibliothek zum Testen der Passwortstärke
- playwright: Bibliothek zum Tests schreiben
- nodemailer: Bibliothek zum Versenden von E-Mails

Datenbank Schema



REGISTRIERUNG

- Formfelder ausfüllen
- Starkes Passwort eingeben
- Darauf achten, dass der Balken grün ist
- Sign Up klicken
- Bei Erfolg wird der Nutzer auf Sign In weitergeleitet

A mockup of a 'Sign Up' form. The form is white with rounded corners and a subtle shadow. It contains four input fields: 'Username', 'Email', 'Password', and 'Confirm Password'. Below the 'Password' field is a link 'Passwortanforderungen anzeigen' in blue. A red progress bar is shown below the link. At the bottom of the form is a grey 'Sign Up' button. Below the button is a link 'Schon ein Account? Sign In' in blue.

Sign Up

Username

Email

Password

Confirm Password

[Passwortanforderungen anzeigen](#)

Sign Up

[Schon ein Account? Sign In](#)

REGISTRIERUNG

RISIKEN

- SQL-Injections
- Zu viele Anfragen senden
- Zu schwaches Passwort

SCHUTZMAßNAMEN

- Prepared Statement mit Prisma
- Rate Limiting
- Passwortstärke prüfen

Registrierung/ Route

```
• signup: publicProcedure
•   .input(
•     z.object({
•       username: z.string().min(2),
•       email: z.string().email(),
•       password: z.string(),
•     }),
•   )
•   .mutation(async ({ ctx, input }) => {
•     // Check if the user already has a session
•     if (ctx.session?.user) {
•       throw new Error("You are already logged in.");
•     }
•   })
```

Registrierung/ Passwortstärke prüfen

```
const options = {
  translations: zxcvbnDePackage.translations,
  graphs: zxcvbnCommonPackage.adjacencyGraphs,
  dictionary: {
    ...zxcvbnCommonPackage.dictionary,
    ...zxcvbnDePackage.dictionary,
  },
};

zxcvbnOptions.setOptions(options);
const passValidation = zxcvbn(input.password);
const pwstrength = passValidation.score;
if (pwstrength < 4) {
  throw new Error(
    passValidation.feedback.warning ?? "Passwort ist zu schwach",
  );
}
```

Registrierung/ Feldervalidierung

```
if (  
    !input.username.trim() ||  
    !input.email.trim() ||  
    !input.password.trim()  
) {  
    throw new Error("All fields must be filled out.");  
}
```

Registrierung/ Prüfung

```
const hashedPassword = await bcrypt.hash(input.password, 10);
const existingUser = await db.user.findFirst({
  where: {
    name: input.username,
  }
});
if (existingUser) {
  throw new Error("Username or Email already exists");
}
const existingEmail = await db.user.findFirst({
  where: {
    email: input.email,
  }
});
if (existingEmail) {
  throw new Error("Username or Email already exists");
}
```

Registrierung/ Nutzer erstellen

```
return await db.user.create({  
  data: {  
    name: input.username,  
    email: input.email,  
    password: hashedPassword,  
  },  
});  
}),
```

ANMELDUNG

- Über Auth.js
- Benutzeroberfläche und Passwort eingeben
- Sign In klicken
- Weiterleitung zum Dashboard
- Alternativ Discord-Sign-in
- Anweisungen von Discord befolgen

Sign In

Username

Password

Sign in with Discord

Sign In

[Noch kein Account? Sign Up](#)

[Passwort vergessen?](#)

Anmeldung/ Credentials

RISIKEN

- SQL-Injections
- Bruteforce
- CSRF

SCHUTZMAßNAMEN

- Prepared Statements mit Prisma
- Rate Limiting über traefik middleware
- Auth.js verwendet CSRF-Token

Anmeldung/ Discord

RISIKEN

- Abfangen des Access/ Refresh Token

SCHUTZMAßNAMEN

- Verbindung über HTTPS
- OAuth2-Standardprotokoll

Anmeldung/ Eingabenprüfung

```
const credentialsSchema = z.object({  
  username: z  
    .string()  
    .min(1, "Username is required")  
    .max(100, "Username is too long"),  
  password: z  
    .string()  
    .min(1, "Password is required")  
    .max(200, "Password is too long"),  
});
```

Anmeldung/ Provider Konfiguration

```
port const authConfig = {  
  providers: [  
    DiscordProvider,  
  
    CredentialsProvider({  
      name: "Credentials",  
      credentials: {  
        username: { label: "Username", type: "text" },  
        password: { label: "Password", type: "password" },  
      },  
    })  
  ],  
}
```

Anmeldung/ Auth.js Konfiguration

```
export const { handlers, auth, signIn, signOut } = NextAuth({  
  trustHost: true,  
  session: {  
    strategy: "jwt",  
  },  
  ...authConfig,  
});
```

Anmeldung/ Aufruf der Funktionen

```
async function handleCredentialsSignIn() {  
  if (isFormValid) {  
    const result = await signIn("credentials", {  
      username,  
      password,  
      redirect: false,  
    }); ...  
  }  
  async function handleDiscordSignIn() {  
    await signIn("discord", { redirectTo: "/" });  
  }  
}
```

Anmeldung/ Funktion

```
async authorize(credentials, req: Request) {  
  try {  
    const validatedCredentials = credentialsSchema.parse(credentials);  
    const { username, password } = validatedCredentials;  
    console.log("User-Agent:", req.headers.get("user-agent"));  
    console.log("Redirect URL:", req.redirect);  
  }  
}
```

Anmeldung/ Funktion

```
const user = await db.user.findUnique({ where: { name: username } });

if (!user || !password || !user.password) {
  throw new Error("Invalid username or password");
}
const isValidPassword = await bcrypt.compare(password, user.password);

if (!isValidPassword) {
  throw new Error("Invalid username or password");
}

return user;
} catch (error) {
  console.error("Validation error:", error);
  throw new Error("Invalid username or password");
}
},
```


Anmeldung/ Callback

```
async jwt({ token, user, account, profile }) {  
  if (account?.provider === "discord") {  
    const existingUser = profile?.email  
      ? await db.user.findUnique({ where: { email: profile.email } }) : null;  
    if (!profile?.email) throw new Error("Email is required");  
    if (!existingUser) {  
      user = await db.user.create({  
        data: {  
          name: profile?.username as string,  
          email: profile?.email  
        },  
      },  
    );  
  }  
});
```

```
    } else {  
        user = existingUser;  
    }  
}  
if (user) {  
    token.id = user.id;  
}  
return token;  
},
```

Autorisierung

- Regex-Matcher in der Middleware legt öffentlich erreichbare Ressourcen fest
 - > *für alle anderen muss man angemeldet sein*
- Verwendung von JWT Token um dem Nutzer eine Session zu geben
- Über den tRPC-Kontext wird die Session des Nutzers abgefragt (public/protected procedures)
 - > *Keine Session: Kein Zugriff auf protected procedures*

Autorisierung/ tRPC-Kontext

```
export const createTRPCContext = async (opts: { headers: Headers })  
=> {  
  const session = await auth();  
  return {  
    db,  
    session,  
    ...opts,  
  };  
};
```

Autorisierung/ Middleware

```
export const { auth: middleware } = NextAuth(authConfig);
```

```
export const config = { matcher:  
  [ "/((?!api|_next/static|_next/image|favicon.ico|auth/signup|auth/getmail|  
auth/reset-password/*).*)", ], };
```

Autorisierung/ Middleware

```
authorized: async ({ auth }) => {  
  return !!auth;  
},
```

Notiz Erstellung

- Titel in die Notiz eintragen
- Notiz mit Markdown und/ oder HTML Syntax schreiben
- Vorschau zeigt die Notiz an
- Speichern leitet den Nutzer weiter zu der erstellten Notiz

Erstelle eine Notiz

nach einem html tag muss eine Zeile frei sein, um mit markdown fortzufahren

Titel der Notiz

Schreibe hier deine Markdown/HTML-Tag Notiz...

☐ Mache diese Notiz privat

Speichern

Vorschau:

Notiz Erstellung

RISIKEN

- SQL injections
- XXS-Scripting

SCHUTZMECHANISMEN

- Prisma Prepared Statements
- Eingabe wird durch DOMPurify geprüft
- Vor jedem rendern einer Notiz wird diese mit DOMPurify gesäubert

Notiz Erstellung/ Preview

```
useEffect(() => {  
  if (content) {  
    setPreview(DOMPurify.sanitize(content));  
  }  
}, [content]);
```

```
<ReactMarkdown  
  className="prose"  
  rehypePlugins={[rehypeRaw]}  
  remarkPlugins={[remarkGfm]}  
  >  
  {preview}  
</ReactMarkdown>
```

Erstelle eine Notiz

nach einem html tag muss eine Zeile frei sein, um mit markdown fortzufahren

Titel

```
# Dies ist eine Notiz
## Eine tolle Notiz
<h3> Hier geht sogar HTML </h3>

<script>Scripts gehen hier nicht</script>
```

☐ Mache diese Notiz privat

Speichern

Vorschau:

Dies ist eine Notiz

Eine tolle Notiz

Hier geht sogar HTML

Notiz Erstellung/ Speichern

```
const sanitizedContent = DOMPurify.sanitize(content);

.mutation(async ({ ctx, input }) => {
  if (!ctx.session.user) {
    throw new Error("Unauthorized: User not logged in");
  }

  const note = await ctx.db.note.create({
    data: {
      title: input.title,
      content: input.content,
      createdBy: { connect: { id: ctx.session.user.id } },
      isPrivate: input.isPrivate,
    },
  });
});
```

Linkstruktur der Notizen/ Notiz anzeigen

localhost:3000/documents/ab50e763-9325-461a-9e05-4f5e3f0ea9c3

- documents/uuid
- Teilbar mit anderen Nutzern
- Notiz wird vor dem Rendern mit DOMPurify geprüft

Linkstruktur der Notiz/ Notiz anzeigen

RISIKEN

- Gezielter Zugriff auf Notizen eines anderen Nutzers
- XSS durch kompromittierte Datenbank

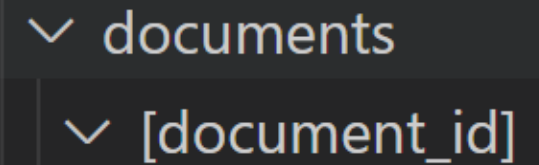
SCHUTZMECHANISMEN

- Dynamischer Link mit der UUID der Notiz
- Säuberung des Notiz durch DOMPurify

Linkstruktur der Notiz

```
async function DocumentPage({
  params,
}: {
  params: Promise<{ document_id: string }>;
}) {
  const document_id = (await params).document_id;
```

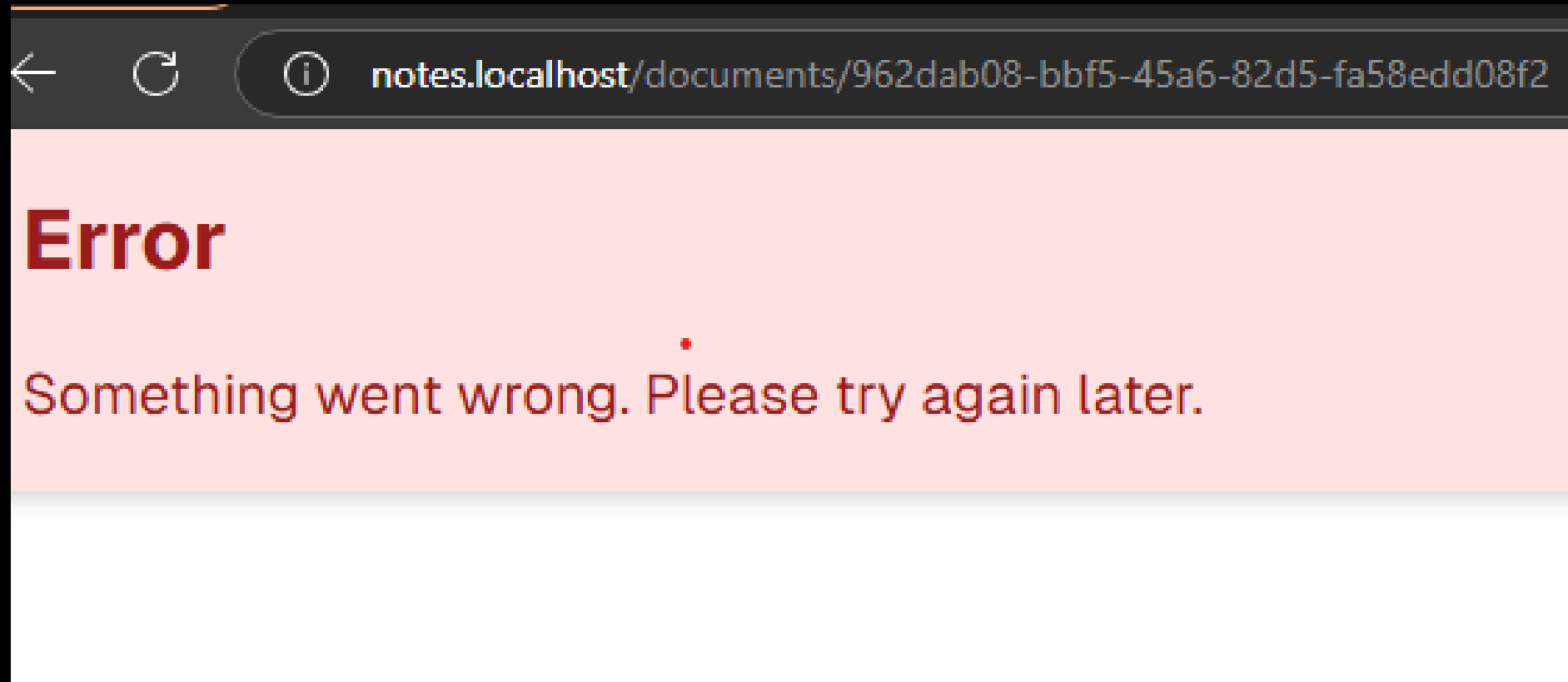
UUID wird aus der URL geholt



```
▼ documents
  ▼ [document_id]
```

Notiz anzeigen

```
export default function Note({ content }: { content: string }) {
  const [sanitizedContent, setSanitizedContent] = useState("");
  useEffect(() => {
    setSanitizedContent(DOMPurify.sanitize(content));
  }, [content]);
  return (
    <div className="preview w-auto rounded-md border p-4 shadow-sm">
      <ReactMarkdown
        className="prose items-center justify-center,,
        rehypePlugins={[rehypeRaw]}
        remarkPlugins={[remarkGfm]}
      >
        {sanitizedContent}
      </ReactMarkdown>
    </div>
  );
}
```



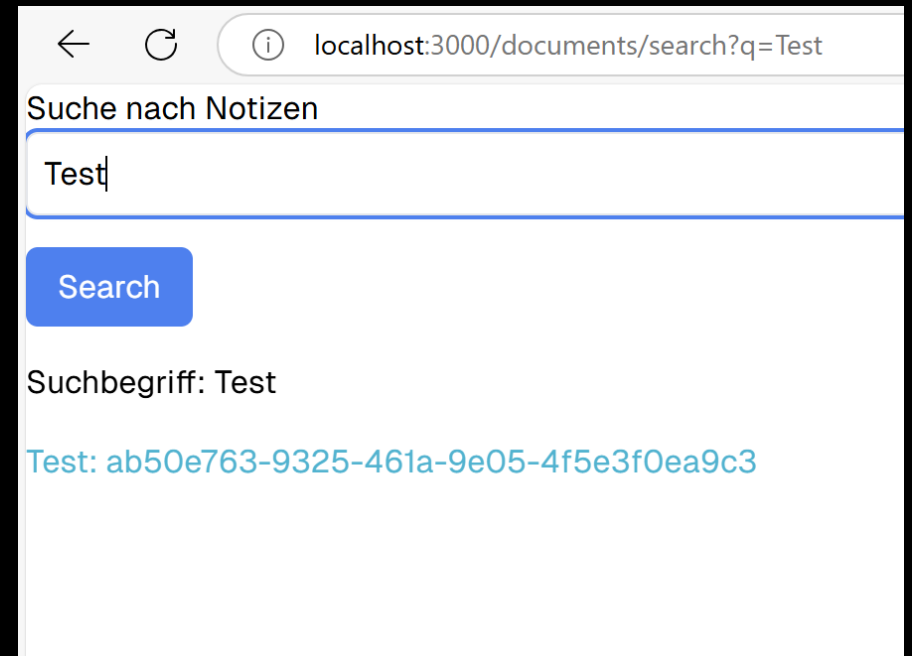
Morgentliche Aufgaben

- Aufstehen
- Frühstück
- Duschen
- **SSE Machen**

Erstellt von awawddawd

Notiz Suche

- Ein Wort des Titels eingeben
- Suchen Klicken
- Link zur Notiz wird angezeigt
- Klicken zum Weiterleiten
- Suchbegriff wird in der URL angezeigt



Notiz Suche

RISIKEN

- Angriff über gefährliche Zeichen in der URL
- SQL-Injections

SCHUTZMAßNAHMEN

- Encodierung der Eingabe vor dem Hinzufügen zur URL
- Prepared Statement mit Prisma

Notiz Suche/ Frontend

```
const pathname = usePathname();  
const searchParams = useSearchParams();  
const query = searchParams.get("q");  
const { data } = api.note.getNoteIdByTitle.useQuery(query);  
  
router.push(`?q=${encodeURIComponent(title)}`);
```

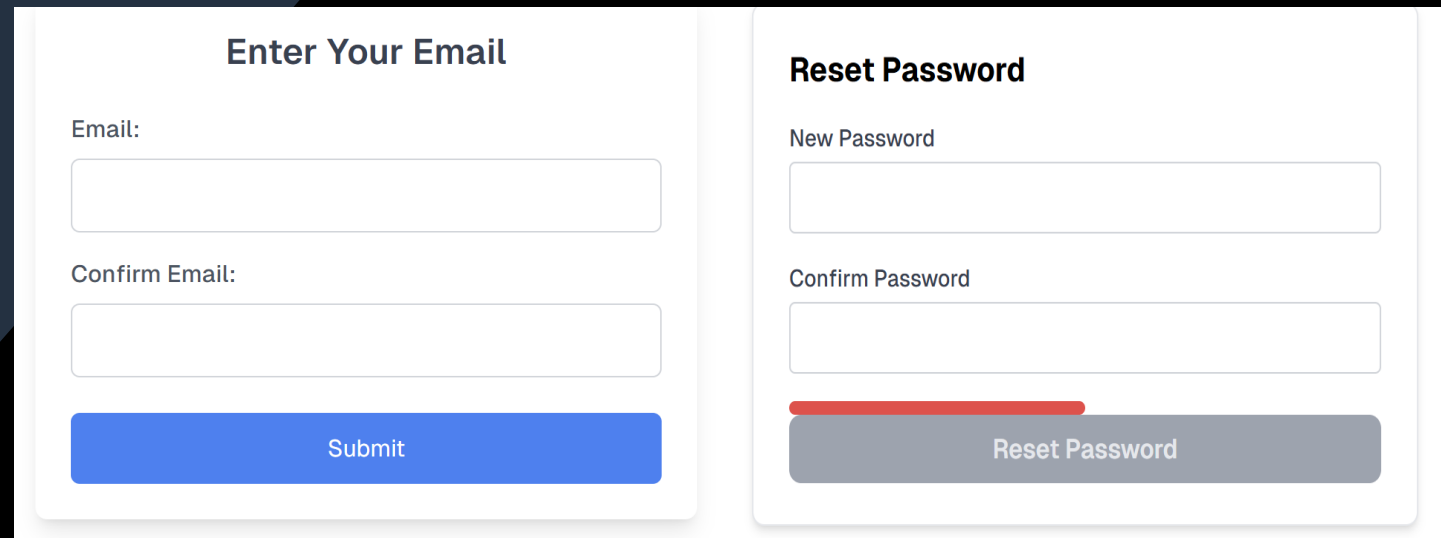
Notiz Suchen/ Backend Query

```
if (!input) return [];  
const ids = await ctx.db.note.findMany({  
  where: {  
    title: {  
      search: input,  
    },  
    createdById: ctx.session.user.id,  
  },  
  select: {  
    title: true,  
    id: true,  
  },  
});
```

```
if (!ids) throw new TRPCError({ code: "NOT_FOUND" });
```

Passwort Zurücksetzen

- E-Mail eingeben und auf Submit klicken
- Mailhog empfängt die E-Mail
- Auf Link klicken
- Passwort zurücksetzen

A user interface for password reset, divided into two panels. The left panel, titled 'Enter Your Email', contains two input fields labeled 'Email:' and 'Confirm Email:', followed by a blue 'Submit' button. The right panel, titled 'Reset Password', contains two input fields labeled 'New Password' and 'Confirm Password', followed by a red progress bar and a grey 'Reset Password' button.

Passwort Zurücksetzen

RISIKEN

- SQL-Injections
- Unsicherer Token
- Bruteforce

SCHUTZMAßNAHMEN

- Prepared Statements durch Prisma
- Generierung eines pseudzufälligen 32 Byte Token
- Token ist einmal Nutzbar
- Token ist eine Stunde haltbar
- Ratelimiting

Passwort Zurücksetzen/ E-Mail senden/ Token erstellen

```
const user = await db.user.findUnique({  
  where: { email: input.email },  
});  
  
if (!user) {  
  return { message: "Password reset email sent!" };  
}  
  
const resetToken: string = crypto.randomBytes(32).toString("hex");  
const expiresAt = new Date(Date.now() + 3600000);
```


Password Zurücksetzen/ E-Mail senden/ Token erstellen

```
await db.passwordReset.create({  
  data: {  
    userId: user.id,  
    token: resetToken,  
    expiresAt: expiresAt,  
  },  
});
```

```
await sendPasswordResetEmail(input.email, resetToken);  
return { message: "Password reset email sent!" };
```

Password Zurücksetzen/ Token Prüfen

```
const reset = await db.passwordReset.findFirst({
  where: {
    token: input.token,
    expiresAt: {
      gte: new Date(),
    },
  },
});

if (!reset) {
  throw new Error("Invalid or expired token");
}
```

Passwort Zurücksetzen/ Passwort Prüfen und Hashen

```
const passValidation = zxcvbn(input.newPassword);
const pwstrength = passValidation.score;

if (pwstrength < 4) {
  throw new Error(
    passValidation.feedback.warning ?? "Password ist zu schwach",
  );
}

const hashedPassword = await bcrypt.hash(input.newPassword, 10);
```

Passwort Zurücksetzen/ Token löschen

//user wurde geupdated

```
await db.passwordReset.delete({  
  where: {  
    id: reset.id,  
  },  
});  
return { message: "Password reset successful" };
```

CI/CD

- Docker Build and Test ci-cd.yml
- Dependabot in Github

CI/CD Docker Build and Test

CODE

```
on:
  push:
    branches:
      - main
      - feature/*
  pull_request:
    branches:
      - main
      - feature/*
```

ERKLÄRUNG

- Definiert, wann der Runner ausgeführt werden soll
- Bei Push und Pullrequest auf main und feature/* Branch

CI/CD Docker Build and Test

CODE

```
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: lts/*
```

ERKLÄRUNG

- Definiert was gemacht werden soll
- Läuft auf Ubuntu
- Kopiert den Code des Repos
- Installiert Node.js

CI/CD Docker Build and Test

CODE

- name: Install Docker using Docker's official script

```
run: |  
    curl -fsSL https://get.docker.com -o get-docker.sh  
    sudo sh get-docker.sh
```

- name: Install Docker Compose

```
run: |  
    sudo curl -L "https://github.com/docker/compose/releases/download/v2.3.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
    sudo chmod +x /usr/local/bin/docker-compose  
    docker-compose --version
```

ERKLÄRUNG

- Docker und Docker Compose werden installiert

CI/CD Docker Build and Test

CODE

- name: Install dependencies
run: npm ci
- name: Start application using Docker Compose
run: docker compose up --build -d
working-directory: .
- name: Install Playwright Browser
run: npx playwright install --with-deps

ERKLÄRUNG

- Dependencies installieren
- Anwendung im Hintergrund starten
- Playwrightbrowser installieren

CI/CD Docker Build and Test

CODE

```
- name: Run Frontend tests
  run: npx playwright test tests/frontendTest.spec.ts
- name: Run Backend tests
  run: npx playwright test tests/backendTest.spec.ts

- uses: actions/upload-artifact@v4
  if: ${{ !cancelled() }}
  with:
    name: playwright-report
    path: playwright-report/
    retention-days: 30
```

ERKLÄRUNG

- Frontend und Backend Tests werden ausgeführt
- Lädt Test-Report hoch

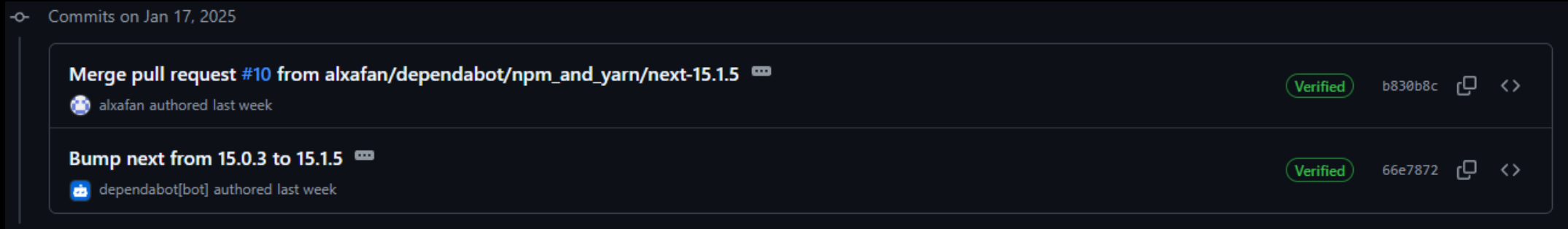
CI/CD Dependabot GitHub-Einstellungen

- Dependabot Alerts

-> Signalisierung von Schwachstellen in verwendeten Bibliotheken

- Dependabot Security Updates

-> Automatische Pullrequest für obengenannte Schwachstellen, falls es einen Patch gibt



Weitere Schwachstellen

- Umgebungsvariablen im Repository
 - > *Verwendung einer .env-Datei*
- DDOS/ Bruteforce
 - > *Verwendung einer Reverse Proxy mit Ratelimiter über Traefik*
- HTTPS
 - > Haben es leider nicht geschafft mit Traefik HTTPS einzustellen

Traefik/ Konfiguration

traefik:

image: "traefik:v3.3"

container_name: "traefik"

command:

- "--api.insecure=true"
- "--providers.docker=true"
- "--providers.docker.exposedbydefault=false"
- "--entryPoints.web.address=:80"

ports:

- "80:80"
- "8080:8080"

volumes:

- "/var/run/docker.sock:/var/run/docker.sock:ro"

Traefik/ Konfiguration

labels:

- "traefik.enable=true"
- "traefik.http.routers.app.rule=Host(`notes.localhost`)"
- "traefik.http.middlewares.app.ratelimit.average=50"
- "traefik.http.middlewares.app.rateLimit.burst=100"
- "traefik.http.routers.app.middlewares=app@docker"