

SecureNotes

Einführung

User

Session

Notiz

Bonus

CI/CD

Ende

ABSCHLUSSPROJEKT: SECURE SOFTWARE ENGINEERING

Osman Ali Usal & Egesel Bozgeyik



SecureNotes

Einführung

User

Session

Notiz

Bonus

CI/CD

Ende

WAS WIR HABEN:

1. User Management (Anmelden, Einloggen)
2. Application Functions (Notiz erstellen, mit Link teilen)
3. Social Plugins (YouTube)
4. Suche
5. CI/CD Pipeline
6. Password vergessen
7. Google OAuth
8. API Key

SecureNotes

Einführung

User

Session

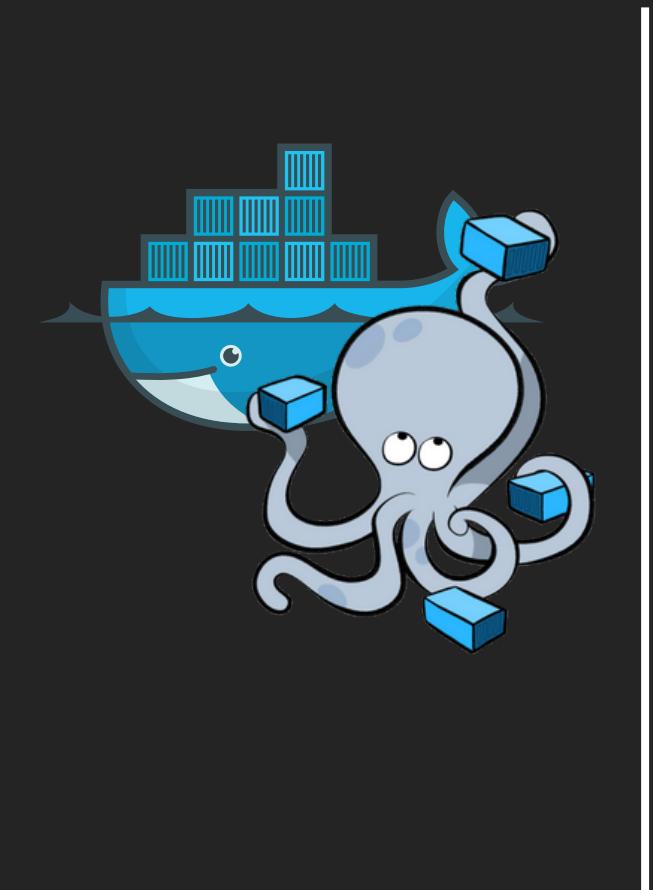
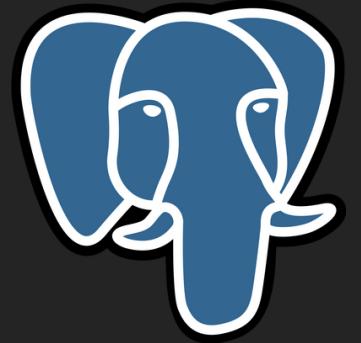
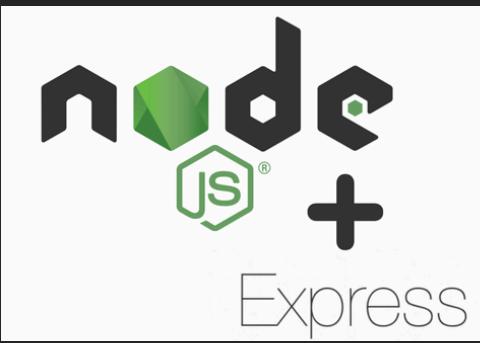
Notiz

Bonus

CI/CD

Ende

TECH STACK

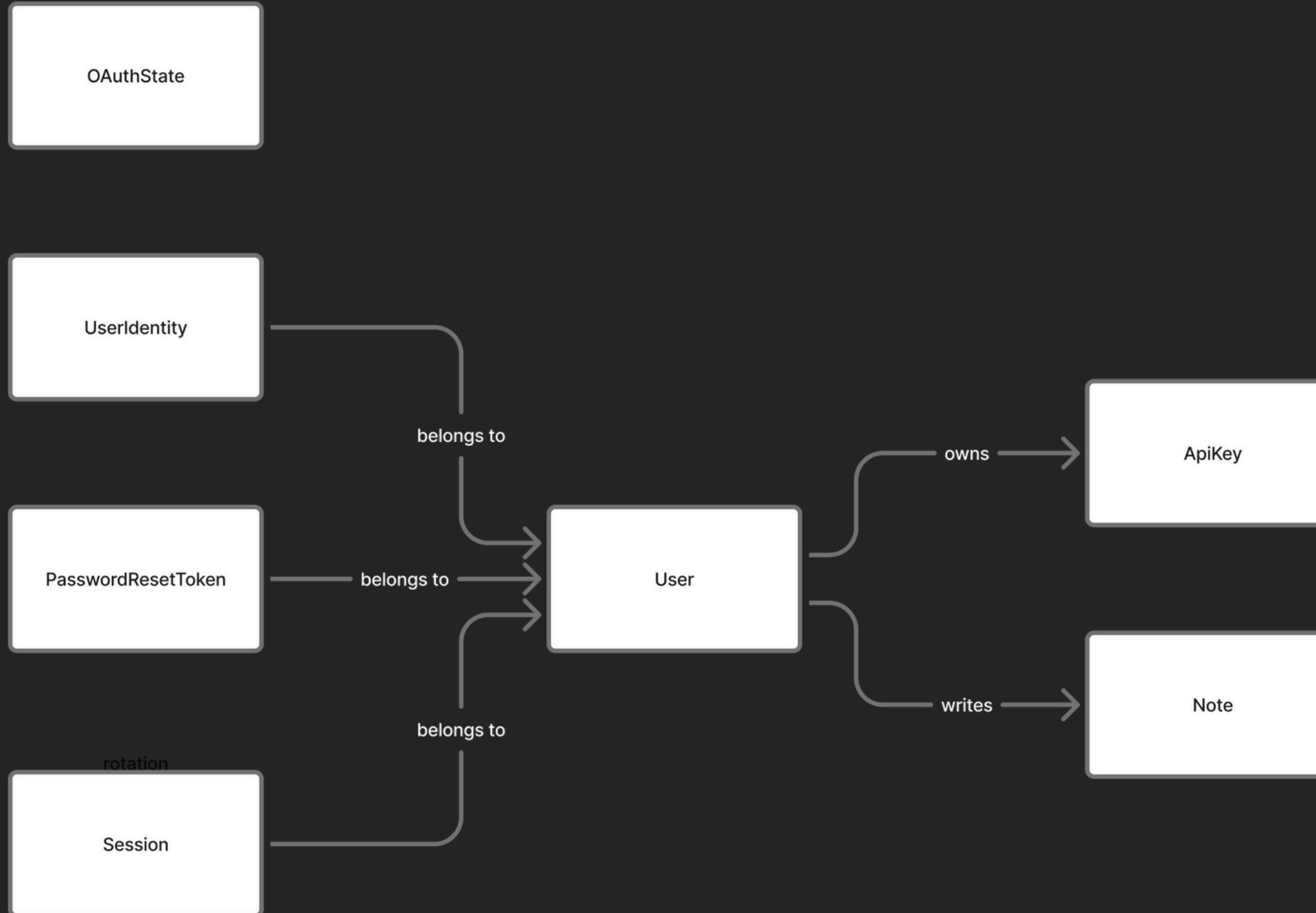


 tailwindcss

Osman Ali Usal & Egesel Bozgeyik

 THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Entity-Relationship-Diagramm



SecureNotes

Einführung

User

Session

Notiz

Bonus

CI/CD

Ende

BENUTZERVERWALTUNG

Anmeldung und Registrierung
durch E-Mail Adresse + Passwort



Osman Ali Usal & Egesel Bozgeyik

PASSWORTSICHERHEIT | Input Validation - Zod

```
export const passwordSchema = z
  .string()
  .min(8, { message: "Passwort muss mindestens 8 Zeichen lang sein." })
  .max(64, { message: "Passwort muss höchstens 64 Zeichen lang sein." })
  .refine((password) => /[A-Z]/.test(password), {
    message: "Passwort muss mindestens ein Großbuchstaben enthalten.",
  })
  .refine((password) => /[a-z]/.test(password), {
    message: "Passwort muss mindestens ein Kleinbuchstaben enthalten.",
  })
  .refine((password) => /[0-9]/.test(password), {
    message: "Passwort muss mindestens eine Zahl enthalten.",
  })
  .refine((password) => /[-+!@#$%^&*]/.test(password), {
    message: "Passwort muss mindestens ein Sonderzeichen enthalten.",
  });

```



PASSWORTSICHERHEIT | Password Strength - zxcvbn

```
export const passwordService = {
  assertStrong(password: string, opts: { userInputs: string[] }) {
    // basic min/max is already in Zod; here the policy
    const r = zxcvbn(password, opts.userInputs);
    if (r.score < 3) throw AuthError.badRequest("WEAK_PASSWORD");
  },
};
```

Das Zod-Schema erfordert formale Komplexität.

zxcvbn misst hingegen die Vorhersagbarkeit.

z. B.

Password1!

Berlin2026!

JohnDoe1!



PASSWORTSICHERHEIT | Hashing - Argon2id

```
async hash(password: string) {
  return argon2.hash(password, {
    type: argon2.argon2id,
    memoryCost: 19456,
    timeCost: 2,
    parallelism: 1,
  });
}

async verify(hash: string, password: string) {
  return argon2.verify(hash, password);
};
```



Das Passwort wird nur gehashed in die DB gespeichert.

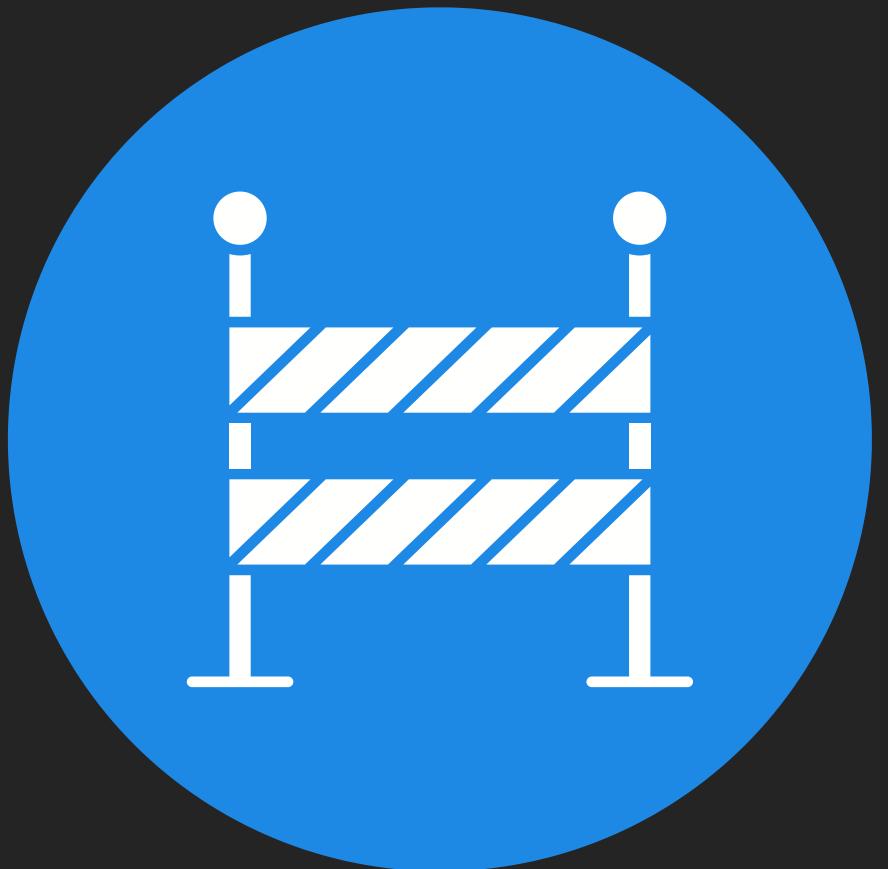
Argon2id: Industriestandard und gegen GPU-basierte Brute-Force-Angriffe
resistent.

PASSWORTSICHERHEIT | Rate Limiting

```
import rateLimit from "express-rate-limit";
```

Wenn innerhalb kurzer Zeit zu viele Anmelde- oder Registrierungsversuche von derselben IP-Adresse aus erfolgen, werden diese Anfragen vorübergehend blockiert.

```
router.post(
  "/register",
  createRateLimiter({ windowMs: 60 * 1000, max: 10 }),
  createAuthHandler
);
```



EMAILSICHERHEIT | Zod Schema + Normalization

```
export const emailSchema = z
  .string()
  .trim()
  .toLowerCase()
  .min(6, { message: "E-Mail ist zu kurz." })
  .max(254, { message: "E-Mail ist zu lang." })
  .pipe(z.email({ message: "Ungültige E-Mail-Adresse." }))
  .refine(
    (email) => {
      return !isDisposableEmail(extractDomain(email));
    },
    {
      message: "Ungültige E-Mail-Adresse.",
    }
  );

```

```
export function normalizeEmail(email: string): string {
  return email.trim().toLowerCase();
}
```



EMAILSICHERHEIT | isDisposableEmail

```
import { isDisposableEmail } from "disposable-email-domains-js";
```

Die Nutzung von temporären/Wegwerf-E-Mail-Diensten
(z. B. 10minutemail, mailinator) ist untersagt.

```
.refine(  
  (email) => {  
    return !isDisposableEmail(extractDomain(email));  
  },  
  r
```



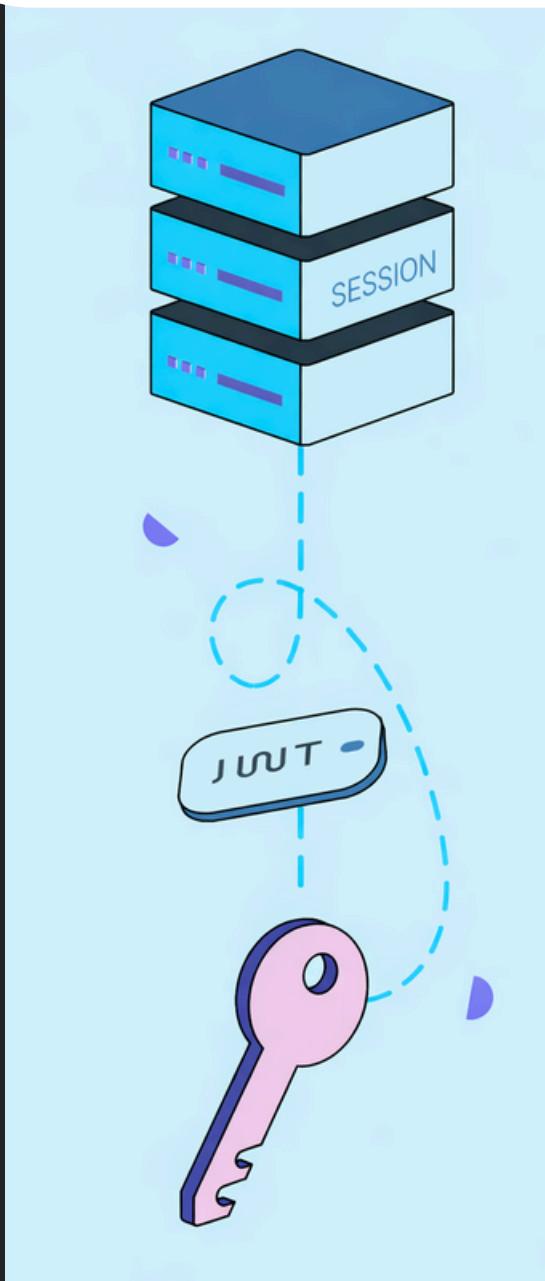
EMAILSICHERHEIT | Verifikation

- Token: 32 Bytes Random (SHA256-gehashed in DB)
- Mailpit für E-Mail-Testing in Development
- Verifikationslink: /verify-email?token=...

HTML **HTML Source** Text Headers Raw Checks ▾

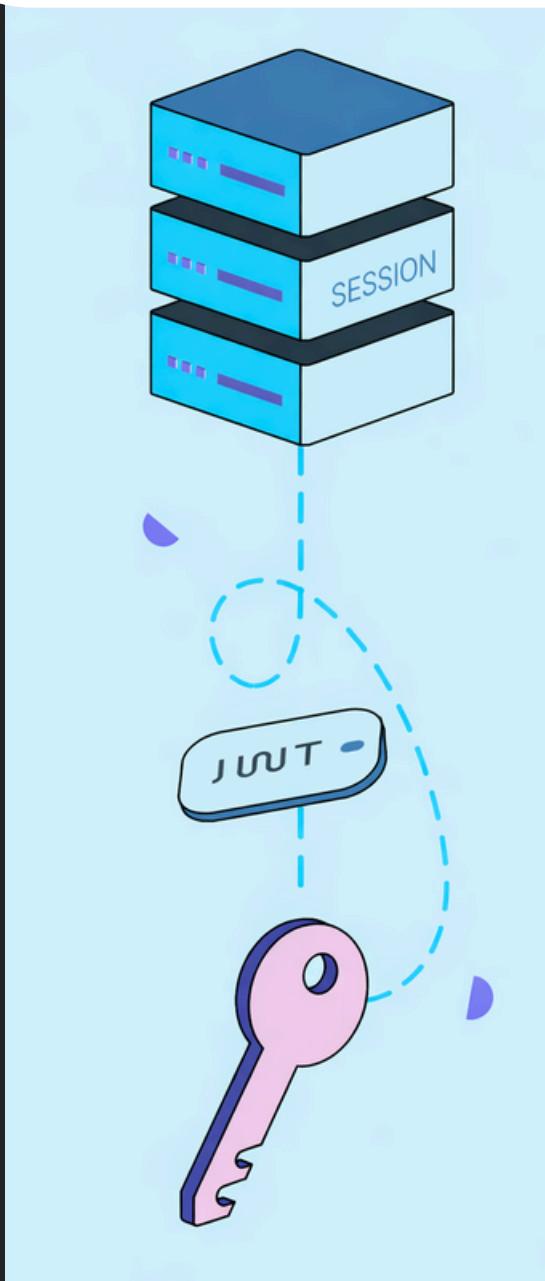
```
<div style="font-family:system-ui,Segoe UI,Roboto,Arial;line-height:1.4">
  <h2>E-Mail bestätigen</h2>
  <p>Hallo,</p>
  <p>bitte bestätige deine E-Mail-Adresse, indem du auf den folgenden Link klickst:</p>
  <p><a href="http://localhost/verify-email?token=d4b7bdf0852c86f23484a03dc6f01845c3bf0d58c51e73718006372a898d8bf0">E-Mail bestätigen</a></p>
  <p>Falls du dich nicht registriert hast, ignoriere diese Nachricht.</p>
</div>
```





SESSION MANAGEMENT

1. Access & Refresh Tokens
2. Cookie Sicherheit
3. CSRF-Schutz
4. Token-Rotation



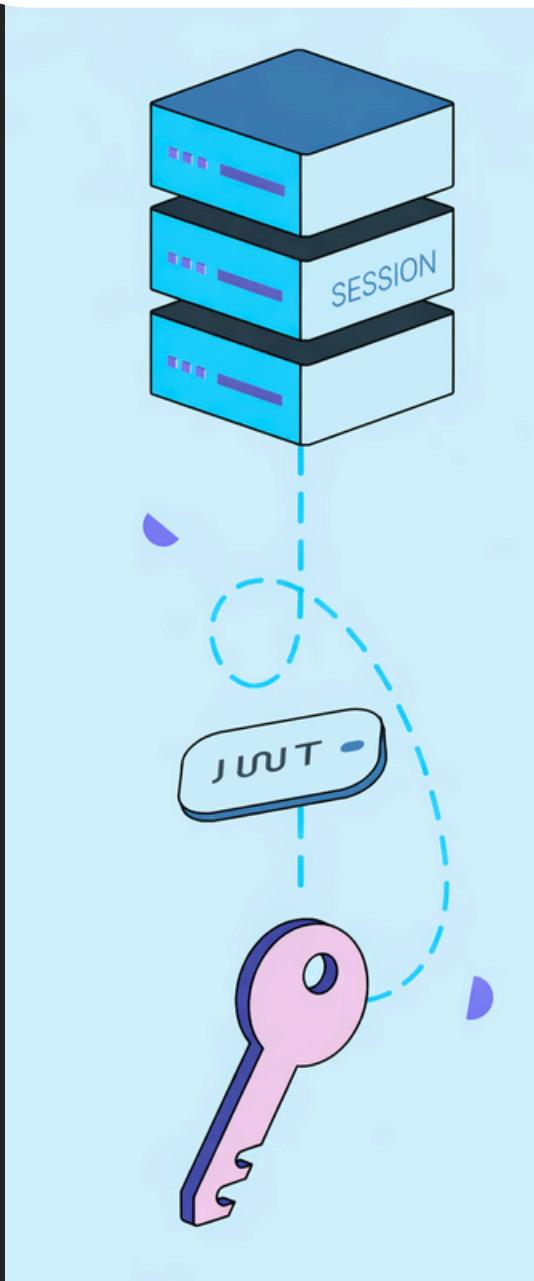
ACCESS & REFRESH TOKENS

Token Service: createRefreshToken und createAccessToken

Access Token: JWT + 10 min. API Abfragen *Authorization: Bearer <token>-*. In Memory clientseitig gespeichert.

```
async createAccessToken(params: { userId: string; sessionId: string }) {
  const { userId, sessionId } = params;

  return new SignJWT({
    sid: sessionId,
  })
    .setProtectedHeader({ alg: "HS256", typ: "JWT" })
    .setSubject(userId)
    .setIssuedAt()
    .setExpirationTime(`>${ACCESS_TOKEN_TTL_SECONDS}s`)
    .sign(jwtKey);
},
```



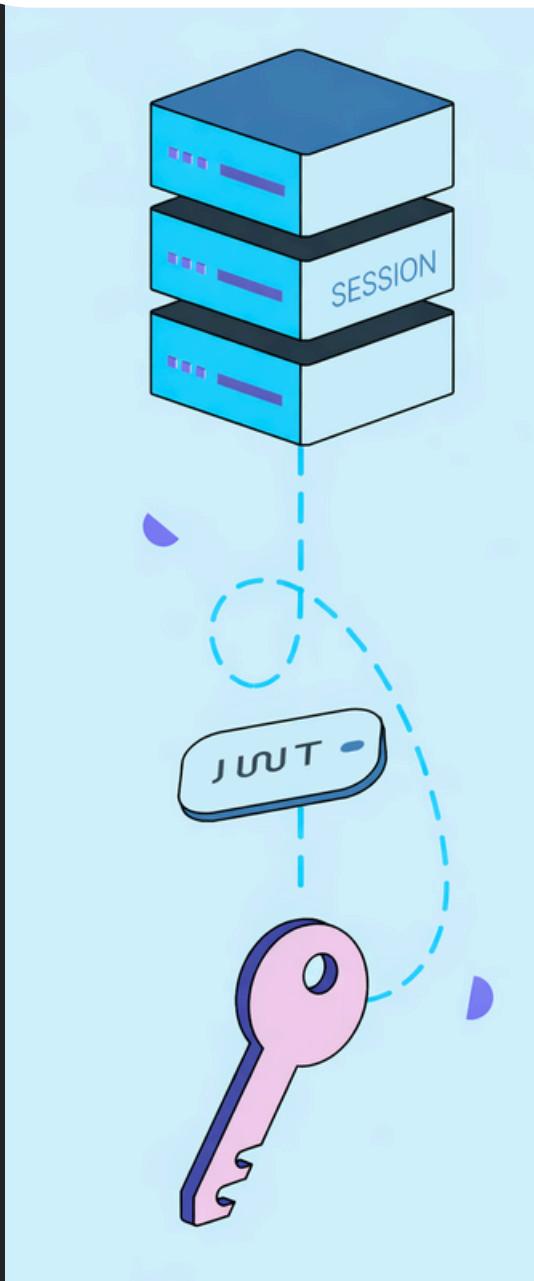
ACCESS & REFRESH TOKENS

Refresh Token: Random 48 Byte + 7 Tage. API Abfragen *Authorization: Bearer <token>-*.

In HttpOnly Cookie gespeichert. JS kann es nicht lesen.

Verwendung: Wird verwendet, um einen neuen Access-Token zu erhalten, wenn der alte abgelaufen ist.

```
createRefreshToken() {  
  const tokenPlain = crypto.randomBytes(48).toString("base64url");  
  const tokenHash = this.hashToken(tokenPlain);  
  return { tokenPlain, tokenHash };  
},
```



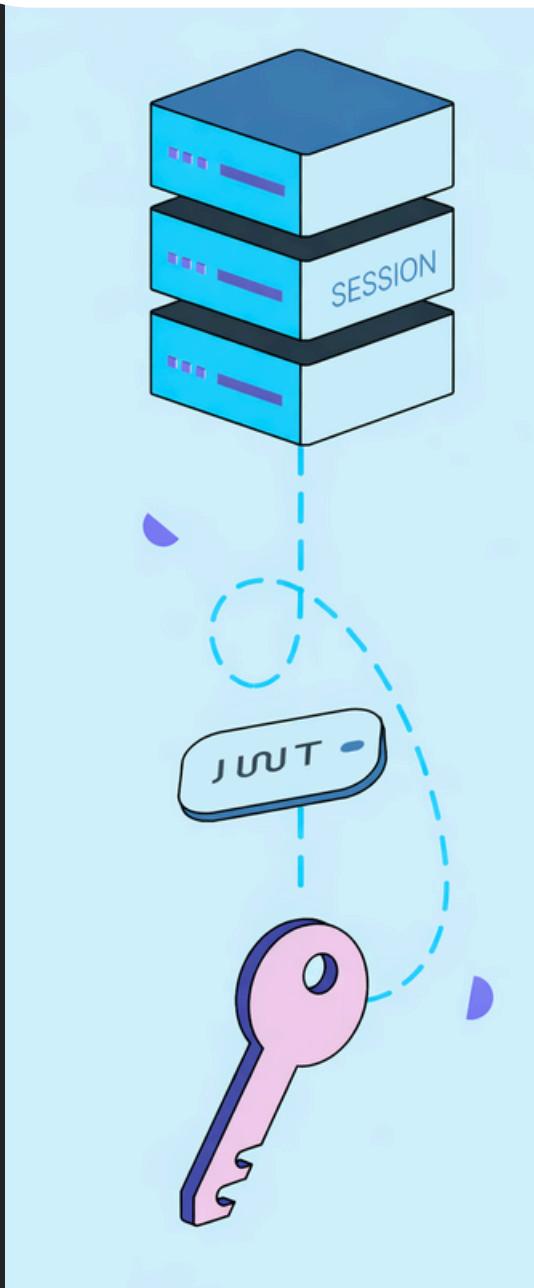
COOKIE SICHERHEIT

`httpOnly: true` → JS kann dieses Token nicht erreichen. Bei einem XSS-Angriff kann der Angreifer den Sitzungsschlüssel nicht stehlen.

`sameSite: "lax"` → Gegen CSRF Angriffe

`path: "/auth/refresh"` → Dieses Cookie wird NUR an den Erneuerungsendpunkt gesendet, nicht an andere unnötige Anfragen.

```
res.cookie(REFRESH_COOKIE_NAME, result.refreshTokenPlain, {  
    httpOnly: true,  
    secure: isProd,  
    sameSite: "lax",  
    path: "/auth/refresh",  
    maxAge: REFRESH_TOKEN_TTL_SECONDS * 1000,  
});
```



CSRF-SCHUTZ

Double Submit Cookie Pattern.

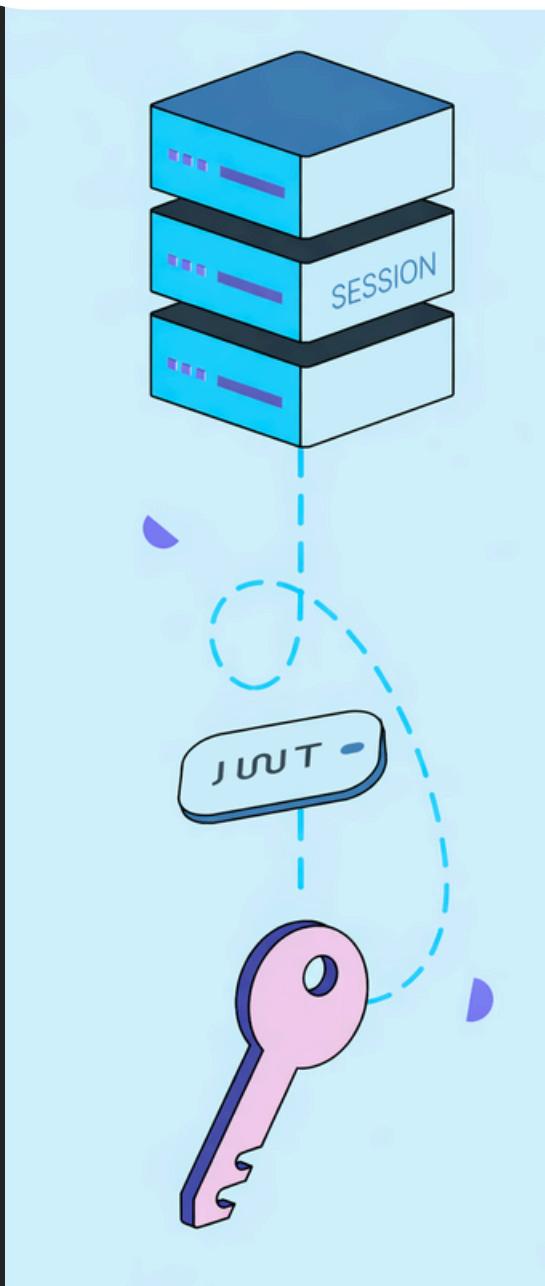
csrfGuard-Middleware überprüft bei sensiblen Anfragen (POST, DELETE), ob die Tokens sowohl im Cookie als auch im Header (x-csrf-token) übereinstimmen.

```
export function csrfGuard(req: Request, res: Response, next: NextFunction) {
  const m = req.method.toUpperCase();
  if (m === "GET" || m === "HEAD" || m === "OPTIONS") return next();

  const cookieToken = req.cookies?.[CSRF_COOKIE_NAME];
  const headerToken = req.get("x-csrf-token");

  if (!cookieToken || !headerToken) return res.status(403).json({ ok: false });
  if (cookieToken !== headerToken) return res.status(403).json({ ok: false });

  return next();
}
```



TOKEN-ROTATION

Normaler Ablauf (Token-Rotation): Bei jeder Verwendung eines Refresh-Tokens wird das alte Refresh-Token ungültig und der Benutzer erhält ein brandneues Refresh-Token.

Angriffsszenario (Wiederverwendung): Token_A wurde bereits verwendet und durch einen neuen ersetzt. Wenn jemand versucht, ihn erneut zu verwenden, bedeutet dies, dass dieser Token GESTOHLEN wurde!“

→ **Revoke All**

Sobald der Server die Gefahr erkennt, werden ALLE Sitzungen dieses Benutzers abgebrochen.

```
if (session.rotatedAt !== null) {  
    // if rotated: possible token reuse attack  
    await sessionRepository.revokeAllForUser(session.userId);  
    throw AuthError.unauthorized("UNAUTHORIZED");  
}
```

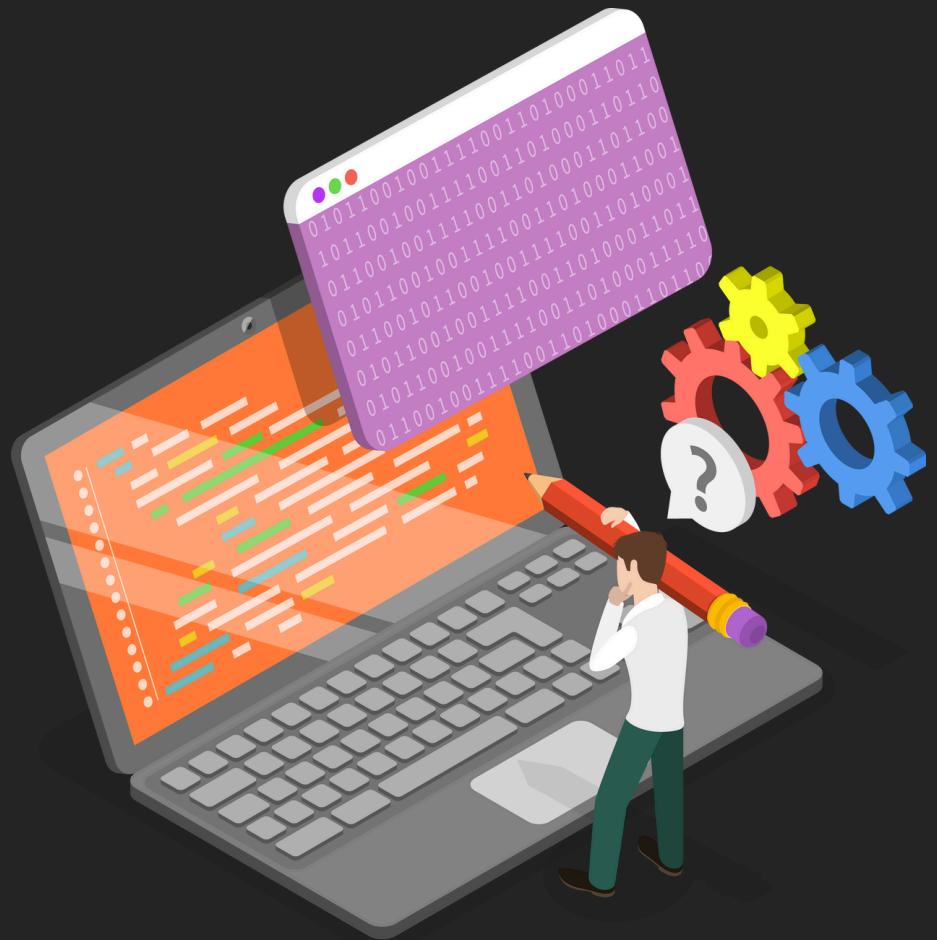
NOTIZ | Bearbeiten und Speichern

Notizen werden im Rohformat (Raw Markdown) in DB gespeichert.

Ist das sicher? → Ja, weil

- wir Schutz gegen SQL Injections haben (Prisma ORM)

```
export const createNote = async (input: CreateNoteInput, userId: string) => {
  const newNote = await prisma.note.create({
    data: {
      ...input,
      authorId: userId,
    },
  });
}
```



NOTIZ | Anzeigen (Markdown → HTML)

import { marked } from "marked";

Aufgabe: Markdown Symbole → HTML

z.B. **bold** → bold

Zusätzlich: Ein Custom Renderer für Youtube Videos.

```
renderer.image = ({  
    href,  
    title,  
    text,  
}: {  
    href: string;  
    title: string | null;  
    text: string;  
) => {  
    if (href && href.startsWith("embed:")) {  
        const rawUrl = href.substring(6);  
        const match = rawUrl.match(youtubeRegex);  
  
        if (match && match[4]) {  
            const videoId = match[4];  
            // Render responsive iframe  
            return `<div class="video-container">  
                <iframe  
                    src="https://www.youtube.com/embed/${videoId}"  
                    title="${text || "YouTube video"}"  
                    frameborder="0"  
                    allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture"  
                    allowfullscreen  
                </iframe>  
            </div>`;  
        }  
        return '<p style="color:red">[Unsupported YouTube URL]</p>';  
    }  
  
    // Fallback to default image rendering for standard images  
    return ``;  
};
```

NOTIZ | Anzeigen (HTML → Sicher HTML)

import sanitizeHtml from "sanitize-html";

Nimmt alles was von marked ausgegeben ist,
löscht alle gefährliche Inputs.

z.B.

<script>(Tags), onclick, onload(Event Attributes)...

Was nicht auf der allowed-Liste ist, wird gelöscht.

Was allowed: h1,h2,p, b, ul, li, iframe(nur für Youtube),
class, src usw.

```
renderer.image = ({  
  href,  
  title,  
  text,  
}: {  
  href: string;  
  title: string | null;  
  text: string;  
) => {  
  if (href && href.startsWith("embed:")) {  
    const rawUrl = href.substring(6);  
    const match = rawUrl.match(youtubeRegex);  
  
    if (match && match[4]) {  
      const videoId = match[4];  
      // Render responsive iframe  
      return `<div class="video-container">  
        <iframe  
          src="https://www.youtube.com/embed/${videoId}"  
          title="${text || "YouTube video"}"  
          frameborder="0"  
          allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture"  
          allowfullscreen>  
        </iframe>  
      </div>`;  
    }  
    return '<p style="color:red">[Unsupported YouTube URL]</p>';  
  }  
  
  // Fallback to default image rendering for standard images  
  return ``;  
};
```

NOTIZ | Zod Schemas

```
export const createNoteSchema = z
  .object({
    title: z.string().optional(),
    content: z
      .string()
      .min(1, "Content must not be empty")
      .max(100000, "Content must not exceed 100,000 characters"),
    visibility: z
      .enum([NoteVisibility.PUBLIC, NoteVisibility.PRIVATE])
      .default(NoteVisibility.PRIVATE),
  })
  .strict();

export type CreateNoteInput = z.infer<typeof createNoteSchema>;
```



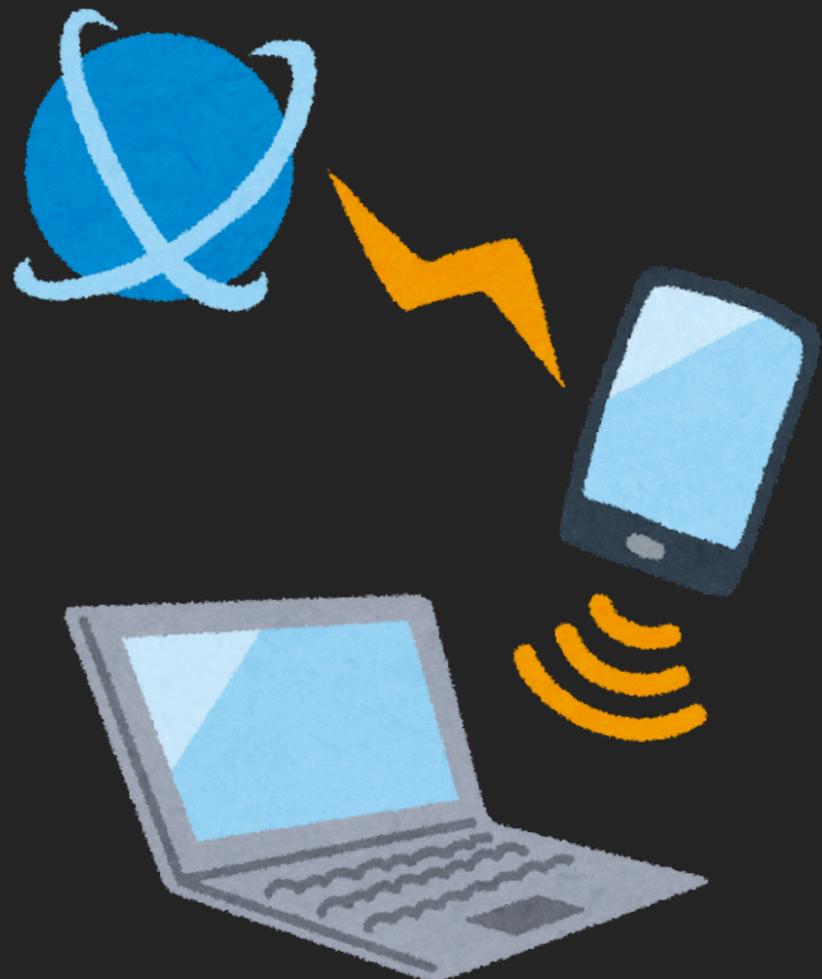
NOTIZ | Teilen

Link-Struktur: xyz.com/#/note/{note_id}

- Sicherheitsannahme: Nur authentifizierte Users können lesen

Keine ID Enumeration Protection:

```
model Note {  
    id      String @id @default(uuid())  
...  
}
```



API Keys

Benutzern können über Anwendung API Keys erstellen/löschen.

API Key: expiredDate

API Keys werden mit SHA-256 gehashed gespeichert.

Bei einer Abfrage wird **x-api-key** Header von Middleware geprüft.



SUCHE

/api/notes/search?query=gesuchtes-wort

Sicher gegen SQL Injections, da wir Prisma ORM haben.

Gegen reflected XSS: React macht HTML Escaping.

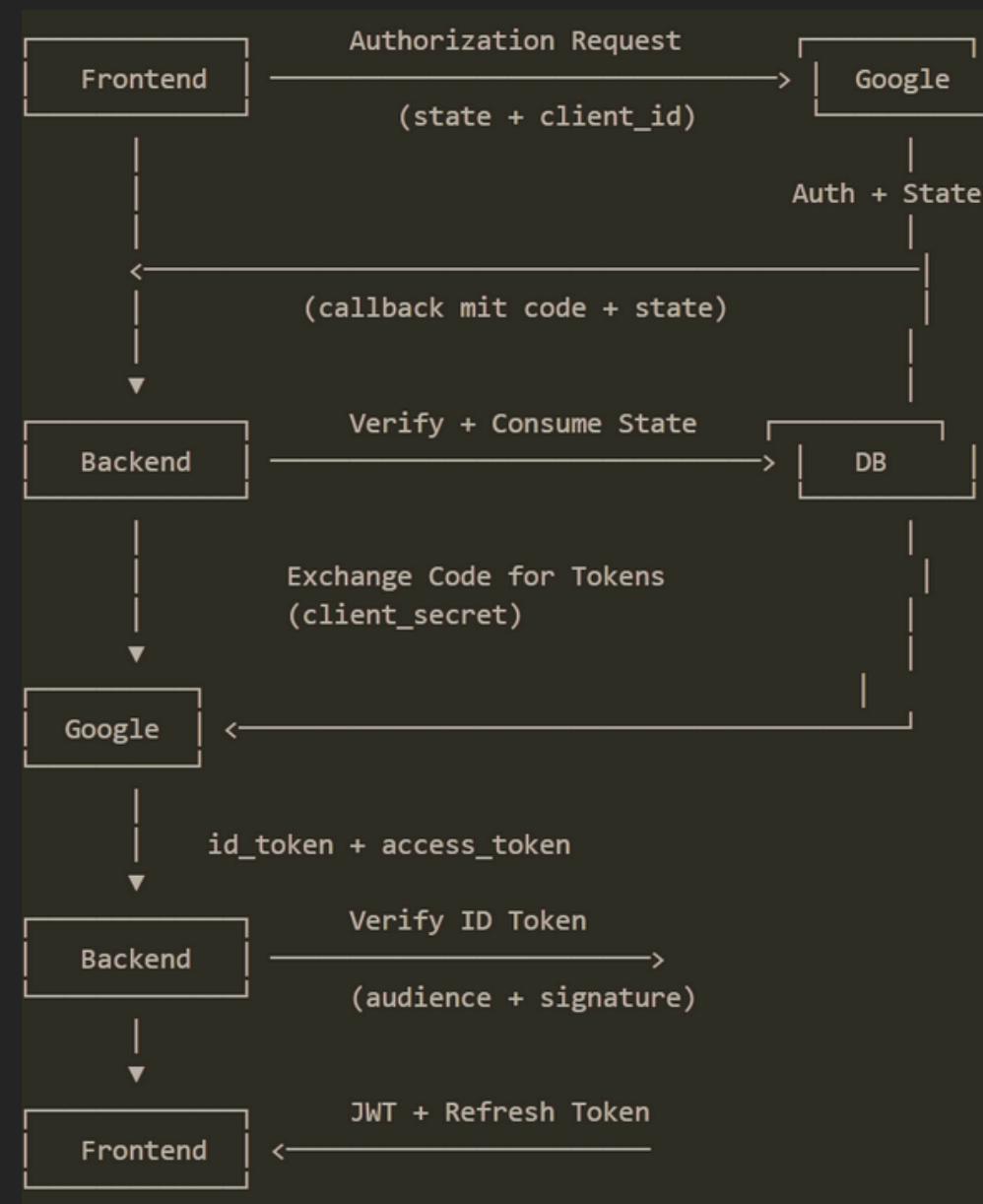
z. B. "<" wird "<"

Nur die Notizen ausgeben, die entweder Public sind oder dem Benutzer gehören.

```
visibilityCondition = {
    OR: [
        { visibility: NoteVisibility.PUBLIC },
        { visibility: NoteVisibility.PRIVATE, authorId: userId },
    ],
};
```



Google OAuth 2.0



Google OAuth 2.0

1- ID Token & Audience Kontrolle:

- *ID Token wird mit Audience-Parameter verifiziert.*
- *Schutz vor Token-Replay*
- *Nur für unsere App gültig*

2- State Parameter CSRF-Schutz:

- *Zufälliger 256-Bit State*
- *In DB gespeichert (10 Minuten gültig)*
- *Einmaliger Gebrauch (consumedAt)*

3- Server-seitiger Token Austausch:

- *Authorization Code Flow statt Implicit Flow*
- *Client Secret auf Backend*
- *Keine Google-Tokens an Client*



Google OAuth

4- OAuth State Management:

- Database-Modell
- Einmalig + Ablaufdatum

5- Least Privilege:

- Nur minimale Scopes
- OpenID, Email, Profile
- Keine unnötigen Permissions (Google Drive usw.)

6- Verification Libraries:

- google-auth-library
- Regelmäßige Security-Updates
- Community-getestet



Password Reset Flow

- Token-basiert: 30 Minuten gültig

Flow:

1. User klickt "Passwort vergessen"
2. E-Mail mit Reset-Link (Token in URL)
3. User setzt neues Passwort
4. Alle Sessions werden rotiert

Security:

- Token: SHA256-gehashed in DB
- Single-Use-Token nach Reset
- IP und User-Agent Tracking



SecureNotes

Einführung

User

Session

Notiz

Bonus

CI/CD

Ende

GitHub Actions Pipeline

1. *Pull Request Workflow*
2. *Merge Workflow*
3. *Dependency Updates*



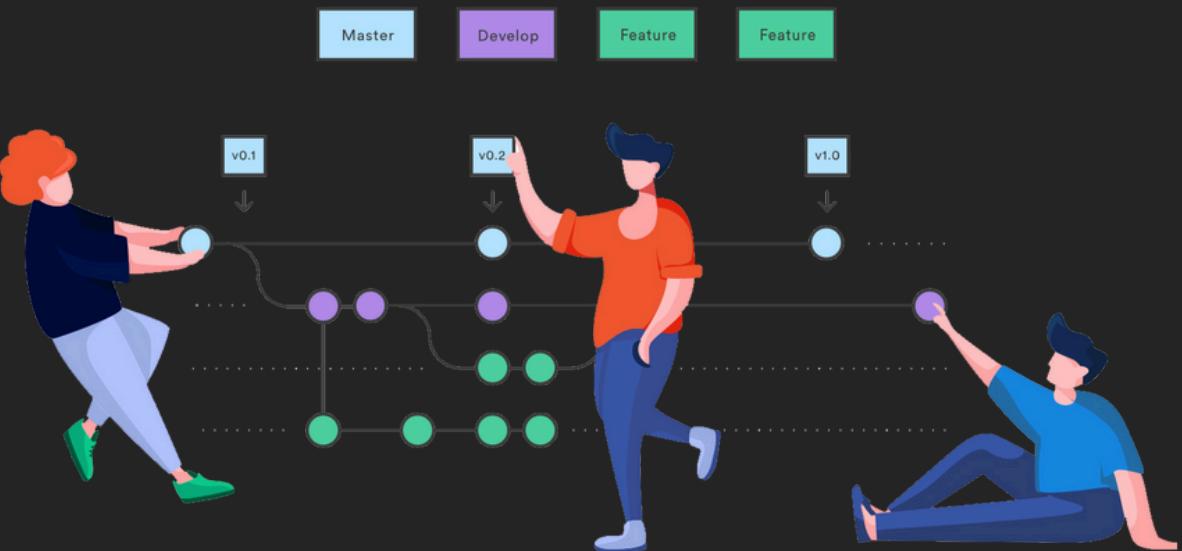
Osman Ali Usal & Egesel Bozgeyik

1- Pull Request Workflow

Workflow: verify.yml + pr-docker-build.yml + dast-scan.yml

Trigger: Pull Request auf main-Branch

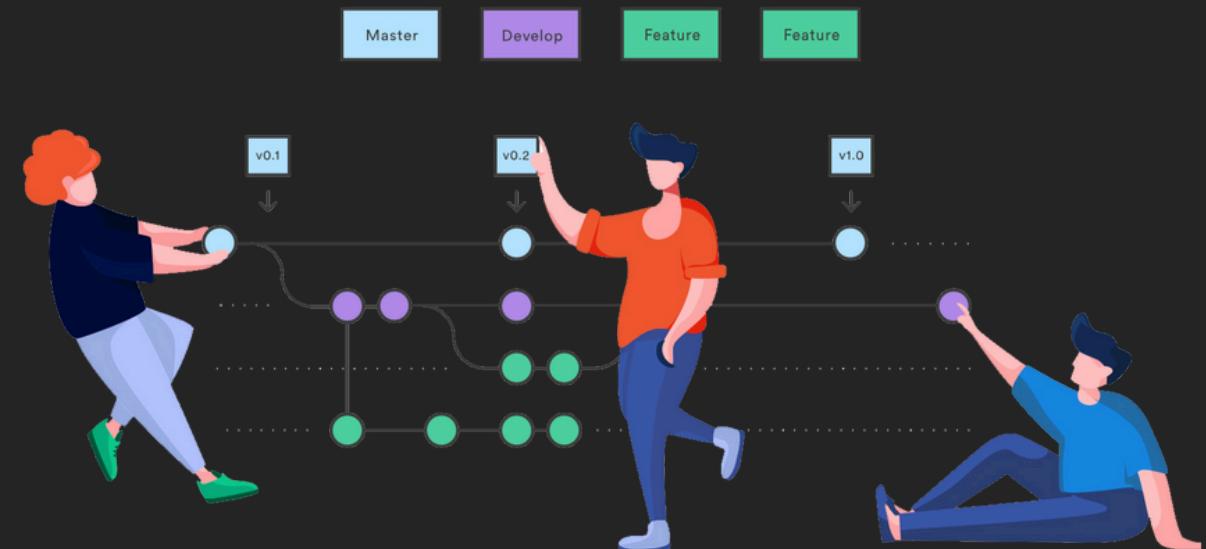
- Security Scans:
 - Gitleaks Secret Scan: Scannet komplette Git-History auf Secrets
 - Backend Security Audit + Frontend Security Audit
- Code Quality Checks
 - Backend: ESLint + TypeScript Type Check + Prisma Generate
 - Frontend: Build Check (kompiliert ohne Fehler)
- Tests
 - Backend Tests: Jest Tests mit Test-Datenbank
 - Frontend Tests: Vitest Tests
- Docker Build Test



1- Pull Request Workflow

- DAST Scan
 - Application starten: Docker Compose startet alle Services
 - **ZAP Scan**: OWASP ZAP scannt die laufende Anwendung
 - **Vulnerabilities reportet**: XSS, SQL Injection, etc.

Pipeline bricht ab, wenn ein Schritt fehlschlägt → PR wird blockiert

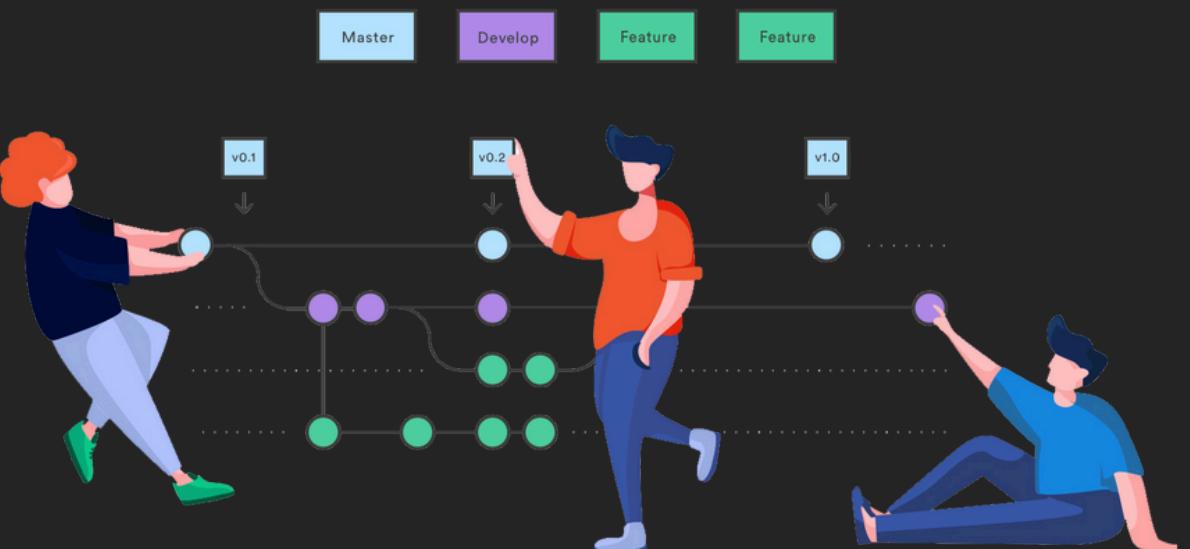


2- Merge Workflow

Workflow: deploy-to-registry.yml + deploy-docs.yml

Trigger: Push auf main-Branch (nach Merge)

- Docker Images bauen
- Container Registry Login
- Images pushen
- Dokumentation deployen (deploy-docs.yml)
 - MkDocs Build: Static HTML wird generiert
 - GitHub Pages: Wird auf egeselbzgyk.github.io/securenotes/deployt

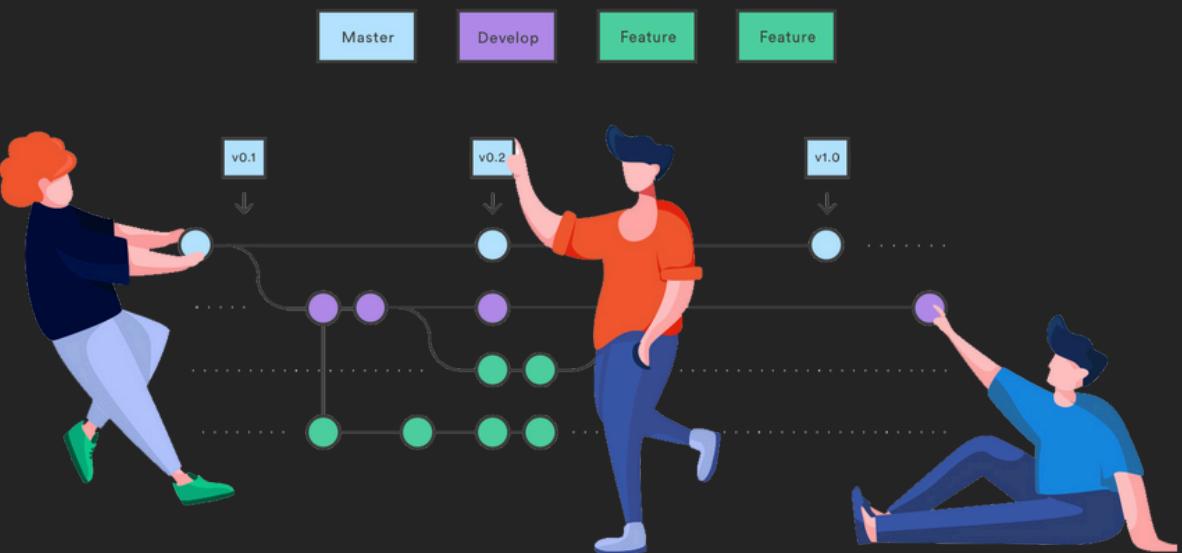


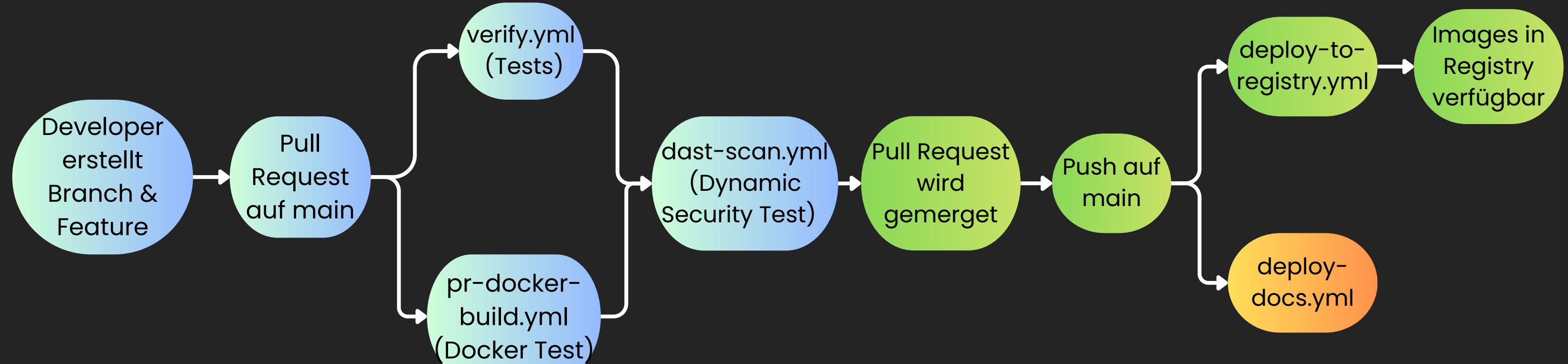
3- Dependency Updates

Workflow: dependabot.yml

Trigger: Wöchentlich (automatisch)

- NPM Dependencies
- Docker Dependencies
- GitHub Actions Dependencies





DATENSCHUTZ & IMPRESSUM

- Datenschutz: Informationen zur Datenverarbeitung
 - Impressum: Rechtliche Haftung und Kontakt
- Diese Seiten können auch ohne Anmeldung besucht werden.



OWASP TOP 10 2025

| | | | |
|--------------------------------|---|----------------------------------|---|
| A01: Broken Access Control | Auth Middleware, Public/Private Routes, Authorization Checks, User vs. API Key | A06: Vulnerable Components | npm audit, Dependabot, Package Overrides |
| A02: Cryptographic Failures | Argon2id, Secure Token Generation, JWT Signing, Refresh Token | A07: Authentication Failures | Strong Password Requirements, zxcvbn, Brute-Force Protection, Google OAuth (OIDC) |
| A03: Injection | Prisma ORM, Input Validation, HTML Sanitization | A08: Data Integrity Failures | TypeScript, Zod Validation, Database Constraints, Token Signature Verification |
| A04: Insecure Design | Defense-in-Depth(Auth, Input Validation, Output Encoding), Security-by-Default, Fail-Safe Defaults(enumration protection) | A09: Security Logging | Console Logging in Backend, Error Tracking(zod val. errors), Audit Trail (Session, IP, User-Agent), No Sensitive Data Logging |
| A05: Security Misconfiguration | Environment Variables, Gitleaks Scanning, CORS , Rate Limiting | A10: Server-Side Request Forgery | CSRF Protection, Origin Guard, SameSite Cookies |

SecureNotes

Einführung

User

Session

Notiz

Bonus

CI/CD

Ende

DANKE
FÜR AUFMERKSAMKEIT

Osman Ali Usal & Egesel Bozgeyik

