Architekturdokumentation Techradar

Inhalt

Einführung und Ziele

Anforderungen

Der Zweck der Applikation ist ein Technologie-Radar mit zwei Sichten:

- Administration (CTO/Tech-Lead): Technologien erfassen, ändern, publizieren, reklassifizieren
- Viewer (Mitarbeitende): Publizierte Technologien ansehen

Die wesentlichen Funktionen sind:

- Erfassen einer Technologie (Name, Kategorie, Tech-Beschreibung)
- Ändern von Basisfeldern
- Publizieren (erfordert Ring und Ringbeschreibung, setzt Timestamp des Publish)
- Reklassifizieren (neuer Ring und Ringbeschreibung, setzt Timestamp des Updates)
- Viewer-Anzeige: Nur publizierte, gruppiert nach Kategorie und Ring
- Anmeldung: Admin (CTO/Tech-Lead, mit Audit), Viewer (Mitarbeitende)

Qualitätsziele

- Der Technologie-Radar-Viewer soll neben der Desktop-Ansicht, auch für die Mobile/Tablet-Ansicht optimiert sein.
- Der Technologie-Radar-Viewer soll bei einer 4G-Verbindung innerhalb 1s geladen werden.
- Sämtliche Anmeldungen an die Technologie-Radar-Administration werden aufgezeichnet
- Die Funktionalitäten sollen mittels sinnvollen automatisierten Unit/Integration-Tests überprüft werden

Stakeholder

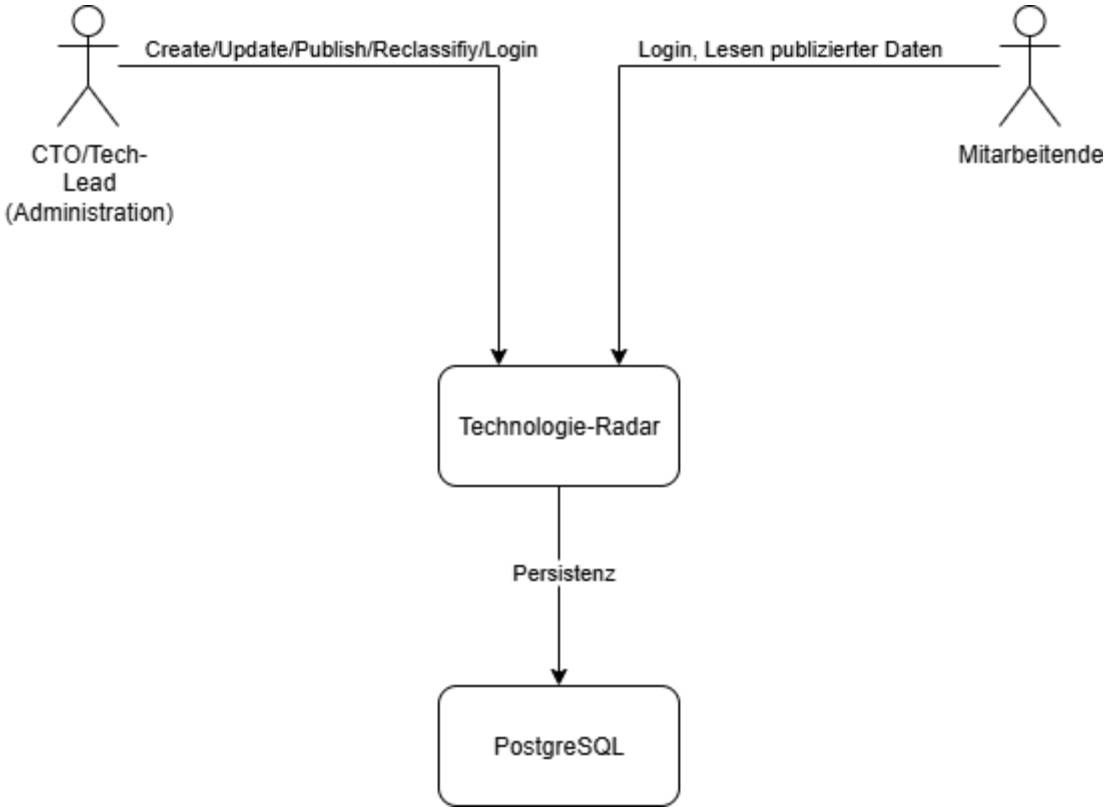
CTO/Tech-Lead	Verwaltung, Pflege, Klassifikation, Publikation
Mitarbeitende	Information bzw. Orientierung bei Technologieentscheidungen
Security/Compliance	Nachweisbarkeit von Admin-Zugriffen über Audits

Randbedingungen

Tech-Stack	Backend: NestJS, Prisma, PostgreSQL, JWT Frontend: Angular, Jest, Cypress
Schnittstellen	REST über HTTP Authentisierung via Bearer JWT im Authorization-Header
Ports	Frontend: http://localhost:4200 Backend: http://localhost:3000
Persistenz	PostgreSQL als produktives DBMS, Datenzugriff über Prisma
Security	JWT-Secret aus ENV, Passwörter gehasht Admin-Endpoints durch JWT + Rollen geschützt Viewer-API login-pflichtig

Kontextabgrenzung

Fachlicher Kontext



Technischer Kontext

Interaktionsebene

- Frontend kommuniziert per http/JSON mit Backend
- Backend greift über Prisma auf PostgreSQL zu
- Authentisierung über JWT Bearer im Authorization-Header

Schnittstellen

Auth	POST /api/auth/login -> { token, user } (Viewer) POST /api/auth/admin/login -> { token, user } (Admin mit Audit)
Admin	POST /api/technologies GET /api/technologies ?status={draft|published|all} GET /api/technologies/:id PATCH /api/technologies/:id PATCH /api/technologies/:id/publish PATCH /api/technologies/:id/reclassify
Viewer	GET /api/radar

Lösungsstrategie

Architektur- und Designprinzipien

Klarer Zustandsfluss	Draft -> Published -> Reclassified Drafts erlauben unvollständige Klassifikation Publish/Reclassify erzwingen Ring und Ringbeschreibung
API nach Use-Cases	PATCH /publish und PATCH /reclassify statt generischem Update
Sicherheit	Alle nichtöffentlichen Endpunkte hinter JWT + Rollenprüfung Auch Viewer-API nur nach Login
KISS	Keine überflüssigen Frameworks/Schichten Schlanke DTOs Services kapseln DB-Zugriff

Sicherheitsstrategie

Authentifizierung	E-Mail/Passwort -> JWT Passwörter gehasht
Autorisierung	JwtAuthGuard + RolesGuard auf Controller-Ebene
Audit	Admin-Logins werden in LoginAudit protokolliert (Zeit, Erfolg, IP, userId)
Transport/Headers	JWT nur im Authorization-Header
Validierung	ValidationPipe und class-validator auf allen DTOs

Persistenzstrategie

DB	PostgreSQL Zugriff via Prisma Migrations als Quelle für Schemaänderungen
Zeitstempel	createdAt, updatedAt, publishedAt

Teststrategie

Unit-Tests	Service-Logik
Integrationstests	Echte DB + signiertes Test-JWT Flow (Create -> Publish -> Reclassify)
E2E	Controllertests
Cypress	UI-Flows JWT via Cypress-Task signiert

UX-/Responsiveness-Strategie

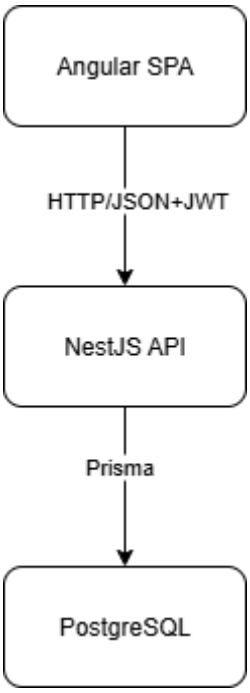
Admin	Klare Toolbars Badges (Draft/Published) Modal für Publish/Reclassify Auf Mobile Buttons umbrechen Tabellen als kompakte Karten
Viewer	Kategorien als Karten Ringe zeilenweise Technologien als Pills

Trade-offs / Alternativen

JWT vs. Session	JWT gewählt, da weniger aufwändig
Spezialisierte Endpunkte vs. Generisches Update	Spezialisierte Endpunkte sichern Domänenregeln und erleichtern Tests
ORM-Wahl (Prisma)	Typsicherheit & Migrationsqualität Alternativen bieten ähnliche Features

Bausteinsicht

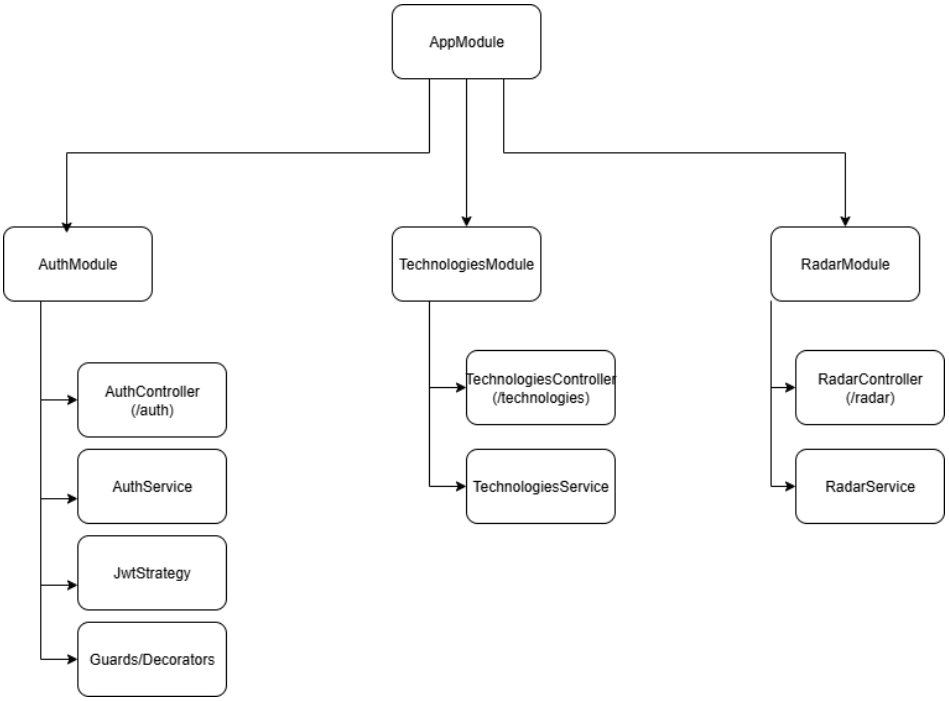
Whitebox Gesamtsystem (Level 1)



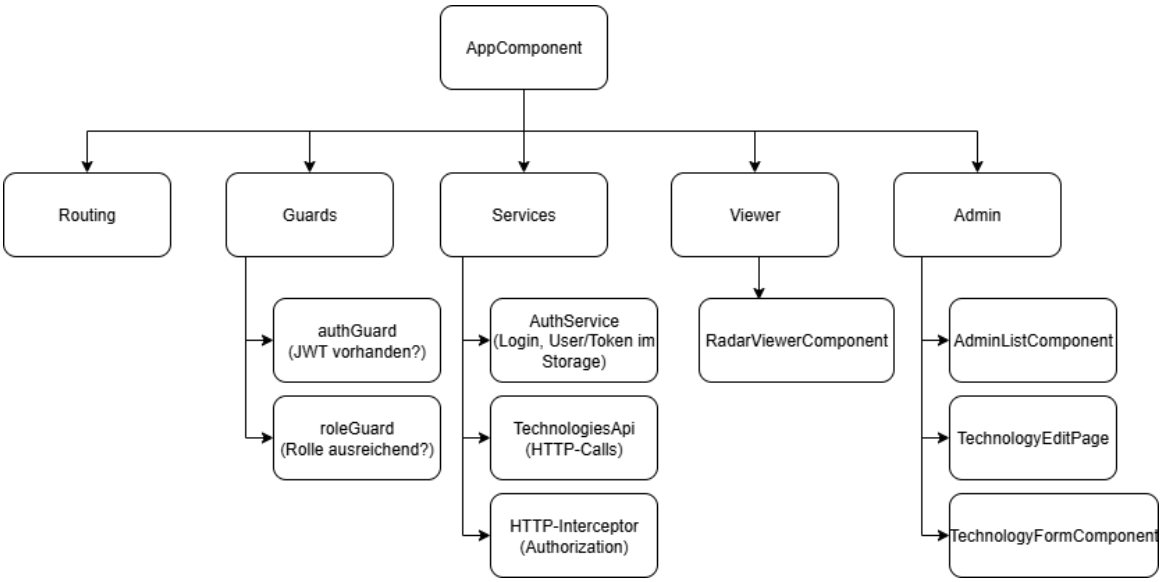
Zerlegung in Frontend (SPA) und Backend (API) mit gemeinsamer DB.

- Frontend: Darstellung/Interaktion, Routing, Auth-Token-Handling
- Backend: Auth, Logik, Validierung, Persistenz
- DB: Dauerhafte Speicherung von Technologie, User und LoginAudit

Backend-Module (Level 2)



Frontend-Struktur (Level 2)



Datenmodell

Technology	Required: id, name, category, techDescription Optional: ringDescription Timestamp : createdAt, updatedAt, publishedAt?
User	Id, email (unique), passwordHash, role
LoginAudit	Id, email, success, ip?, userId?, context, createdAt

Externe Sicht der API

Endpunkt	Methode	Auth/Rolle	Zweck	Request DTO
/api/auth/login	POST	-	Viewer-Login	LoginDto
/api/auth/admin/login	POST	-	Admin-Login	LoginDto
/api/technologies	POST	JWT + Rolle	Draft anlegen	CreateTechnologyDto
/api/technologies?status	GET	JWT + Rolle	Liste filtern	-
/api/technologies/:id	GET	JWT + Rolle	Einzelnes lesen	-
/api/technologies/:id	GET	JWT + Rolle	Felder ändern	UpdateTechnologyDto
/api/technologies/:id/publish	PATCH	JWT + Rolle	Publizieren	PublishTechnologyDto
/api/technologies/:id/reclassify	PATCH	JWT + Rolle	Reklassifizieren	PublishTechnologyDto
/api/radar	GET	JWT	Publizierte laden	-

Backend-Bausteine (Level 3)

TechnologiesService	
create(dto)	Erzeugt Draft
update(id, dto)	Ändert Felder, ändert nicht den Publish-Status
publish(id, ring, desc)	Validierung, setzt ring, publishedAt
reclassify(id, ring, desc)	Validierung, aktualisiert ring, updatedAt
AuthService	
login(email, pwd)	Hash-Check, Signatur JWT
loginAdmin(email, pwd)	Wie oben + Audit schreiben
Guards/Strategy	
JwtStrategy	Verifiziert Token und stellt user bereit
JwtAuthGuard	Verlangt gültiges Token
RolesGuard	Prüft Rolle aus Requests gegen geforderte Rolle

Frontend-Bausteine (Level 3)

TechnologiesApi	Wrapper um HttpClient, Pfade/DTOs typisiert Nutzt globalen Auth-Interceptor
AuthService	Login, Speichern von {token, user} in localStorage Abmeldung, einfache Getter (isLoggedIn, role)
authGuard/roleGuard	Prüfen Storage-Zustand, Navigation auf Login
UI Komponenten	AdminListComponent: Status-Badges, Filter, Aktionen, Modals TechnologyFormComponent: Forms, Validierung RadarViewerComponent: Gruppierung in Karten, Ringe als Zeilen, Techs als Pills

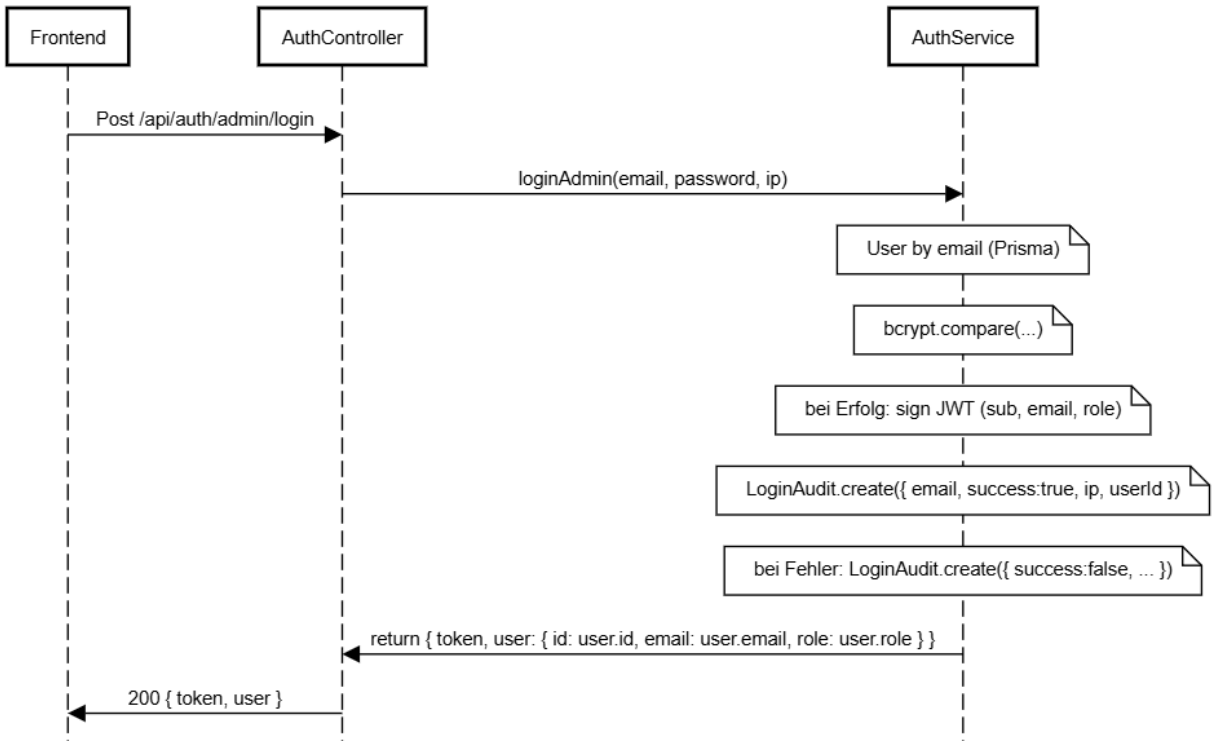
Wichtige Entscheidungen in der Bausteinstruktur

- Spezifische Endpunkte (publish, reclassify) statt offenem Partial-Update
- Viewer-API gekapselt (/radar) -> minimierter Payload, keine Admin-Felder
- Guards pro Controller statt global -> feinere Kontrolle, klarer Scope
- Shared Styles -> konsistente UX in Admin & Viewer

Laufzeitsicht

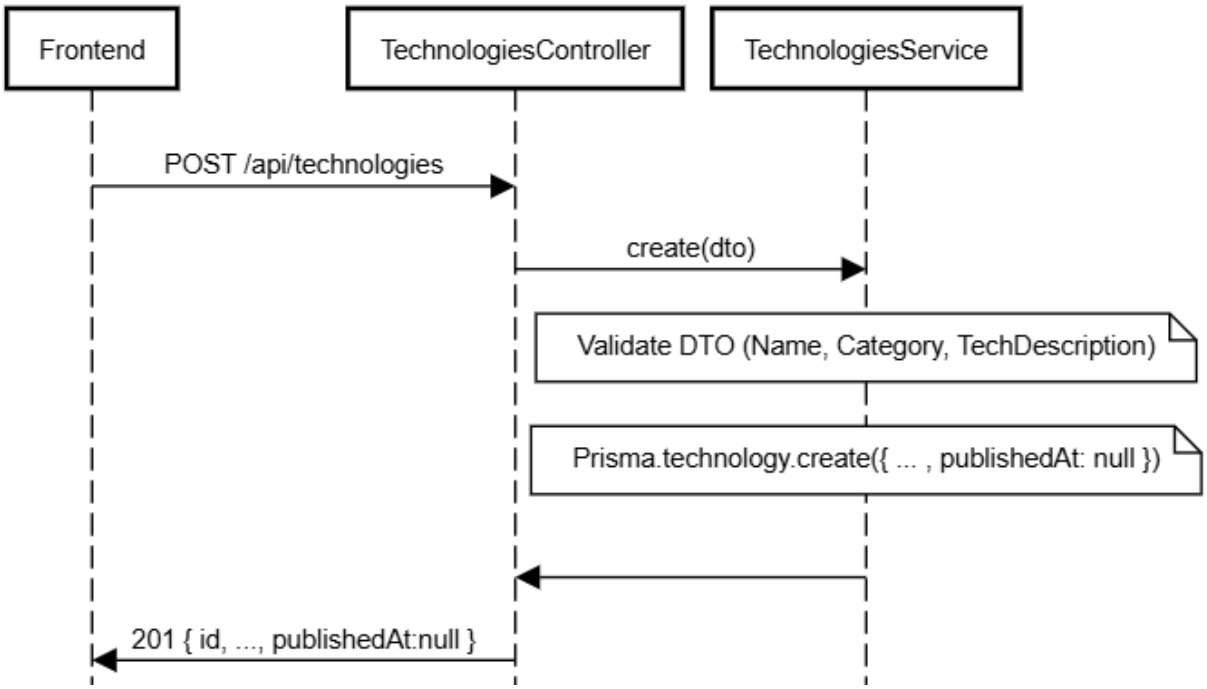
Anmeldung (Admin) inkl. Audit

Ziel: CTO/Tech-Lead erhält JWT, Admin-Login wird protokolliert



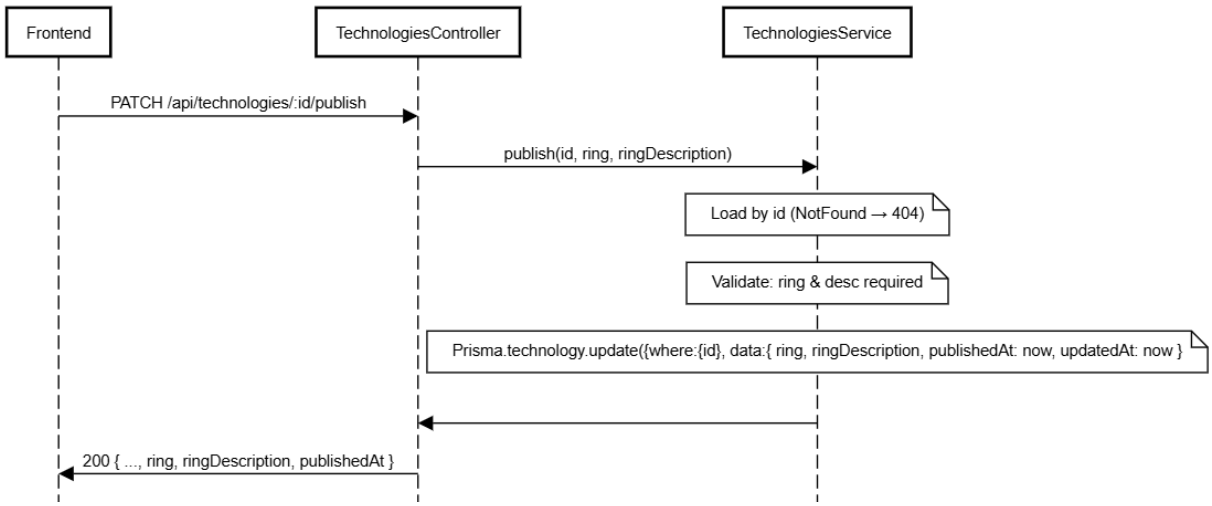
Technologie anlegen (Draft)

Ziel: Neue Technology im Draft-Status



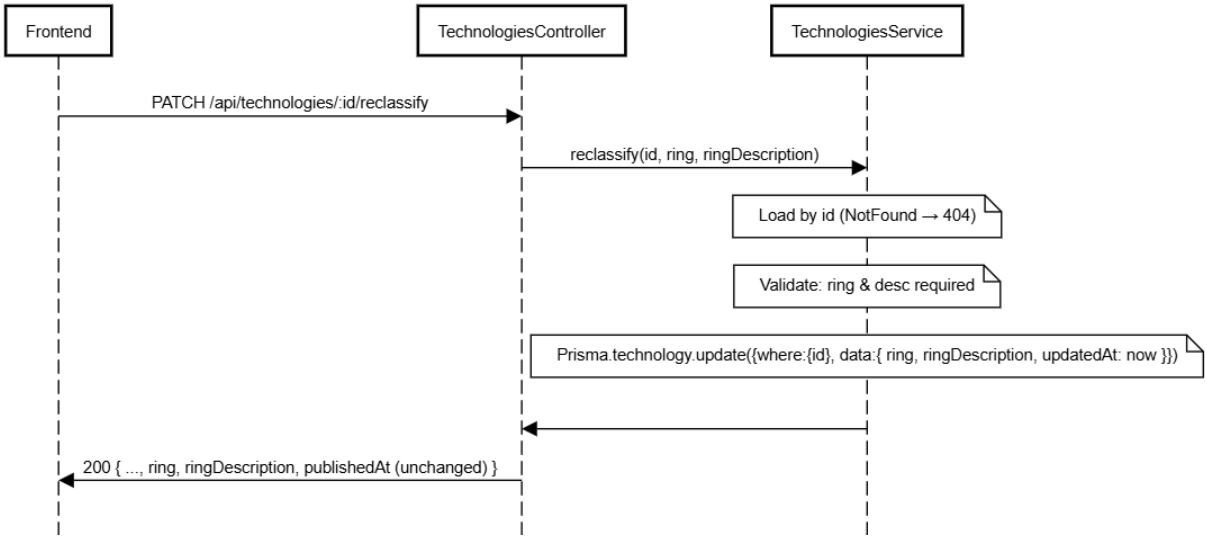
Publish

Ziel: Draft wird veröffentlicht, Ring + Begründung zwingend



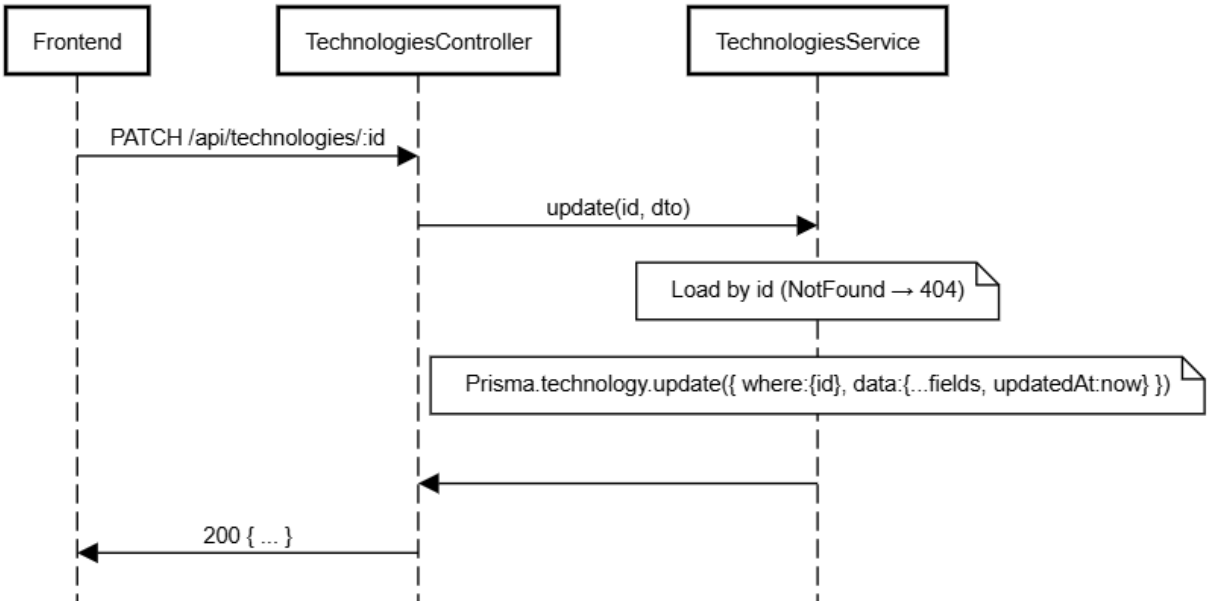
Reclassify

Ziel: Ring/Begründung aktualisieren, publishedAt bleibt erhalten



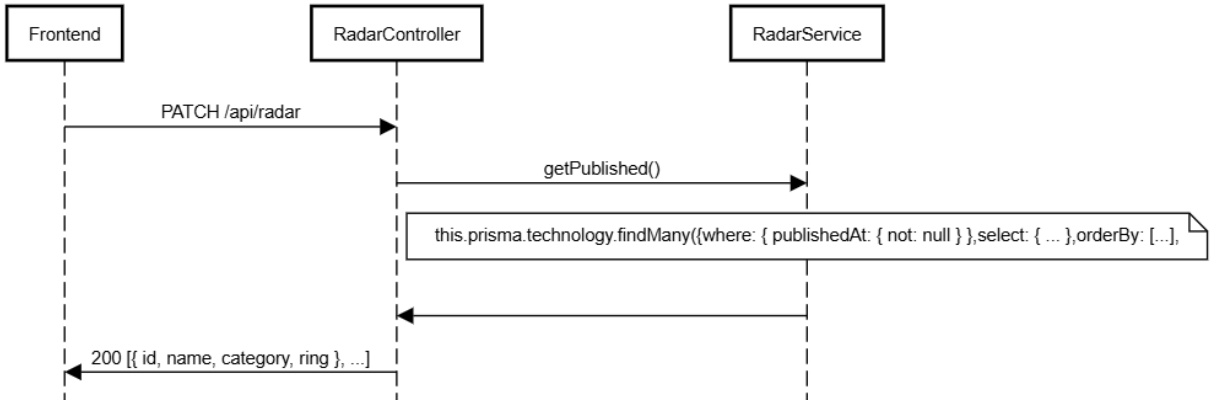
Basisfelder ändern

Ziel: Name/Kategorie/Tech-Beschreibung anpassen, Status unverändert



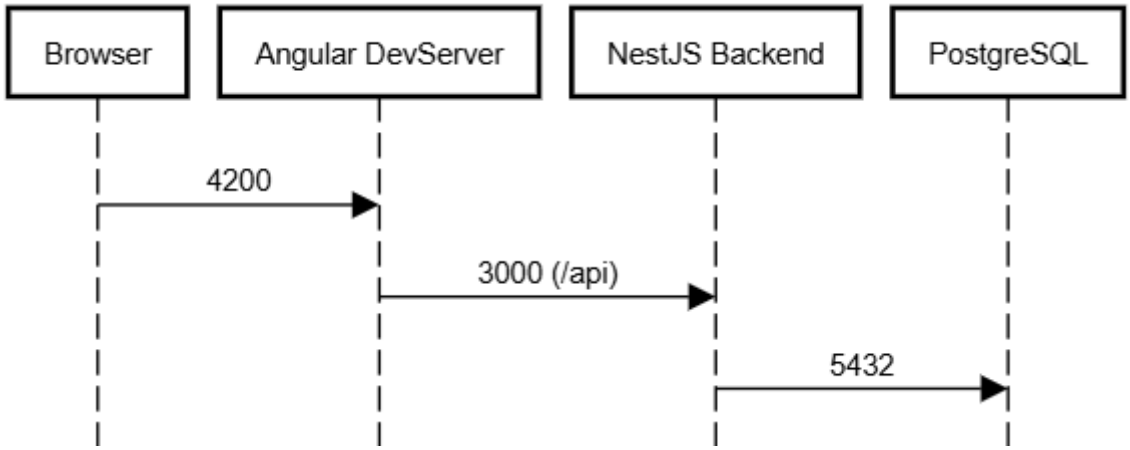
Viewer: publizierte Technologien abrufen

Ziel: Minimaler Datensatz für Anzeige



Verteilungssicht

Infrastrukturübersicht (Topologie)



Entwicklungsumgebung

Frontend	Angular DevServer auf http://localhost:4200
Backend	NestJS auf http://localhost:3000
Datenbank	Lokaler Postgres Migration via npx prisma migrate dev Seeds via npx prisma db seed

Querschnittliche Konzepte

Domänenregeln

States einer Technology	Draft (published=null) Published (erstmalig publishedAt gesetzt) Beliebige Reklassifikationen (Ring/Begründung ändern, publishedAt bleibt)
Pflichtfelder je Use-Case	Create/Update: name, category, techDescription Publish/Reclassify: ring, ringDescription
Zeitstempel	createdAt: DB-Default updatedAt: bei jeder Änderung publishedAt: nur beim ersten Publish

Sicherheit

JWT Bearer	Signatur über JWT_SECRET (ENV)
Guards	JwtAuthGuard (Token-Pflicht) RolesGuard (Rollen : CTO, TECH_LEAD, EMPLOYEE)
Admin-Login-Audit	Tabelle LoginAudit

Validierung & Fehlerbehandlung

- Global ValidationPipe: whitelist=true, forbidNonWhitelisted=true
- DTOs per class-validator (z.B. IsEmail, IsEnum, IsNotEmpty)
- Error-Mapping:
    - 400 -> DTO/Validierung verletzt
    - 401 -> ungültiges/fehlendes Token
    - 403 -> Rolle unzureichend
    - 404 -> Entität nicht gefunden
    - 5xx -> unerwarteter Serverfehler

Persistenz & Datenzugriff

Prisma	Alleiniger DB-Zugriff Schema via Migrations versioniert
Transaktionen	Einzelne Updates genügen (Publish/Reclassify atomar pro Request)
Selektive Felder	/api/radar liefert keine Admin-Felder
Indizes	Primärschlüssel
Migrations/Seeds	Strikt über CLI, keine manuelle Schemaänderung

Testbarkeit & Testkonzept

Unit	Services
Integration	Reale DB + signiertes Test-JWT
E2E	Controller
Cypress	UI-Flows, JWT in Node-Task signiert

Architekturentscheidungen (ADRs)

ADR-01: Authentifizierung via JWT statt Session

Kontext	SPA (Angular) + API (NestJS)
Entscheidung	Bearer JWT im Authorization-Header Claims: sub, email, role
Begründung	Stateless, einfache Horizontal-Skalierung
Konsequenzen	Token-Invalidierung nur über Expire Secret muss sicher verteilt werden
Status	Akzeptiert

ADR-02: Explizite Use-Case-Endpunkte statt generischem PATCH

Kontext	Strikte Domänenregeln
Entscheidung	Zwei spezialisierte Endpunkte mit eigener Validierung
Begründung	Bessere Testbarkeit & Nachvollziehbarkeit
Konsequenzen	Etwas mehr API-Oberfläche
Status	Akzeptiert

ADR-03: Prisma + PostgreSQL als Persistenz

Kontext	Typsichere Entwicklung, Migrationsdisziplin
Entscheidung	Prisma ORM, Postgres als DB
Begründung	Robuste Migrations
Konsequenzen	Lernkurve für Schema/Migrate
Status	Akzeptiert

ADR-04: ValidationPipe (whitelist + forbidNonWhitelisted)

Kontext	Schutz vor «silent acceptance» unbekannter Felder
Entscheidung	Global aktiv, DTOs strikt
Begründung	Sicherheit % Korrektheit Klare 400er bei Fehlbedienung
Konsequenzen	Strengere Tests
Status	Akzeptiert

ADR-05: Rollenprüfung via Guards auf Controller-Ebene

Kontext	Unterschiedliche Schutzbedarfe (Admin vs. Viewer)
Entscheidung	JwtAuthGuard überall, RolesGuard nur bei Admin
Begründung	Transparente, deklarative Policy nahe des Interfaces
Konsequenzen	Mehr Deklaration pro Controller
Status	Akzeptiert

ADR-06: Cypress-Auth über JWT-Task statt realem Login-Endpoint

Kontext	Flaky E2E durch Login-Routen/DB-Zustand vermeiden
Entscheidung	Token im Cypress Node-Task signieren und ablegen
Begründung	Stabil, unabhängig von Auth-Endpoint
Konsequenzen	Test-Secret muss Backend-Secret entsprechen Kein echter Login-Flow in UI-Tests
Status	Akzeptiert

Qualitätsanforderungen

Qualitätsbaum

Sicherheit	Authentifizierung, Autorisierung, Audit
Korrektheit & Integrität	Zustandsregeln, Validierung
Benutzbarkeit	Verständliche UI, Responsiveness
Zuverlässigkeit	Atomare Updates, robuste Fehlercodes
Leistung	Geringe Latenz
Wartbarkeit	Modulgrenzen, DTOs, Migrations
Testbarkeit	Unit/Integration/E2E/Cypress
Beobachtbarkeit	Audits

Qualitätsszenarien

Rollenschutz Admin-APIs	Authentisierter Nutzer ohne Rolle CTO/TECH_LEAD ruft POST /api/technologies auf. Request wird abgelehnt
Viewer Geschützt	Unangemeldeter Nutzer ruft GET /api/radar auf Request wird abgelehnt
Admin-Login-Audit	CTO-Login Audit-Eintrag mit Zeit, E-Mail, Erfolg, IP
Responsiveness	Viewer/Admin auf Smartphone-Viewport. Layout wechselt entsprechend

Risiken und technische Schulden

Risiken

Risiko	**Folgen**	**Gegenmassnahme**
Fehlkonfiguration JWT-Secret	Unsicheres Secret in allen Umgebungen Token-Fälschung möglich	Secret-Management
Fehlender Rollen-Schutz	Vergessene Guards/Decorator	Security-Tests
Migrationsfehler	Downtime, Datenverlust	Migrations in Staging testen
CORS/Proxy-Fehler	Fehlende Proxy-Konfiguration Gebrochene Logins/Requests	Infrastruktur-Smoke-Tests
Wachsendes Datenvolumen	Get request ohne Paging	Frühzeitig mit Paging/filter ergänzen

Technische Schulden

Schulden	**Folge**	**Abhilfe**
Keine API-Versionierung	Erschwerte Breaking-Changes	Einführung /api/v1
Viewer-Endpoint ohne Cache	Potenziell höhere Last bei Wachstum	Cache evaluieren
Keine Business-Audits	Geringere Nachvollziehbarkeit	Audit-Events/Tabelle ergänzen

Glossar

Admin	Benutzer mit Rolle CTO oder TECH_LEAD
API	http-Schnittstelle des Backends unter /api
Audit / LoginAudit	Protokollierung von Admin-Anmeldung
Auth	Authentifizierung: Nachweis der Identität Autorisierung: Rechteprüfung
Category / Kategorie	Zuordnung einer Technologie: Techniques, Platforms, Tools, LanguagesFrameworks
CORS	Cross-Origin Ressource Sharing
CTO	Chief Technology Officer
Cypress	End-to-End-Testframework für das Frontend
DTO	Data Transfer Object
Draft	Nicht veröffentlichte Technologie
EMPLOYEE	Standardrolle für Mitarbeitende
E2E-Tests	End-to-End-Tests
ENV / .env	Umgebungsvariablen
Guard	NestJS-Mechanismus zur Zugriffskontrolle
JWT	JSON Web Token
Jest	Test-Runner/Assertion-Bibliothek
Migration	Versionskontrollierte DB-Schemaänderungen
NestJS	Backend-Framework
Prisma	Type-safe ORM
Publish / Publizieren	Übergang von Draft zu Published
Radar (Viewer)	Read-Only-Ansicht für Mitarbeitende
Reclassify / Reklassifizieren	Änderung von ring + ringDescription
Ring	Maturität/Empfehlung: Assess, Trial, Adopt, Hold
Rolle	CTO, TECH_LEAD, EMPLOYEE
Seed	Initialbefüllung der DB
Technology	Kerndomänenobjekt
Timestamps	createdAt: Erstellzeitpunkt updatedAt: Änderungszeitpunkt publishedAt: Zeitpunkt der erstmaligen Veröffentlichung
ValidationPipe	Globale Eingabevalidierung für DTOs