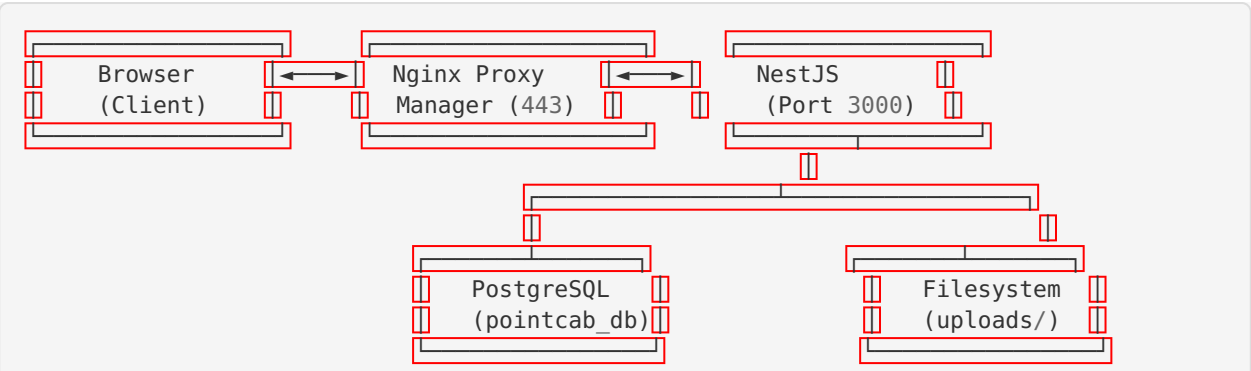
System-Architektur - PointCab Webexport Server

Diese Dokumentation beschreibt die technische Architektur des Systems.

🏗️ Übersicht



📂 Verzeichnisstruktur

```
/var/www/pointcab_webexport_server/
├── nodejs_space/                 # Haupt-Anwendung
│   ├── src/
│   │   ├── controllers/          # HTTP-Endpunkte
│   │   │   ├── admin.controller.ts
│   │   │   ├── projects.controller.ts
│   │   │   └── root.controller.ts
│   │   └── services/             # Business-Logik
│   │       ├── admin.service.ts
│   │       ├── projects.service.ts
│   │       ├── upload.service.ts
│   │       └── prisma.service.ts
│   ├── prisma/
│   │   └── schema.prisma         # Datenbank-Schema
│   ├── dist/                     # Kompilierter Code
│   ├── uploads/                  # Hochgeladene Projekte
│   ├── package.json
│   ├── tsconfig.json
│   └── .env                      # Konfiguration
└── backups/                      # Deployment-Backups
```

🛠️ Komponenten

1. Controllers

ProjectsController (`projects.controller.ts`)

Verantwortlich für:
- Projekt-Anzeige (`GET /:shareId/view`)

- Passwort-Authentifizierung (`POST /:shareId/auth`)
- Asset-Serving (`GET /:shareId/*`)

Wichtige Funktionen:

```
// Projekt anzeigen
@Get(':shareId/view')
async viewProject()

// Assets laden (JS, CSS, Bilder)
@Get(':shareId/*')
async getProjectResource()

// Passwort-Seite
@Get(':shareId')
async showPasswordPage()
```

AdminController (`admin.controller.ts`)

Verantwortlich für:
- Dashboard (`GET /admin/dashboard`)
- Projekt-Verwaltung (CRUD)
- RAR-Entpacken
- Datei-Upload

2. Services

ProjectsService (`projects.service.ts`)

Kernfunktionen:

1. **Web-Subfolder-Erkennung:**
   `typescript`
   `detectWebSubfolder(projectPath: string): string | null`
   `// Erkennt z.B. "Web_0_web/" Ordner`

2. **Asset-Pfad-Auflösung:**
   `typescript`
   `resolveAssetPath(projectPath: string, assetPath: string): string`
   `// Löst relative Pfade auf`

3. **Base-Tag-Injection:**
   `typescript`
   `injectBaseTag(html: string, shareId: string, htmlPath?: string): string`
   `// Fügt <base href> für korrekte Asset-Pfade ein`

AdminService (`admin.service.ts`)

Kernfunktionen:

1. **RAR-Entpacken:**
   `typescript`
   `extractRar(projectId: string, rarPath: string): Promise<void>`
   `// Verwendet spawn() für große Dateien`

2. **HTML-Erkennung:**
   `typescript`

```
  findHtmlFiles(projectPath: string): string[]
  // Findet alle HTML-Dateien im Projekt
```

3. **Multi-HTML-Logik:**
   `typescript`
   ```
   processExtractedProject(projectId: string): Promise<void>
   // Setzt htmlfilename = null bei mehreren HTMLs
   ```

UploadService (`upload.service.ts`)

Kernfunktionen:
- ZIP/RAR-Upload verarbeiten
- Projekt in Datenbank erstellen
- Multi-HTML-Erkennung

3. Datenbank-Schema

```
model project {
  id           String    @id @default(uuid())
  name         String    @unique
  shareid      String    @unique
  password     String    // Klartext (kein Hash!)
  htmlfilename String?   // NULL bei Multi-HTML
  uploaddate   DateTime  @default(now())
  expirydate   DateTime?
  createdat    DateTime  @default(now())
}
```

Wichtig: `htmlfilename` ist **nullable** für Multi-HTML-Unterstützung!

🔄 Request-Flow

Projekt anzeigen

```
1. Browser: GET /abc123/view
   ↓
2. Nginx Proxy: Weiterleitung an :3000
   ↓
3. ProjectsController.viewProject()
   │
   ├─ Projekt aus DB laden
   ├─ Passwort-Check (Cookie)
   ├─ htmlfilename prüfen
   │   ├─ null → HTML-Auswahl-Seite
   │   └─ vorhanden → HTML laden
   ├─ Web-Subfolder erkennen
   ├─ Base-Tag injecten
   └─ HTML zurückgeben
   ↓
4. Browser: Rendert HTML
   ↓
5. Browser: Lädt Assets (GET /abc123/js/main.js)
   ↓
6. ProjectsController.getProjectResource()
   ├─ Pfad auflösen (mit Subfolder)
   └─ Datei zurückgeben
```

RAR entpacken

```
1. Admin: POST /admin/projects/:id/extract-rar
   ↓
2. AdminController.extractRar()
   ↓
3. AdminService.extractRar()
   ├─ Platzhalter-HTML löschen
   ├─ RAR entpacken (spawn)
   ├─ HTML-Dateien finden
   ├─ Web-Subfolder erkennen
   ├─ htmlfilename setzen
   │    ├─ 1 HTML → Dateiname
   │    └─ >1 HTML → null
   └─ DB aktualisieren
```

🔐 Sicherheit

Passwort-Handling

- Passwörter werden als **Klartext** gespeichert
- Kein bcrypt-Hashing (bewusste Entscheidung für einfache Verwaltung)
- Cookie-basierte Session nach erfolgreicher Authentifizierung

Pfad-Sicherheit

```
// Pfad-Normalisierung verhindert Directory Traversal
const safePath = path.normalize(requestedPath).replace(/^(\.\.\/)+/, '');
```

Datei-Zugriff

- Nur Dateien innerhalb des Projekt-Verzeichnisses
- Keine direkten Pfade vom Client
- MIME-Type-Validierung

📊 Technologie-Stack

Schicht	Technologie	Version
Runtime	Node.js	18.x LTS
Framework	NestJS	10.x
Sprache	TypeScript	5.x
Datenbank	PostgreSQL	16.x
ORM	Prisma	5.x
Process Manager	PM2	5.x
Reverse Proxy	Nginx Proxy Manager	Latest
OS	Ubuntu	24.04 LTS

🔧 Konfiguration

Umgebungsvariablen (.env)

```
PORT=3000                    # Server-Port
NODE_ENV=production          # Umgebung
DATABASE_URL="postgresql://..."  # DB-Verbindung
UPLOAD_DIR=/path/to/uploads  # Upload-Verzeichnis
SESSION_SECRET=...           # Session-Verschlüsselung
ADMIN_PASSWORD=...           # Admin-Zugang
```

PM2 Konfiguration

```
// ecosystem.config.js
module.exports = {
  apps: [{
    name: 'pointcab-server',
    script: './dist/main.js',
    instances: 1,
    autorestart: true,
    max_memory_restart: '1G'
  }]
};
```

📈 Performance

Optimierungen

1. **Spawn statt Exec:** Für RAR-Entpacken (kein Buffer-Limit)
2. **Lazy Loading:** Assets werden on-demand geladen
3. **PM2 Clustering:** Möglich für Skalierung

Limits

- Max Upload: 500 MB (konfigurierbar)
- Max Projekte: Unbegrenzt (Speicherplatz-abhängig)
- Gleichzeitige Verbindungen: Node.js Standard

Siehe auch: CHANGELOG.md (CHANGELOG.md)