Zeiterfassungssystem mit Swiss Compliance - Vollständige Dokumentation

Projekt: cflux - Time Tracking & Workflow Management System
Version: 1.0
Datum: Dezember 2025
Technologie: TypeScript, Node.js, React, PostgreSQL

Inhaltsverzeichnis

Projektübersicht

Das cflux Zeiterfassungssystem ist eine vollständige Lösung für Zeiterfassung, Rechnungsstellung und Workflow-Management mit spezieller Unterstützung für Schweizer Arbeitsrecht-Compliance (ArG/ArGV 1).

Hauptmerkmale

- **Zeiterfassung** mit Ein-/Ausstempeln auf Projekte und Standorte
- **Rechnungsverwaltung** mit PDF-Generierung und Vorlagen-System
- **Visueller Workflow-Editor** für Genehmigungs- und Benachrichtigungsprozesse
- **Swiss Compliance** mit automatischer Verletzungserkennung
- **Urlaubsverwaltung** mit Jahresplaner
- **Multi-User** mit Rollen (Admin/User)
- **Dark Mode** mit persistenter Speicherung
- **Mobile-responsive** Design

Zielgruppe

- Kleine bis mittelgroße Unternehmen in der Schweiz
- Dienstleister mit Projektabrechnung
- Unternehmen mit Compliance-Anforderungen nach ArG/ArGV 1

- Teams mit komplexen Genehmigungs-Workflows

Technologie-Stack

Backend

Technologie	Version	Verwendung
Node.js	20	Runtime Environment
Express	4.18	Web Framework
TypeScript	5.3	Typsicherheit
PostgreSQL	16	Relationale Datenbank
Prisma	5.8	ORM & Migrations
JWT	-	Authentication
bcrypt	-	Passwort-Hashing
nodemailer	6.9	E-Mail-Versand
PDFKit	-	PDF-Generierung

Frontend

Technologie	Version	Verwendung
React	18	UI Framework
TypeScript	4.9	Typsicherheit
React Router	6	Navigation
Axios	-	HTTP Client
Recharts	-	Diagramme & Visualisierung
ReactFlow	-	Workflow-Editor

DevOps & Infrastruktur

Technologie	Verwendung
Docker	Containerisierung
Docker Compose	Multi-Container Orchestrierung
Nginx	Reverse Proxy & Static Files
PostgreSQL	Datenpersistenz

Kernfunktionen

1. Zeiterfassung

Für Benutzer:

- Ein-/Ausstempeln mit Live-Uhr in Titelleiste
- Buchung auf Projekte und Standorte
- Übersicht eigener Arbeitszeiten
- Mobile-optimierte Bedienung

Für Administratoren:

- Zeitkorrekturen durchführen
- Reports für alle Mitarbeiter
- Überstunden-Tracking
- Export-Funktionen

2. Rechnungsverwaltung

Rechnungserstellung:

- Entwürfe erstellen und bearbeiten
- Positionen mit Beschreibung, Menge, Preis, MwSt.
- Automatische Berechnung von Summen und Steuern
- Status-Management (Entwurf → Versendet → Bezahlt)

Vorlagen-System:

- Wiederverwendbare Rechnungsvorlagen
- Vordefinierte Positionen
- Workflow-Zuordnung zu Vorlagen
- PDF-Design-Integration

PDF-Generierung:

- Professionelle Rechnungs-PDFs
- Firmen-Logo und -Daten
- Mehrere MwSt.-Sätze
- Swiss QR-Code für Zahlungen (geplant)

3. Workflow-System

Visueller Editor:

- Node-basierter Flow-Editor mit ReactFlow
- Drag & Drop Interface
- Echtzeit-Vorschau

Workflow-Typen:

- **START** - Workflow-Einstiegspunkt
- **APPROVAL** - Genehmigungsschritte mit Benutzerzuweisung
- **EMAIL** - Automatischer E-Mail-Versand

- **VALUE_CONDITION** - Bedingungen basierend auf Rechnungswerten
- **DATE_CONDITION** - Zeitbasierte Bedingungen (geplant)
- **END** - Workflow-Abschluss

Features:

- Edge-basierte Ausführung (folgt visuellen Verbindungen)
- Bedingte Verzweigungen (true/false Outputs)
- Platzhalter in E-Mails ({invoiceNumber}, {totalAmount}, etc.)
- Genehmigungs-Queue für Benutzer
- Automatische Workflow-Auslösung bei Rechnungsstatus-Änderung

4. Swiss Compliance (ArG/ArGV 1)

Automatische Prüfungen:

- ⚠ Ruhezeit (min. 11h zwischen Arbeitstagen) - Art. 15a ArG
- ⚠ Tägliche Höchstarbeitszeit (max. 12.5h) - Art. 9 ArG
- ⚠ Wöchentliche Höchstarbeitszeit (45h/50h) - Art. 9 ArG
- ⚠ Pausenregelung (15min/30min/60min) - Art. 15 ArGV 1
- ⚠ Überstundenlimits und Mehrarbeit

Compliance Dashboard:

- Echtzeit-Statistiken zu Violations
- Kritische vs. Warnungs-Violations
- Top-Benutzer mit Verstößen
- Detaillierte Violations-Tabelle mit Filterung
- Violations auflösen mit Notizen
- Kantonsbasierte Feiertage (alle 26 Kantone)

Überstunden-Management:

- Reguläre Überstunden (vertragliche vs. gesetzliche Stunden)
- Mehrarbeit (über gesetzliche Höchstarbeitszeit)
- Jahres-Saldo pro Benutzer
- Automatische Berechnung

5. Urlaubsverwaltung

Abwesenheitsanträge:

- Urlaub, Krankheit, persönliche Gründe, unbezahlter Urlaub
- Start- und Enddatum mit Begründung
- Genehmigung/Ablehnung durch Admin
- E-Mail-Benachrichtigungen

Urlaubsplaner:

- Jahresübersicht aller Mitarbeiter
- Farbcodierung nach Typ

- Filter nach Benutzer und Typ
- Kalender-Grid-Ansicht

6. Backup & Restore

- Automatische PostgreSQL Backups
- Manuelle Backup-Erstellung über Admin Panel
- Download, Upload und Restore von Backups
- JSON-Export für Datenanalyse
- Backup-Verwaltung (Auflisten, Löschen)

Installation & Setup

Schnellstart mit Docker (Empfohlen)

Voraussetzungen:

- Docker Desktop (Windows/Mac) oder Docker Engine (Linux)
- Docker Compose

3 einfache Schritte:

```
# 1. Repository klonen
git clone <repository-url>
cd cflux

# 2. Starten
docker-compose up --build -d

# 3. Warten bis bereit (ca. 30-60 Sekunden)
docker-compose logs -f backend
# Warten auf: "Server running on port 3001"
```

System-URLs:

- Frontend: http://localhost:3002
- Backend API: http://localhost:3001
- Health Check: http://localhost:3001/health

Erster Login:

- Email: `admin@timetracking.local`
- Passwort: `admin123`
- Beim ersten Login: Passwortänderung erforderlich

Manuelle Installation

Backend Setup:

```
cd backend
npm install
cp .env.example .env
# .env konfigurieren mit DATABASE_URL und JWT_SECRET
npm run prisma:migrate
npm run prisma:generate
npm run dev
```

Frontend Setup:

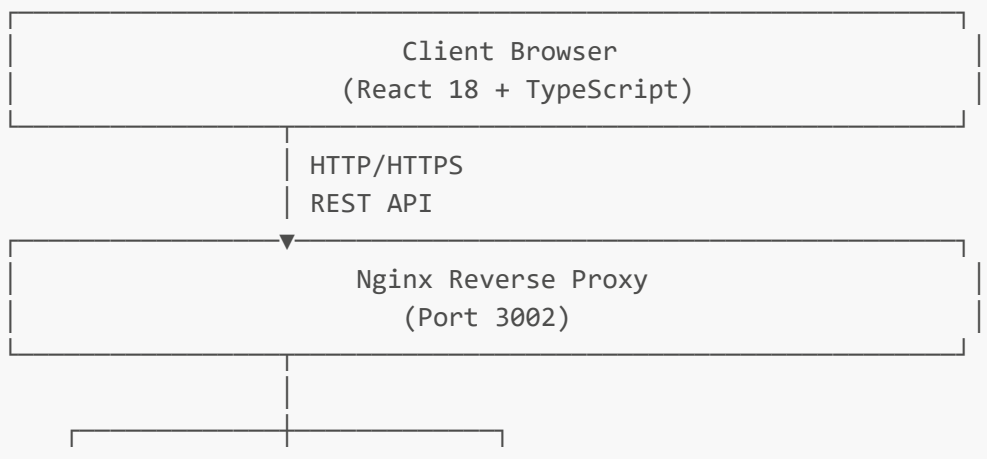
```
cd frontend
npm install
npm start
```

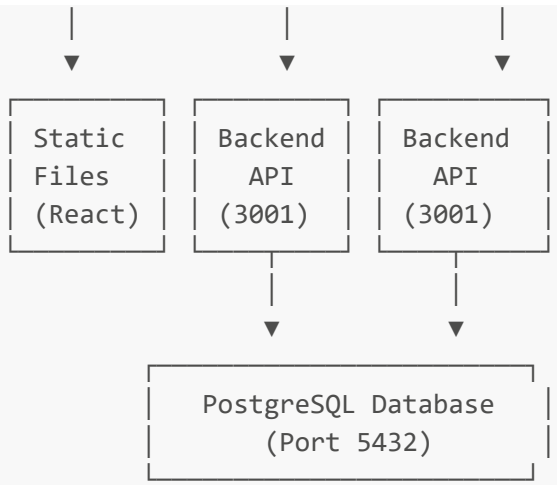
Produktions-Deployment

```
# Mit Docker
docker-compose up -d --build

# Backend manuell
cd backend
npm run build
npm start

# Frontend manuell
cd frontend
npm run build
# Build-Ordner mit Nginx oder anderem Server hosten
```

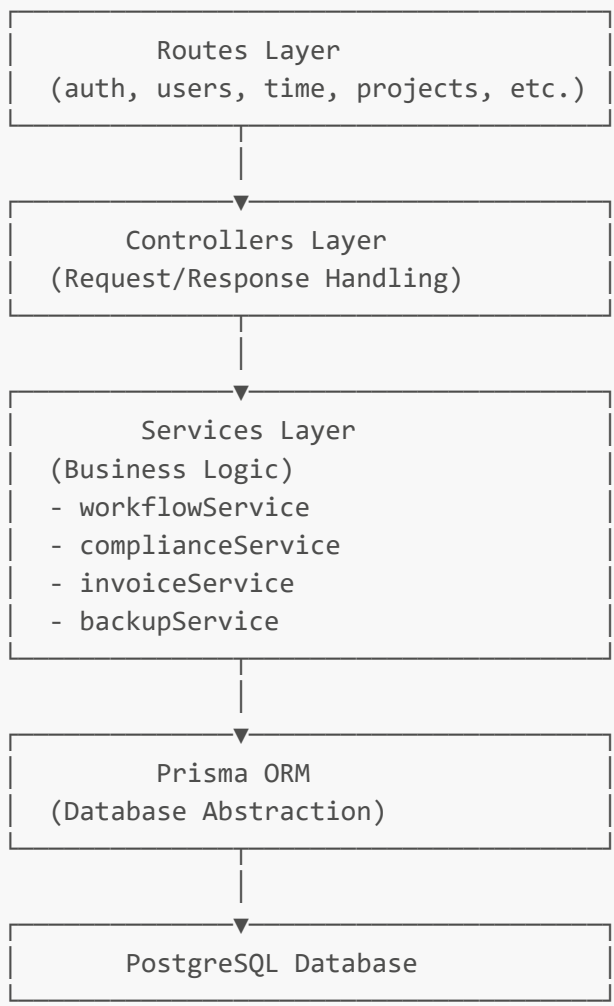
Architektur

Systemarchitektur





Backend-Architektur (Layered)



Frontend-Architektur (Component-Based)

```
src/
├── components/           # Wiederverwendbare UI-Komponenten
│   ├── admin/           # Admin-spezifische Komponenten
```

```
│   │   ├── NodeBasedWorkflowEditor.tsx
│   │   ├── InvoiceTemplateEditor.tsx
│   │   └── ComplianceDashboard.tsx
│   ├── ThemeToggle.tsx
│   └── ...
├── pages/              # Routen-Komponenten
│   ├── Dashboard.tsx
│   ├── AdminPanel.tsx
│   ├── InvoicePage.tsx
│   └── ...
├── contexts/           # React Context Provider
│   ├── AuthContext.tsx
│   └── ThemeContext.tsx
├── services/           # API-Kommunikation
│   ├── api.ts
│   ├── authService.ts
│   ├── workflowService.ts
│   └── ...
└── types/              # TypeScript Typdefinitionen
```

Datenbankmodell

Haupttabellen

users

```
- id: UUID (PK)
- email: String (unique)
- password: String (hashed)
- firstName: String
- lastName: String
- role: Enum (USER, ADMIN)
- isActive: Boolean
- vacationDays: Integer
-- Erweiterte Felder:
- dateOfBirth, placeOfBirth, nationality
- phone, mobile
- street, streetNumber, zipCode, city, country
- employeeNumber, entryDate, exitDate
- iban, bankName
- civilStatus, religion, ahvNumber
- isCrossBorderCommuter: Boolean
-- Swiss Compliance:
- weeklyHours: Integer (45 oder 50)
- canton: String (ZH, BE, etc.)
- exemptFromTracking: Boolean
- contractHours: Decimal
```

time_entries

```
- id: UUID (PK)
- userId: UUID (FK → users)
- projectId: UUID (FK → projects)
- locationId: UUID (FK → locations)
- clockIn: DateTime
- clockOut: DateTime (nullable)
- status: Enum (CLOCKED_IN, CLOCKED_OUT)
- description: Text
- createdAt, updatedAt: DateTime
```

invoices

```
- id: UUID (PK)
- invoiceNumber: String (unique)
- invoiceDate: Date
- dueDate: Date
- customerId: UUID (FK → customers)
- status: Enum (DRAFT, SENT, PAID, CANCELLED)
- subtotal, vatAmount, totalAmount: Decimal
- notes: Text
- templateId: UUID (FK → invoice_templates)
- createdAt, updatedAt: DateTime
```

workflows

```
- id: UUID (PK)
- name: String
- description: Text
- isActive: Boolean
- definition: JSON (nodes & edges für Flow-Editor)
- createdAt, updatedAt: DateTime
```

workflow_steps

```
- id: UUID (PK)
- workflowId: UUID (FK → workflows)
- name: String
- type: Enum (APPROVAL, EMAIL, VALUE_CONDITION, etc.)
- order: Integer
- approverUserIds: JSON
- approverGroupIds: JSON
- requireAllApprovers: Boolean
- config: JSON (step-spezifische Konfiguration)
```

workflow_instances

```
- id: UUID (PK)
- workflowId: UUID (FK → workflows)
- invoiceId: UUID (FK → invoices)
- status: Enum (IN_PROGRESS, COMPLETED, CANCELLED)
- currentStepId: UUID (FK → workflow_steps)
- createdAt, updatedAt, completedAt: DateTime
```

compliance_violations

```
- id: UUID (PK)
- userId: UUID (FK → users)
- type: Enum (REST_PERIOD, DAILY_HOURS, WEEKLY_HOURS, PAUSE, OVERTIME)
- severity: Enum (CRITICAL, WARNING)
- date: Date
- description: Text
- actualValue, requiredValue: Decimal
- resolved: Boolean
- resolvedAt: DateTime
- resolvedBy: UUID (FK → users)
- resolvedNote: Text
```

Wichtige Relationen

```
users ←─┐
        ├── time_entries
projects ←┘
locations ←┘

users ←── absence_requests

invoices ←─── invoice_items
customers ←┘

workflows ←── workflow_steps
workflows ←── workflow_instances ──→ invoices

invoice_templates ←── invoice_template_workflows ──→ workflows
```

API-Endpunkte

Authentication

Methode	Endpoint	Beschreibung	Auth
POST	`/api/auth/register`	Neuen Benutzer registrieren	-
POST	`/api/auth/login`	Benutzer anmelden	-
GET	`/api/auth/me`	Aktuellen Benutzer abrufen	✓

Time Tracking

Methode	Endpoint	Beschreibung	Auth
POST	`/api/time/clock-in`	Einstempeln	✓
POST	`/api/time/clock-out`	Ausstempeln	✓
GET	`/api/time/current`	Aktuellen Zeiteintrag	✓
GET	`/api/time/my-entries`	Eigene Zeiteinträge	✓
GET	`/api/time/user/:userId`	Zeiteinträge eines Users	Admin
PUT	`/api/time/:id`	Zeiteintrag korrigieren	Admin
DELETE	`/api/time/:id`	Zeiteintrag löschen	Admin

Invoices

Methode	Endpoint	Beschreibung	Auth
GET	`/api/invoices`	Alle Rechnungen	Admin
GET	`/api/invoices/:id`	Rechnung Details	Admin
POST	`/api/invoices`	Rechnung erstellen	Admin
PUT	`/api/invoices/:id`	Rechnung aktualisieren	Admin
DELETE	`/api/invoices/:id`	Rechnung löschen	Admin
GET	`/api/invoices/:id/pdf`	Rechnung als PDF	Admin

Workflows

Methode	Endpoint	Beschreibung	Auth
GET	`/api/workflows`	Alle Workflows	Admin
GET	`/api/workflows/:id`	Workflow Details	Admin
POST	`/api/workflows`	Workflow erstellen	Admin
PUT	`/api/workflows/:id`	Workflow aktualisieren	Admin
DELETE	`/api/workflows/:id`	Workflow löschen	Admin
GET	`/api/workflows/my-approvals`	Eigene Genehmigungen	✓

Methode	Endpoint	Beschreibung	Auth
POST	`/api/workflows/:id/approve`	Workflow-Schritt genehmigen	✓
POST	`/api/workflows/:id/reject`	Workflow-Schritt ablehnen	✓

Compliance

Methode	Endpoint	Beschreibung	Auth
GET	`/api/compliance/violations`	Violations (gefiltert)	Admin
GET	`/api/compliance/violations/stats`	Violations Statistiken	Admin
PATCH	`/api/compliance/violations/:id/resolve`	Violation auflösen	Admin
GET	`/api/compliance/overtime/:userId`	Überstunden-Saldo	Admin
GET	`/api/compliance/holidays`	Feiertage nach Jahr/Kanton	✓

Workflow-System

Konzept

Das Workflow-System ermöglicht die Automatisierung von Geschäftsprozessen, insbesondere für Rechnungsgenehmigungen und Benachrichtigungen.

Workflow-Editor

ReactFlow Integration:

- Visueller Node-Editor mit Drag & Drop
- Nodes repräsentieren Workflow-Schritte
- Edges definieren die Ausführungsreihenfolge
- Echtzeit-Vorschau der Workflow-Struktur

Workflow-Ausführung

Edge-basierte Execution:

1. Workflow wird durch Rechnungsstatus-Änderung ausgelöst (z.B. DRAFT → SENT)
2. System parst `workflow.definition` (JSON mit nodes & edges)
3. Start-Node wird identifiziert
4. Edges werden gefolgt, um nächste Steps zu finden
5. VALUE_CONDITION Nodes werden evaluiert
6. Nur verbundene Steps (via Edges) werden ausgeführt
7. APPROVAL Steps werden als PENDING markiert, zugewiesene Benutzer benachrichtigt
8. EMAIL Steps werden sofort ausgeführt
9. Workflow schreitet fort nach Genehmigung/Ablehnung

VALUE_CONDITION Logic:

```
// Beispiel: Rechnungsbetrag > 1000 CHF
{
  "type": "VALUE_CONDITION",
  "config": {
    "field": "totalAmount",
    "operator": "greater",  // >, >=, <, <=, ==, !=
    "value": 1000
  }
}

// Bei true: Folge edges mit sourceHandle="true"
// Bei false: Folge edges mit sourceHandle="false"
```

EMAIL Step:

```
{
  "type": "EMAIL",
  "config": {
    "recipients": ["user@example.com"],  // oder User-IDs
    "subject": "Rechnung {invoiceNumber} genehmigt",
    "body": "Die Rechnung über {totalAmount} CHF wurde genehmigt."
  }
}

// Platzhalter:
// - {invoiceNumber}
// - {totalAmount}
// - {customerName}
// - {invoiceDate}
// - {dueDate}
```

Workflow-Template-Zuordnung

1. Invoice-Template erstellen
2. Workflows zuordnen mit Reihenfolge
3. Bei Rechnungserstellung mit Template: Workflows automatisch zugeordnet
4. Bei Status-Änderung auf SENT: Workflows starten automatisch

---

Swiss Compliance

Rechtliche Grundlage

Das System implementiert Überwachung gemäss:

- **Art. 9 ArG** - Höchstarbeitszeit (45h/50h Woche, 12.5h Tag)
- **Art. 15 ArGV 1** - Pausen (15min ab 5.5h, 30min ab 7h, 60min ab 9h)
- **Art. 15a ArG** - Ruhezeit (11h zwischen Arbeitstagen)

Violation-Typen

Typ	Severity	Beschreibung	Schwellenwert
REST_PERIOD	CRITICAL	Ruhezeit verletzt	< 11 Stunden
DAILY_HOURS	CRITICAL	Tägliche Arbeitszeit überschritten	> 12.5 Stunden
WEEKLY_HOURS	WARNING/CRITICAL	Wöchentliche Stunden überschritten	> 45h oder 50h
PAUSE	WARNING	Pausenregelung verletzt	Je nach Arbeitszeit
OVERTIME	WARNING	Überstunden akkumuliert	Konfigurierbar

Automatische Prüfung

Bei Clock-Out:

1. Berechne Arbeitszeit des Tages
2. Prüfe tägliche Höchstarbeitszeit (12.5h)
3. Prüfe Pausen (15min/30min/60min je nach Arbeitszeit)
4. Berechne wöchentliche Arbeitszeit
5. Prüfe wöchentliche Höchstarbeitszeit (45h/50h)
6. Prüfe Ruhezeit zum vorherigen Tag (11h)
7. Erstelle Violations bei Verstößen
8. Berechne Überstunden (regular + extra time)

Überstunden-Berechnung

```
Regular Overtime = Wochenarbeitszeit - Vertragsstunden
Extra Time = Wochenarbeitszeit - Gesetzliche Höchstarbeitszeit

Beispiel:
- Vertragsstunden: 42h
- Gesetzliche Höchstarbeitszeit: 45h
- Tatsächliche Arbeitszeit: 47h

→ Regular Overtime: 47h - 42h = 5h
→ Extra Time: 47h - 45h = 2h
```

Feiertage

- Alle 26 Schweizer Kantone unterstützt
- Nationale Feiertage + kantonale Feiertage
- Prozentsatz für halbe Feiertage (z.B. 50% = halber Tag)
- Automatische Berücksichtigung bei Arbeitszeitberechnung

Rechnungsverwaltung

Rechnungserstellung

Workflow:

1. Kunde auswählen oder erstellen
2. Rechnungsdatum und Fälligkeitsdatum festlegen
3. Positionen hinzufügen (Beschreibung, Menge, Preis, MwSt.)
4. Notizen hinzufügen (optional)
5. Als Entwurf speichern
6. PDF-Vorschau generieren
7. Auf "Versendet" setzen → Workflows starten

Positionen:

- Freitextbeschreibung
- Menge (Dezimal)
- Einzelpreis (Dezimal)
- MwSt.-Satz (0%, 2.6%, 8.1%)
- Automatische Berechnung von Zwischensumme, MwSt., Gesamt

Rechnungsvorlagen

Vorteile:

- Wiederverwendbare Strukturen
- Vordefinierte Positionen
- Standard-MwSt.-Sätze
- Workflow-Automatisierung

Workflow-Zuordnung:

- Mehrere Workflows pro Vorlage
- Reihenfolge definierbar
- Aktiv/Inaktiv Schalter
- Automatischer Start bei Rechnungsversand

PDF-Generierung

Implementierung:

- PDFKit für serverseitige Generierung
- Firmen-Logo und -Daten aus System-Einstellungen
- Mehrsprachige Unterstützung (Deutsch)
- Professionelles Layout mit Tabellen

Inhalt:

- Firmenkopf mit Logo
- Rechnungsnummer und Datum
- Kundenadresse
- Positionstabelle mit MwSt.

- Summen (Zwischensumme, MwSt., Gesamt)
- Zahlungsbedingungen
- Fußzeile mit Kontaktdaten

Sicherheit

Authentication & Authorization

JWT-basierte Authentifizierung:

- Tokens mit konfigurierbarem Secret
- Expiration Zeit: 7 Tage (konfigurierbar)
- Refresh-Mechanismus (optional)

Rollen-System:

- `USER` - Standardbenutzer (Zeiterfassung, eigene Abwesenheiten)
- `ADMIN` - Administrator (voller Zugriff)

Middleware:

```
// Auth-Check
requireAuth: Prüft JWT-Token

// Admin-Check
requireAdmin: Prüft Token + Admin-Rolle

// Optional Auth
optionalAuth: Token prüfen, aber nicht zwingend
```

Passwort-Sicherheit

- bcrypt Hashing mit 10 Salt Rounds
- Mindestlänge: 6 Zeichen (konfigurierbar)
- Beim ersten Login: Passwortänderung erzwungen
- Passwort-Reset via E-Mail (geplant)

API-Sicherheit

Implementierte Maßnahmen:

- CORS aktiviert mit konfigur ierbaren Origins
- Helmet.js für Security Headers
- Rate Limiting (geplant)
- Input-Validierung mit express-validator
- SQL-Injection Schutz durch Prisma ORM
- XSS-Schutz durch React (automatisch)

Security Headers:

```
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000
```

Datenschutz

DSGVO/DSG Compliance:

- Passwörter werden nur gehasht gespeichert
- Keine Klartext-Speicherung sensibler Daten
- Backup-Funktion für Datenportabilität
- Admin-Zugriffe werden geloggt (geplant)
- Löschfunktion für Benutzer-Daten

Deployment

Docker Deployment (Empfohlen)

Produktions-Setup:

```yaml
# docker-compose.yml
version: '3.8'
services:
  db:
    image: postgres:16-alpine
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: always

  backend:
    build: ./backend
    environment:
      DATABASE_URL: postgresql://timetracking:${DB_PASSWORD}@db:5432/timetracking
      JWT_SECRET: ${JWT_SECRET}
      NODE_ENV: production
    depends_on:
      - db
    restart: always

  frontend:
    build: ./frontend
    ports:
      - "80:80"
      - "443:443"
    depends_on:
```

```
      - backend
    restart: always
```

Umgebungsvariablen (.env):

```
DB_PASSWORD=sicheres-passwort-hier
JWT_SECRET=sehr-sicherer-jwt-secret
NODE_ENV=production
```

Deployment-Schritte:

```
# 1. .env Datei erstellen
echo "DB_PASSWORD=..." > .env
echo "JWT_SECRET=..." >> .env

# 2. Containers bauen und starten
docker-compose up -d --build

# 3. Logs prüfen
docker-compose logs -f

# 4. Backup einrichten (cron job)
0 2 * * * docker exec timetracking-backend npm run backup
```

Nginx Konfiguration

SSL/TLS Setup:

```
server {
    listen 443 ssl http2;
    server_name your-domain.com;

    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;

    # Frontend
    location / {
        root /usr/share/nginx/html;
        try_files $uri $uri/ /index.html;
    }

    # Backend API
    location /api {
        proxy_pass http://backend:3001;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
```

```
    }
}
```

Backup-Strategie

Automatische Backups:

- Tägliche PostgreSQL Dumps
- Rotation: 7 Tage behalten
- Offsite-Backup empfohlen

Backup-Skript:

```
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
docker exec timetracking-db pg_dump -U timetracking timetracking >
backup_$DATE.sql
# Upload zu S3/Cloud Storage
```

Monitoring

Health Checks:

- Backend: `/health` Endpoint
- Database: `pg_isready` Check
- Frontend: Nginx Process Check

Empfohlene Tools:

- Prometheus + Grafana für Metriken
- ELK Stack für Logs
- Uptime-Monitoring (UptimeRobot, etc.)

Skalierung

Horizontal Scaling:

- Mehrere Backend-Instanzen hinter Load Balancer
- Shared PostgreSQL Database
- Session-Storage in Redis (geplant)

Vertical Scaling:

- PostgreSQL Connection Pooling
- Backend Worker Prozesse
- Nginx Caching

Anhang

Nützliche Befehle

Docker:

```
# Alle Container neu starten
docker-compose restart

# Backend neu bauen
docker-compose up -d --build backend

# Logs anzeigen
docker-compose logs -f

# Container-Shell öffnen
docker exec -it timetracking-backend sh

# Datenbank-Shell
docker exec -it timetracking-db psql -U timetracking
```

Prisma:

```
# Migration erstellen
npm run prisma:migrate

# Datenbank zurücksetzen
npm run prisma:reset

# Prisma Studio öffnen
npm run prisma:studio

# Schema synchronisieren
npm run prisma:push
```

Entwicklung:

```
# Backend mit Hot-Reload
cd backend && npm run dev

# Frontend mit Hot-Reload
cd frontend && npm start

# TypeScript kompilieren
npm run build

# Tests ausführen
npm test
```

Bekannte Einschränkungen

1. **Zeiterfassung:**
   - Keine automatische Pause-Erkennung
   - Keine Geofencing-Unterstützung
   - Keine Offline-Funktionalität
2. **Workflows:**
   - Keine parallelen Genehmigungen
   - Keine Eskalations-Mechanismen
   - Keine Workflow-Vorlagen
3. **Compliance:**
   - Nachtarbeit (23:00-06:00) nicht speziell geprüft
   - Sonntagsarbeit keine separate Violation
   - Keine flexible Arbeitszeit-Modelle
4. **Rechnungen:**
   - Kein Swiss QR-Code (geplant)
   - Keine E-Rechnung (ZUGFeRD/Factur-X)
   - Keine Mahnungs-Automatisierung

Roadmap

Q1 2026:

- ☐ Mobile Apps (iOS/Android)
- ☐ Swiss QR-Code Integration
- ☐ Workflow-Vorlagen
- ☐ Eskalations-Mechanismen

Q2 2026:

- ☐ Multi-Mandanten-Fähigkeit
- ☐ SSO/LDAP Integration
- ☐ Erweiterte Reporting (PDF/Excel)
- ☐ API-Dokumentation (Swagger)

Q3 2026:

- ☐ Geofencing für Clock-In
- ☐ Nachtarbeits-Violations
- ☐ Flexible Arbeitszeit-Modelle
- ☐ Integration mit Lohnbuchhaltung

Q4 2026:

- ☐ E-Rechnung Support

- [ ] Automatische Mahnungen
- [ ] Projektbudget-Tracking
- [ ] Ressourcen-Planung

Kontakt & Support

Dokumentation:

- GitHub: [Repository-URL]
- Docs: `/docs` Verzeichnis

Lizenz: MIT

Autor: cflux Development Team

Letzte Aktualisierung: Dezember 2025 Version: 1.0