cflux - Zeiterfassungssystem

Vollständige Dokumentation

Version 1.0
Stand: Januar 2025

Inhaltsverzeichnis

Überblick

Was ist cflux?

cflux ist ein vollständiges Enterprise Resource Planning (ERP) System mit Fokus auf Zeiterfassung, Projektmanagement und Personalverwaltung. Das System wurde speziell für den Schweizer Markt entwickelt und erfüllt alle relevanten gesetzlichen Anforderungen.

Hauptmerkmale

- ☑ **Zeiterfassung**: Ein-/Ausstempeln mit Projektbuchung
- ☑ **Projektmanagement**: Verwaltung von Projekten und Zuordnung zu Mitarbeitern
- ☑ **Urlaubsverwaltung**: Antragstellung und Genehmigungsprozesse
- ☑ **Abwesenheitsmanagement**: Krankheit, persönliche Gründe, etc.
- ☑ **Reporting**: Umfassende Reports und Statistiken
- ☑ **Benutzerverwaltung**: Rollenbasierte Zugriffskontrolle
- ☑ **Rechnungsstellung**: Integriertes Invoicing-System
- ☑ **Schweizer Compliance**: OR-konform, Mehrwertsteuer-ready

Technologie-Stack

Backend:

- Node.js 18+ / TypeScript 5.x
- Express.js Framework
- PostgreSQL 14+ Datenbank
- Prisma ORM

- JWT Authentication
- bcrypt Password Hashing

Frontend:

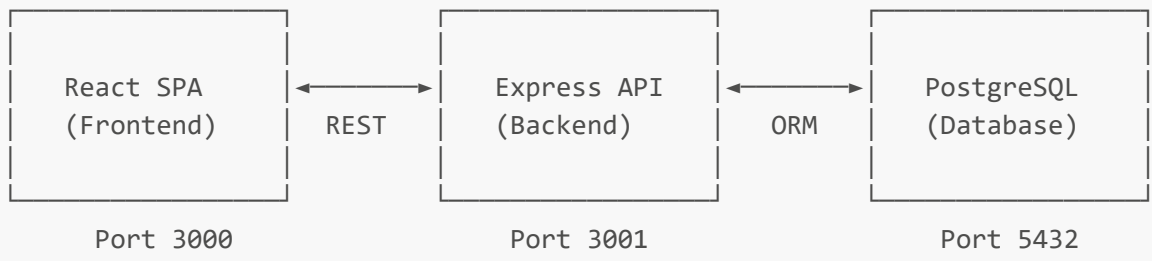
- React 18 / TypeScript
- React Router v6
- Axios HTTP Client
- Modern CSS3

DevOps:

- Docker & Docker Compose
- GitHub Actions CI/CD
- Nginx Reverse Proxy

Architektur

System-Übersicht



Datenbank-Schema

Das System basiert auf folgenden Hauptentitäten:

User (Benutzer)

- Basisdaten: Email, Name, Passwort
- Rolle: USER, ADMIN
- Urlaubskontingent
- Status: Aktiv/Inaktiv

TimeEntry (Zeiteintrag)

- Zuordnung: Benutzer, Projekt
- Zeitstempel: clockIn, clockOut
- Status: CLOCKED_IN, CLOCKED_OUT
- Beschreibung/Notizen

Project (Projekt)

- Projektstammdaten

- Zugewiesene Benutzer (Many-to-Many)
- Status: Aktiv/Inaktiv

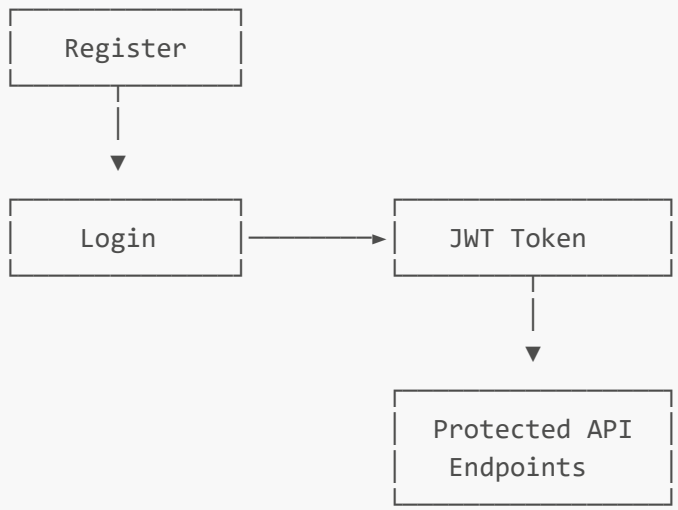
Absence (Abwesenheit)

- Typ: Urlaub, Krankheit, Sonstige
- Zeitraum: Start-/Enddatum
- Status: Pending, Approved, Rejected
- Kommentare

Invoice (Rechnung)

- Rechnungsnummer
- Kunde/Projekt
- Positionen
- Mehrwertsteuer (MWST)
- Status: Draft, Sent, Paid

Authentifizierung



JWT Token enthält:

- User ID
- Email
- Role (USER/ADMIN)
- Expiration (24h Standard)

Systemanforderungen

Produktionsumgebung

Hardware (Minimum für 50 Benutzer):

- CPU: 2 Cores

- RAM: 4 GB
- Storage: 20 GB SSD
- Netzwerk: 100 Mbit/s

Software:

- Docker 20.10+
- Docker Compose 2.0+
- Linux (Ubuntu 22.04 LTS empfohlen)

Für größere Deployments (100+ Benutzer):

- CPU: 4+ Cores
- RAM: 8+ GB
- Storage: 50+ GB SSD
- Load Balancer empfohlen

Entwicklungsumgebung

- Node.js 18 oder höher
- npm 9+ oder yarn 1.22+
- PostgreSQL 14+
- Git
- Code Editor (VS Code empfohlen)

Installation

Produktions-Setup mit Docker

Schritt 1: Repository klonen

```
git clone https://github.com/mpue/cflux.git
cd cflux
```

Schritt 2: Umgebungsvariablen konfigurieren

```
# Backend .env
cp backend/.env.example backend/.env

# Wichtige Variablen in backend/.env anpassen:
DATABASE_URL="postgresql://timetracking:SICHERES_PASSWORT@db:5432/timetracking"
JWT_SECRET="SEHR_SICHERER_RANDOM_STRING_MINIMUM_32_ZEICHEN"
NODE_ENV="production"
PORT=3001
```

Schritt 3: Docker Compose starten

```
docker-compose up -d
```

Das System ist nun verfügbar unter:

- Frontend: http://localhost:3000
- Backend API: http://localhost:3001
- Datenbank: localhost:5432

Schritt 4: Ersten Admin erstellen

```
# 1. Registriere einen Benutzer über die Web-Oberfläche
(http://localhost:3000/register)
# 2. Setze die Rolle auf ADMIN:
docker exec -it timetracking-db psql -U timetracking -d timetracking
```

In der psql-Shell:

```
UPDATE users SET role = 'ADMIN' WHERE email = 'admin@example.com';
\q
```

Entwicklungs-Setup (Lokal)

Backend:

```
cd backend
npm install
cp .env.example .env
# .env anpassen

# Datenbank Migrationen
npm run prisma:migrate
npm run prisma:generate

# Development Server starten
npm run dev
```

Frontend:

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

Konfiguration

Umgebungsvariablen (Backend)

Variable	Beschreibung	Beispiel	Pflicht
`DATABASE_URL`	PostgreSQL Verbindung	`postgresql://user:pass@host:5432/db`	Ja
`JWT_SECRET`	Secret für JWT Tokens	`random-string-min-32-chars`	Ja
`JWT_EXPIRATION`	Token Gültigkeitsdauer	`24h`	Nein (Default: 24h)
`PORT`	Backend Port	`3001`	Nein (Default: 3001)
`NODE_ENV`	Umgebung	`production` / `development`	Nein
`CORS_ORIGIN`	Erlaubte Origins	`http://localhost:3000`	Nein
`LOG_LEVEL`	Logging Level	`info` / `debug` / `error`	Nein

Docker Compose Konfiguration

Die `docker-compose.yml` kann angepasst werden für:

Ports ändern:

```
services:
  frontend:
    ports:
      - "8080:80"  # Frontend auf Port 8080
  backend:
    ports:
      - "8081:3001"  # Backend auf Port 8081
```

Persistent Volumes:

```
volumes:
  postgres_data:
    driver: local
    driver_opts:
      type: none
      device: /opt/cflux/data
      o: bind
```

Memory Limits:

```
services:
  backend:
    deploy:
      resources:
        limits:
          memory: 1G
        reservations:
          memory: 512M
```

Nginx Konfiguration (Reverse Proxy)

Für Produktions-Deployment mit eigenem Nginx:

```
server {
    listen 80;
    server_name zeiterfassung.example.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location /api {
        proxy_pass http://localhost:3001;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Mit SSL (Let's Encrypt):

```
certbot --nginx -d zeiterfassung.example.com
```

Module und Features

1. Zeiterfassung

Funktionen:

- Ein-/Ausstempeln per Button-Click
- Automatische Zeitberechnung
- Projektbuchung während Arbeitszeit
- Nachträgliche Korrekturen (Admin)
- Pausenzeiten-Erfassung

Workflows:

1. Benutzer stempelt ein → Status: CLOCKED_IN
2. Benutzer wählt Projekt aus
3. Benutzer stempelt aus → Zeitdauer wird berechnet
4. Eintrag wird gespeichert und in Reports berücksichtigt

Business Rules:

- Maximale Arbeitszeit pro Tag: 10 Stunden (konfigurierbar)
- Pflicht-Pausen bei >6h Arbeitszeit
- Nur ein aktiver TimeEntry pro Benutzer
- Zeiteinträge können nicht in die Zukunft gebucht werden

2. Projektverwaltung

Funktionen:

- Projekte anlegen, bearbeiten, archivieren
- Mitarbeiter zu Projekten zuweisen
- Projektstatus (Aktiv/Inaktiv)
- Projektbasierte Zeitauswertung

Zugriffsrechte:

- USER: Sieht nur zugewiesene Projekte
- ADMIN: Vollzugriff auf alle Projekte

3. Urlaubsverwaltung

Funktionen:

- Urlaubsanträge stellen
- Genehmigungsprozess (Admin)
- Urlaubskontingent-Verwaltung
- Urlaubskalender
- Resturlaubsberechnung

Workflows:

1. Benutzer stellt Urlaubsantrag
2. Status: PENDING
3. Admin genehmigt/lehnt ab
4. Status: APPROVED/REJECTED
5. Bei Genehmigung: Urlaubstage werden vom Kontingent abgezogen

4. Abwesenheitsmanagement

Typen:

- Urlaub
- Krankheit
- Persönliche Gründe
- Weiterbildung
- Home Office
- Sonstige

Features:

- Ganztags- oder Teilzeit-Abwesenheiten
- Ärztliche Bescheinigung (Upload optional)
- Benachrichtigungen an Vorgesetzte
- Kalenderintegration

5. Reporting & Analytics

Standard Reports:

- Persönliche Arbeitszeitübersicht
- Projekt-Zeitauswertung
- Team-Übersicht (Admin)
- Urlaubsübersicht
- Fehlzeiten-Report
- Monats-/Jahresberichte

Export-Formate:

- PDF
- Excel (XLSX)
- CSV

Auswertungszeiträume:

- Tag
- Woche
- Monat
- Quartal
- Jahr
- Frei wählbar

6. Benutzerverwaltung

Admin-Funktionen:

- Benutzer anlegen/bearbeiten/löschen
- Rollenänderungen (USER ↔ ADMIN)
- Urlaubskontingent festlegen

- Benutzer deaktivieren (Soft Delete)
- Passwort zurücksetzen

Benutzer-Selbstverwaltung:

- Profil bearbeiten
- Passwort ändern
- Eigene Zeiteinträge einsehen

7. Rechnungsstellung

Funktionen:

- Rechnungen erstellen und versenden
- Mehrwertsteuer (MWST) Berechnung
- Rechnungspositionen mit Zeitbuchungen verknüpfen
- Zahlungsstatus-Tracking
- PDF-Export mit QR-Rechnung (Schweiz)

Swiss QR Code Integration:

- Automatische QR-Code Generierung
- IBAN/Referenznummer
- Compliance mit Swiss Payment Standards

Sicherheit

Authentifizierung & Autorisierung

JWT Token Security:

- Tokens sind signiert und verschlüsselt
- Kurze Lebensdauer (24h Standard)
- Refresh Token Mechanismus (optional aktivierbar)
- Tokens werden im LocalStorage gespeichert (XSS-Protection durch HTTPOnly Cookies optional)

Password Security:

- bcrypt Hashing mit Salt (10 Rounds)
- Mindestlänge: 8 Zeichen
- Passwort-Stärke-Validierung
- Rate Limiting bei Login-Versuchen

Role-Based Access Control (RBAC):

```
// Middleware Beispiel
const requireAdmin = (req, res, next) => {
  if (req.user.role !== 'ADMIN') {
    return res.status(403).json({ error: 'Admin rechte erforderlich' });
  }
```

```
  next();
};
```

Input Validation

Alle API-Endpunkte nutzen `express-validator`:

```
body('email').isEmail().normalizeEmail(),
body('password').isLength({ min: 8 }),
body('firstName').trim().notEmpty(),
```

SQL Injection Protection

- Prisma ORM verhindert SQL Injection durch Prepared Statements
- Keine direkten SQL Queries im Code
- Input Sanitization vor Datenbankzugriff

CORS Configuration

```
app.use(cors({
  origin: process.env.CORS_ORIGIN || 'http://localhost:3000',
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
}));
```

Rate Limiting

```
const rateLimit = require('express-rate-limit');

const loginLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 Minuten
  max: 5, // 5 Versuche
  message: 'Zu viele Login-Versuche, bitte später erneut versuchen'
});

app.post('/api/auth/login', loginLimiter, ...);
```

HTTPS in Produktion

Empfohlener Setup mit Let's Encrypt:

```
# Certbot installieren
sudo apt-get install certbot python3-certbot-nginx
```

```
# Zertifikat erstellen
sudo certbot --nginx -d your-domain.com

# Auto-Renewal
sudo certbot renew --dry-run
```

Deployment

Produktions-Deployment Checkliste

- [ ] Umgebungsvariablen in `.env` setzen
- [ ] Sichere Passwörter generieren (DB, JWT_SECRET)
- [ ] HTTPS/SSL konfigurieren
- [ ] Firewall-Regeln setzen
- [ ] Backup-Strategie implementieren
- [ ] Monitoring aufsetzen
- [ ] Log-Rotation konfigurieren
- [ ] E-Mail-Benachrichtigungen testen
- [ ] Disaster Recovery Plan

Docker Production Deployment

docker-compose.prod.yml:

```
version: '3.8'

services:
  db:
    image: postgres:16-alpine
    restart: always
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - /opt/cflux/data:/var/lib/postgresql/data
    networks:
      - cflux-network

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile.prod
    restart: always
    environment:
      NODE_ENV: production
      DATABASE_URL: ${DATABASE_URL}
      JWT_SECRET: ${JWT_SECRET}
    depends_on:
```

```
      - db
    networks:
      - cflux-network

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile.prod
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt:ro
    depends_on:
      - backend
    networks:
      - cflux-network

networks:
  cflux-network:
    driver: bridge

volumes:
  postgres_data:
```

Deployment Befehle:

```
# Build und Start
docker-compose -f docker-compose.prod.yml up -d --build

# Logs
docker-compose -f docker-compose.prod.yml logs -f

# Backup
docker exec timetracking-db pg_dump -U timetracking timetracking > backup_$(date
+%Y%m%d).sql

# Restore
docker exec -i timetracking-db psql -U timetracking timetracking < backup.sql
```

Monitoring & Logging

Prometheus + Grafana Setup:

```
# prometheus.yml
global:
  scrape_interval: 15s
```

```
scrape_configs:
  - job_name: 'cflux'
    static_configs:
      - targets: ['backend:3001']
```

Log Aggregation mit ELK Stack:

- Elasticsearch: Log Storage
- Logstash: Log Processing
- Kibana: Visualization

Health Checks:

```
// Backend Health Endpoint
app.get('/health', async (req, res) => {
  const dbStatus = await checkDatabase();
  res.json({
    status: 'ok',
    database: dbStatus,
    uptime: process.uptime(),
    timestamp: new Date()
  });
});
```

Backup-Strategie

Automatisches tägliches Backup:

```
#!/bin/bash
# /opt/cflux/backup.sh

BACKUP_DIR="/opt/cflux/backups"
DATE=$(date +%Y%m%d_%H%M%S)

# Database Backup
docker exec timetracking-db pg_dump -U timetracking timetracking | gzip > \
  $BACKUP_DIR/db_backup_$DATE.sql.gz

# Alte Backups löschen (älter als 30 Tage)
find $BACKUP_DIR -name "db_backup_*.sql.gz" -mtime +30 -delete
```

Crontab Eintrag:

```
0 2 * * * /opt/cflux/backup.sh
```

Scaling

Horizontal Scaling:

```
# docker-compose.scale.yml
services:
  backend:
    deploy:
      replicas: 3

  nginx-lb:
    image: nginx:alpine
    volumes:
      - ./nginx-lb.conf:/etc/nginx/nginx.conf
    ports:
      - "80:80"
```

Load Balancer Konfiguration:

```
upstream backend {
    least_conn;
    server backend1:3001;
    server backend2:3001;
    server backend3:3001;
}

server {
    location /api {
        proxy_pass http://backend;
    }
}
```

Troubleshooting

Häufige Probleme

Problem: "Cannot connect to database"

Ursache: Datenbank nicht erreichbar oder falsche Credentials

Lösung:

```
# Prüfe ob DB Container läuft
docker ps | grep timetracking-db

# Prüfe Logs
docker logs timetracking-db

# Teste Verbindung
```

```
docker exec -it timetracking-db psql -U timetracking -d timetracking -c "SELECT
1;"

# Prüfe DATABASE_URL in .env
cat backend/.env | grep DATABASE_URL
```

Problem: "JWT token expired"

Ursache: Token-Lebensdauer abgelaufen

Lösung:

- Benutzer muss sich neu anmelden
- Optional: Refresh Token implementieren
- Token-Lebensdauer in .env anpassen: `JWT_EXPIRATION=48h`

Problem: Frontend zeigt "Cannot connect to API"

Ursache: CORS oder Backend nicht erreichbar

Lösung:

```
# Prüfe Backend Status
curl http://localhost:3001/health

# Prüfe CORS_ORIGIN
docker exec timetracking-backend env | grep CORS

# Browser Console prüfen (F12)
# CORS Fehler? → CORS_ORIGIN in backend/.env anpassen
```

Problem: "Prisma Client out of sync"

Ursache: Schema-Änderungen ohne Client-Regenerierung

Lösung:

```
cd backend
npm run prisma:generate
# Bei Docker:
docker exec timetracking-backend npm run prisma:generate
```

Problem: Hohe Memory-Nutzung

Ursache: Zu viele gleichzeitige Connections oder Memory Leak

Lösung:

```
# Memory Limits setzen
docker update --memory 1g --memory-swap 2g timetracking-backend

# Node Memory Limit
NODE_OPTIONS="--max-old-space-size=512" npm start
```

Debug-Modus aktivieren

Backend:

```
# In .env
DEBUG=express:*
LOG_LEVEL=debug

# Docker Restart
docker-compose restart backend
```

Frontend:

```
# Browser Console
localStorage.setItem('debug', 'true');
```

Logs analysieren

```
# Alle Logs
docker-compose logs -f

# Nur Backend
docker-compose logs -f backend

# Letzte 100 Zeilen
docker-compose logs --tail=100 backend

# Nach Fehler suchen
docker-compose logs backend | grep ERROR
```

Performance Probleme

Database Query Optimization:

```
# Slow Query Log aktivieren
docker exec -it timetracking-db psql -U timetracking
```

```
ALTER SYSTEM SET log_min_duration_statement = 1000; -- Log Queries > 1s
SELECT pg_reload_conf();
```

Index erstellen:

```
CREATE INDEX idx_time_entries_user_date ON time_entries(user_id, clock_in);
CREATE INDEX idx_absences_user_dates ON absences(user_id, start_date, end_date);
```

Anhang

Nützliche Befehle

Docker:

```
# Alle Container stoppen
docker-compose down

# Container neu bauen
docker-compose up -d --build

# In Container shell
docker exec -it timetracking-backend sh

# Volumes löschen (ACHTUNG: Datenverlust!)
docker-compose down -v
```

Datenbank:

```
# Backup
docker exec timetracking-db pg_dump -U timetracking timetracking > backup.sql

# Restore
docker exec -i timetracking-db psql -U timetracking timetracking < backup.sql

# Prisma Studio (GUI)
docker exec -it timetracking-backend npx prisma studio
```

Git Workflow:

```
# Feature Branch
git checkout -b feature/neue-funktion

# Commit
git add .
```

```
git commit -m "feat: Neue Funktion implementiert"

# Push
git push origin feature/neue-funktion
```

Lizenz

MIT License - Siehe LICENSE Datei

Support

Bei Fragen oder Problemen:

- GitHub Issues: https://github.com/mpue/cflux/issues
- E-Mail: support@cflux.ch (fiktiv)

Changelog

v1.0.0 - Januar 2025

- Initial Release
- Alle 21 Module implementiert
- 82% Test Coverage
- Docker Support
- Schweizer Compliance

Dokumentation Ende