The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found.

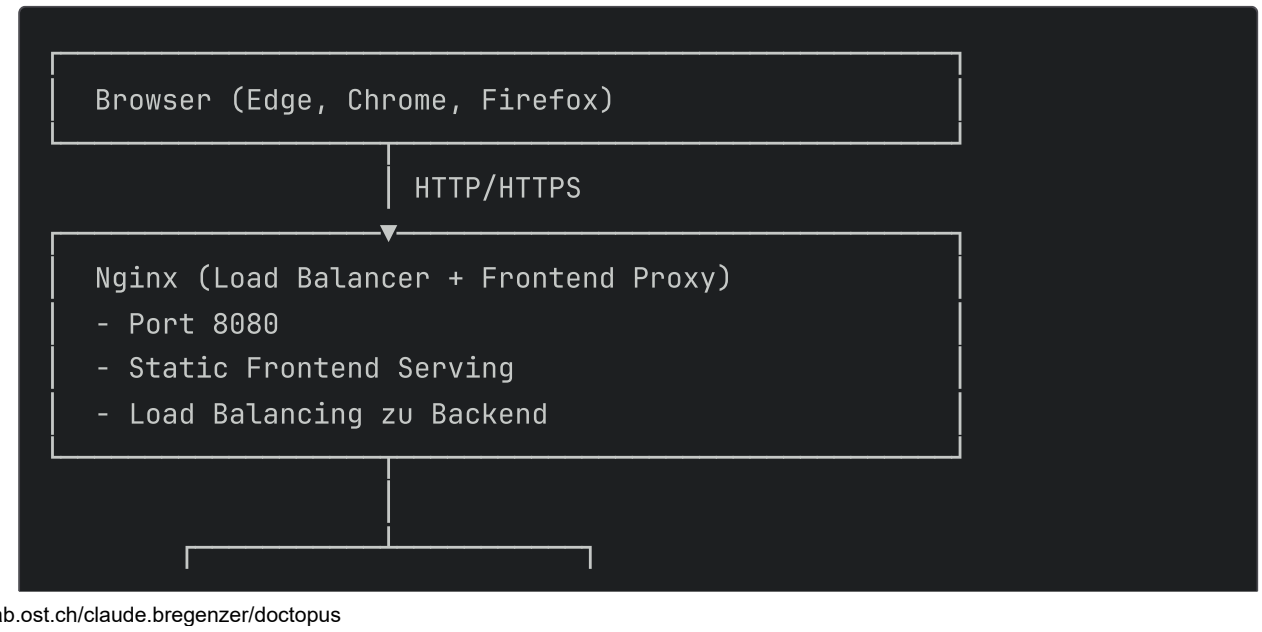
solved api.js
Claude Bregenzer authored 37 minutes ago

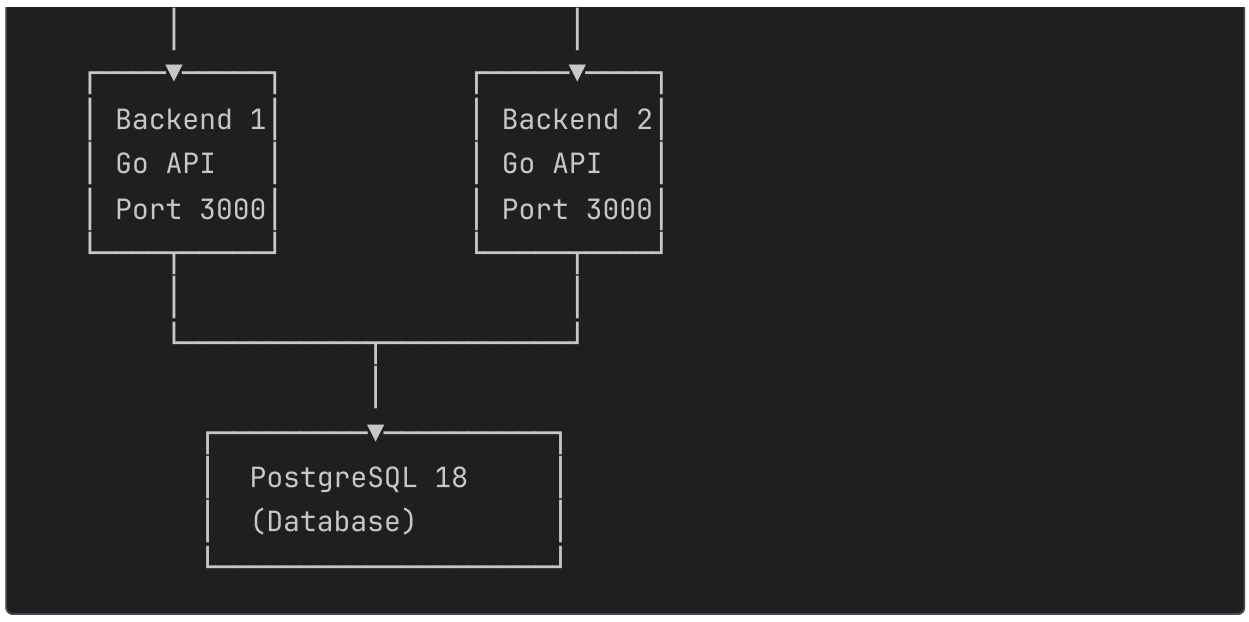
Name	Last commit	Last update
backend	solved api.js	37 minutes ago
db	improved security, added d...	7 hours ago
frontend	solved api.js	37 minutes ago
lb	modified rate limiting	7 hours ago
.dockerignore	add git and dockerignore	6 days ago
.gitignore	add git and dockerignore	6 days ago
LICENSE	Initial commit	2 weeks ago
README.md	solved api.js	37 minutes ago
agent.md	guidelines for ai	6 days ago
compose.yml	improved security, added d...	7 hours ago
package-lock.json	Add files via upload	2 weeks ago

README.md

DocTopus ist ein modernes, sicheres Dokumentenverwaltungssystem mit AsciiDoc-Editor, JWT-basierter Authentifizierung und Load-Balancing.

- **Sichere Authentifizierung** - JWT Tokens mit Access & Refresh Token Rotation
- **Live AsciiDoc Editor** - Echtzeit-Vorschau von dokumentationen
- **Dokumentenverwaltung** - CRUD-Operationen für Dokumente
- **Load Balancing** - Nginx-basiertes Load Balancing über 2 Backend-Instanzen
- **PostgreSQL Database** - Persistente Datenspeicherung
- **Docker Compose** - Vollständig containerisiert
- **TypeScript + React** - Modernes Frontend mit Next.js
- **Go Backend** - Performanter REST API Server





- **Framework:** Next.js 16.2.1
- **UI Library:** React 19.2.3
- **Language:** TypeScript
- **Editor:** AsciiDoctor Core 3.0.4
- **Linting:** ESLint 9.x

- **Language:** Go 1.23
- **HTTP:** Net/HTTP (Standard Library)
- **Auth:** JWT (golang-jwt/jwt v5.3.0)
- **Encryption:** crypto/bcrypt (golang.org/x/crypto)
- **Database:** PostgreSQL Driver (lib/pq)

- **Load Balancer:** Nginx 1.29-alpine
- **Database:** PostgreSQL 18-alpine
- **Container Runtime:** Docker + Docker Compose

- Docker & Docker Compose installed
- Git

```
# 1. Repository clonen
git clone <repository-url>
cd doctopus

# 2. Environment-Variablen setzen
cat > .env << EOF
POSTGRES_USER=doctopus
POSTGRES_PASSWORD=secretpassword
POSTGRES_DB=doctopus_db
ADMIN_PASSWORD=admin123
JWT_SECRET=your-super-secret-jwt-key-here
ENV=development
INSTANCE_NAME=api-instance
DB_HOST=db
DB_USER=doctopus
DB_PASSWORD=secretpassword
DB_NAME=doctopus_db
EOF
```

```
# 3. Docker Compose ausführen
docker compose down
docker compose build --no-cache
docker compose up -d

# 4. Zugriff
# Frontend: http://localhost:8080
# Backend API: http://localhost:8080/api
# Health: http://localhost:8080/health
```

- **Username:** `admin`
- **Password:** (aus `ADMIN_PASSWORD` env variable)

DocTopus nutzt ein **sicheres Dual-Token System**:

- **Gültigkeitsdauer:** 15 Minuten
- **Zweck:** API-Zugriff
- **Storage:** HttpOnly Cookie (`access_token`)
- **Type:** `"access"`

- **Gültigkeitsdauer:** 7 Tage
- **Zweck:** Neuen Access Token generieren
- **Storage:** HttpOnly Cookie (`refresh_token`)
- **Type:** `"refresh"`

```
{
  "user": "admin",
  "type": "access",
  "exp": 1234567890,
  "iat": 1234567800
}
```

- [OK] **HttpOnly Cookies** - JavaScript kann nicht zugreifen (XSS-Schutz)
- [OK] **HMAC-SHA256 Signierung** - Manipulation unmöglich
- [OK] **Type-Validierung** - Access Token für API, Refresh für Token-Refresh
- [OK] **SameSite-Policy** - CSRF-Schutz

```
POST /auth/login
  - Body: { username, password }
  - Response: { status: "logged in" }
  - Sets: access_token, refresh_token (Cookies)

POST /auth/refresh
  - Reads: refresh_token (Cookie)
  - Response: { status: "token refreshed" }
  - Updates: access_token (Cookie)

POST /auth/logout
```

Project information

Created on
February 16, 2026

```
  - Clears: access_token, refresh_token (Cookies)
  - Response: { status: "logged out" }
```

```
GET /health
  - Response: { instance: "api-1" | "api-2" }
  - Used by Load Balancer
```

```
GET /api/documents
  - Requires: access_token Cookie
  - Response: [{ id, title, content, author, created_at, updated_at }]

POST /api/documents
  - Requires: access_token Cookie
  - Body: { title, content }
  - Response: { id, title, content, ... }

GET /api/documents/:id
  - Requires: access_token Cookie
  - Response: { id, title, content, ... }

PUT /api/documents/:id
  - Requires: access_token Cookie
  - Body: { title, content }
  - Response: { id, title, content, ... }

DELETE /api/documents/:id
  - Requires: access_token Cookie
  - Response: 200 OK
```

```
doctopus/
├── backend/                 # Go Backend
│   ├── main.go             # Server Entry Point
│   ├── go.mod              # Go Dependencies
│   ├── dockerfile          # Backend Container
│   └── auth/
│       ├── jwt.go          # JWT Token Generation & Verification
│       ├── middleware.go    # JWT Middleware
│       └── user.go         # User Management
├── frontend/               # Next.js Frontend
│   ├── src/
│   │   ├── app/
│   │   │   ├── page.tsx    # Home/Dashboard
│   │   │   ├── pages/
│   │   │   │   ├── login/   # Login Page
│   │   │   │   └── dashboard/ # Dashboard
│   │   │   └── layout.tsx
│   │   ├── components/
│   │   │   ├── Editor.tsx  # AsciiDoc Editor
│   │   │   └── DocsTable.tsx
│   │   ├── api.js          # API Client (fetch wrapper)
│   │   └── app/
│   ├── package.json
│   ├── tsconfig.json
│   ├── Dockerfile          # Frontend Container
│   └── README.md
├── db/                     # Database
│   ├── init.sql           # Database Initialization
│   ├── insert_docs.sql    # Sample Documents
```

```
│   └── dockerfile          # DB Container (if needed)
├── lb/                     # Load Balancer
│   └── nginx.conf          # Nginx Configuration
├── compose.yml             # Docker Compose Setup
└── .env                    # Environment Variables
```

```
# Go Module abhängigkeiten installieren
cd backend
go mod tidy

# Backend lokal starten (+ .env setzen)
go run main.go

# Tests ausführen
go test ./...
```

Wichtige Go-Dependencies:

- `github.com/golang-jwt/jwt/v5` - JWT Token Handling
- `golang.org/x/crypto` - Bcrypt für Password Hashing
- `github.com/lib/pq` - PostgreSQL Driver

```
# Dependencies installieren
cd frontend
npm install

# Development Server starten (Port 3000)
npm run dev

# Build für Production
npm run build

# Production Server starten
npm run start

# Linting
npm run lint
```

```
# PostgreSQL Container durchsuchen
docker exec -it doctopus-db-1 psql -U doctopus -d doctopus_db

# Beispiel Queries
SELECT * FROM users;
SELECT * FROM documents;
```

---

db - PostgreSQL Database

- Image: `postgres:18-alpine`
- Port: 5432 (internal)
- Volume: `postgres_data:/var/lib/postgresql/data`

backend-1 & backend-2 - Go API Instances

- Image: `golang:1.23-alpine`
- Port: 3000 (internal)
- Depends On: db (health check)

lb - Nginx Load Balancer

- Image: `nginx:1.29-alpine`
- Port: 8080 → 80 (external)
- Serves: Frontend + Routes to Backend

```
# Services hochfahren
docker compose up -d

# Logs anschauen
docker compose logs -f backend-1
docker compose logs -f lb

# Services herunterfahren
docker compose down

# Volumes löschen (WARNUNG: Datenbank wird gelöscht!)
docker compose down -v

# Neu bauen ohne Cache
docker compose build --no-cache
```

---

1. **Cookies überprüfen:**
   - F12 → Application → Cookies → `localhost:3000`
   - Siehst du: `access_token` (HttpOnly), `refresh_token` (HttpOnly)
2. **Network Requests überprüfen:**
   - F12 → Network Tab → Einen Request anklicken
   - Request Headers → Cookie: `access_token=...` wird automatisch gesendet
3. **JWT Token dekodieren:**
   - Gehe zu https://jwt.io
   - Kopiere den Token-String
   - Siehst du die Claims inkl. `"type": "access"` oder `"type": "refresh"`

```
# Backend Logs
docker compose logs backend-1 -f

# Database Logs
docker compose logs db -f

# Nginx Logs
docker compose logs lb -f

# Alle Logs
docker compose logs -f
```

```
# Database
POSTGRES_USER=doctopus          # PostgreSQL User
```

```
POSTGRES_PASSWORD=secret          # PostgreSQL Password
POSTGRES_DB=doctopus_db           # Database Name
ADMIN_PASSWORD=admin123           # Initial Admin Password

# JWT
JWT_SECRET=your-secret-key        # HMAC Secret (min. 32 Zeichen!)

# Environment
ENV=development|production        # development = insecure cookies, produ
INSTANCE_NAME=api-1               # Instance Identifier

# Backend Database Connection
DB_HOST=db                        # Database Host
DB_USER=doctopus                  # Database User
DB_PASSWORD=secret                # Database Password
DB_NAME=doctopus_db               # Database Name
```

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    author VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Lösung: `.env` Datei mit `JWT_SECRET=...` erstellen

Lösung:

```
docker compose down
docker system prune -f
docker compose build --no-cache
docker compose up -d
```

Cookies im Browser löschen (F12 → Application → Clear All)

Lösung:

- Überprüfe Cookies (F12 → Application → Cookies)
- Überprüfe Network Tab auf 401 Unauthorized
- Melde dich neu an

Lösung:

```
docker compose down -v  # Volumes löschen
docker compose up -d    # Neu starten
```

Für Fragen oder Issues:

- GitHub Issues: Issues
- Email: support@doctopus.local

Made by the DocTopus Team

Zuletzt aktualisiert: März 2026