# 🚀 Terminkalender - Installations- und Inbetriebnahme Anleitung

**Version:** 1.1 (Production-Ready)  
**Datum:** 26.September 2025  
**Autor:** SWTP1 Team  
**Status:** Vollständig getestet und produktionsbereit

## 📋 Inhaltsverzeichnis

1. [Überblick](#überblick)
2. [Systemanforderungen](#systemanforderungen)
3. [Schnellstart (5-Minuten-Setup)](#schnellstart)
4. [Detaillierte Installation](#detaillierte-installation)
5. [Konfiguration](#konfiguration)
6. [Datenbank-Setup](#datenbank-setup)
7. [Erste Schritte](#erste-schritte)
8. [Admin-Funktionen](#admin-funktionen)
9. [Monitoring & Wartung](#monitoring--wartung)
10. [Troubleshooting](#troubleshooting)
11. [Produktionsbereitstellung](#produktionsbereitstellung)

## 🎯 Überblick

Der **Terminkalender** ist eine moderne, vollständig ausgestattete Kalender-Webanwendung mit:

### ✨ **Hauptfeatures:**

* 📅 **Terminverwaltung**: CRUD-Operationen mit erweiterten Metadaten
* 🎉 **Deutsche Feiertage**: Automatische Integration aller Bundesländer
* 👥 **Benutzerverwaltung**: JWT-basierte Authentifizierung
* 📊 **Admin-Dashboard**: Zentrale Systemverwaltung
* 💻 **System-Monitoring**: Echtzeit-Performance-Überwachung
* 🗄️ **Azure MySQL**: Cloud-Database-Integration
* 📱 **Responsive Design**: Funktioniert auf allen Geräten

### 🏗️ **Technologie-Stack:**

* **Backend**: Spring Boot 3.1.0, Java 17+
* **Frontend**: HTML5, CSS3, JavaScript (ES6+)
* **Database**: Azure MySQL Flexible Server (Prod), H2 (Dev)
* **Security**: JWT Authentication, Spring Security
* **Monitoring**: Spring Boot Actuator
* **Build**: Maven 3.8+

## 💻 Systemanforderungen

### **Minimum-Anforderungen:**

* **Java**: JDK 17 oder höher
* **Maven**: 3.8.0 oder höher
* **Node.js**: 18.x oder höher
* **npm**: 9.x oder höher
* **Angular CLI**: 17.x oder höher
* **RAM**: 2 GB verfügbarer Arbeitsspeicher
* **Speicher**: 500 MB freier Festplattenspeicher
* **Browser**: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

### **Empfohlene Anforderungen:**

* **Java**: JDK 21 (LTS)
* **Maven**: 3.9.x (neueste Version)
* **Node.js**: 20.x (LTS)
* **npm**: 10.x (neueste Version)
* **Angular CLI**: 18.x (neueste Version)
* **RAM**: 4 GB oder mehr
* **Speicher**: 2 GB freier Festplattenspeicher
* **Database**: Azure MySQL Flexible Server (Produktion)

### **Unterstützte Betriebssysteme:**

* ✅ Windows 10/11
* ✅ macOS 12+ (Monterey)
* ✅ Linux (Ubuntu 20.04+, CentOS 8+, Debian 11+)

## ⚡ Schnellstart (5-Minuten-Setup)

Für den schnellsten Einstieg folgen Sie diesen Schritten:

### **1. Repository klonen**

git clone https://github.com/Finnbmb/Projekt1.git  
cd Projekt1

### **2. Java & Maven prüfen**

java -version # Sollte Java 17+ anzeigen  
mvn -version # Sollte Maven 3.8+ anzeigen

### **3. Backend starten**

mvn spring-boot:run

### **4. Frontend starten** (Neues Terminal)

cd frontend  
npm install  
npm start

### **5. Browser öffnen**

# Frontend (Hauptanwendung)  
http://localhost:4200  
  
# Backend APIs + Admin Tools  
http://localhost:8080

### **6. Fertig! 🎉**

#### **🎯 Hauptanwendung (Angular Frontend)**

http://localhost:4200

* **📅 Kalenderansicht** - Interaktive Termin-Übersicht
* **📝 Terminverwaltung** - CRUD-Operationen für Appointments
* **� Benutzer-Login** - JWT-basierte Authentifizierung
* **📱 Responsive Design** - Mobile & Desktop optimiert

#### **🔧 Backend & Admin-Tools**

http://localhost:8080

* **🛠️ Admin-Dashboard**: /admin-dashboard.html
* **📊 System-Monitoring**: /system-monitoring.html
* **🗄️ API-Dokumentation**: /debug-interface.html
* **👥 User-Management**: /user-management.html
* **🎄 Feiertage-Viewer**: /holiday-viewer.html

**💡 Wichtig**: Die Hauptanwendung läuft unter **Port 4200** (Angular), die Admin-Tools unter **Port 8080** (Spring Boot).

**Hinweis**: Beim ersten Start wird automatisch die H2-Datenbank initialisiert und deutsche Feiertage für 2024-2030 generiert.

## 🔧 Detaillierte Installation

### **Schritt 1: Voraussetzungen installieren**

#### **Java JDK 17+ installieren**

**Windows:**

# Option 1: Mit Chocolatey  
choco install openjdk17  
  
# Option 2: Manual Download  
# Besuchen Sie: https://adoptium.net/temurin/releases/  
# Laden Sie JDK 17 für Windows herunter und installieren Sie es

**macOS:**

# Mit Homebrew  
brew install openjdk@17  
  
# Pfad setzen  
echo 'export PATH="/opt/homebrew/opt/openjdk@17/bin:$PATH"' >> ~/.zshrc  
source ~/.zshrc

**Linux (Ubuntu/Debian):**

sudo apt update  
sudo apt install openjdk-17-jdk

#### **Maven installieren**

**Windows:**

# Mit Chocolatey  
choco install maven  
  
# Oder manuell von: https://maven.apache.org/download.cgi

**macOS:**

brew install maven

**Linux:**

sudo apt install maven

### **Schritt 2: Projekt einrichten**

#### **Repository klonen**

git clone https://github.com/Finnbmb/Projekt1.git  
cd Projekt1

#### **Abhängigkeiten herunterladen**

mvn clean install

#### **Tests ausführen (optional)**

mvn test

### **Schritt 3: Node.js & Angular CLI installieren**

#### **Node.js installieren**

**Windows:**

# Option 1: Download von nodejs.org  
# Besuchen Sie: https://nodejs.org/  
# Laden Sie die LTS-Version herunter  
  
# Option 2: Mit Chocolatey  
choco install nodejs  
  
# Option 3: Mit winget  
winget install OpenJS.NodeJS

**macOS:**

# Mit Homebrew  
brew install node  
  
# Oder mit MacPorts  
sudo port install nodejs18

**Linux (Ubuntu/Debian):**

# NodeSource Repository  
curl -fsSL https://deb.nodesource.com/setup\_20.x | sudo -E bash -  
sudo apt-get install -y nodejs  
  
# Oder mit snap  
sudo snap install node --classic

#### **Angular CLI global installieren**

npm install -g @angular/cli  
  
# Version prüfen  
ng version

### **Schritt 4: Frontend Setup**

#### **Frontend-Abhängigkeiten installieren**

cd frontend  
npm install

#### **Frontend im Entwicklungsmodus starten**

npm start  
# Oder alternativ:  
ng serve

**Frontend läuft dann unter: http://localhost:4200**

### **Schritt 5: Backend starten**

#### **Backend (Spring Boot) starten**

# Im Projekt-Root-Verzeichnis  
mvn spring-boot:run

**Backend-APIs laufen unter: http://localhost:8080**

#### **Oder: JAR-Datei erstellen und ausführen**

mvn clean package  
java -jar target/terminkalender-1.0.0.jar

### **Schritt 4: Funktionstest**

1. **Browser öffnen**: http://localhost:8080
2. **Registrierung testen**: Neuen Benutzer anlegen
3. **Login testen**: Mit neuem Benutzer anmelden
4. **Termine testen**: Ersten Termin erstellen
5. **Feiertage prüfen**: Kalender auf deutsche Feiertage prüfen

## ⚙️ Konfiguration

### **Konfigurationsdateien**

Das Projekt verwendet mehrere Konfigurationsdateien:

src/main/resources/  
├── application.yml # Hauptkonfiguration  
├── application-dev.properties.backup # Entwicklung (Backup)  
└── application-prod.properties # Produktion (Azure MySQL)

### **Umgebungsprofile**

#### **Entwicklung (Standard)**

# application.yml  
spring:  
 profiles:  
 active: dev # H2 Database für Entwicklung

#### **Produktion**

# application.yml  
spring:  
 profiles:  
 active: prod # Azure MySQL für Produktion

### **Wichtige Konfigurationsparameter**

#### **Server-Konfiguration**

server:  
 port: 8080 # HTTP-Port  
 servlet:  
 context-path: / # Context-Pfad

#### **Database-Konfiguration**

# H2 (Entwicklung)  
spring:  
 datasource:  
 url: jdbc:h2:file:./data/terminkalender  
 driver-class-name: org.h2.Driver  
   
# MySQL (Produktion) - in application-prod.properties  
spring:  
 datasource:  
 url: jdbc:mysql://your-server:3306/terminkalender  
 username: ${DB\_USERNAME:admin}  
 password: ${DB\_PASSWORD:password}

#### **Actuator-Konfiguration**

management:  
 endpoints:  
 web:  
 exposure:  
 include: health,info,metrics,env  
 endpoint:  
 health:  
 show-details: always

## 🗄️ Datenbank-Setup

### **Option 1: H2-Database (Entwicklung)**

**Automatische Konfiguration - keine weiteren Schritte erforderlich!**

* **Speicherort**: ./data/terminkalender.mv.db
* **Web-Console**: http://localhost:8080/h2-console
* **JDBC URL**: jdbc:h2:file:./data/terminkalender
* **Username**: sa
* **Password**: *(leer)*

### **Option 2: Azure MySQL (Produktion)**

#### **Voraussetzungen**

* Azure-Account
* Azure MySQL Flexible Server

#### **Azure MySQL Server erstellen**

# Azure CLI verwenden  
az mysql flexible-server create \  
 --resource-group myResourceGroup \  
 --name terminkalender-mysql \  
 --admin-user dbadmin \  
 --admin-password 'SecurePassword123!' \  
 --sku-name Standard\_B1ms \  
 --tier Burstable \  
 --public-access 0.0.0.0 \  
 --storage-size 20 \  
 --version 8.0

#### **Datenbank erstellen**

CREATE DATABASE terminkalender   
CHARACTER SET utf8mb4   
COLLATE utf8mb4\_unicode\_ci;

#### **Konfiguration anpassen**

# application-prod.properties  
spring.datasource.url=jdbc:mysql://terminkalender-mysql.mysql.database.azure.com:3306/terminkalender?useSSL=true&requireSSL=false&serverTimezone=Europe/Berlin  
spring.datasource.username=dbadmin  
spring.datasource.password=SecurePassword123!

#### **Mit Azure MySQL starten**

mvn spring-boot:run -Dspring.profiles.active=prod

### **Option 3: Lokale MySQL (Alternative)**

#### **MySQL installieren**

# Ubuntu/Debian  
sudo apt install mysql-server  
  
# macOS  
brew install mysql  
  
# Windows  
# MySQL Installer von https://dev.mysql.com/downloads/installer/

#### **Datenbank einrichten**

CREATE DATABASE terminkalender;  
CREATE USER 'terminalender'@'localhost' IDENTIFIED BY 'password';  
GRANT ALL PRIVILEGES ON terminkalender.\* TO 'terminalender'@'localhost';  
FLUSH PRIVILEGES;

## 🚀 Erste Schritte

### **1. Anwendung starten**

mvn spring-boot:run

**Erfolgreiche Ausgabe:**

. \_\_\_\_ \_ \_\_ \_ \_  
 /\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \  
( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  
 \\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  
 ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /  
 =========|\_|==============|\_\_\_/=/\_/\_/\_/  
 :: Spring Boot :: (v3.1.0)  
  
2025-09-26 10:30:00.000 INFO --- [main] TerminkalenderApplication : Started TerminkalenderApplication in 5.234 seconds

### **2. Erste Benutzer-Registrierung**

1. **Browser öffnen**: http://localhost:8080/login.html
2. **“Registrieren” klicken**
3. **Daten eingeben**:
   * Name: Max Mustermann
   * E-Mail: max@example.com
   * Passwort: password123
4. **“Registrieren” klicken**

### **3. Erste Termine erstellen**

1. **Nach Login**: Hauptkalender öffnet sich automatisch
2. **“+ Neuer Termin” klicken**
3. **Termin-Details eingeben**:
   * Titel: Wichtiger Termin
   * Beschreibung: Mein erster Termin
   * Startzeit: Heute + 1 Stunde
   * Endzeit: Heute + 2 Stunden
   * Kategorie: Arbeit
   * Priorität: Hoch
4. **“Speichern” klicken**

### **4. Deutsche Feiertage prüfen**

1. **Kalender-Ansicht**: Feiertage sind automatisch sichtbar
2. **Feiertag-Viewer**: http://localhost:8080/holiday-viewer.html
3. **Jahr auswählen**: 2025, 2026, etc.
4. **Bundesland filtern**: Optional spezifisches Bundesland

## 👨‍💼 Admin-Funktionen

### **Admin-Dashboard aufrufen**

http://localhost:8080/admin-dashboard.html

### **Verfügbare Admin-Tools**

#### **1. Benutzer-Verwaltung**

* **URL**: http://localhost:8080/user-management.html
* **Funktionen**:
  + Alle Benutzer anzeigen
  + Benutzer-Statistiken
  + Benutzer löschen (mit Terminen)
  + Live-Aktualisierung

#### **2. Datenbank-Verwaltung**

* **Azure DB Viewer**: http://localhost:8080/azure-database-viewer.html
* **Tabellen-Ansicht**: http://localhost:8080/database/view
* **Users API**: http://localhost:8080/database/api/users
* **Appointments API**: <http://localhost:8080/database/api/appointments>

#### **3. Feiertag-System**

* **Holiday Viewer**: http://localhost:8080/holiday-viewer.html
* **Initialisierung**: Über Admin-Dashboard
* **API-Tests**: Über Debug-Interface

#### **4. System-Monitoring**

* **Monitor-Dashboard**: http://localhost:8080/system-monitoring.html
* **System-Konfiguration**: http://localhost:8080/system-config.html
* **Health-Check**: http://localhost:8080/actuator/health
* **Metriken**: http://localhost:8080/actuator/metrics

#### **5. Debug-Tools**

* **Debug-Interface**: http://localhost:8080/debug-interface.html
* **Appointment-Tests**: http://localhost:8080/appointment-test.html
* **Database-Tools**: http://localhost:8080/database-tools.html

## 📊 Monitoring & Wartung

### **System-Gesundheit überwachen**

#### **Health-Check-Endpunkte**

# Basis-Gesundheit  
curl http://localhost:8080/actuator/health  
  
# Detaillierte Gesundheit  
curl http://localhost:8080/actuator/health/db  
  
# System-Metriken  
curl http://localhost:8080/actuator/metrics

#### **Wichtige Metriken**

* **JVM Memory**: Heap/Non-Heap Speicherverbrauch
* **System CPU**: CPU-Auslastung
* **HTTP Requests**: Request-Rate und Response-Zeit
* **Database**: Connection-Status
* **Uptime**: System-Laufzeit

### **Log-Überwachung**

#### **Log-Level konfigurieren**

# application.yml  
logging:  
 level:  
 de.swtp1.terminkalender: DEBUG # Für detaillierte Logs  
 org.springframework.security: INFO  
 org.hibernate.SQL: DEBUG # Für SQL-Queries

#### **Log-Dateien**

# Standard-Output  
mvn spring-boot:run  
  
# In Datei umleiten  
mvn spring-boot:run > application.log 2>&1  
  
# Mit Log-Rotation (Produktion)  
mvn spring-boot:run | tee -a logs/terminkalender-$(date +%Y%m%d).log

### **Backup-Strategien**

#### **H2-Database Backup**

# Datei-basierte H2-Database  
cp ./data/terminkalender.mv.db ./backups/terminkalender-$(date +%Y%m%d).mv.db

#### **MySQL Backup**

# MySQL Dump  
mysqldump -h your-server -u username -p terminkalender > backup-$(date +%Y%m%d).sql  
  
# Azure MySQL Backup  
az mysql flexible-server backup create \  
 --resource-group myResourceGroup \  
 --server-name terminkalender-mysql \  
 --backup-name manual-backup-$(date +%Y%m%d)

## 🔧 Troubleshooting

### **Häufige Probleme und Lösungen**

#### **Problem: Port 8080 bereits belegt**

# Port-Nutzung prüfen  
netstat -tulpn | grep :8080  
  
# Alternativen Port verwenden  
mvn spring-boot:run -Dserver.port=8081  
  
# Oder in application.yml  
server:  
 port: 8081

#### **Problem: Java-Version nicht kompatibel**

# Java-Version prüfen  
java -version  
  
# JAVA\_HOME setzen (Windows)  
set JAVA\_HOME=C:\Program Files\Java\jdk-17  
  
# JAVA\_HOME setzen (Linux/macOS)  
export JAVA\_HOME=/usr/lib/jvm/java-17-openjdk

#### **Problem: Maven-Build schlägt fehl**

# Dependencies aktualisieren  
mvn clean install -U  
  
# Ohne Tests bauen  
mvn clean install -DskipTests  
  
# Maven-Cache löschen  
rm -rf ~/.m2/repository/de/swtp1  
mvn clean install

#### **Problem: Database-Connection-Fehler**

# H2-Console prüfen  
http://localhost:8080/h2-console  
  
# MySQL-Connection testen  
mysql -h hostname -u username -p terminkalender  
  
# Azure MySQL Connection testen  
mysql -h server.mysql.database.azure.com -u username -p terminkalender

#### **Problem: Feiertage werden nicht angezeigt**

# Feiertag-Initialisierung prüfen  
curl http://localhost:8080/api/v1/holidays/year/2025  
  
# Über Admin-Dashboard neu initialisieren  
# http://localhost:8080/admin-dashboard.html -> Feiertag-System -> Initialisieren

### **Debug-Modi aktivieren**

#### **Debug-Logging**

# application.yml  
logging:  
 level:  
 de.swtp1.terminkalender: DEBUG  
 org.springframework.web: DEBUG  
 org.hibernate.SQL: DEBUG  
 org.hibernate.type.descriptor.sql.BasicBinder: TRACE

#### **Development-Profile**

# Mit Debug-Profil starten  
mvn spring-boot:run -Dspring.profiles.active=dev -Ddebug=true  
  
# Remote-Debugging aktivieren  
mvn spring-boot:run -Dspring-boot.run.jvmArguments="-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=n,address=5005"

## 🏭 Produktionsbereitstellung

### **Vorbereitung für Produktion**

#### **1. Produktions-JAR erstellen**

# Clean Build  
mvn clean package -DskipTests  
  
# Mit Tests  
mvn clean install  
  
# JAR-Datei prüfen  
ls -la target/terminkalender-\*.jar

#### **2. Produktions-Konfiguration**

# application-prod.properties  
spring.profiles.active=prod  
spring.datasource.url=jdbc:mysql://production-server:3306/terminkalender  
server.port=8080  
management.endpoints.web.exposure.include=health,info,metrics  
logging.level.de.swtp1.terminkalender=INFO

#### **3. Umgebungsvariablen setzen**

# Linux/macOS  
export DB\_USERNAME=produser  
export DB\_PASSWORD=securepassword  
export JWT\_SECRET=your-production-jwt-secret  
  
# Windows  
set DB\_USERNAME=produser  
set DB\_PASSWORD=securepassword  
set JWT\_SECRET=your-production-jwt-secret

## 🎨 Frontend-Entwicklung (Angular)

### **📁 Frontend-Struktur**

frontend/  
├── src/  
│ ├── app/  
│ │ ├── components/ # Wiederverwendbare Komponenten  
│ │ ├── services/ # API-Services & Business Logic  
│ │ ├── models/ # TypeScript-Interfaces  
│ │ ├── guards/ # Route Guards  
│ │ ├── interceptors/ # HTTP-Interceptors  
│ │ ├── appointments/ # Terminverwaltung  
│ │ ├── calendar-view/ # Kalenderansicht  
│ │ └── dashboard/ # Dashboard  
│ ├── environments/ # Umgebungskonfiguration  
│ └── assets/ # Statische Dateien  
├── angular.json # Angular CLI Konfiguration  
├── package.json # NPM-Abhängigkeiten  
└── tsconfig.json # TypeScript-Konfiguration

### **🔧 Frontend-Entwicklung**

#### **Development Server starten**

cd frontend  
npm start  
  
# Oder mit spezifischen Optionen:  
ng serve --host 0.0.0.0 --port 4200 --open

#### **Production Build erstellen**

npm run build  
  
# Build-Artefakte in: frontend/dist/

#### **Frontend-Tests ausführen**

# Unit Tests  
npm test  
  
# E2E Tests   
npm run e2e  
  
# Code Coverage  
npm run test:coverage

### **🌐 Frontend-Features**

#### **🎯 Hauptfunktionen**

* **📅 Interaktive Kalenderansicht** (Monat/Woche/Tag)
* **📝 Terminverwaltung** mit CRUD-Operationen
* **🔐 JWT-basierte Authentifizierung**
* **📱 Responsive Design** (Mobile-First)
* **🎨 Material Design** UI-Komponenten
* **🔔 Real-time Notifications**

#### **🏗️ Architektur-Pattern**

* **Standalone Components** (Angular 17+)
* **Signal-basiertes State Management**
* **Service-orientierte Architektur**
* **Reactive Forms** mit Validation
* **HTTP-Interceptors** für API-Kommunikation

#### **📡 API-Integration**

// Beispiel: Appointment Service  
@Injectable({  
 providedIn: 'root'  
})  
export class AppointmentService {  
 private apiUrl = `${environment.apiUrl}/appointments`;  
  
 getAppointments(): Observable<Appointment[]> {  
 return this.http.get<Appointment[]>(this.apiUrl);  
 }  
  
 createAppointment(appointment: AppointmentRequest): Observable<Appointment> {  
 return this.http.post<Appointment>(this.apiUrl, appointment);  
 }  
}

### **⚙️ Frontend-Konfiguration**

#### **Environment-Konfiguration**

// src/environments/environment.ts  
export const environment = {  
 production: false,  
 apiUrl: 'http://localhost:8080/api/v1',  
 features: {  
 enableDebugMode: true,  
 enableAnalytics: false  
 }  
};  
  
// src/environments/environment.prod.ts  
export const environment = {  
 production: true,  
 apiUrl: 'https://your-domain.com/api/v1',  
 features: {  
 enableDebugMode: false,  
 enableAnalytics: true  
 }  
};

#### **Proxy-Konfiguration für Development**

// proxy.conf.json  
{  
 "/api/\*": {  
 "target": "http://localhost:8080",  
 "secure": true,  
 "changeOrigin": true,  
 "logLevel": "debug"  
 }  
}

# Mit Proxy starten  
ng serve --proxy-config proxy.conf.json

### **🎭 UI/UX Features**

#### **Material Design Integration**

// Angular Material Modules  
import { MatButtonModule } from '@angular/material/button';  
import { MatCardModule } from '@angular/material/card';  
import { MatDatepickerModule } from '@angular/material/datepicker';  
import { MatFormFieldModule } from '@angular/material/form-field';  
import { MatIconModule } from '@angular/material/icon';

#### **Responsive Design**

// Mobile-First Approach  
.appointment-grid {  
 display: grid;  
 gap: 1rem;  
   
 // Mobile (default)  
 grid-template-columns: 1fr;  
   
 // Tablet  
 @media (min-width: 768px) {  
 grid-template-columns: repeat(2, 1fr);  
 }  
   
 // Desktop  
 @media (min-width: 1024px) {  
 grid-template-columns: repeat(3, 1fr);  
 }  
}

### **🔄 Frontend-Backend Kommunikation**

#### **HTTP-Client Setup**

// app.config.ts  
export const appConfig: ApplicationConfig = {  
 providers: [  
 provideHttpClient(withInterceptors([authInterceptor])),  
 // ... andere Provider  
 ]  
};

#### **Authentication Interceptor**

// auth.interceptor.ts  
export const authInterceptor: HttpInterceptorFn = (req, next) => {  
 const token = localStorage.getItem('authToken');  
   
 if (token) {  
 req = req.clone({  
 setHeaders: {  
 Authorization: `Bearer ${token}`  
 }  
 });  
 }  
   
 return next(req);  
};

### **📱 PWA-Unterstützung (Optional)**

#### **Service Worker aktivieren**

ng add @angular/pwa

#### **PWA-Features**

* **📱 Installierbare Web-App**
* **🔄 Offline-Funktionalität**
* **🔔 Push-Notifications**
* **📊 App-Shell-Architektur**

### **🐛 Frontend-Debugging**

#### **Development Tools**

# Angular DevTools (Browser Extension)  
# Redux DevTools für State Management  
# Lighthouse für Performance-Analyse  
  
# Debug-Modus aktivieren  
ng serve --source-map --verbose

#### **Error Handling**

// Global Error Handler  
@Injectable()  
export class GlobalErrorHandler implements ErrorHandler {  
 handleError(error: any): void {  
 console.error('Global error:', error);  
 // Sentry, LogRocket, etc. Integration  
 }  
}

## 🚀 Deployment-Optionen

### **Deployment-Optionen**

#### **Option 1: Direkter JAR-Start**

# Produktions-JAR starten  
java -Xmx2g -Xms1g -jar terminkalender-1.0.0.jar --spring.profiles.active=prod  
  
# Mit zusätzlichen JVM-Parametern  
java -Xmx2g -Xms1g \  
 -Dserver.port=8080 \  
 -Dspring.profiles.active=prod \  
 -jar terminkalender-1.0.0.jar

#### **Option 2: Systemd-Service (Linux)**

# /etc/systemd/system/terminkalender.service  
[Unit]  
Description=Terminkalender Application  
After=network.target  
  
[Service]  
Type=simple  
User=terminkalender  
WorkingDirectory=/opt/terminkalender  
ExecStart=/usr/bin/java -Xmx2g -jar terminkalender-1.0.0.jar --spring.profiles.active=prod  
Restart=always  
RestartSec=10  
  
Environment=DB\_USERNAME=produser  
Environment=DB\_PASSWORD=securepassword  
  
[Install]  
WantedBy=multi-user.target

# Service aktivieren  
sudo systemctl enable terminkalender  
sudo systemctl start terminkalender  
sudo systemctl status terminkalender

#### **Option 3: Docker-Container**

# Dockerfile  
FROM openjdk:17-jre-slim  
  
WORKDIR /app  
COPY target/terminkalender-\*.jar app.jar  
  
EXPOSE 8080  
  
ENV SPRING\_PROFILES\_ACTIVE=prod  
ENV DB\_USERNAME=produser  
ENV DB\_PASSWORD=securepassword  
  
CMD ["java", "-Xmx2g", "-jar", "app.jar"]

# Docker-Image bauen  
docker build -t terminkalender:1.0.0 .  
  
# Container starten  
docker run -d \  
 --name terminkalender \  
 -p 8080:8080 \  
 -e DB\_USERNAME=produser \  
 -e DB\_PASSWORD=securepassword \  
 terminkalender:1.0.0

### **Produktions-Monitoring**

#### **Health-Check Setup**

# Automatischer Health-Check  
curl -f http://localhost:8080/actuator/health || exit 1  
  
# Cron-Job für Health-Check (Linux)  
# \*/5 \* \* \* \* curl -f http://localhost:8080/actuator/health > /dev/null 2>&1 || echo "Terminkalender health check failed" | mail -s "Alert" admin@company.com

#### **Log-Monitoring**

# Structured Logging  
java -jar terminkalender-1.0.0.jar \  
 --logging.pattern.console="%d{yyyy-MM-dd HH:mm:ss} - %logger{36} - %level - %msg%n" \  
 --logging.file.name=logs/terminkalender.log  
  
# Log-Rotation mit logrotate  
# /etc/logrotate.d/terminkalender  
/opt/terminkalender/logs/\*.log {  
 daily  
 rotate 30  
 compress  
 missingok  
 notifempty  
 create 644 terminkalender terminkalender  
}

## 📞 Support & Weiterführende Informationen

### **Dokumentation**

* **API-Spezifikation**: docs/specifications/api-specification.md
* **Architektur-Dokumentation**: docs/architecture/arc42-architecture.md
* **Requirements**: docs/specifications/requirements-specification.md

### **Nützliche URLs (bei laufender Anwendung)**

* **Hauptanwendung**: http://localhost:8080
* **Login**: http://localhost:8080/login.html
* **Admin-Dashboard**: http://localhost:8080/admin-dashboard.html
* **System-Monitoring**: http://localhost:8080/system-monitoring.html
* **Health-Check**: http://localhost:8080/actuator/health
* **H2-Console**: http://localhost:8080/h2-console (nur Development)

### **Kontakt & Support**

* **GitHub Repository**: https://github.com/Finnbmb/Projekt1
* **Issue-Tracking**: GitHub Issues
* **Wiki**: GitHub Wiki
* **Discussions**: GitHub Discussions

## ✅ Checkliste: Installation erfolgreich

* Java 17+ installiert und funktionsfähig
* Maven 3.8+ installiert und funktionsfähig
* Repository erfolgreich geklont
* Dependencies heruntergeladen (mvn clean install)
* Anwendung startet ohne Fehler (mvn spring-boot:run)
* Browser zeigt Login-Seite (http://localhost:8080)
* Benutzer-Registrierung funktioniert
* Erster Termin erstellt
* Deutsche Feiertage sichtbar im Kalender
* Admin-Dashboard erreichbar
* System-Monitoring zeigt Metriken

## 🎉 Herzlichen Glückwunsch!

Ihre **Terminkalender-Anwendung** ist jetzt erfolgreich installiert und betriebsbereit!

### **Nächste Schritte:**

1. **Erkunden** Sie alle Features über das Admin-Dashboard
2. **Erstellen** Sie Ihre ersten Termine und Nutzer
3. **Überwachen** Sie das System über das Monitoring-Dashboard
4. **Backup** Ihre Datenbank regelmäßig
5. **Aktualisieren** Sie die Konfiguration nach Ihren Bedürfnissen

**Viel Erfolg mit Ihrer neuen Terminkalender-Anwendung!** 🚀

**Hinweis**: Diese Anleitung wird kontinuierlich aktualisiert. Für die neueste Version besuchen Sie das GitHub-Repository.