

Studienprojekt

Externe Konfiguration von clientseitigen Frontend-Technologien in Angular und React

im Studiengang
Softwaretechnik und Medieninformatik

vorgelegt von

Philipp Schlosser
Matrikel-Nr.: 765167

am 17. Juni 2025
an der Hochschule Esslingen

| | |
|--------------|------------------------------|
| Erstprüfer: | Prof. Dr.-Ing. Rainer Keller |
| Zweitprüfer: | Matthias Häussler |

Inhaltsverzeichnis

| | |
|---|----|
| Abbildungsverzeichnis..... | 1 |
| Tabellenverzeichnis..... | 2 |
| Abkürzungsverzeichnis..... | 3 |
| 1 Kurzfassung..... | 4 |
| 2 Einleitung | 5 |
| 3 Aufgabenstellung | 6 |
| 4 Technologieauswahl | 7 |
| 4.1 Verwendete Technologien | 7 |
| 4.2 Systemübersicht | 8 |
| 4.3 Vorteile der Architektur..... | 8 |
| 4.4 Nachteile der Architektur | 9 |
| 4.5 Verwendete Frameworks..... | 9 |
| 4.5.1 Angular | 9 |
| 4.5.2 React..... | 9 |
| 4.6 Ausgangssituation | 10 |
| 5 Umsetzung | 11 |
| 5.1 Technische Umsetzung des Frontends..... | 11 |
| 5.1.1 Zentrale Funktion: Dynamische API-Konfiguration | 11 |
| 5.1.2 Visuelles Feedback und Statusanzeige | 13 |
| 5.1.3 Optionales Feature: Dark/Bright Mode | 13 |
| 5.2 Ergebnis des UI | 14 |
| 5.3 Implementierung des Backends | 15 |
| 5.4 Initialisierung der Datenbank..... | 16 |
| 6 Fazit und Ausblick..... | 17 |
| Eidesstattliche Erklärung | 18 |
| Literaturverzeichnis | 19 |
| Anhang | 20 |

Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 1 - UI der ursprünglichen Laborübung | 10 |
| Abbildung 2 - Ordnerstruktur | 11 |
| Abbildung 3 - Code Ausschnitt URL handling..... | 12 |
| Abbildung 4 - Code Ausschnitt Dark Mode..... | 13 |
| Abbildung 5 - React Umgebung, Dark Mode | 14 |
| Abbildung 6 - Angular Umgebung, Bright Mode | 14 |
| Abbildung 7 - Datenmodell „Item“ | 15 |
| Abbildung 8 - SQL Skript..... | 16 |

Tabellenverzeichnis

| | |
|--|-----------|
| Tabelle 1 - Anforderungen..... | 6 |
| Tabelle 2 - Technologien..... | 7 |
| Tabelle 3 - API Endpunkte | 20 |
| Tabelle 4 - Portübersicht | 20 |

Abkürzungsverzeichnis

| | |
|-------------|-----------------------------------|
| API | Application Programming Interface |
| CD | Continuous Delivery |
| CI | Continuous Integration |
| CLI | Command Line Interface |
| CRUD | Create, Read, Update and Delete |
| HTTP | Hypertext Transfer Protocol |
| JSON | JavaScript Object Notation |
| REST | Representational State Transfer |
| UI | User Interface |
| URL | Uniform Resource Locator |
| UX | User Experience |
| SPA | Single Page Application |

1 Kurzfassung

Dieses Projekt entstand aus der Laborübung zur Vorlesung „Parallele und Verteilte Systeme“. Ausgangspunkt war das Problem, dass die Frontends im Labor fest mit einem statischen Backend verbunden waren, was Änderungen und parallele Entwicklungen erschwerte.

Ziel des Projekts ist es, zwei Webanwendungen, basierend auf den Frameworks Angular und React, so zu gestalten, dass sie zur Laufzeit dynamisch zwischen mehreren Backend-Instanzen wechseln können. Dies wurde erreicht, indem die API-Endpunkte über eine Konfigurationsdatei (z.B. *config.json*¹) oder benutzerseitige Eingaben zur Laufzeit konfigurierbar sind.

Durch diese Erweiterung können die Frontends flexibel mit unterschiedlichen Backends kommunizieren ohne, dass ein Rebuild erforderlich ist (z. B. für Entwicklungs-, Test- oder Produktionszwecke). Wie im Labor üblich, wurde für jedes Systemkomponentenmodul (Backend, Frontend und Datenbank) ein eigenes *Dockerfile* erstellt. Zusätzlich erfolgte die Bereitstellung über *Kubernetes*² mittels entsprechender Konfigurationsdateien. Die Lösung unterstützt dadurch parallele Umgebungen und steigert die Skalierbarkeit sowie Wartbarkeit des Gesamtsystems.

¹ Die Datei *config.json* wird genutzt, um Konfigurationsdaten wie API-URLs unabhängig vom Quellcode zur Laufzeit bereitzustellen. Vgl. Fowler o. J.

² Open-Source-Orchestrierungswerkzeug zur automatisierten Verwaltung, Skalierung und Verteilung containerisierter Anwendungen. Vgl. Hightower/Burns/Beda 2019.

2 Einleitung

Im Rahmen der Vorlesung „Parallele und Verteilte Systeme“ entstand dieses Projekt aus einer Laborübung, in der eine einfache Webanwendung mit einem statisch angebundenen Backend realisiert wurde. Dabei war die Verbindung zwischen Frontend und Backend fest im Code verankert, was für moderne, dynamische Anwendungen nur eingeschränkt praktikabel ist. Ziel der Übung war es, ein grundlegendes Verständnis für die Kommunikation zwischen verteilten Komponenten zu vermitteln und erste Erfahrungen im Umgang mit client-server-basierten Architekturen zu sammeln.

Im Projekt kamen unter anderem Technologien wie z. B. React, Node.js, REST-API, Docker und Kubernetes zum Einsatz. Die Wahl einer festen Backend-Anbindung diene der Vereinfachung, spiegelt jedoch nicht die Flexibilität wider, die in heutigen skalierbaren Systemen gefordert ist. Diese Dokumentation beschreibt den Aufbau und die Umsetzung der Anwendung inkl. der umzusetzenden Anforderungen.

Alle verwendeten Abkürzungen sind im Abkürzungsverzeichnis am Anfang der Dokumentation aufgeführt.

3 Aufgabenstellung

Ziel dieses Projekts war es daher, ein System zu entwerfen und umzusetzen, indem die Frontend-Anwendungen zur Laufzeit flexibel mit verschiedenen Backend-Instanzen verbunden werden können, ohne ein erneutes Kompilieren oder Deployment. Dies bildet den Kern für eine dynamische Konfiguration, wie sie etwa in produktiven *CI/CD*³-Umgebungen oder bei verschiedenen Test- und Entwicklungsstufen notwendig ist.

| Funktionale Anforderungen |
|--|
| Initiales Laden der API-URL aus einer externen Konfigurationsdatei |
| Die manuelle Anpassung der API-URL im ist im UI sichtbar |
| Visuelles Feedback zur aktiven Verbindung im UI |
| Framework-übergreifende Anwendbarkeit in React und Angular |

Tabelle 1 - Anforderungen

Um die funktionalen Anforderungen umzusetzen, wurden zwei Varianten des Frontends wie beschrieben implementiert. Beide greifen auf eine identische API zu, wobei der Zugriff zur Laufzeit über eine konfigurierbare URL gewährleistet wird. Im Hintergrund stehen zwei voneinander unabhängige Go-Backends bereit, die jeweils mit ihrer eigenen Datenbank (realisiert mit PostgreSQL) verbunden sind.

Das Projekt kombiniert moderne Technologien wie Docker, Kubernetes und *GitHub-Codespaces*⁴ und clientseitige Konfigurationsmechanismen, um ein realistisches Setup für parallele und verteilte Systeme im Webkontext zu demonstrieren.

³ CI/CD bezeichnet Prozesse zur kontinuierlichen Integration und Auslieferung von Software, um Qualität und Geschwindigkeit zu verbessern. Vgl. *Humble/Farley 2010*.

⁴ GitHub Codespaces ist eine cloudbasierte Entwicklungsumgebung, die direkt in GitHub integriert ist und vollständige Dev-Container für die Softwareentwicklung bereitstellt. Vgl. *GitHub Inc. (2025)*.

4 Technologieauswahl

Das Projekt entstand aus einer Laborarbeit, in welcher bereits diverse Technologien festgelegt wurden. Diese wurden ohne direkte Vorgaben ausgewählt und daher nicht mit anderen Technologien verglichen, was zu einer Vereinfachung der Umsetzung führte.

4.1 Verwendete Technologien

Folgende Technologien wurden für die Umsetzung des Projektes verwendet:

| Kategorie | Technologie | Zweck |
|--------------------------------|----------------------------|---------------------------------------|
| Frontend | React | Moderne Webentwicklung |
| | Angular | Alternative SPA ⁵ |
| Backend | Go | REST-API ⁶ |
| Datenbank | PostgreSQL | Relationale Datenspeicherung |
| Containerisierung ⁷ | Docker & Docker Compose | Aufbau der lokalen Container-Umgebung |
| Orchestrierung ⁸ | Kubernetes + Minikube | Container-Orchestrierung |
| Kommunikation | HTTP/REST | API-Zugriff der Frontends |

Tabelle 2 - Technologien

⁵ Bei einer SPA handelt es sich um eine Webanwendung, die Inhalte dynamisch nachlädt und dabei ohne vollständiges Neuladen der Seite auskommt. Vgl. Flanagan 2020.

⁶ REST ist ein Architekturstil für Webdienste, der auf HTTP basiert und auf Ressourcen zugreift, die über eindeutige URLs identifizierbar sind. Vgl. Fielding 2000.

⁷ Containerisierung beschreibt die Ausführung von Anwendungen in isolierten, portablen Containern, die alle benötigten Abhängigkeiten enthalten. Vgl. Merkel 2014.

⁸ Orchestrierung bezeichnet die automatisierte Verwaltung und Koordination mehrerer Dienste oder Container in verteilten Systemen. Vgl. Burns et al. 2016.

4.2 Systemübersicht

Das System ist so aufgebaut, dass beide Frontends vollständig unabhängig und parallel betrieben werden können. Über eine *config.json* oder per UI kann jederzeit zwischen den zwei Backends umgeschaltet werden, ohne dass eine neue Build-Version erforderlich ist.

- React- und Angular-Frontends laufen in separaten Containern und laden beim Start eine Konfigurationsdatei, die die Backend-URL angibt
- Backend A und B sind jeweils eigenständige Go-Services, die *CRUD*⁹ Funktionalitäten über REST bereitstellen
- PostgreSQL A und B speichern jeweils die Daten des zugehörigen Backends. Es gibt keine geteilte Datenbasis
- Die Komponenten kommunizieren ausschließlich über die definierten HTTP-Schnittstellen

4.3 Vorteile der Architektur

- **Dynamische Umschaltung der API zur Laufzeit**
Die Frontends können ohne Rebuild zur Laufzeit das Backend wechseln (oft in der Softwareentwicklung für Dev-/Test-/Prod-Szenarien verwendet)
- **Isolierte Backend-/Datenbankinstanzen**
Jede Backend-Instanz arbeitet mit einer eigenen PostgreSQL-Datenbank, sodass parallele Zugriffe keine Konflikte verursachen
- **Gleichzeitiger Vergleich von React und Angular möglich**
Beide Frontends greifen auf denselben Backend-Mechanismus zu und demonstrieren unterschiedliche Architekturstile
- **Skalierbar und „cloud-ready“ durch Containerisierung**
Die gesamte Anwendung ist in Docker-Containern gekapselt und lässt sich portabel ausführen. Zudem können die Container mithilfe von Kubernetes orchestriert werden.

⁹ CRUD bezeichnet grundlegende Datenbankoperationen. Der Begriff steht für Create, Read, Update und Delete. Vgl. *Elmasri/Navathe, 2015*.

4.4 Nachteile der Architektur

- **Komplexität der Infrastruktur und steile Lernkurve**
Durch den Einsatz von mehreren Frontends, Backends und Datenbanken sowie Kubernetes steigt die technische Komplexität erheblich, sowohl beim Aufbau als auch im Betrieb. Zudem steigt die Lernkurve bei Zunahme der Komplexität
- **Höherer Ressourcenverbrauch**
Zwei parallele Backends mit jeweils eigener Datenbank beanspruchen mehr Systemressourcen (RAM, CPU, Speicher) vor allem in einer lokalen Entwicklungsumgebung. Durch den Codespace ist dies jedoch zu vernachlässigen.

4.5 Verwendete Frameworks

4.5.1 Angular

Angular ist ein webbasiertes Frontend-Framework, das von Google entwickelt wird. Es basiert auf TypeScript und bietet ein umfassendes Set an Funktionen für die Entwicklung strukturierter, modularer und dynamischer SPAs mit eingebautem Routing, Formularmanagement und Datenbindung.¹⁰

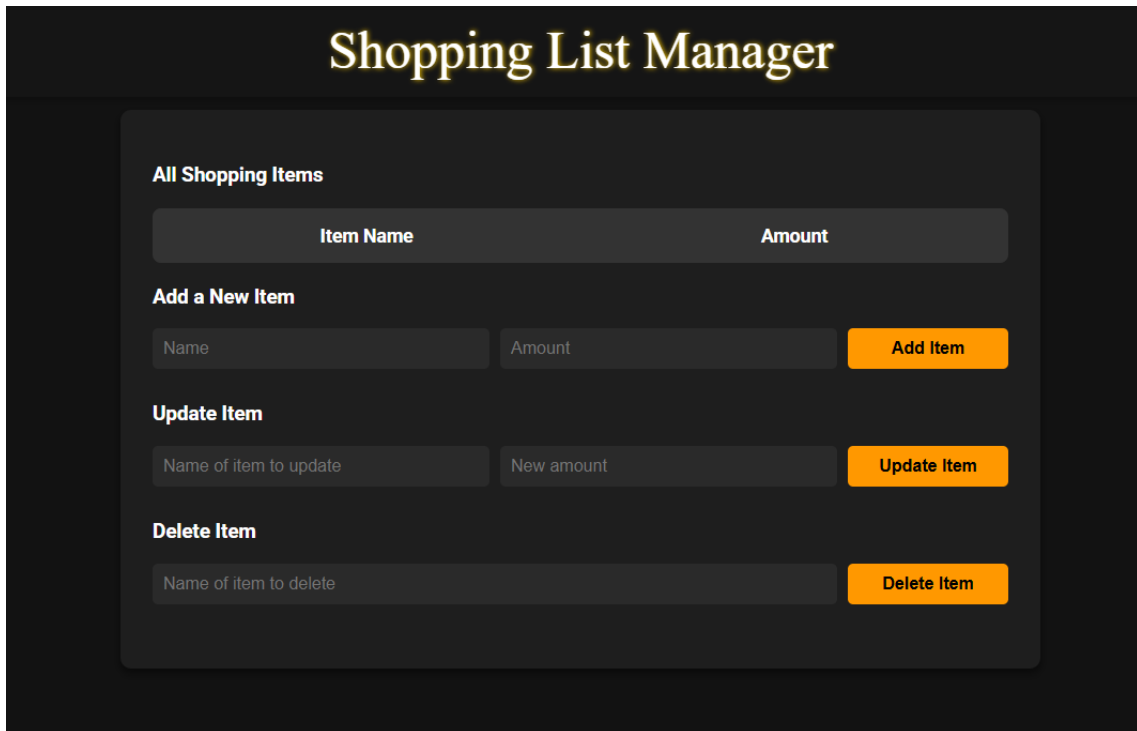
4.5.2 React

React ist eine JavaScript-Bibliothek zur Entwicklung von Benutzeroberflächen, die von Meta (ehemals Facebook) entwickelt wurde. Sie basiert auf einem komponentenbasierten Ansatz und ermöglicht die effiziente Erstellung interaktiver, einseitiger Webanwendungen.¹¹

¹⁰ Vgl. Google LLC: Angular – The modern web developer’s platform, 2024.

¹¹ Vgl. Meta Platforms, Inc.: React – A JavaScript library for building user interfaces, 2024.

4.6 Ausgangssituation



The image shows a web application titled "Shopping List Manager" with a dark theme. It features a central panel with the following sections:

- All Shopping Items:** A table with two columns: "Item Name" and "Amount".
- Add a New Item:** Two input fields labeled "Name" and "Amount", followed by an orange "Add Item" button.
- Update Item:** Two input fields labeled "Name of item to update" and "New amount", followed by an orange "Update Item" button.
- Delete Item:** One input field labeled "Name of item to delete", followed by an orange "Delete Item" button.

Abbildung 1 - UI der ursprünglichen Laborübung

Als Projektumgebung dient eine Webanwendung zur Verwaltung einer Einkaufsliste. Diese wurde mit dart.js, GO und PostgreSQL umgesetzt. Die Funktionalität wurde anschließend um Docker und Kubernetes erweitert, um deren Einsatz in verteilten Systemen zu demonstrieren und praktisch zu erlernen.

Wie in Abbildung 1 dargestellt, umfasst die Anwendung bewusst einfache Funktionen wie das Hinzufügen, Bearbeiten und Löschen von Listeneinträgen. Ein Eintrag besteht aus einem Textfeld für die Bezeichnung und einer Ganzzahl zur Angabe der Menge. Die Benutzerinteraktion erfolgt über Buttons, die entsprechende Hyper-Text-Transfer-Protocol (kurz http) - Requests an die API senden. Änderungen werden daraufhin persistent in der PostgreSQL-Datenbank gespeichert.

Durch Docker werden Frontend, Backend und Datenbank jeweils als eigenständige Container bereitgestellt. Kubernetes übernimmt dabei die Orchestrierung und das Management dieser Container und ermöglicht einen skalierbaren und modularen Betrieb.

5 Umsetzung

Zunächst wurde das Labor in ein neues GitHub-Repository geklont und die notwendigen Module via Command-Line-Interface (CLI) installiert. Unter anderem wurde die Ordner-Struktur und in den Unterordnern die Projektstruktur entsprechend ihrem Zweck angelegt.

```
> backend  
> db  
> frontend-angular  
> frontend-react  
> k8s
```

Abbildung 2 - Ordnerstruktur

5.1 Technische Umsetzung des Frontends

Im Rahmen des Projekts wurden zwei SPAs mit den Frameworks React und Angular entwickelt. Beide Anwendungen verfolgen denselben funktionalen Umfang, um einen Vergleich der Technologien zu ermöglichen.

5.1.1 Zentrale Funktion: Dynamische API-Konfiguration

Das zentrale Ziel dieses Projekts war die Möglichkeit, die API-URL der Backend-Instanz zur Laufzeit über die Benutzeroberfläche, ohne einen Rebuild des Frontends, zu ändern.

Umsetzung:

- Beim Start wird eine Konfigurationsdatei (*config.json*) aus dem *assets*-Verzeichnis geladen.
- In der Benutzeroberfläche gibt es zwei Buttons: „Backend A“ und „Backend B“. Diese setzen die API-URL zur Laufzeit (z. B. auf Port 5000 oder 5001).
- Über ein weiteres Eingabefeld kann eine individuelle URL eingegeben und angewendet werden.
- Alle API-Methoden (*GET*, *POST*, *PUT*, *DELETE*) verwenden zur Laufzeit den jeweils aktuellen Wert der API-URL.

In React erfolgt diese Steuerung über *useState()*, während in Angular ähnliche Zustände in der *AppComponent* gepflegt werden. Beide Implementierungen folgen dem Prinzip: **Trennung von Konfiguration und Funktionalität**.

Ein wesentlicher Bestandteil der Frontend-Logik besteht in der dynamischen Umwandlung der Host-URL, um unterschiedliche Backend-Instanzen anzusprechen. Anstatt feste URLs zu verwenden, wird zur Laufzeit die aktuelle Host-Adresse der Anwendung manipuliert, indem ein definierter Portabschnitt ersetzt wird.

Auszüge aus der *App.tsx* und der *app.component.ts*:

```
// React - Wechsel zu Backend A
const hostname = window.location.hostname;
const newUrl = `https://${hostname.replace('-6000', '-5000')}`;
// Angular - Wechsel zu Backend B
const host = window.location.hostname;
const targetUrl = `https://${host.replace('-6001', '-5001')}`;
```

Abbildung 3 - Code Ausschnitt URL handling

Da das Projekt in GitHub Codespaces umgesetzt wurde, wird für jede Instanz eine individuelle Subdomain generiert, inklusive eines Ports, der vom jeweiligen Container abhängt. Diese URLs sind dynamisch und ändern sich mit jedem neuen Codespace. Eine statische API-URL wäre daher unzuverlässig und würde zu Verbindungsfehlern führen.

Was wird dadurch bezweckt:

- Die Codespace-URL wird nicht fest kodiert, sondern zur Laufzeit anhand des aktuellen Frontend-Ports modifiziert. Dies funktioniert zuverlässig, solange die verwendeten Ports bekannt sind und einer festen Logik folgen.
- Die resultierende URL wird im *localStorage* ¹²gespeichert, wodurch auch nach neuem Laden die gewählte Backend-Verbindung erhalten bleibt.

Der Vorteil besteht darin, dass diese Lösung manuelle Eingaben vermeidet und es dem User erlaubt, durch einfaches Klicken ohne komplexe Konfigurationsmechanismen oder Umgebungsvariablen zwischen mehreren Backends zu wechseln.

¹² Ein Web-API-Feature, das es ermöglicht, Daten lokal im Browser des Benutzers zu speichern – persistent über Sitzungen hinweg.

5.1.2 Visuelles Feedback und Statusanzeige

Die jeweils aktive API-Verbindung wird durch den Buttontext („Aktiv“) und visuelle, grüne Hervorhebung angezeigt. Tritt ein Fehler beim Laden der Items auf, wird dieser deutlich als rote Fehlermeldung in der UI eingeblendet.

5.1.3 Optionales Feature: Dark/Bright Mode

Zusätzlich zu den essenziellen Anforderungen wurde ein *Dark-Mode-Toggle* implementiert. Dieser erlaubt es, zwischen hellem und dunklem Layout der Anwendung zu wechseln. Dabei wird der *dark*-CSS-Klassenname dynamisch auf das *<body>*-Element angewendet und mittels eines Buttons mit Icons (typischerweise Sonne/Mond) steuerbar gemacht.

```
1 // Beispiel in React
2 v const toggleDarkMode = () => {
3   document.body.classList.toggle('dark');
4   setIsDarkMode(!isDarkMode);
5 };

1 // Beispiel in Angular
2 v toggleDarkMode() {
3   document.body.classList.toggle('dark');
4   this.isDarkMode = !this.isDarkMode;
5 }
6
```

Abbildung 4 - Code Ausschnitt Dark Mode

Ein Dark Mode ergibt vor allem deshalb Sinn, weil er die Benutzerfreundlichkeit einer Anwendung deutlich erhöhen kann. Besonders in dunkler Umgebung reduziert er die Belastung für die Augen und erleichtert die Interaktion der Anwendung. Darüber hinaus wirkt ein dunkles Design oft moderner und professioneller und entspricht dem gewachsenen Nutzerbedürfnis nach Individualisierung. Viele Nutzer bevorzugen eine dunkle Oberfläche, etwa aus Gewohnheit oder ästhetischen Gründen. Auch technisch bringt er Vorteile mit sich: Auf Geräten mit OLED-Displays kann er den Energieverbrauch senken. In Summe zeigt die Integration eines Dark Modes, dass die Anwendung nicht nur funktional, sondern auch auf Nutzerkomfort und modernes Design ausgerichtet ist.

5.2 Ergebnis des UI

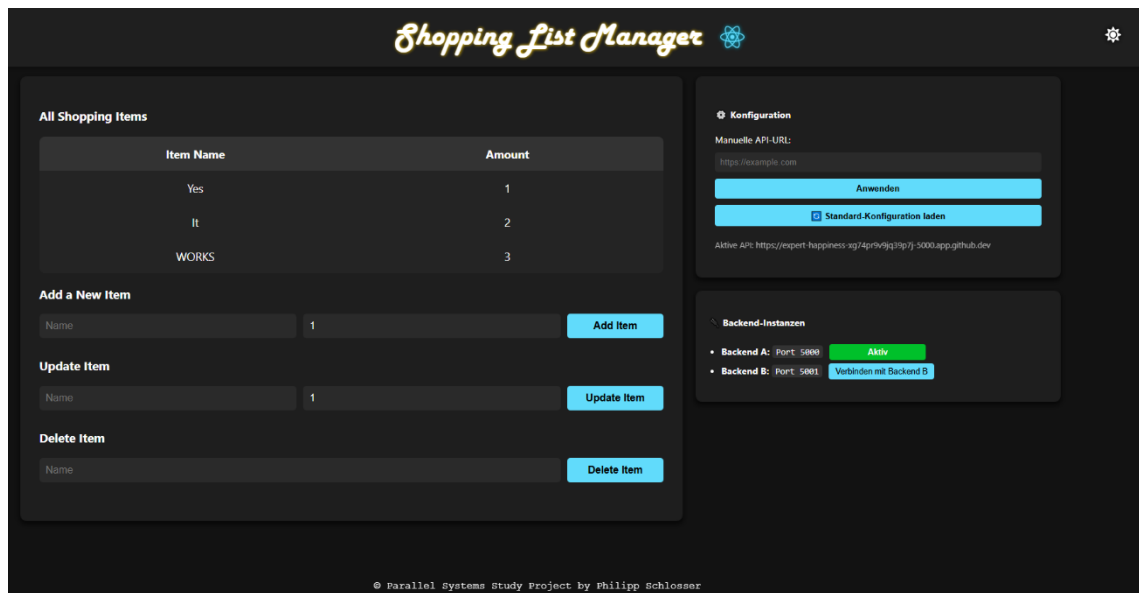


Abbildung 5 - React Umgebung, Dark Mode

Das UI wurde schlicht gehalten die Farben und das Logo sowie das Favicon auf das jeweilige Framework angepasst, um dem Benutzer direkt aufzuzeigen, auf welchem Interface er sich gerade befindet. Die obere Abbildung zeigt die Anwendung in React und im Dark Mode, während die untere Abbildung die Anwendung in Angular und im Bright Mode abbildet. Das Layout der Anwendung wurde identisch gehalten, um die Funktion so einfach wie möglich zu gestalten.

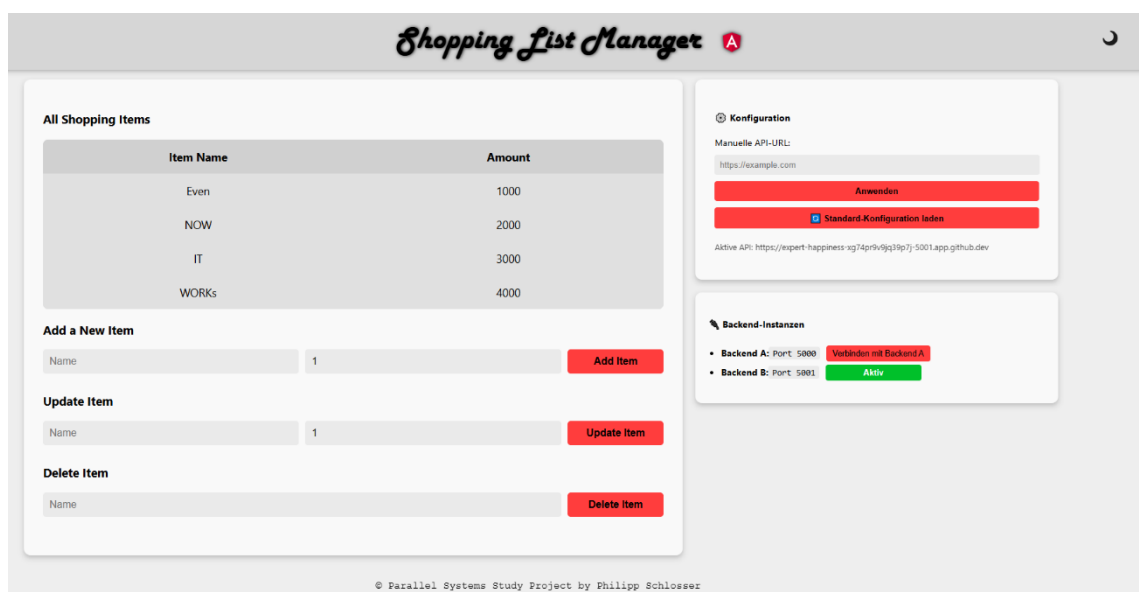


Abbildung 6 - Angular Umgebung, Bright Mode

5.3 Implementierung des Backends

Das Backend dieser Anwendung wurde in der Programmiersprache Go entwickelt und bildet die Schnittstelle zwischen den Frontends (React und Angular) und der persistenten Datenspeicherung in einer PostgreSQL-Datenbank. Es stellt eine einfache, aber robuste REST-API bereit, um CRUD-Operationen auf einer Einkaufsliste durchzuführen.

Beim Start der Go-Anwendung wird eine Verbindung zur Datenbank hergestellt. Die Verbindungsdaten (Host, Benutzername, Passwort und Datenbankname) werden aus Umgebungsvariablen gelesen, was eine flexible und containerfreundliche Konfiguration ermöglicht. Falls die Verbindung erfolgreich aufgebaut wird, prüft das Backend, ob die benötigte Tabelle (*items*) existiert und legt sie nach Bedarf automatisch an.

Das zentrale Datenmodell ist der Typ *Item*, der aus zwei Feldern besteht:

```
1  type Item struct {  
2      Name    string `json:"name"`  
3      Amount  int    `json:"amount"`  
4  }  
5
```

Abbildung 7 – Datenmodell „Item“

API-Endpunkte:

Die API stellt folgende Endpunkte bereit (Siehe Anhang):

- **GET /items:** Gibt alle gespeicherten Items als *JSON*¹³ zurück.
- **POST /items:** Fügt ein neues Item zur Datenbank hinzu.
- **PUT /items/:name:** Aktualisiert die Anzahl eines Items.
- **DELETE /items/:name:** Löscht ein Item anhand seines Namens.

Die gesamte Kommunikation erfolgt über JSON, sowohl für Ein- als auch Ausgabe.

Deployment-Fähigkeit

Da das Backend vollständig auf Umgebungsvariablen und Ports konfigurierbar ist, eignet es sich ideal für den Betrieb in Docker-Containern oder in Kubernetes. Diese Architektur erlaubt eine einfache horizontale Skalierung.

¹³ JSON steht für „JavaScript Object Notation“ und ist ein leichtgewichtiges Datenformat zur strukturierten Datenübertragung, das insbesondere in Webanwendungen zur Kommunikation zwischen Client und Server verwendet wird.

5.4 Initialisierung der Datenbank

Die Initialisierung der PostgreSQL-Datenbank erfolgt automatisch beim Start der Container durch Ausführung eines SQL-Skripts. Dieses legt zunächst eine Tabelle *items* mit den Spalten *name* (als Primärschlüssel) und *amount* an. Anschließend werden drei Beispiel-Datensätze eingefügt:

```
1 ✓ CREATE TABLE items (  
2     name TEXT PRIMARY KEY,  
3     amount INT  
4 );  
5  
6 ✓ INSERT INTO items (name, amount) VALUES  
7     ('Yes', 1),  
8     ('It', 2),  
9     ('WORKS', 3);  
10
```

Abbildung 8 - SQL Skript

Durch die automatische Initialisierung ist die Anwendung direkt nach dem Start lauffähig und demonstriert die Funktionsweise der API ohne manuelles Hinzufügen von Einträgen. Besonders in containerisierten Umgebungen wie Docker oder Kubernetes ist dies hilfreich für reproduzierbare Setups und Testbarkeit.

6 Fazit und Ausblick

Das Projekt zeigt, wie moderne Webanwendungen durch dynamische Konfiguration deutlich flexibler gestaltet werden können. Statt fest verdrahteter Verbindungen können Frontends nun zur Laufzeit mit verschiedenen Backends kommunizieren. Dies erleichtert insbesondere Testszenarien und den parallelen Betrieb von Entwicklungs- und Produktivumgebungen. Die Verwendung von Docker und Kubernetes steigert zusätzlich die Skalierbarkeit und Wartbarkeit des Gesamtsystems. Insgesamt konnte so ein praxisnahes, zukunftsfähiges Architekturkonzept realisiert werden.

Das System könnte um folgende Features und Optimierungen ergänzt werden:

- **Authentifizierung & Autorisierung:** Aktuell ist der Zugriff auf die API-Endpunkte „offen“. In einem produktiven Umfeld könnte eine Zugriffskontrolle über *Keycloak*¹⁴ integriert werden, um ggfs. sensible Daten zu schützen.
- **UI-Optimierung & UX:** Die bereits integrierte Visualisierung des Backend sowie der Dark/Bright Mode könnten weiter verfeinert werden. (z. B. durch animierte Übergänge)
- **Multi-Tenant-Unterstützung:** Durch die Trennung von Datenbanken ist die Grundlage für eine mandantenfähige Architektur gelegt. Eine mögliche, nächste Ausbaustufe könnte die dynamische Erzeugung von Datenbankinstanzen für unterschiedliche Nutzer sein.

Langfristig kann das Projekt somit vielfältige Einsatzmöglichkeiten als leichtgewichtige, containerisierte Verwaltungsplattform bieten.


¹⁴ Keycloak ist ein Open-Source-Tool für Identitäts- und Zugriffsmanagement (IAM), das unter anderem Single Sign-On (SSO), Benutzerregistrierung und OAuth2-basierte Authentifizierung bietet.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 17.06.2025

Ort, Datum


Unterschrift

Literaturverzeichnis

1. **FOWLER, Martin**, o. J. *Configuration File* [online]. Verfügbar unter: <https://martinfowler.com/bliki/ConfigurationFile.html> [Zugriff am 14.06.2025].
2. **HIGHTOWER, Kelsey; BURNS, Brendan; BEDA, Joe**, 2019. *Kubernetes: Up and Running*. 2. Aufl. Sebastopol: O'Reilly Media.
3. **ELMASRI, R.; NAVATHE, S. B.**, 2015. *Fundamentals of Database Systems*. 7th ed. Boston: Pearson Education.
4. **FLANAGAN, David**, 2020. *JavaScript – The Definitive Guide*. 7. Aufl. Sebastopol: O'Reilly Media.
5. **MERKEL, Dirk**, 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. In: *Linux Journal*, 239.
6. **BURNS, Brendan; GRANT, Brian; OPPENHEIMER, David; et al.**, 2016. *Design Patterns for Container-Based Distributed Systems*. Redmond: Microsoft Research.
7. **FIELDING, Roy T.**, 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation, University of California, Irvine.
8. **GITHUB INC.**, 2025. *GitHub Codespaces Documentation* [online]. Verfügbar unter: <https://docs.github.com/en/codespaces> [Zugriff am 15.06.2025].
9. **HUMBLE, Jez; FARLEY, David**, 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison-Wesley.
10. **GOOGLE LLC**, 2024. *Angular – The modern web developer's platform* [online]. Verfügbar unter: <https://angular.io> [Zugriff am 15.06.2025].
11. **META PLATFORMS, INC.**, 2024. *React – A JavaScript library for building user interfaces* [online]. Verfügbar unter: <https://react.dev> [Zugriff am 15.06.2025].

Anhang

| Endpunkt | Methode | Beschreibung |
|--------------|---------|-------------------------------|
| /items | GET | Alle Items abrufen |
| /items | POST | Neues Item anlegen |
| /items/:name | PUT | Item aktualisieren (per Name) |
| /items/:name | DELETE | Item löschen (per Name) |

Tabelle 3 - API Endpunkte

| Komponente | Port | Beschreibung |
|------------------|------|-----------------------|
| Backend A | 5000 | Go-API (Datenbank A) |
| Backend B | 5001 | Go-API (Datenbank B) |
| Frontend React | 6000 | React Web-Anwendung |
| Frontend Angular | 6001 | Angular Web-Anwendung |
| PostgreSQL A | 5432 | Datenbank A |
| PostgreSQL B | 5433 | Datenbank B |

Tabelle 4 - Portübersicht