Dokumentation „Adventskalender-Manager“

Carla Erb
ON24, November 2025

Projektdokumentation

1. Thema des Projekts

1.1 Projektbeschreibung

Der **Adventskalender-Manager** ist eine webbasierte Client-Server-Anwendung zur Verwaltung von handgemachten Adventskalendern mit 24 Säckchen. Die Anwendung unterstützt Benutzer bei der Planung, Befüllung und Organisation ihrer Adventskalender für die Weihnachtszeit.

Was kann die Anwendung?

Die Anwendung ermöglicht es Benutzern:

- Mehrere Adventskalender anzulegen und zu verwalten
- Für jeden Kalender 24 Säckchen (nummeriert 1-24) mit Inhalten zu befüllen
- Jeden Säckchen Inhalte, Notizen und einen "Gepackt"-Status zuzuordnen
- Den Fortschritt zu verfolgen (z.B. "17 von 24 Säckchen gepackt")
- Inhalte zufällig neu zu mischen, falls gewünscht
- Kalender als JSON oder CSV zu exportieren (für Backup oder Druckvorlagen)
- (Als Admin) Benutzerkonten zu verwalten

1.2 Warum dieses Thema?

Praktischer Nutzen: Meine Mutter verkauft in ihrem Etsy-Shop selbstgenähte Adventskalender. Die App könnte sie ihren Kundinnen zeigen (z. B. als Link auf ihrer Website), damit diese das Befüllen planen. Das bringt echten Nutzen und zusätzlich Aufmerksamkeit für den Shop.

Lernziele:

- Moderne Webentwicklung mit Vue 3 Composition API
- REST-API-Design und -Implementierung
- Deno als Node.js-Alternative kennenlernen
- TypeScript in der Praxis anwenden
- Datenbankmodellierung mit SQLite

1.3 Projektumfang

Technische Anforderungen (T4-Modul):

- Client-Server-Architektur mit getrennten Runtimes
- REST-API für asynchrone Kommunikation
- Multi-User-fähig mit Authentifizierung
- Vue.js mit Single File Components
- Composition API (Pflicht)
- Mindestens ein vollständiger CRUD-Zyklus
- Zentrale Datenhaltung mit Datenisolation
- Server-seitige Validierung

Zusätzliche Features:

- TypeScript für Frontend und Backend
- Admin-Bereich mit Rollenverwaltung
- Export-Funktionalität (JSON/CSV)
- Fortschritts-Tracking
- Shuffle-Feature mit Fisher-Yates-Algorithmus

2. Ausgangssituation

2.1 Vorkenntnisse vor dem Projekt (Skill-Übersicht)

Programmiererfahrung: Ich habe die vorhandenen Grundkenntnisse aus den beiden vorherigen Semestern.

Framework-Erfahrung: Ich habe Grunderfahrungen in TypeScript und Javascript.

Backend-Erfahrung: Node.js und Deno waren komplett neu.

Datenbankkenntnisse: SQL-Grundlagen aus Datenbanken-Kurs, aber noch nie praktisch angewendet.

Zusammenfassung Ausgangslage: Solide Grundkenntnisse in Webentwicklung, aber keine Erfahrung mit modernen Frameworks oder fullstack-Entwicklung.

2.2 Bisherige Erfahrungen bei der Entwicklung von Webanwendungen

Vorherige Projekte: Nur kleine statische Webseiten mit HTML/CSS/JavaScript im ersten und zweiten Semester.

Lücken/Schwächen:

- Keine große Erfahrung mit Frontend-Frameworks
- Backend-Entwicklung war komplett neu
- Noch nie mit Datenbanken gearbeitet
- REST-API-Design unbekannt

3. Vorgehen

3.1 Planung des Projekts

Initiale Planung: Ich habe einen detaillierten Projektplan (`PROJEKTPLAN.md`) erstellt mit 11 Phasen. Dabei hatte jede Phase klare Deliverables und Testziele.

Zeitmanagement: Ca. 18-20 Stunden über 2 Wochen

3.2 Aktivitäten und Entwicklungsprozess

Vorgehen pro Phase:

1. **Planung:** Anforderungen in `PROJEKTPLAN.md` definieren
2. **Recherche:** Dokumentationen lesen, Tutorials schauen
3. **Implementierung:** Backend-Endpoints → Frontend-Komponenten → Integration
4. **Testing:** Postman (Backend) + Browser (Frontend) + Manuelle Testchecklisten
5. **Review:** Projektplan aktualisieren, nächste Phase planen

Iterativer Ansatz:

- Start mit Minimal-Features (MVP)
- Schrittweise Erweiterung in 10 Phasen
- Nach jeder Phase: Testing & Feedback

Git-Workflow: Regelmäßige Commits nach jeder Phase, klare Commit-Messages

3.3 Wissenserwerb

- Offizielle Dokumentationen (Vue, Deno, TypeScript)
- YouTube-Tutorials für Grundlagen
- GitHub Copilot mit Claude für Code-Vorschläge und konzeptionelle Fragen

4. Anforderungsliste

4.1 Pflichtanforderungen (T4-Modul)

#	Anforderung	Status	Implementiert in
1	Trennung Client/Server	✅	Vue (Frontend) + Deno (Backend)
2	REST-API	✅	`/api/*` Endpunkte
3	Asynchrone Kommunikation	✅	Fetch-API mit Promises
4	Session Management	✅	Cookie-basierte Sessions
5	Multi-User-fähig	✅	Benutzer-Isolation in DB
6	Mindestens ein CRUD-Zyklus	✅	Kalender + Säckchen
7	Vue.js mit SFC	✅	13 `.vue`-Komponenten
8	Composition API	✅	`<script setup>` überall
9	Mehrere Komponenten	✅	13 Komponenten + 6 Views
10	Reactivity sinnvoll	✅	`ref()`, `computed()`, Pinia
11	Server-seitige Validierung	✅	Alle Inputs im Backend geprüft
12	Zentrale Datenhaltung	✅	SQLite-Datenbank
13	Datenisolation	✅	User sieht nur eigene Daten
14	Vollständige Dokumentation	✅	Hauptdokument + Phasen-Docs

Ergebnis: 14/14 Pflichtanforderungen erfüllt (100 %)

4.2 Zusätzliche Features (über Pflicht hinaus)

#	Feature	Status	Anmerkung
1	TypeScript (Frontend + Backend)	✅	Typsicherheit, bessere DX
2	Shuffle-Algorithmus (Fisher-Yates)	✅	Mathematisch korrekt
3	Export-Funktionalität (JSON + CSV)	✅	2 Formate
4	Admin-Bereich (RBAC)	✅	Rollenverwaltung
5	Passwort-Hashing (bcrypt)	✅	Sicherheit
6	Progress-Tracking (X/24)	✅	Visuelle Fortschrittsanzeige
7	Responsive Design	✅	Mobile-freundlich
8	Dark/Light Mode (CSS-Variablen)	✅	Weihnachtliches Design
9	Automatisches Session-Cleanup	✅	Keine "Leichen" in DB
10	Foreign Key Constraints	✅	Datenintegrität

4.3 Nicht umgesetzte Features (bewusste Entscheidung)

#	Feature	Grund
1	Passwort-Reset-Funktion	Zeitlich nicht machbar, Admin kann Workaround nutzen
2	E-Mail-Benachrichtigungen	Außerhalb des Projekt-Scopes
3	Kalender-Duplikation	Nice-to-have, nicht essenziell
4	Import-Funktion	Export reicht für Backup-Zweck
5	Bilder/Dateien hochladen	Würde Komplexität erhöhen
6	Mehrsprachigkeit (i18n)	Projekt nur für deutschen Markt

5. Konzeption

5.1 Technologie-/Werkzeugauswahl

5.1.1 Frontend-Technologien

Vue 3 (v3.5.22)

- **Warum Vue?** Pflicht laut Projektspezifikation, außerdem:
  - Einfacher Einstieg durch klare Dokumentation
  - Composition API ermöglicht bessere Code-Organisation
  - Single File Components (SFC) für übersichtliche Struktur
  - Reaktivität: UI aktualisiert sich automatisch bei Datenänderungen

TypeScript (v5.9.3)

- **Warum TypeScript?**
  - Statische Typisierung verhindert Laufzeitfehler
  - Bessere IDE-Unterstützung (Autocomplete, Refactoring)
  - Selbstdokumentierender Code durch Typen
  - Frühzeitiges Erkennen von Fehlern

Vite (v7.1.7)

- **Warum Vite?**
  - Extrem schneller Dev-Server (ESM-basiert)
  - Hot Module Replacement (HMR) in Millisekunden
  - Optimierte Production-Builds

Vue Router (v4.6.3)

- Client-seitige Navigation ohne Seiten-Reload
- Navigation Guards für geschützte Routen
- Typsichere Route-Parameter

Pinia (v3.0.4)

- **Warum Pinia statt Vuex?**
  - Offizielles State Management für Vue 3
  - Weniger Boilerplate als Vuex
  - Bessere TypeScript-Integration
  - Modularer: Jeder Store ist unabhängig

5.1.2 Backend-Technologien

Deno (v1.37+)

- **Warum Deno statt Node.js?** Pflicht laut Projektspezifikation, außerdem:
  - TypeScript ohne Build-Step
  - Sicherheit: Permissions explizit vergeben (`--allow-net`, `--allow-read`, etc.)
  - Modernes Modul-System (ESM, URL-Imports)
  - Kein `node_modules` → sauberere Struktur
  - Built-in Formatter, Linter, Test-Runner

SQLite (v3.9.1)

- **Warum SQLite?** Empfohlen in Projektspezifikation, außerdem:
  - Dateibasiert → keine separate DB-Server-Installation
  - Einfaches Setup (eine `.db`-Datei)
  - Ausreichend performant für < 100.000 Einträge
  - ACID-Transaktionen, Foreign Keys, Check Constraints

bcrypt (v0.4.1)

- Passwort-Hashing mit Salt
- Industry-Standard für Passwort-Sicherheit
- Schutz vor Rainbow-Table-Attacks

5.1.3 Entwicklungswerkzeuge

Tool/Library	**Version**	**Zweck**
`@vitejs/plugin-vue`	v6.0.1	Vue-SFC-Kompilierung in Vite
`vue-tsc`	v3.1.0	TypeScript-Check für Vue-Komponenten
`@types/node`	v24.6.0	TypeScript-Typen für Node.js-APIs
Postman	-	API-Testing während Entwicklung
Git + GitHub	-	Versionskontrolle
VS Code	-	IDE mit Vite/Vue/TypeScript-Support

5.2 Entwurf

5.2.1 Architektur

Gesamtarchitektur: Client-Server-Trennung

Die Anwendung besteht aus drei Hauptschichten:

1. Frontend-Schicht (Browser, Port 5173)

- **Vue 3 Frontend (Single Page Application)**
    - Components: Wiederverwendbare UI-Bausteine (z.B. CalendarCard, PouchItem)
    - Views: Vollständige Seiten (z.B. DashboardView, CalendarDetailView)
    - Router: Client-seitige Navigation zwischen Views
    - Pinia Stores: Zentrales State Management für globalen Zustand
    - Composables: Gekapselte API-Calls (useApi.ts)

Kommunikation: Frontend ↔ Backend

- Protokoll: REST API mit JSON-Format
- Transport: HTTP-Requests mit Session-Cookie für Authentifizierung
- Richtung: Bidirektional (Request vom Client, Response vom Server)

2. Backend-Schicht (Deno Server, Port 8000)

Die Backend-Schicht ist in mehrere Unter-Schichten aufgeteilt:

- **HTTP Server Layer**
    - Routing: URL-Matching zu entsprechenden Handlern
    - Middleware: Auth-Checks (requireAuth) und CORS-Konfiguration
    - Request Parsing: JSON-Body-Parsing, Cookie-Extraktion

- **Route Handlers Layer**
    - `/api/auth/*`: Authentifizierung (Login, Register, Logout, Session-Check)
    - `/api/calendars/*`: Kalender-CRUD-Operationen
    - `/api/pouches/*`: Säckchen-Update und Toggle
    - `/api/admin/*`: Benutzerverwaltung (nur für Admins)

- **Business Logic Layer**

    - Eingabe-Validierung (Länge, Format, Pflichtfelder)
    - Shuffle-Algorithmus: Fisher-Yates für zufällige Neuverteilung
    - Export-Generierung: JSON- und CSV-Format-Erstellung
    - Session Management: Erstellen, Prüfen, Löschen von Sessions

- **Database Layer (database.ts)**

    - SQL-Queries als Prepared Statements (SQL-Injection-Schutz)
    - Transaktions-Management (BEGIN, COMMIT, ROLLBACK)
    - CRUD-Funktionen für alle Tabellen

Kommunikation: Backend ↔ Datenbank

- Protokoll: SQL-Queries
- Transport: SQLite-Driver (deno.land/x/sqlite)
- Richtung: Bidirektional (Queries zum Server, Ergebnisse zurück)

3. Datenschicht (SQLite-Datenbank)

- **adventskalender.db** (Datei-basierte SQLite-Datenbank)
    - users: Benutzerkonten mit Zugangsdaten
    - calendars: Adventskalender-Einträge
    - pouches: 24 Säckchen pro Kalender
    - sessions: Login-Sessions für Authentifizierung

Architekturentscheidungen:

1. **Trennung Frontend/Backend (Separate Runtimes):**

    - Frontend läuft auf Vite Dev-Server (Port 5173)
    - Backend läuft auf Deno HTTP-Server (Port 8000)
    - Kommunikation über REST-API
    - Vorteile: Klare Verantwortlichkeiten, unabhängiges Deployment

2. **REST-API mit JSON:**

    - Alle Endpunkte unter `/api/*`
    - Stateless (außer Session-Cookie)
    - Standard HTTP-Methoden (GET, POST, PUT, PATCH, DELETE)
    - Konsistente Response-Struktur

3. **Schichtenmodell:**

    - **Presentation Layer:** Vue-Komponenten (UI)
    - **Application Layer:** Pinia Stores, Router Guards
    - **API Layer:** REST-Endpunkte, Middleware
    - **Business Logic Layer:** Validierung, Shuffle, Export
    - **Data Access Layer:** Database-Funktionen
    - **Data Layer:** SQLite

5.2.2 Datenmodell / Persistenz

Entity-Relationship-Modell (ER-Modell):

Das Datenmodell besteht aus vier Hauptentitäten mit folgenden Beziehungen:

1. Tabelle: users (Benutzerkonten)

- **Primärschlüssel:** id (INTEGER, AUTOINCREMENT)
- **Unique Key:** username (eindeutiger Benutzername)
- **Attribute:**
    - password_hash: Bcrypt-Hash des Passworts
    - role: Benutzerrolle (user oder admin)
    - created_at: Registrierungszeitpunkt
- **Beziehungen:** Ein User hat N Kalender (1:N Beziehung zu calendars)

2. Tabelle: calendars (Adventskalender)

- **Primärschlüssel:** id (INTEGER, AUTOINCREMENT)
- **Fremdschlüssel:** user_id → Referenz auf users.id
- **Attribute:**
    - name: Kalender-Name
    - description: Optionale Beschreibung
    - created_at: Erstellungszeitpunkt
- **Beziehungen:**
    - Ein Kalender gehört zu genau einem User (N:1 zu users)
    - Ein Kalender enthält 24 Säckchen (1:24 zu pouches)

3. Tabelle: pouches (Säckchen)

- **Primärschlüssel:** id (INTEGER, AUTOINCREMENT)
- **Fremdschlüssel:** calendar_id → Referenz auf calendars.id
- **Attribute:**
    - number: Säckchen-Nummer (1-24)
    - content: Inhaltsbeschreibung
    - notes: Zusätzliche Notizen
    - is_packed: Gepackt-Status (Boolean, 0 oder 1)
    - created_at: Erstellungszeitpunkt
- **Beziehungen:** Ein Säckchen gehört zu genau einem Kalender (N:1 zu calendars)

4. Tabelle: sessions (Login-Sessions)

- **Primärschlüssel:** id (TEXT, UUID)
- **Fremdschlüssel:** user_id → Referenz auf users.id
- **Attribute:**
  - expires_at: Ablaufzeitpunkt der Session
  - created_at: Login-Zeitpunkt
- **Beziehungen:** Eine Session gehört zu genau einem User (N:1 zu users)

Datenintegrität:

- **FOREIGN KEY Constraints** mit `ON DELETE CASCADE`
  - Beim Löschen eines Users: Alle Kalender werden gelöscht
  - Beim Löschen eines Kalenders: Alle Säckchen werden gelöscht
- **UNIQUE Constraints:**
  - `username` muss eindeutig sein
  - `(calendar_id, number)` muss eindeutig sein (keine doppelten Nummern)
- **CHECK Constraints:**
  - `role IN ('user', 'admin')`
  - `number BETWEEN 1 AND 24`
  - `is_packed IN (0, 1)`

Transaktionen: Beim Kalender-Erstellen werden in einer Transaktion:

1. Kalender eingefügt (`INSERT INTO calendars`)
2. 24 leere Säckchen eingefügt (`INSERT INTO pouches` × 24)
3. Bei Fehler: Rollback (alles oder nichts)

5.2.3 UI-Entwurf

Komponentenstruktur:

Die Frontend-Architektur folgt einer hierarchischen Komponentenstruktur:

Root-Komponente:

- **App.vue**: Hauptkomponente mit Router-View

Views (Seiten-Komponenten):

1. **LoginView.vue** - Authentifizierungs-Seite
   - LoginForm.vue: Login-Formular
   - RegisterForm.vue: Registrierungs-Formular

2. **DashboardView.vue** - Hauptseite mit Kalenderübersicht

    - CalendarList.vue: Liste aller Kalender
        - CalendarCard.vue: Einzelne Kalender-Karte
        - ProgressBar.vue: Fortschrittsanzeige (X/24 gepackt)

3. **CalendarDetailView.vue** - Detailansicht eines Kalenders

    - PouchList.vue: Liste aller 24 Säckchen
        - PouchItem.vue: Einzelnes Säckchen mit Inline-Editing

4. **CalendarEditView.vue** - Kalender bearbeiten

    - CalendarForm.vue: Formular für Name/Beschreibung

5. **AdminDashboardView.vue** - Admin-Bereich

    - UserList.vue: Liste aller Benutzer
    - UserForm.vue: Formular zum Benutzer anlegen

Design-Entscheidungen:

1. **Single Page Application (SPA):**

    - Keine Seiten-Reloads
    - Schnelle Navigation zwischen Views
    - Vue Router für Client-seitiges Routing

2. **Responsive Design:**

    - Mobile-First-Ansatz
    - CSS Grid & Flexbox für Layouts
    - Media Queries für verschiedene Bildschirmgrößen

3. **Scoped CSS:**

    - Jede Komponente hat eigene Styles
    - Keine Style-Konflikte zwischen Komponenten
    - Globale Variablen für Farben/Fonts in `style.css`

4. **Dark/Light Mode Vorbereitung:**

    - CSS Custom Properties (Variablen)
    - Weihnachtliches Farbschema (rot, grün, gold)

Alternativen erwogen:

- **MPA (Multi-Page Application):** Verworfen wegen langsamerer Navigation
- **Server-Side Rendering (SSR):** Nicht nötig für diesen Use Case
- **Tailwind CSS:** Verzichtet zugunsten von Custom CSS (Lernzweck)

5.2.4 API-Design

REST-Konventionen:

- `GET /api/resources` - Liste aller Ressourcen
- `POST /api/resources` - Neue Ressource erstellen
- `GET /api/resources/:id` - Einzelne Ressource
- `PUT /api/resources/:id` - Ressource vollständig aktualisieren
- `PATCH /api/resources/:id` - Ressource teilweise aktualisieren
- `DELETE /api/resources/:id` - Ressource löschen

Implementierte Endpunkte:

- **Auth:** `/api/auth/register`, `/api/auth/login`, `/api/auth/logout`, `/api/auth/session`
- **Calendars:** `/api/calendars` (CRUD), `/api/calendars/:id/pouches`, `/api/calendars/:id/shuffle`, `/api/calendars/:id/export`
- **Pouches:** `/api/pouches/:id`, `/api/pouches/:id/toggle`
- **Admin:** `/api/admin/users` (CRUD), `/api/admin/users/:id/role`

Beispiel Request-Flow (Kalender erstellen):

1. User füllt Formular aus (CalendarForm.vue)

2. Submit-Handler ruft `createCalendar()` im Store (calendar.ts)

3. Store ruft `createCalendar()` in useApi.ts

4. useApi macht POST-Request zu /api/calendars mit JSON-Body

5. Backend empfängt Request (server.ts)

6. Middleware prüft Session (requireAuth)

7. Route-Handler validiert Input

8. Handler ruft Database-Funktion (database.ts)

9. SQL INSERT in calendars + 24x pouches (Transaktion)

10. Handler sendet JSON-Response mit 201 Created

11. Store aktualisiert State (calendars.value.push(...))

12. Vue Reactivity aktualisiert UI automatisch

Sicherheit:

- Server-seitige Validierung aller Inputs
- Prepared Statements gegen SQL-Injection
- bcrypt für Passwort-Hashing
- HttpOnly-Cookies für Sessions (XSS-Schutz)
- SameSite=Lax für CSRF-Schutz
- CORS konfiguriert für Frontend-Origin

6. Ergebnis des Projekts

6.1 Projektstatistik

Entwicklungsumfang:

- **Entwicklungszeit:** ~18-20 Stunden über 2 Wochen
- **Entwicklungsphasen:** 10 abgeschlossene Phasen
- **Code:** ~6.000 Zeilen (Frontend + Backend)
- **Dateien:** ~70 Dateien (Code + Dokumentation)
- **Commits:** 16

Frontend:

- 13 Vue-Komponenten
- 6 Views (Seiten)
- 4 Pinia Stores
- 1 Composable (API-Wrapper)

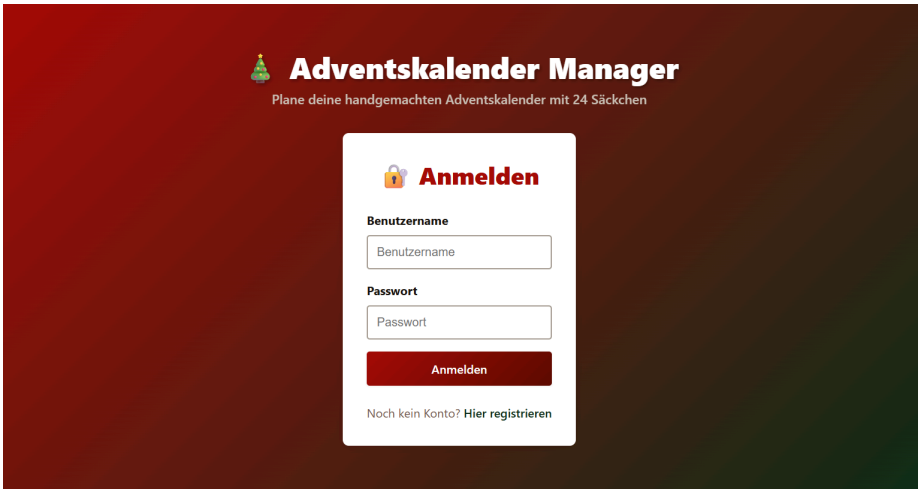
Backend:

- 18 REST-API-Endpunkte
- 4 Datenbank-Tabellen
- ~30 SQL-Funktionen
- 4 Route-Module

6.2 Screenshots und Screenflow

6.2.1 Authentifizierung

Screenshot 1: Login-Seite



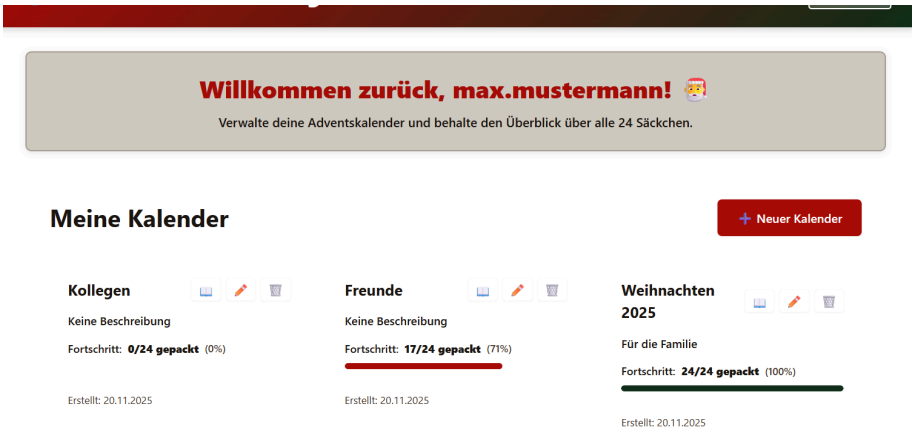
- Einfaches Login-Formular
- Link zur Registrierung
- Fehleranzeige bei falschen Zugangsdaten

Screenshot 2: Registrierung

- Neues Benutzerkonto anlegen
- Validierung: Username 3-20 Zeichen, Passwort min. 6 Zeichen
- Direkte Weiterleitung zum Dashboard nach Erfolg

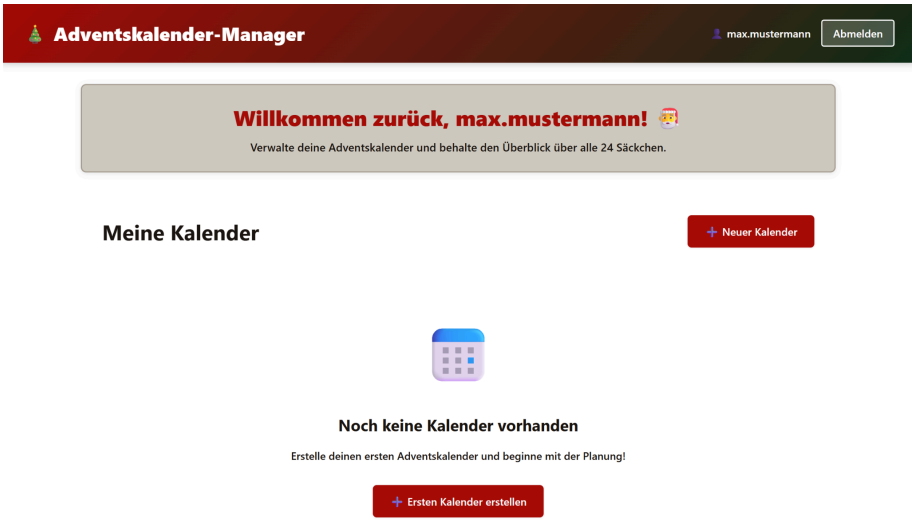
6.2.2 Dashboard (Kalenderübersicht)

Screenshot 3: Dashboard mit Kalendern



- Liste aller eigenen Kalender
- Fortschrittsanzeige pro Kalender (z.B. "17/24 gepackt")
- Button "Neuer Kalender"
- Schnellzugriff: Bearbeiten, Löschen, Details

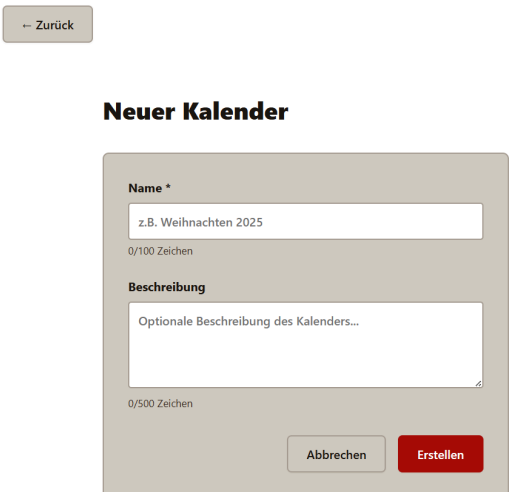
Screenshot 4: Leeres Dashboard



- Anzeige wenn noch keine Kalender vorhanden
- Call-to-Action: "Erstelle deinen ersten Kalender"

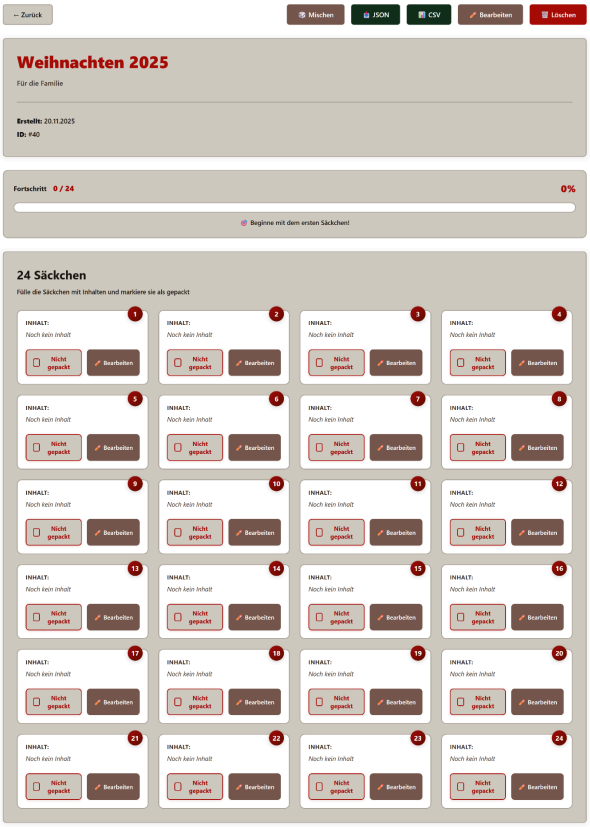
6.2.3 Kalender-Verwaltung

Screenshot 5: Neuen Kalender erstellen



- Formular: Name (Pflicht), Beschreibung (optional)
- Bei Erstellung werden automatisch 24 leere Säckchen angelegt

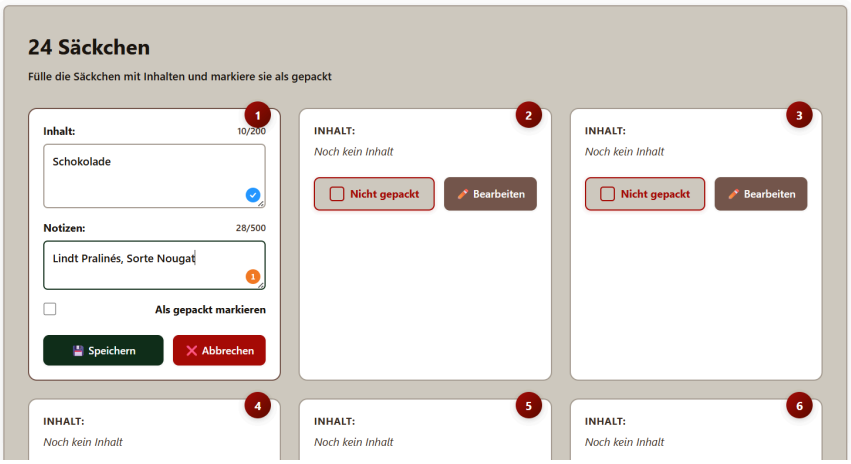
Screenshot 6: Kalender-Detail mit Säckchen



- Übersicht aller 24 Säckchen (Nummer 1-24)
- Jedes Säckchen zeigt: Nummer, Inhalt, Notizen, Status
- Inline-Editing: Direkt im Detail-View bearbeiten
- Aktionsbuttons: Mischen, Exportieren, Kalender bearbeiten

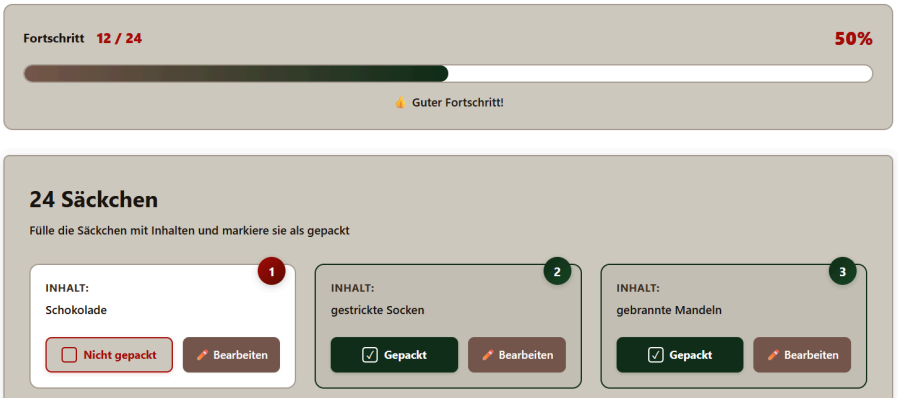
6.2.4 Säckchen-Verwaltung

Screenshot 7: Säckchen bearbeiten



- Inline-Bearbeitung von Inhalt und Notizen
- Toggle-Button für "Gepackt"-Status
- Änderungen werden sofort gespeichert (Auto-Save)

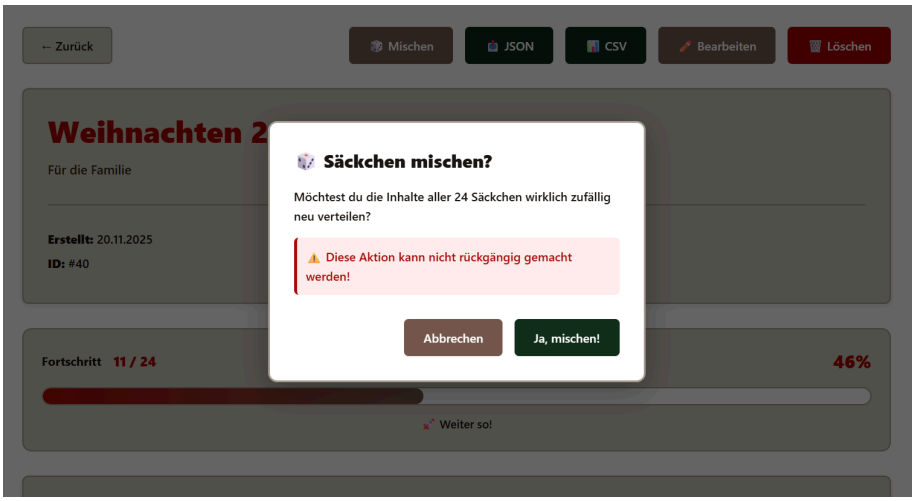
Screenshot 8: Gepackte Säckchen



- Visuelle Unterscheidung: Gepackte Säckchen grün markiert
- Fortschrittsbalken oben zeigt X/24 gepackt

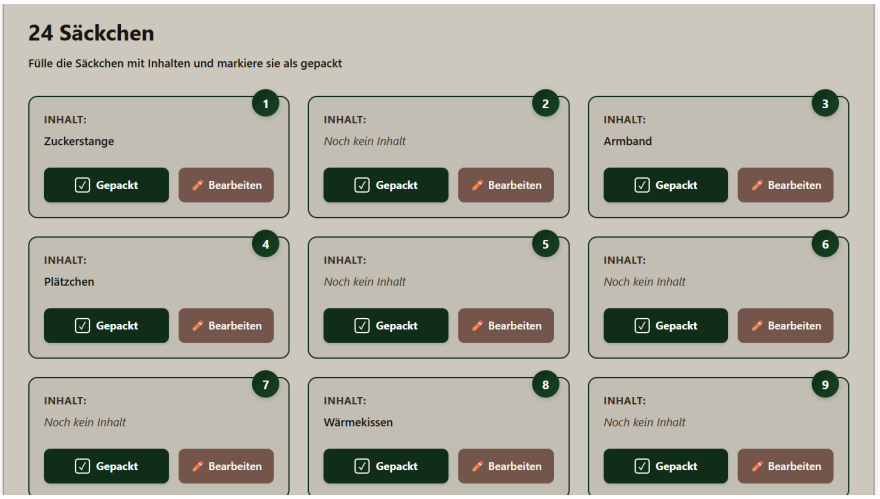
6.2.5 Special Features

Screenshot 9: Mischen-Dialog



- Bestätigungsdialog vor dem Mischen
- Warnung: "Inhalte werden zufällig neu verteilt"

Screenshot 10: Ergebnis nach Mischen



- Inhalte sind über die 24 Säckchen neu verteilt
- Nummerierung 1-24 bleibt gleich, nur Inhalte ändern sich

Screenshot 11: Export-Optionen

- Buttons für JSON- und CSV-Export
- Download startet automatisch mit Dateiname `kalender-export-nummer.json/csv`

Screenshot 12: Exportierte CSV-Datei

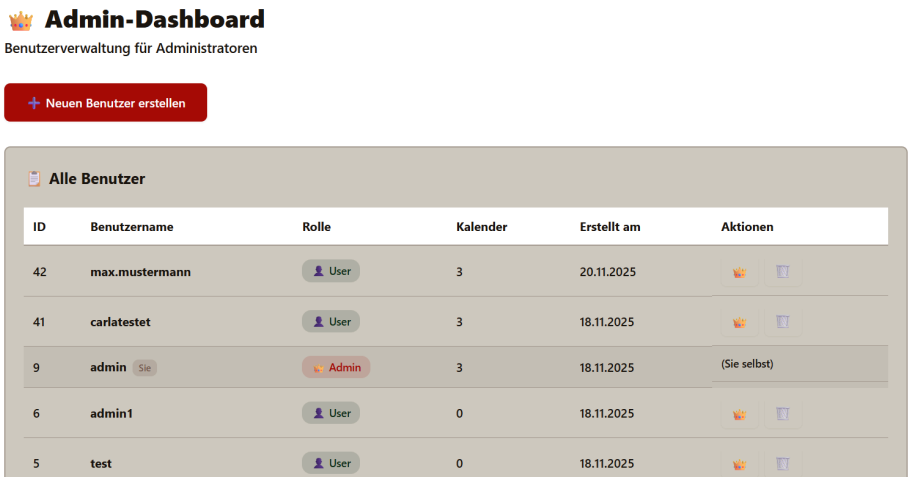
C: > Users > Carla > Downloads > kalender_export_1763675576534.csv

```
Kalendername;Beschreibung;Erstellt am
"Weihnachten 2025";"Für die Familie";"20.11.2025"

Nummer;Inhalt;Notizen;Gepackt
1;"Zuckerstange";"";"Ja"
2;"";"";"Ja"
3;"Armband";"";"Ja"
4;"Plätzchen";"";"Ja"
5;"";"";"Ja"
6;"";"";"Ja"
7;"";"";"Ja"
8;"Wärmekissen";"";"Ja"
9;"";"";"Ja"
10;"";"";"Ja"
11;"Schokolade";"Lindt Pralinés, Sorte Nougat";"Ja"
12;"Chips";"";"Ja"
13;"Socken";"gestrickt";"Ja"
14;"";"";"Ja"
15;"";"";"Ja"
16;"Labello";"in rot";"Ja"
17;"Lottoschein";"";"Ja"
18;"";"";"Ja"
19;"";"";"Ja"
20;"gebrante Mandeln";"";"Ja"
21;"Nikolaus";"";"Ja"
22;"";"";"Ja"
23;"";"";"Ja"
24;"";"";"Ja"

```

- Geöffnete CSV-Datei in VSC
- Spalten: Nummer, Inhalt, Notizen, Gepackt

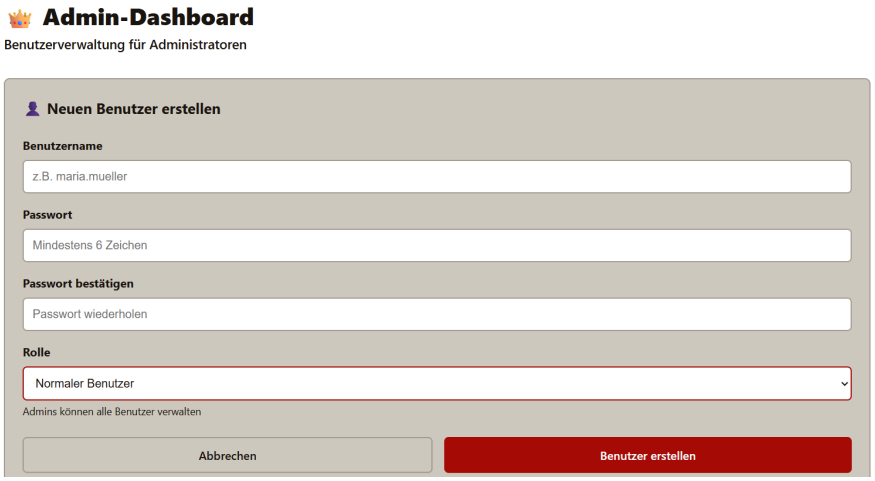
6.2.6 Admin-Bereich

Screenshot 13: Admin-Dashboard



- Übersicht aller Benutzerkonten
- Anzeige: Username, Rolle, Kalender-Anzahl, Registrierungsdatum
- Aktionen: Rolle ändern, Benutzer löschen

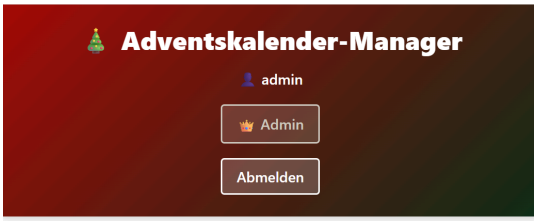
Screenshot 14: Neuen Benutzer anlegen

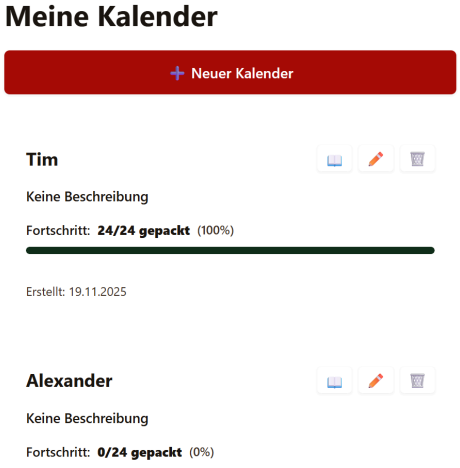


- Formular: Username, Passwort, Rolle
- Nur Admins können andere Admins erstellen

6.2.7 Responsive Design

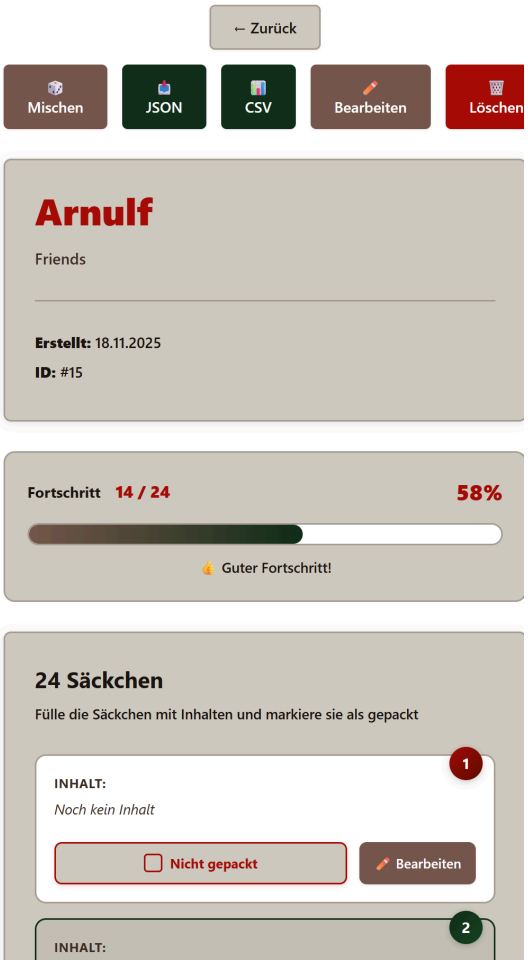
Screenshot 15: Mobile-Ansicht Dashboard





- Kalender-Cards stacken vertikal
- Touch-freundliche Buttons

Screenshot 16: Mobile-Ansicht Kalender-Detail



- Säckchen-Liste scrollbar
- Kompakte Darstellung

6.3 Selbstbeurteilung des Ergebnisses

Was funktioniert gut:

- Alle Core-Features funktionieren ohne Fehler
- UI ist intuitiv bedienbar
- Performance ist gut, keine Ladezeiten

Was könnte besser sein:

- Design könnte professioneller sein
- Einige Edge Cases nicht abgedeckt
- Keine Unit-Tests vorhanden

Gesamtbewertung: Sehr zufrieden. Alle Anforderungen erfüllt, viele Zusatzfeatures implementiert, stabiles System.

7. Reflexion

7.1 Herausforderungen und deren Bewältigung

Challenge #1: Deno vs. Node.js (Neue Technologie)

Problem: Deno war komplett neu, musste URL-basierte Imports verstehen

Lösung:

- Deno-Dokumentation gelesen
- Standard Library angeschaut
- Unterschiede zu Node.js bemerkt

Learnings: Permissions-System ist anfangs nervig, aber sicherer als Node.js

Challenge #2: TypeScript in Vue-Komponenten

Problem: Unterschiedliche Types waren anfangs verwirrend

Lösung:

- TypeScript-Docs für Vue 3 angeschaut
- Viele Trial-and-Error-Versuche

Learnings: TypeScript zahlt sich aus, findet Fehler vor dem Testen

Challenge #3: Session Management mit Cookies

Problem: CORS + Credentials war tricky, Cookies wurden nicht gesendet

Lösung:

- `Access-Control-Allow-Credentials: true` im Backend
- `credentials: 'include'` im Frontend-Fetch
- `SameSite=Lax` für CSRF-Schutz

Learnings: Cookies sind komplexer als gedacht, aber sicherer als localStorage

Challenge #4: Kalender-Säckchen-Beziehung

Problem: Wie erstelle ich automatisch 24 Säckchen beim Kalender-Erstellen?

Lösung:

- Transaktion: Kalender INSERT → Loop 24x Pouch INSERT → COMMIT
- Bei Fehler: ROLLBACK (alles oder nichts)

Learnings: Transaktionen sind wichtig für Datenintegrität

Challenge #5: Shuffle-Algorithmus

Problem: Wie mische ich fair und zufällig? Naive Sortierung ist nicht gleichverteilt

Lösung:

- Fisher-Yates-Algorithmus recherchiert
- Implementiert in TypeScript
- Getestet: Verteilung ist gleichmäßig

Learnings: Algorithmen aus der Informatik sind wichtig, nicht einfach drauf los coden

Challenge #6: User-Isolation und Security

Problem: Wie stelle ich sicher, dass User A nicht Daten von User B sieht?

Lösung:

- Jede DB-Query filtert nach `user_id`
- Middleware `requireAuth()` prüft Ownership
- Tests: User A versucht, Kalender-ID von User B zu öffnen → 403 Forbidden

Learnings: Security by default, niemals auf Frontend-Checks verlassen

Challenge #7: Zeitmanagement

Problem: Projekt parallel zu anderen Kursen, Klausuren, Privatleben

Lösung:

- Detaillierter Projektplan mit Phasen
- Klare Priorisierung: Pflichtfeatures zuerst
- Puffer für Unvorhergesehenes

Learnings: Planung ist die halbe Miete, spart am Ende Zeit

Challenge #8: Dokumentation während des Projekts

Problem: Dokumentation am Ende wäre zu viel auf einmal

Lösung:

- Nach jeder Phase: Zusammenfassung schreiben
- Testing-Checklisten parallel erstellen
- Projektplan laufend aktualisieren

Learnings: Kontinuierliche Dokumentation ist weniger Arbeit als alles am Schluss

7.2 Unterstützung

7.2.1 Was habe ich unternommen, wenn ich nicht weiterwusste?

Vorgehen bei Problemen:

1. **Recherche:** Erst versuchen, Problem selbst zu lösen
2. **Dokumentationen:** Offizielle Docs durchlesen
3. **Tutorials:** YouTube-Videos für Grundlagen
4. **Community:** Stack Overflow, Reddit
5. **KI-Tools:** Claude, GitHub Copilot für Erklärungen

7.2.2 Offizielle Dokumentationen

- **Vue 3:** https://vuejs.org/ → Composition API, Reactivity, SFC
- **Vue Router:** https://router.vuejs.org/ → Navigation Guards, Route Meta
- **Pinia:** https://pinia.vuejs.org/ → State Management
- **Deno:** https://deno.land/manual → Standard Library, Permissions
- **Deno SQLite:** https://deno.land/x/sqlite@v3.9.1 → DB-Operationen
- **bcrypt:** https://deno.land/x/bcrypt@v0.4.1 → Password Hashing
- **MDN Web Docs:** https://developer.mozilla.org/ → Fetch, Cookies, CORS

7.2.3 KI-Tools (GitHub Copilot mit Claude)

Eingesetzte KI-Tools:

GitHub Copilot mit Claude

- **Wofür genutzt:** Code-Completion, Code-Review, Debugging, komplexe Erklärungen, repetitive Tasks
- **Beispiele:**
  - Auto-Complete für TypeScript-Interfaces
  - Vorschläge für SQL-Queries
  - Generierung von Test-Daten
- **Bewertung:** Sehr hilfreich, aber manchmal falsche Vorschläge

ChatGPT (https://chat.openai.com/)

- **Wofür genutzt:** Konzeptionelle Fragen, Algorithmen, Erklärungen
- **Beispiele:**
  - "Wie funktioniert Fisher-Yates-Algorithmus?"
  - "Wie implementiere ich Session Management in Deno?"
- **Bewertung:** Gute Erklärungen, aber Code musste ich oft anpassen

Verantwortung und Eigenleistung:

Mein Verständnis:

- Ich bin verantwortlich für ALLEN Code, auch KI-generierten
- KI ist ein Werkzeug wie Stack Overflow oder Tutorials
- Ich habe jeden generierten Code getestet
- Bei Unklarheiten: Nachforschen, nicht blind kopieren

Eigenleistung:

- ~20 % selbst geschrieben, ~80 % KI-unterstützt
- Alle konzeptionellen Entscheidungen selbst getroffen
- Alle Bugs selbst gefixed
- Alle Tests selbst durchgeführt

KI-Grenzen erlebt:

- KI generiert manchmal veralteten Code
- KI versteht Projekt-Kontext nicht immer
- Debugging ist Handarbeit, KI kann nur Tipps geben

7.3 Lernerfolge / Fazit

7.3.1 Technische Skills

Neu gelernt:

- Vue 3 Composition API
- Deno Runtime und Standard Library
- Pinia State Management
- REST-API-Design
- SQLite mit Foreign Keys
- bcrypt Password Hashing
- Session Management mit Cookies
- CORS richtig konfigurieren

Vertieft:

- SQL (Joins, Transaktionen, Constraints)
- TypeScript in Praxis (nicht nur Theorie)
- Git (sinnvolle Commits, Branching)
- Debugging (Browser DevTools, Netzwerk-Tab)
- CSS (Flexbox, Grid, Responsive Design)

7.3.2 Soft Skills

Verbessert:

- Zeitmanagement (Phasenplanung)
- Dokumentation (laufend statt am Ende)
- Selbstständigkeit (Probleme selbst lösen)
- Frustrationstoleranz (Bugs, Errors, Trial & Error)

Gelernt über mich selbst: Ich arbeite besser mit klarem Plan als chaotisch drauf los.

7.3.3 Architektur und Design

Wichtige Erkenntnisse:

- Trennung von Concerns (Frontend/Backend, Components/Logic)
- API-First-Design: Erst API planen, dann UI
- Datenbank-Schema von Anfang an gut durchdenken
- Security von Anfang an mitdenken, nicht nachträglich

Best Practices verinnerlicht:

- RESTful-API-Konventionen
- Komponentenbasierte UI
- Single Responsibility Principle
- DRY (Don't Repeat Yourself)

7.3.4 Was lief gut?

- Phasenweise Planung hat Struktur gegeben
- Frühe Tests haben viele Bugs verhindert
- Dokumentation parallel zum Code war kluge Entscheidung
- Alle Pflichtanforderungen erfüllt (14/14)

7.3.5 Was würde ich in Zukunft verbessern?

- Mehr Code-Reviews
- Mehr Pausen einlegen (manchmal zu lange am Stück gecoded)
- Früher mit UI-Design anfangen statt nur Funktionalität

7.3.6 Wie finde ich mein Ergebnis?

Sehr zufrieden. Alle Anforderungen erfüllt, viele Zusatzfeatures implementiert, stabiles System. Die Anwendung ist benutzbar und erfüllt ihren Zweck.

7.3.7 Was kann man nun damit machen?

Praktischer Nutzen:

- Echte Nutzung zur Adventskalender-Planung
- Hilfreich für Etsy-Shop oder privaten Gebrauch
- Exportfunktion ermöglicht Backup und Druckvorlagen

Portfolio-Projekt:

- Zeigt Fullstack-Fähigkeiten (Frontend + Backend)
- Moderne Tech-Stack-Erfahrung (Vue 3, TypeScript, Deno)
- Demonstriert selbstständiges Arbeiten an größerem Projekt

7.3.8 Weiterentwicklungsmöglichkeiten

Mögliche Erweiterungen:

- Passwort-Reset per E-Mail
- Kalender-Vorlagen (z.B. "Schokoladen-Kalender", "Beauty-Kalender")
- Teilen-Funktion (Kalender mit anderen Usern)
- Kalender-Duplikation (Vorjahres-Kalender als Vorlage)
- Mobile App (React Native, Flutter)
- Bilder hochladen für Säckchen
- Erinnerungsfunktion (Benachrichtigungen)
- Import-Funktion (CSV/JSON importieren)
- Dark Mode (vollständig implementiert)
- Mehrsprachigkeit (i18n)

Berufliche Relevanz:

- Fullstack-Projekt im Portfolio
- Moderne Tech-Stack-Erfahrung (Vue 3, TypeScript, Deno)
- Selbstständiges Arbeiten an größerem Projekt
- Erfahrung mit REST-API-Design
- Datenbank-Modellierung und SQL

7.3.9 Persönliches Fazit

Dieses Projekt war herausfordernd, aber unglaublich lehrreich. Ich habe mehr gelernt als in jedem Tutorial, weil ich echte Probleme lösen musste. Besonders die Kombination aus Frontend (Vue 3) und Backend (Deno) hat mir gezeigt, wie komplex moderne Webanwendungen sind. Die größte Herausforderung war das Zeitmanagement, aber durch den detaillierten Projektplan habe ich es geschafft, alle Anforderungen zu erfüllen.

Ich bin stolz auf das Ergebnis: Eine funktionierende, sichere und gut dokumentierte Anwendung, die ich auch tatsächlich nutzen werde. Die KI-Tools (Copilot, Claude) waren hilfreich, aber ich musste viel selbst lernen und verstehen. Das nächste Projekt würde ich mit diesem Wissen viel schneller und strukturierter angehen.

A. Installationsanleitung

1. Systemvoraussetzungen

Betriebssystem

- **Windows** 10/11 (getestet)
- **macOS** 26 (getestet)
- **Linux** (Ubuntu 20.04+, sollte funktionieren)

Erforderliche Software

- **Node.js** Version 18.x oder höher
- **npm** Version 9.x oder höher (kommt mit Node.js)
- **Deno** Version 1.37 oder höher
- **Git** (optional, zum Klonen des Repositories)
- **PowerShell** (Windows) oder **Bash** (macOS/Linux)

Hardware-Anforderungen

- **RAM:** Mindestens 4 GB (8 GB empfohlen)
- **Festplatte:** 500 MB freier Speicherplatz
- **Prozessor:** Dual-Core oder besser

2. Installation der Voraussetzungen

Node.js und npm installieren

Windows

1. Besuche https://nodejs.org/
2. Lade die **LTS-Version** (z.B. 20.x) herunter
3. Führe den Installer aus und folge den Anweisungen
4. Öffne PowerShell und prüfe die Installation:

   node --version

   npm --version

macOS

```
# Mit Homebrew

brew install node

# Oder lade den Installer von nodejs.org herunter
```

Linux (Ubuntu/Debian)

```
# NodeSource Repository hinzufügen

curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -

sudo apt-get install -y nodejs

# Installation prüfen

node --version

npm --version
```

Deno installieren

Windows (PowerShell)

```
irm https://deno.land/install.ps1 | iex
```

Nach der Installation:

```
# Deno zum PATH hinzufügen (falls nicht automatisch geschehen)

$env:Path += ";$env:USERPROFILE\.deno\bin"

# Permanent machen (optional)

[Environment]::SetEnvironmentVariable("Path", $env:Path,
[EnvironmentVariableTarget]::User)

# Installation prüfen

deno --version
```

macOS/Linux

```
curl -fsSL https://deno.land/install.sh | sh

# Zum PATH hinzufügen (füge dies zu ~/.bashrc oder ~/.zshrc hinzu)

export DENO_INSTALL="$HOME/.deno"

export PATH="$DENO_INSTALL/bin:$PATH"

# Installation prüfen

deno --version
```

3. Projekt herunterladen

Option A: Git Clone (empfohlen)

In gewünschtes Verzeichnis wechseln

cd C:\Users\DeinName\Projekte

Repository klonen

git clone https://github.com/carloerbse/adventskalender-manager.git

cd adventskalender-manager

Option B: ZIP herunterladen

1. Lade das Projekt als ZIP herunter
2. Entpacke die Datei in ein Verzeichnis deiner Wahl
3. Öffne PowerShell/Terminal in diesem Verzeichnis

4. Frontend-Installation

Schritt 1: Dependencies installieren

Im Projekt-Hauptverzeichnis

npm install

Was passiert hier?

- npm lädt alle benötigten Pakete herunter (Vue 3, TypeScript, Vite, Router, Pinia, etc.)
- Dies kann 1-2 Minuten dauern
- Erstellt einen `node_modules/` Ordner mit ~200 MB

Schritt 2: Installation prüfen

Dependencies anzeigen

npm list --depth=0

Du solltest sehen:

- `vue@3.x`
- `vue-router@4.x`
- `pinia@2.x`
- `vite@5.x`
- etc.

5. Backend-Vorbereitung

Schritt 1: Deno-Dependencies cachen

cd server

Cache alle Dependencies (ca. 30 Sekunden)

deno cache server.ts

cd ..

Was passiert hier?

- Deno lädt alle benötigten Module herunter und cached sie
- SQLite-Binaries werden heruntergeladen
- bcrypt-Module werden kompiliert

Schritt 2: Datenbank-Initialisierung

Die Datenbank wird automatisch beim ersten Start des Servers erstellt. Du musst nichts manuell tun!

Hinweis: Die Datei `server/adventskalender.db` wird automatisch generiert und enthält:

- Alle Tabellen (users, calendars, pouches, sessions)
- Ist in `.gitignore` eingetragen (wird nicht committed)

6. Admin-Account erstellen

Wichtig: Ohne Admin-Account kannst du keine Benutzer verwalten!

Schritt 1: Skript ausführen

cd server

deno run --allow-read --allow-write create_admin_user.ts

Schritt 2: Admin-Daten eingeben

Das Skript fragt nach:

1. **Username:** z.B. `admin`
2. **Passwort:** z.B. `admin123` (mindestens 6 Zeichen)
3. **Passwort wiederholen:** Gleiche Eingabe zur Bestätigung

Erfolgreiche Ausgabe:

✅ Admin-Benutzer 'admin' erfolgreich erstellt!

Hinweis: Falls der Benutzer bereits existiert, erhältst du eine Fehlermeldung. Das ist normal und kein Problem.

7. Anwendung starten

Du benötigst **zwei Terminal-Fenster** (oder Tabs), da Frontend und Backend parallel laufen müssen.

Option A: Automatisiert mit Skript (Windows)

Im Projekt-Hauptverzeichnis

.\start-dev.ps1

Das Skript startet automatisch:

1. Backend auf Port 8000
2. Frontend auf Port 5173

Option B: Manuell (empfohlen für Debugging)

Terminal 1: Backend starten

Im Projekt-Hauptverzeichnis

cd server

deno run --allow-net --allow-read --allow-write server.ts

Erfolgreiche Ausgabe:

- Datenbank initialisiert

- Server läuft auf http://localhost:8000

- API verfügbar unter http://localhost:8000/api/

Terminal 2: Frontend starten

Im Projekt-Hauptverzeichnis (neues Terminal!)

npm run dev

Erfolgreiche Ausgabe:

VITE v5.x.x ready in XXX ms

➜ Local: http://localhost:5173/

➜ Network: use --host to expose

➜ press h + enter to show help

Anwendung ist bereit!

Öffne deinen Browser und gehe zu: **http://localhost:5173/**

8. Anwendung nutzen

Erster Login als Admin

1. Öffne **http://localhost:5173/**
2. Klicke auf **"Noch kein Account? Hier registrieren"** ODER logge dich mit dem erstellten Admin-Account ein:
    - **Username:** `admin` (oder was du eingegeben hast)
    - **Passwort:** `admin123` (oder was du eingegeben hast)
3. Du landest im Dashboard

Benutzer verwalten (als Admin)

1. Im Dashboard: Klicke auf **"👑 Admin"**
2. Hier kannst du:
    - Neue Benutzer anlegen
    - Benutzer löschen
    - Rollen ändern (user ↔ admin)

Kalender erstellen

1. Im Dashboard: Klicke auf **"+ Neuer Kalender"**
2. Fülle das Formular aus:
    - **Name:** z.B. "Weihnachten 2025"
    - **Beschreibung:** Optional, z.B. "Für die Familie"
3. Klicke **"Speichern"**
4. Der Kalender wird mit 24 leeren Säckchen erstellt

Säckchen befüllen

1. Klicke auf einen Kalender
2. Scrolle zu den Säckchen
3. Klicke in ein Säckchen, um es zu bearbeiten:
    - **Inhalt:** z.B. "Schokolade"
    - **Notiz:** z.B. "Lindt Pralinés"
4. Änderungen werden automatisch gespeichert
5. Häkchen setzen, wenn das Säckchen gepackt ist

Weitere Funktionen

- **Mischen:** Verteilt Inhalte zufällig neu
- **Exportieren:** Download als JSON oder CSV
- **Fortschritt:** Siehst du in der Kalender-Karte (z.B. "17/24 gepackt")

9. Problemlösungen

Problem: "deno: command not found"

Lösung:

```
# Windows: Deno zum PATH hinzufügen

$env:Path += ";$env:USERPROFILE\.deno\bin"

# Terminal neu starten
```

Problem: "Port 8000 already in use"

Lösung:

```
# Prozess finden und beenden (Windows)

Get-Process | Where-Object {$_.ProcessName -like '*deno*'} | Stop-Process -Force

# Oder Port ändern in server/server.ts (Zeile mit "serve")
```

Problem: "Port 5173 already in use"

Lösung:

```
# Prozess finden und beenden (Windows)

Get-Process | Where-Object {$_.ProcessName -like '*node*'} | Stop-Process -Force

# Oder Vite wählt automatisch einen anderen Port
```

Problem: "CORS error" im Browser

Lösung:

- Stelle sicher, dass Backend auf Port 8000 läuft
- Stelle sicher, dass Frontend auf Port 5173 läuft
- CORS ist bereits konfiguriert, sollte automatisch funktionieren

Problem: "Database locked"

Lösung:

Backend neu starten (Ctrl+C, dann erneut starten)

cd server

deno run --allow-net --allow-read --allow-write server.ts

Problem: "Cannot find module" (Frontend)

Lösung:

node_modules löschen und neu installieren

Remove-Item -Recurse -Force node_modules

npm install

Problem: Login funktioniert nicht

Lösung:

- Überprüfe, ob Backend läuft (http://localhost:8000/api/)
- Überprüfe Browser-Konsole (F12) auf Fehler
- Session-Cookie könnte abgelaufen sein → Browser-Cookies löschen
- Admin-Account neu erstellen (siehe Schritt 6)

Problem: Datenbank zurücksetzen

Lösung:

```
# Im server-Verzeichnis

Remove-Item adventskalender.db

# Server neu starten (erstellt neue leere DB)

deno run --allow-net --allow-read --allow-write server.ts

# Admin-Account neu erstellen

deno run --allow-read --allow-write create_admin_user.ts
```

Server stoppen

Windows (PowerShell)

```
# Alle Deno- und Node-Prozesse beenden

Get-Process | Where-Object {$_.ProcessName -like '*deno*' -or $_.ProcessName -like '*node*'} | Stop-Process -Force
```

macOS/Linux (Terminal)

```
# Mit Ctrl+C im jeweiligen Terminal

# Oder:

pkill -f "deno run"

pkill -f "vite"
```

B. Benutzerdokumentation

1. Einführung

Was ist der Adventskalender-Manager?

Der Adventskalender-Manager ist eine Webanwendung zur Planung und Verwaltung von handgemachten Adventskalendern. Mit dieser Anwendung kann man:

- **Mehrere Kalender** gleichzeitig verwalten
- **24 Säckchen** pro Kalender befüllen und organisieren
- **Fortschritt verfolgen** (z.B. "17/24 gepackt")
- **Inhalte mischen** für Überraschungseffekt
- **Daten exportieren** als JSON oder CSV
- **Multi-User-fähig** - jeder sieht nur seine eigenen Kalender

Für wen ist diese Anwendung?

Meine Mutter verkauft in ihrem Etsy-Shop selbstgenähte Adventskalender. Die App könnte sie ihren Kundinnen zeigen (z. B. als Link auf ihrer Website), damit diese das Befüllen planen. Das bringt echten Nutzen und zusätzlich Aufmerksamkeit für den Shop. Dennoch ist dieses Feature auch für andere nützlich:

- Etsy-Shop-Betreiber, die Kunden beim Befüllen unterstützen
- Familien, die gemeinsam Adventskalender planen
- Alle, die den Überblick über 24 kleine Geschenke behalten möchten
- Hobby-Bastler, die handgemachte Adventskalender erstellen

Hauptfunktionen auf einen Blick

1. **Registrierung & Login** - Sicherer Zugang mit eigenem Account
2. **Kalender-CRUD** - Erstellen, Bearbeiten, Löschen von Kalendern
3. **Säckchen-Verwaltung** - 24 individuell befüllbare Säckchen
4. **Status-Tracking** - Markiere gepackte Säckchen
5. **Zufallsgenerator** - "Mischen"-Feature verteilt Inhalte neu
6. **Datenexport** - Sichere deine Daten als JSON oder CSV
7. **Admin-Bereich** - Benutzerverwaltung (nur für Admins)

2. Erste Schritte

2.1 Registrierung

So erstellst du einen Account:

1. Öffne die Anwendung im Browser: `http://localhost:5173/`
2. Du siehst die Login-Seite
3. Klicke auf **"Noch kein Account? Hier registrieren"**
4. Fülle das Formular aus:
    - **Benutzername:** 3-20 Zeichen, nur Buchstaben, Zahlen, Unterstrich
    - **Passwort:** Mindestens 6 Zeichen
    - **Passwort wiederholen:** Zur Bestätigung
5. Klicke auf **"Registrieren"**
6. Bei Erfolg wirst du automatisch eingeloggt und zum Dashboard weitergeleitet

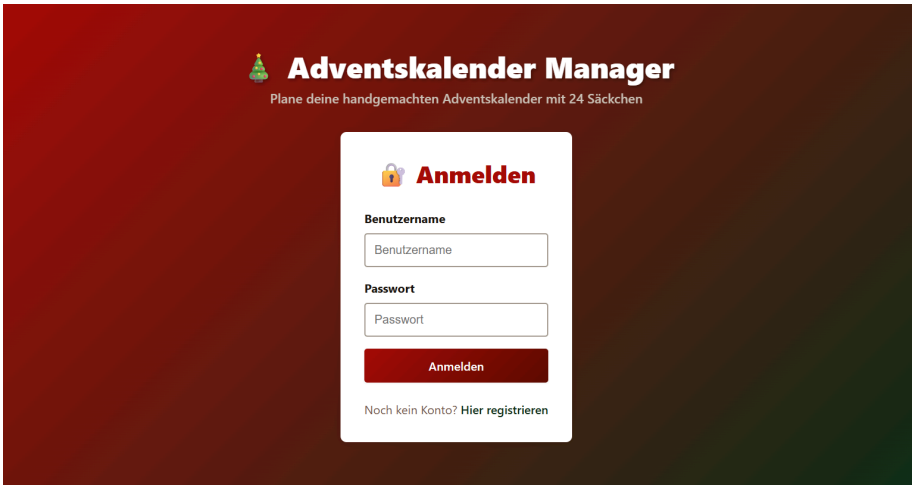
Hinweise:

- Benutzernamen müssen eindeutig sein
- Passwörter werden verschlüsselt gespeichert (bcrypt)
- Du kannst dich jederzeit wieder einloggen

2.2 Login

So loggst du dich ein:

1. Öffne die Anwendung: `http://localhost:5173/`
2. Gib deine Zugangsdaten ein:
    - **Benutzername**
    - **Passwort**
3. Klicke auf **"Anmelden"**
4. Du wirst zum Dashboard weitergeleitet



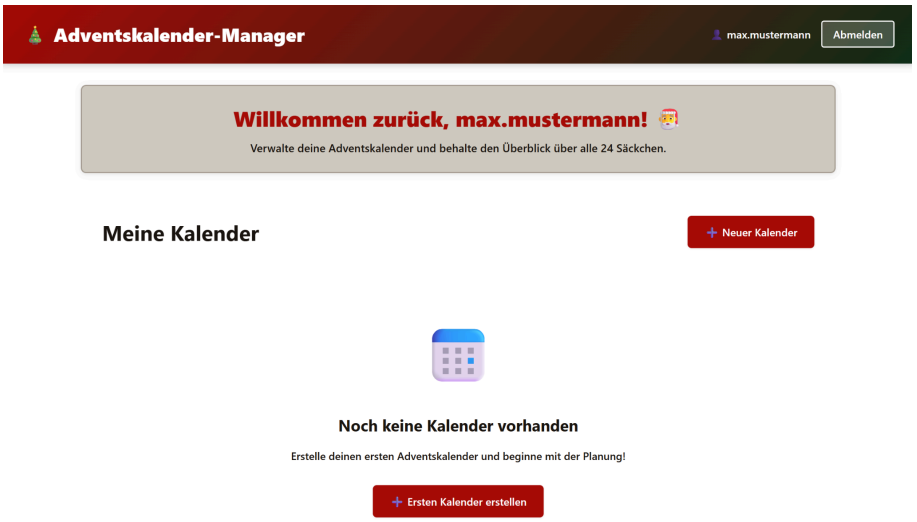
Hinweise:

- Session bleibt 7 Tage gültig (oder bis Logout)
- Bei falschen Zugangsdaten erscheint eine Fehlermeldung

2.3 Dashboard-Übersicht

Nach dem Login landest du im **Dashboard**. Hier siehst du:

- **Willkommens-Nachricht** mit deinem Benutzernamen
- **Navigation:**
  - Dashboard (Startseite)
  - Admin (nur für Admins sichtbar)
  - Abmelden
- **Kalender-Übersicht:**
  - Liste aller deiner Kalender als Karten
  - Button **"+ Neuer Kalender"** zum Erstellen der Kalender
  - Leerer Zustand mit Hinweis, wenn noch keine Kalender vorhanden

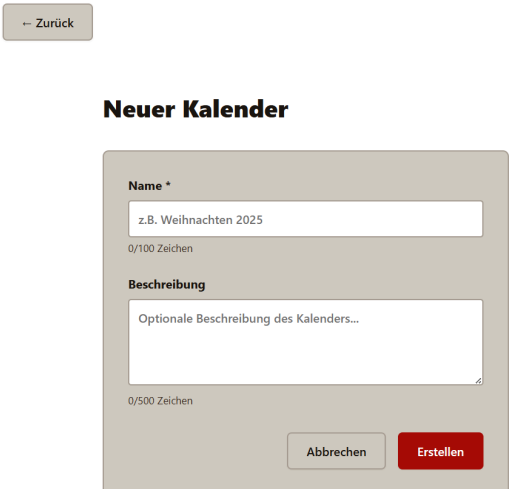


3. Kalender verwalten

3.1 Kalender erstellen

So erstellst du einen neuen Kalender:

1. Klicke im Dashboard auf **"+ Neuer Kalender"**
2. Fülle das Formular aus:
    - **Name:** z.B. "Weihnachten 2025" (3-50 Zeichen, Pflichtfeld)
    - **Beschreibung:** Optional, z.B. "Für die Familie" (max. 200 Zeichen)
3. Klicke auf **"Speichern"**
4. Der Kalender wird erstellt mit:
    - 24 leeren Säckchen (Nummern 1-24)
    - Status: 0/24 gepackt
5. Du wirst zur Detail-Ansicht weitergeleitet



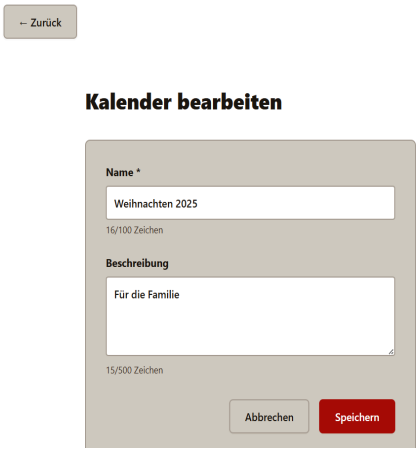
Hinweise:

- Name ist Pflicht, Beschreibung optional
- Zeichenzähler zeigt verbleibende Zeichen an
- Abbrechen-Button führt zurück zum Dashboard

3.2 Kalender bearbeiten

So änderst du Name oder Beschreibung:

1. Navigiere zum Dashboard
2. Finde die Kalender-Karte, die du bearbeiten möchtest
3. Klicke auf den **"✏️ Bearbeiten"**-Button
4. Ändere Name und/oder Beschreibung
5. Klicke auf **"Speichern"**
6. Änderungen werden sofort übernommen



Hinweise:

- Säckchen-Inhalte bleiben unverändert
- Nur Name und Beschreibung sind editierbar
- Abbrechen verwirft alle Änderungen

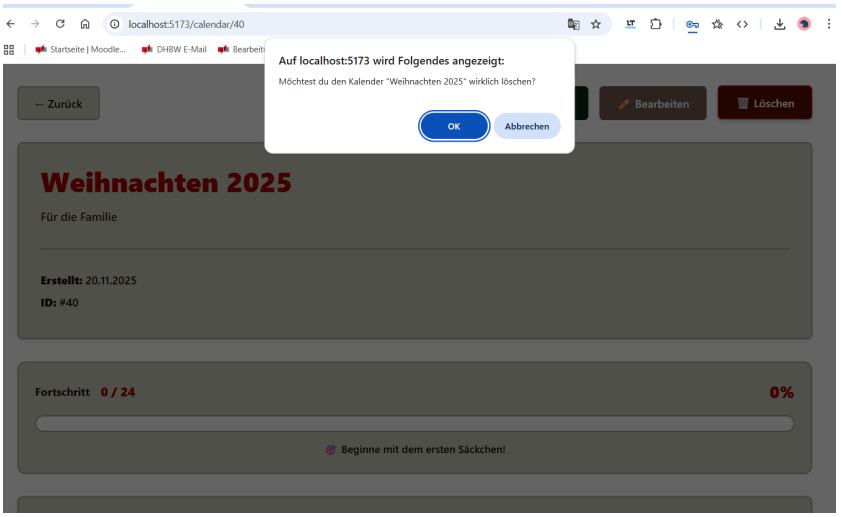
3.3 Kalender löschen

So löschst du einen Kalender:

1. Navigiere zum Dashboard
2. Finde die Kalender-Karte, die du löschen möchtest
3. Klicke auf den "🗑️ **Löschen**"-Button
4. Bestätige die Sicherheitsabfrage
5. Der Kalender wird unwiderruflich gelöscht

⚠️ ACHTUNG:

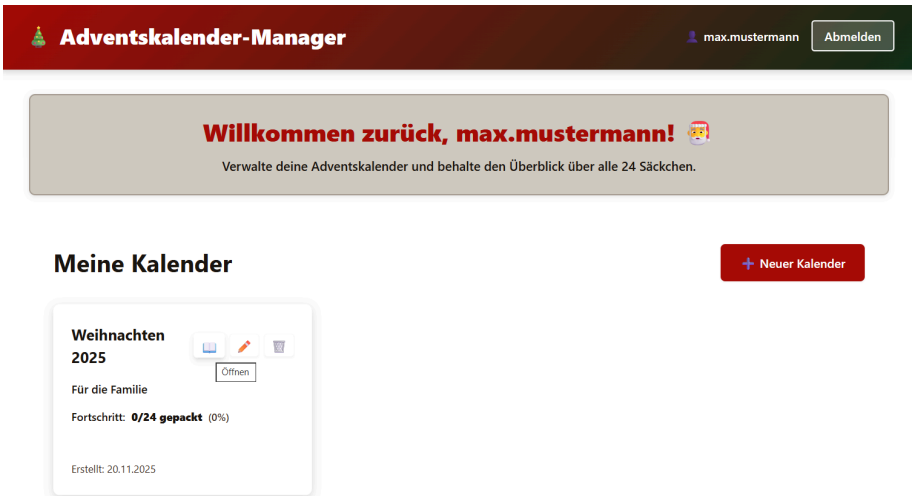
- Gelöschte Kalender können NICHT wiederhergestellt werden!
- Alle 24 Säckchen mit Inhalten werden ebenfalls gelöscht
- Erwäge vorher einen Export (siehe Kapitel 7)



3.4 Kalender öffnen

So öffnest du einen Kalender:

1. Navigiere zum Dashboard
2. Klicke entweder auf:
    - Den **Kalender-Namen** (oben auf der Karte)
    - Den **"📖 Öffnen"**-Button
3. Du gelangst zur Detail-Ansicht mit allen Säckchen

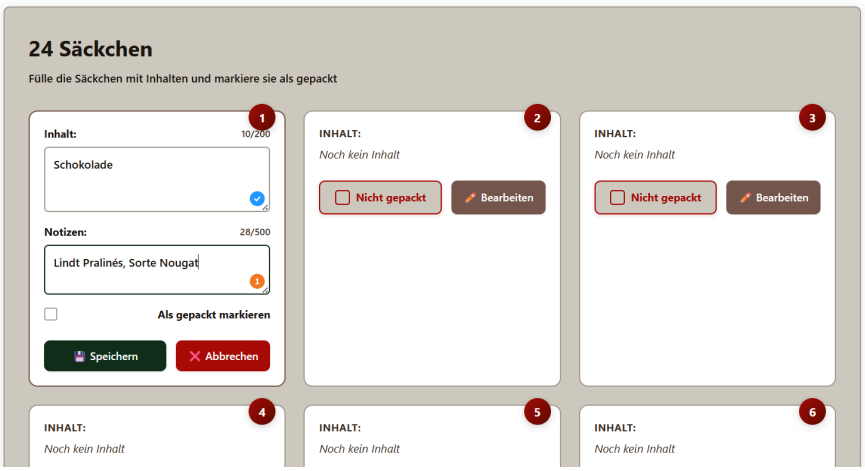


4. Säckchen befüllen

4.1 Säckchen bearbeiten

So befüllst du ein Säckchen:

1. Öffne einen Kalender (siehe 3.4)
2. Scrolle zur Säckchen-Liste (unterhalb des Headers)
3. Du siehst alle 24 Säckchen nummeriert
4. Klicke in ein Säckchen-Feld:
    - **Inhalt:** Gib ein, was in dieses Säckchen kommt (z.B. "Schokolade")
    - **Notiz:** Optional, zusätzliche Infos (z.B. "Lindt Pralinés, Sorte Nougat")
5. Änderungen werden **automatisch gespeichert** (nach 500ms)
6. Ein Häkchen erscheint kurz als Bestätigung



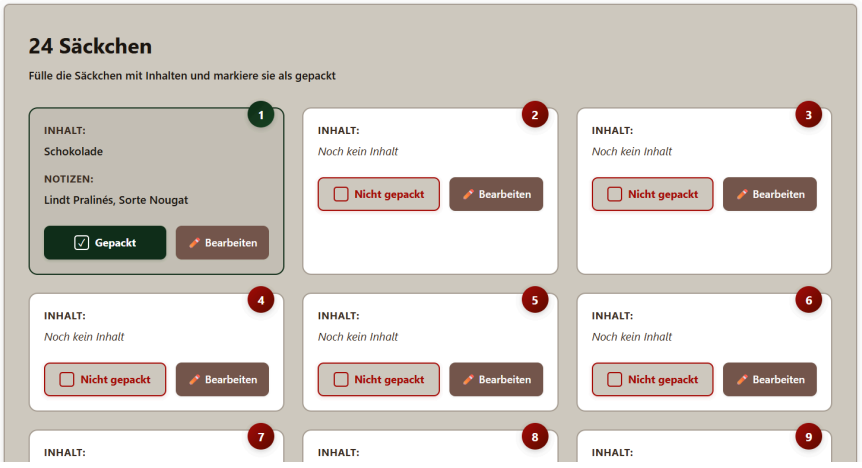
Hinweise:

- Inhaltsfeld: max. 100 Zeichen
- Notizfeld: max. 200 Zeichen
- Änderungen werden live gespeichert (kein "Speichern"-Button nötig)
- Scroll-Position bleibt erhalten beim Bearbeiten

4.2 Gepackt-Status setzen

So markierst du ein Säckchen als gepackt:

1. Öffne einen Kalender
2. Finde das Säckchen, das du gerade gepackt hast
3. Klicke auf den **Button** in der oberen rechten Ecke:
    - **"Nicht gepackt"** (roter Button) → wird zu **"✓ Gepackt"** (grüner Button)
    - **"✓ Gepackt"** (grüner Button) → wird zu **"Nicht gepackt"** (roter Button)
4. Status wird sofort gespeichert
5. Fortschrittsanzeige oben wird aktualisiert



Tipp: Nutze den Gepackt-Status, um den Überblick zu behalten, welche Säckchen bereits physisch befüllt und verschlossen sind.

4.3 Fortschritt einsehen

Die Fortschrittsanzeige findest du an zwei Stellen:

In der Kalender-Karte (Dashboard):

- Progress-Bar mit Farbverlauf (rot → grün)
- Text: "17/24 gepackt" (Beispiel)
- Prozentuale Darstellung als Balken

In der Kalender-Detail-Ansicht:

- Gleiche Anzeige im Header
- Aktualisiert sich live beim Abhaken

Hinweise:

- 0/24 = Progress-Bar ist rot
- 24/24 = Progress-Bar ist grün
- Dazwischen: Farbverlauf von rot nach grün

5. Fortschritt verfolgen

5.1 Übersicht im Dashboard

Das Dashboard zeigt dir auf einen Blick:

- **Anzahl Kalender:** Wie viele Kalender du insgesamt hast
- **Fortschritt pro Kalender:** Jede Karte zeigt den individuellen Status
- **Visuelle Unterscheidung:** Komplett gepackte Kalender haben eine grüne Progress-Bar

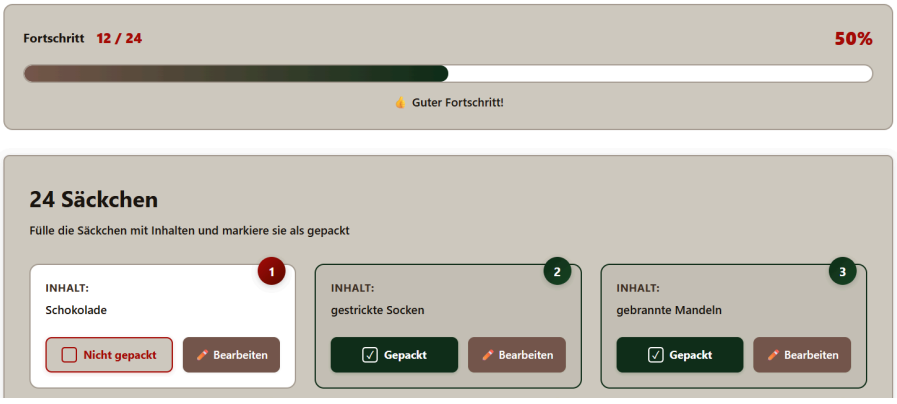
Beispiel-Szenario:

- Kalender "Familie": 24/24 gepackt ✅ (fertig)
- Kalender "Freunde": 17/24 gepackt 🔄 (in Arbeit)
- Kalender "Kolleg:innen": 0/24 gepackt ⏳ (noch nicht gestartet)

5.2 Detaillierte Ansicht

In der Kalender-Detail-Ansicht siehst du:

- **Welche Säckchen** bereits gepackt sind (grüner Button)
- **Welche Säckchen** noch offen sind (roter Button)
- **Welche Säckchen** bereits befüllt sind (haben Inhalt)
- **Welche Säckchen** noch leer sind (kein Inhalt)



Tipp: Scrolle durch die Liste und konzentriere dich auf die roten "Nicht gepackt"-Buttons, um zu sehen, was noch zu tun ist.

6. Mischen-Funktion

6.1 Was macht die Mischen-Funktion?

Die **Mischen-Funktion** verteilt die Inhalte deiner 24 Säckchen **zufällig neu** auf die Nummern 1-24. Das ist nützlich, wenn du:

- Die Reihenfolge der Geschenke ändern möchtest
- Einen Überraschungseffekt haben willst

Wichtig:

- Nur die **Zuordnung von Inhalt zu Nummer** wird geändert
- Die **Inhalte selbst** bleiben erhalten (nichts geht verloren)
- Der **Gepackt-Status** bleibt bei den Nummern (nicht bei den Inhalten)

Beispiel:

Vorher:

Säckchen 1: "Schokolade" (gepackt)

Säckchen 2: "Gutschein" (nicht gepackt)

Säckchen 3: "Teebeutel" (gepackt)

Nachher (nach Mischen):

Säckchen 1: "Gutschein" (gepackt) ← war vorher bei 2

Säckchen 2: "Teebeutel" (nicht gepackt) ← war vorher bei 3

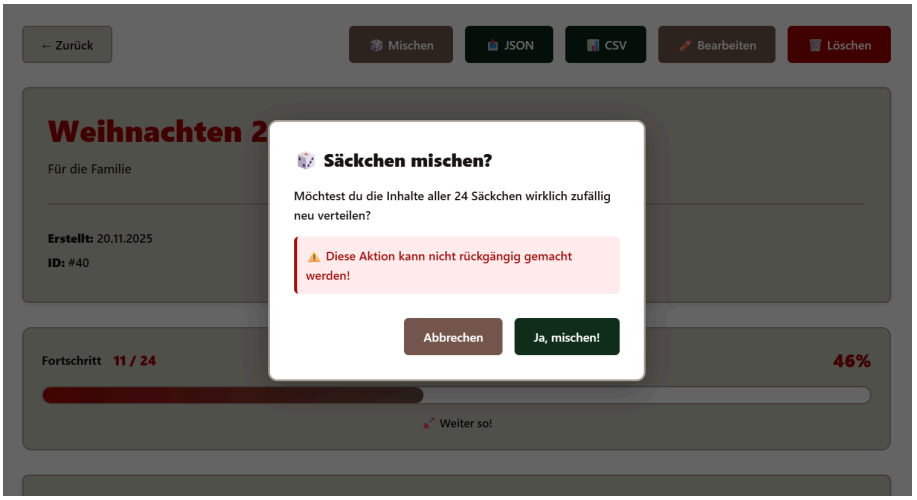
Säckchen 3: "Schokolade" (gepackt) ← war vorher bei 1

6.2 Mischen durchführen

So mischst du einen Kalender:

1. Öffne den Kalender, den du mischen möchtest
2. Scrolle nach oben zum Header-Bereich
3. Klicke auf den **"🔀 Mischen"**-Button (orange)
4. Ein Bestätigungsdialog erscheint:
    - **Text:** "Möchtest du die Inhalte dieses Kalenders wirklich neu mischen?"
    - **Hinweis:** "Die Zuordnung von Inhalten zu Säckchen-Nummern wird zufällig neu verteilt."
5. Klicke auf **"Ja, mischen"** oder **"Abbrechen"**

6. Bei Bestätigung:
    - Button zeigt "🔄 Wird gemischt..."
    - Server berechnet neue Zuordnung (Fisher-Yates-Algorithmus)
    - Säckchen-Liste wird neu geladen
    - Erfolgs-Meldung: "✅ Kalender erfolgreich gemischt!"



Hinweise:

- Mischen ist **nicht rückgängig** zu machen
- Du kannst mehrfach hintereinander mischen
- Jedes Mal entsteht eine neue zufällige Verteilung
- Leere Säckchen bleiben leer, werden aber auch neu verteilt

6.3 Wann sollte ich mischen?

Gute Gründe zum Mischen:

- ✅ Du hast alle Inhalte eingegeben, aber noch nichts gepackt
- ✅ Du möchtest die Reihenfolge ändern, ohne alles manuell umzusortieren
- ✅ Du willst, dass die "großen" Geschenke nicht alle am Anfang sind
- ✅ Überraschungseffekt für den Empfänger

NICHT mischen, wenn:

- ❌ Du bereits einige Säckchen physisch gepackt hast (es sei denn, du willst sie umetikettieren)
- ❌ Du eine bestimmte Reihenfolge haben möchtest (z.B. steigender Wert)
- ❌ Die Nummern bereits auf den physischen Säckchen stehen

7. Export-Funktion

7.1 Wozu Exportieren?

Die Export-Funktion erstellt eine **Sicherungskopie** deiner Kalender-Daten. Du kannst die Dateien:

- Auf deinem Computer speichern
- Per E-Mail versenden
- In Excel/LibreOffice öffnen (CSV)
- Als Backup verwenden
- Für Analysen nutzen

7.2 JSON-Export

Was ist JSON?

- JSON = JavaScript Object Notation
- Enthält **alle Daten** des Kalenders inkl. Metadaten
- Maschinenlesbar, strukturiert
- Gut für Backups und Datenverarbeitung

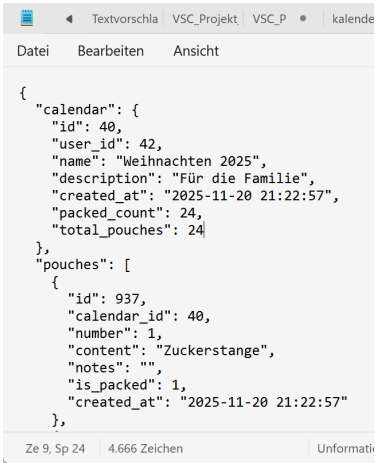
So exportierst du als JSON:

1. Öffne den Kalender, den du exportieren möchtest
2. Scrolle nach oben zum Header-Bereich
3. Klicke auf "📥 **JSON**"-Button
4. Der Download startet automatisch
5. Dateiname: `kalender-export-nummer.json`

Beispiel-Dateiname:

kalender_export_1763590490329.json

Inhalt (Beispiel):



7.3 CSV-Export

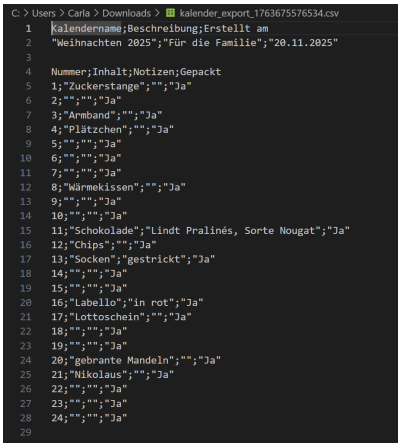
Was ist CSV?

- CSV = Comma-Separated Values
- Einfache **Tabelle** mit Säckchen-Daten
- Öffnet sich in Excel, LibreOffice Calc, Google Sheets
- Gut für Übersicht und Ausdruck

So exportierst du als CSV:

1. Öffne den Kalender
2. Klicke auf "📊 **CSV**"-Button
3. Der Download startet automatisch
4. Dateiname: `kalender-export-nummer.csv`

Inhalt (Beispiel):



So öffnest du die CSV:

- Doppelklick öffnet Excel/Calc/VSC
- Import als Tabelle mit Spalten:
  - Nummer (1-24)
  - Inhalt
  - Notizen
  - Gepackt (Ja/Nein)

7.4 Export-Tipps

Best Practices:

- ✅ Exportiere **vor dem Mischen**, um die alte Reihenfolge zu behalten
- ✅ Exportiere **vor dem Löschen** eines Kalenders
- ✅ Erstelle regelmäßig Backups (z.B. wöchentlich)
- ✅ Nutze JSON für vollständige Backups
- ✅ Nutze CSV für Ausdrucke oder Excel-Bearbeitung

8. Admin-Funktionen

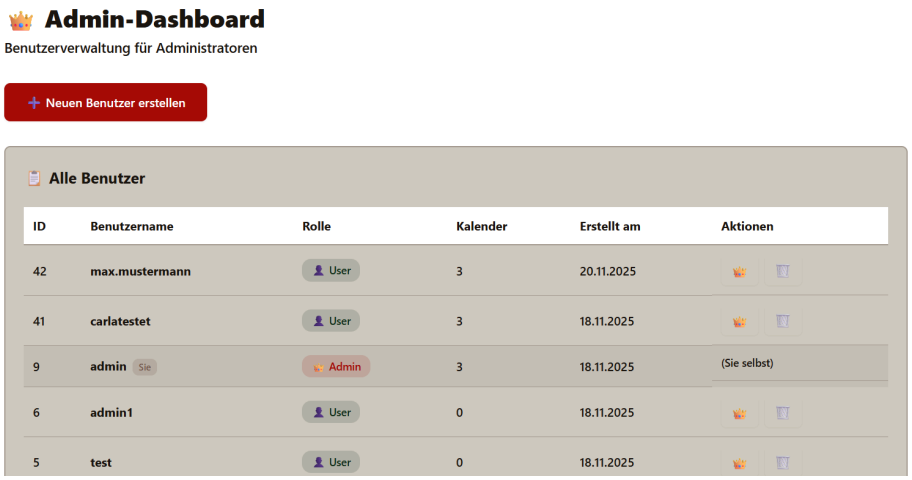
8.1 Admin-Bereich öffnen

Nur für Benutzer mit Admin-Rolle!

So gelangst du zum Admin-Bereich:

1. Logge dich mit einem Admin-Account ein
2. Im Dashboard siehst du einen zusätzlichen Button: "👑 **Admin**"
3. Klicke auf diesen Button
4. Du gelangst zum Admin-Dashboard

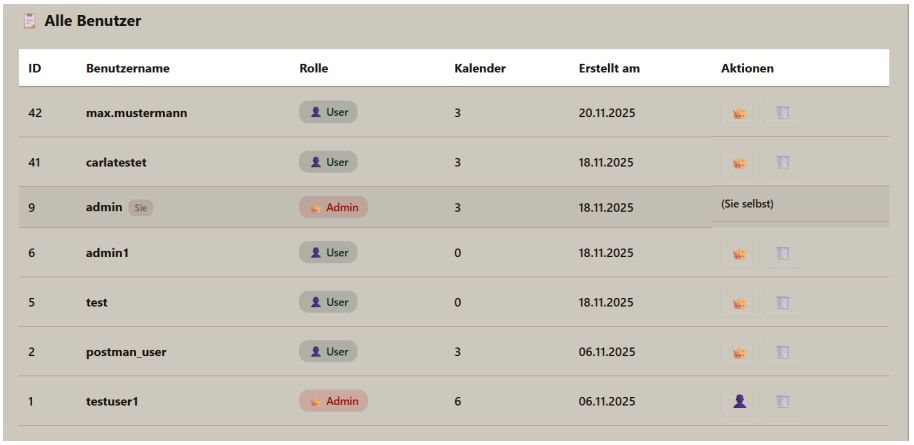
Hinweis: Normale Benutzer sehen diesen Button NICHT.



8.2 Benutzer anzeigen

Das Admin-Dashboard zeigt eine **Liste aller registrierten Benutzer**:

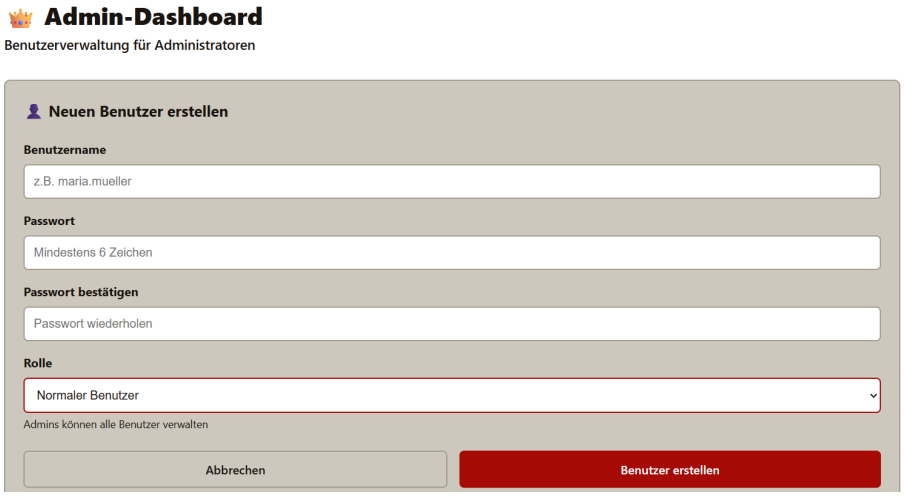
Spalte	Beschreibung
ID	Eindeutige Benutzer-ID
Benutzername	Login-Name des Users
Rolle	`user` oder `admin`
Erstellt am	Registrierungsdatum
Aktionen	Buttons zum Bearbeiten/Löschen



8.3 Neuen Benutzer anlegen

So erstellst du einen neuen Benutzer als Admin:

1. Klicke im Admin-Dashboard auf **"+ Neuer Benutzer"**
2. Fülle das Formular aus:
    - **Benutzername:** 3-20 Zeichen
    - **Passwort:** Mindestens 6 Zeichen
    - **Rolle:** Wähle `user` oder `admin`
3. Klicke auf **"Erstellen"**
4. Der neue Benutzer wird angelegt und erscheint in der Liste



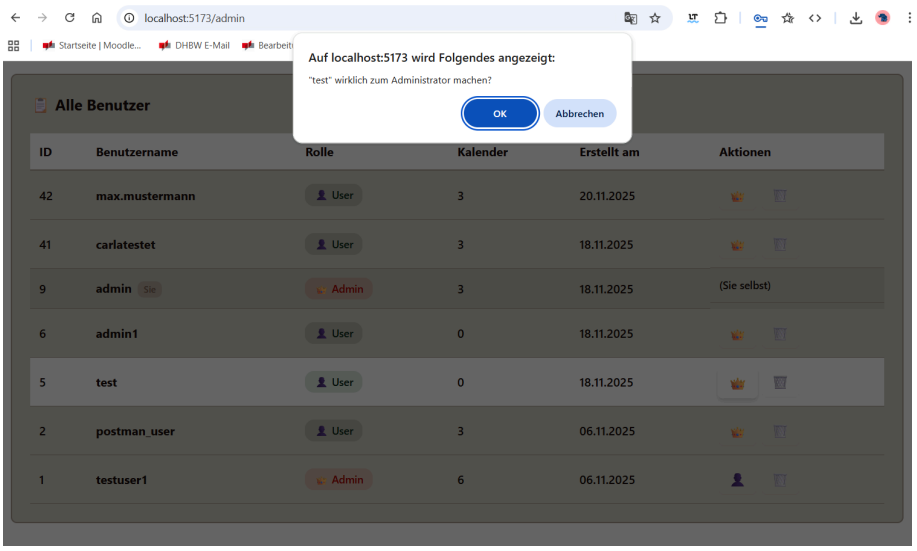
Hinweise:

- Als Admin kannst du andere Admins erstellen
- Der neue Benutzer kann sich sofort einloggen
- Initiales Passwort sollte dem Benutzer sicher mitgeteilt werden

8.4 Benutzer-Rolle ändern

So änderst du die Rolle eines Benutzers:

1. Finde den Benutzer in der Liste
2. Klicke auf den "👤/👑"-Button (Rolle ändern)
3. Die Rolle wechselt zwischen:
    - `user` → `admin` (Benutzer bekommt Admin-Rechte)
    - `admin` → `user` (Admin wird zum normalen Benutzer)
4. Änderung wird sofort gespeichert



Wichtig:

- ⚠️ Du kannst deine **eigene Rolle NICHT ändern** (Schutz vor versehentlichem Rechte-Verlust)
- Ein Admin kann andere Admins degradieren (Vorsicht!)

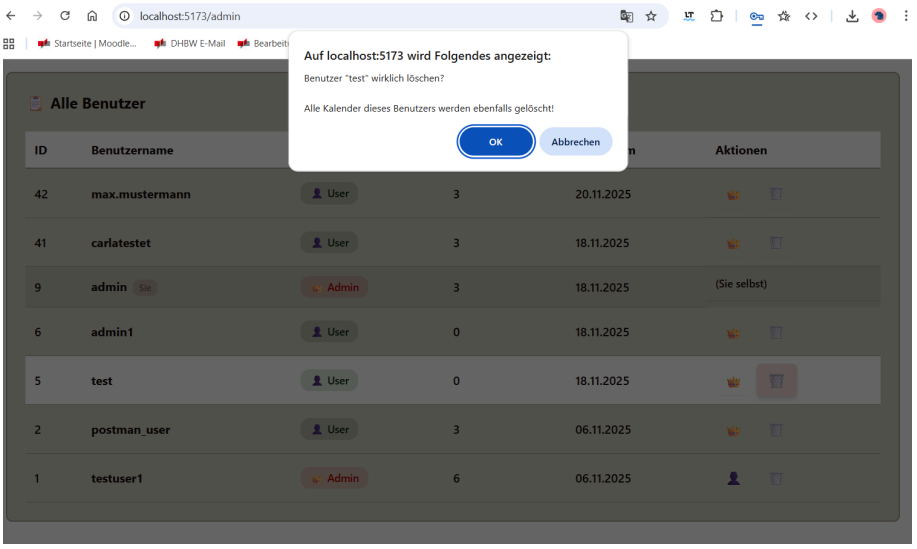
8.5 Benutzer löschen

So löschst du einen Benutzer:

1. Finde den Benutzer in der Liste
2. Klicke auf den "🗑️ **Löschen**"-Button
3. Bestätige die Sicherheitsabfrage
4. Der Benutzer wird gelöscht

⚠️ **ACHTUNG:**

- Du kannst dich **selbst NICHT löschen** (Schutz)
- Alle Kalender und Säckchen des Benutzers werden **ebenfalls gelöscht**!
- Dieser Vorgang ist **nicht umkehrbar**!



Best Practice:

- Informiere den Benutzer vor dem Löschen
- Biete dem Benutzer an, seine Daten zu exportieren
- Überlege, ob Deaktivierung statt Löschen sinnvoller wäre (aktuell nicht implementiert)

8.6 Admin-Rechte und Verantwortung

Was kann ein Admin?

- ✅ Alle Benutzer sehen
- ✅ Neue Benutzer anlegen
- ✅ Benutzer-Rollen ändern
- ✅ Benutzer löschen
- ✅ Zugriff auf Admin-Dashboard

Was kann ein Admin NICHT?

- ❌ Kalender anderer Benutzer sehen/bearbeiten
- ❌ Sich selbst löschen
- ❌ (In der aktuellen Version) Passwörter zurücksetzen (nur: Benutzer löschen und neu anlegen)

Verantwortung:

- Behandle Admin-Rechte verantwortungsvoll
- Lösche Benutzer nur nach Rücksprache
- Erstelle nicht unnötig viele Admins
- Sichere deine Admin-Zugangsdaten besonders gut

9. Tipps und Tricks

9.1 Workflow-Empfehlungen

Optimal Workflow für einen neuen Kalender:

1. **Kalender erstellen**
    - Name und Beschreibung eingeben
    - Kalender öffnen
2. **Inhalte sammeln (ohne Reihenfolge)**
    - Alle 24 Säckchen-Inhalte eingeben
    - Notizen hinzufügen (z.B. Kaufquelle, Preis)
    - NOCH NICHT "gepackt" markieren
3. **Mischen (optional)**
    - Wenn gewünscht: Inhalte zufällig verteilen
    - So entsteht eine überraschende Reihenfolge
4. **Exportieren (Backup)**
    - JSON-Export als Sicherung
    - CSV-Export für Übersicht beim Packen
5. **Physisch packen**
    - Drucke CSV aus oder öffne App auf Tablet
    - Packe Säckchen der Reihe nach
    - Hake jedes gepackte Säckchen ab
    - Verfolge Fortschritt (z.B. "17/24 gepackt")
6. **Finaler Export**
    - Nach Fertigstellung: Erneuter Export als Archiv

9.2 Fehler vermeiden

Häufige Fehler und wie du sie vermeidest:

Fehler	**Vermeidung**
Kalender versehentlich gelöscht	✅ Regelmäßig exportieren
Nach Mischen: alte Reihenfolge weg	✅ Vor dem Mischen exportieren

Fehler	Vermeidung
Säckchen überschrieben	✅ Vorsicht beim Editieren (Auto-Save nach 500ms)
Vergessen, Gepackt-Status zu setzen	✅ Direkt nach Packen abhaken
Passwort vergessen	✅ Admin kann neuen Account erstellen

9.3 Performance-Tipps

Bei vielen Kalendern:

- Die App ist optimiert für bis zu ~50 Kalender pro Benutzer
- Dashboard lädt alle Kalender auf einmal
- Bei Verzögerungen/Langsamkeit: Alte Kalender exportieren und löschen

Bei langsamer Internetverbindung:

- Änderungen werden zwischengespeichert (Browser-Cache)
- Session bleibt 7 Tage gültig (kein ständiges Neueinloggen)

9.4 Mobile Nutzung

Responsive Design:

- App funktioniert auf Smartphones
- Optimierte Touch-Bedienung
- Angepasste Layouts für kleine Bildschirme

Tipp: Öffne die App auf deinem Tablet/Laptop beim physischen Packen der Säckchen!

9.7 Drucken

CSV für Druckliste:

1. Exportiere Kalender als CSV
2. Öffne in Excel/LibreOffice
3. Formatiere nach Wunsch:
    - Spaltenbreiten anpassen
    - Schriftgröße erhöhen
    - Farben hinzufügen
4. Drucke als Checkliste
5. Hake während des Packens ab

10. Häufig gestellte Fragen (FAQ)

Allgemein

F: Kostet die Anwendung etwas? A: Nein, die Anwendung ist kostenlos und Open Source (siehe GitHub).

F: Sind meine Daten sicher? A: Ja, Passwörter werden verschlüsselt (bcrypt), und jeder Benutzer sieht nur seine eigenen Daten.

F: Kann ich die App offline nutzen? A: Nein, die App benötigt eine Verbindung zum Backend-Server (läuft lokal oder auf Deno Deploy).

F: Wie viele Kalender kann ich erstellen? A: Theoretisch unbegrenzt, praktisch empfohlen: bis zu 50 Kalender pro Benutzer.

Account & Login

F: Ich habe mein Passwort vergessen. Was tun? A: Kontaktiere einen Admin. Der Admin kann deinen Account löschen, und du kannst dich neu registrieren.
(Passwort-Reset-Funktion noch nicht implementiert)

F: Kann ich meinen Benutzernamen ändern? A: Nein, aktuell nicht. Du müsstest einen neuen Account erstellen.

F: Wie lange bleibt meine Session gültig? A: 7 Tage oder bis zum manuellen Logout.

F: Kann ich von mehreren Geräten gleichzeitig eingeloggt sein? A: Technisch ja, aber nur eine aktive Session pro Account. Neuer Login invalidiert alte Session.

Kalender & Säckchen

F: Kann ich mehr als 24 Säckchen haben? A: Nein, die App ist auf 24 Säckchen (Adventskalender-Standard) limitiert.

F: Kann ich die Nummern der Säckchen ändern? A: Nein, die Nummern 1-24 sind fest. Du kannst aber die Inhalte mit "Mischen" neu zuordnen.

F: Was passiert beim Mischen mit meinen Notizen? A: Notizen bleiben beim Inhalt. Wenn "Schokolade + Notiz" von Säckchen 1 zu Säckchen 10 wandert, kommt die Notiz mit.

F: Kann ich Säckchen einzeln löschen? A: Nein, aber du kannst den Inhalt einfach leeren (löschen von Text).

F: Kann ich Säckchen kopieren/duplizieren? A: Nein, aber du kannst Text manuell kopieren (Strg+C / Strg+V).

Export & Backup

F: Kann ich einen exportierten Kalender wieder importieren? A: Nein, aktuell nicht. Export ist nur für Sicherung/Archivierung gedacht.

F: Wo werden die Exporte gespeichert? A: In deinem Standard-Download-Ordner (Browser-Einstellung).

F: Was ist besser: JSON oder CSV? A: JSON für vollständiges Backup, CSV für Ausdrucke und Excel-Bearbeitung.

Admin-Funktionen

F: Wie werde ich Admin? A: Entweder beim ersten Setup (mit `create_admin_user.ts`) oder ein bestehender Admin ändert deine Rolle.

F: Kann ich als Admin die Kalender anderer Benutzer sehen? A: Nein, Datenisolation gilt auch für Admins.

F: Was passiert, wenn ich versehentlich alle Admins lösche/degradiere? A: Nutze das Server-Skript `create_admin_user.ts`, um einen neuen Admin anzulegen.

Technisches

F: Welche Browser werden unterstützt? A: Alle modernen Browser (Chrome, Firefox, Edge, Safari). Getestet mit Chrome 120+ und Firefox 121+.

F: Kann ich die App auf einem Server hosten? A: Ja, die App ist für Deno Deploy vorbereitet. Siehe `docs/TECHNISCHE_DOKUMENTATION.md`.

F: Ist die App Open Source? A: Ja, siehe GitHub: https://github.com/carloerbse/adventskalender-manager

F: Wo kann ich Fehler melden oder Features vorschlagen? A: Aktuell: Kontakt über DHBW. In Zukunft: GitHub Issues.