



# GetraenkeIO: Eine Getränkelagerverwaltung mit Bestandsstatistik für Vereinsheime


Technischer Bericht: CL-TR-2025-42, selfref 2025


Dotzler Martin, Ehrles Andreas, Taach Eduard, Wegerer Nikolas,  
Weinhut Justin, Christoph P. Neumann   
CyberLytics-Lab an der Fakultät Elektrotechnik, Medien und Informatik  
Ostbayerische Technische Hochschule Amberg-Weiden  
Amberg, Deutschland


**Zusammenfassung**—Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. {  The abstract does neither mention a teaching module nor a team/project, it is a summary of the content, thus, the objectives and architecture. Do NOT remove the abstract , this section is mandatory. You should consider comparing your self-written abstract with the result of a generative AI that summarizes your content after you have written a nearly stable draft version. However, do not use a verbatim copy to replace your abstract, just use generative AI for inspirational purposes. }


**Index Terms**—D.2.0.c Software engineering for Internet projects, D.2.2.c Distributed/Internet based software engineering tools and techniques


## I. EINLEITUNG UND PROJEKTZIELE

The cs-techrep formatting is adopted both from IEEE [iee2018formattingrules] and IARIA [aria2014formattingrules] styles. The cs-techrep L<sup>A</sup>T<sub>E</sub>X class is based on IEEEtran class [iee2015howto]. In addition, be aware of the supplementary IARIA editorial rules [aria2009editorialrules]  that provide a beginner-friendly set of further advices. It is recommended to use a grammar tool, e. g., the LanguageTool [languagetool] browser plugin in combination with Overleaf [overleaf].

The title of your paper should not exceed two lines . In exceptional cases, three lines might be allowed. A four-line-title is absolutely forbidden (hint: use the longer form in the abstract).

For capitalization of titles and section headings, use a web tool like Capitalize My Title  with the option `chicago` for capitalization rules by Chicago Manual of Style (CMOS).

The pipe symbol „|“ in the section headings represents alternatives! Choose one and remove the others . The selectively provided quoted terms are special German alternatives. You may deviate from the structure of this example document and its exemplary section headings.

The problem statement needs to be written from perspective of a subject-matter expert („Fachkonzept“). Like an elevator pitch / mission statement  and NOT from a technical perspective.

## II. VERWANDTE ARBEITEN

Die Verwaltung von Getränken ist in vielen Vereinen eine Notwendige Aufgabe. Daher ist es nicht verwunderlich, dass es für eine leichtere Handhabung dieses Vorgangs bereits bestehende Anwendungen gibt, welche eine Vereinfachung versprechen. Bei diesem Vergleich interessieren uns vor allem die Unterschiede, insbesondere jene im Bereich der Monetarisierung. Alle drei hier betrachteten Anwendungen basieren auf einer teilweise zahlungspflichtigen Strategie. So ist es bei der Getränke Zähler App [drinkscouter], der Getränkliste App [getraenkelisteapp] und dem Getränkewart 2.0 [getraenkewart] jeweils möglich je nach Anbieter mit bis zu 3 bis 10 Nutzern die Anwendung jeweils kostenlos zu nutzen. Sollte die App jedoch von mehr Benutzern verwendet werden, so werden jeweils monatliche oder jährliche Kosten für die Nutzung fällig. Dies ist ein großer Unterschied zu GetraenkeIO, da diese nicht nur kostenfrei nutzbar ist, sondern dank der MIT Lizenz sogar Open Source. Im Gegenteil zu den drei anderen genannten Lösungen ist es bei GetraenkeIO deshalb ebenfalls möglich die Software auf eigener Hardware laufen zu lassen und so die komplette Kontrolle über die eigenen Daten zu erhalten.

## III. ARCHITEKTURZIELE

Provides (1) a visualization of the external systems and users with which the system interacts („Kontextabgrenzung“), (2) the most important technical and organizational preconditions („Rahmenbedingungen“), (3) quality/non-functional requirements („Qualitätsziele“), and/or (4) architectural style design decisions with formative patterns of the solution („Architekturstil“) as well as (5) the applied programming language(s).

## IV. ARCHITEKTUR VON GETRAENKEIO

### A. Technologie-Stack

Die GetraenkeIO-Anwendung soll auf einer dreischichtigen Architektur, bestehend aus Frontend, Backend und Datenhaltungsschicht bestehen. Jede Schicht soll von den anderen abgegrenzt auf einem eigenen Container [conatiners] laufen. Das Frontend besteht aus einer React-Anwendung [react]. Als Backend wird eine Python-Anwendung [python] verwendet. Diese benutzt das Web-Framework FastAPI [fastapi] und stellt

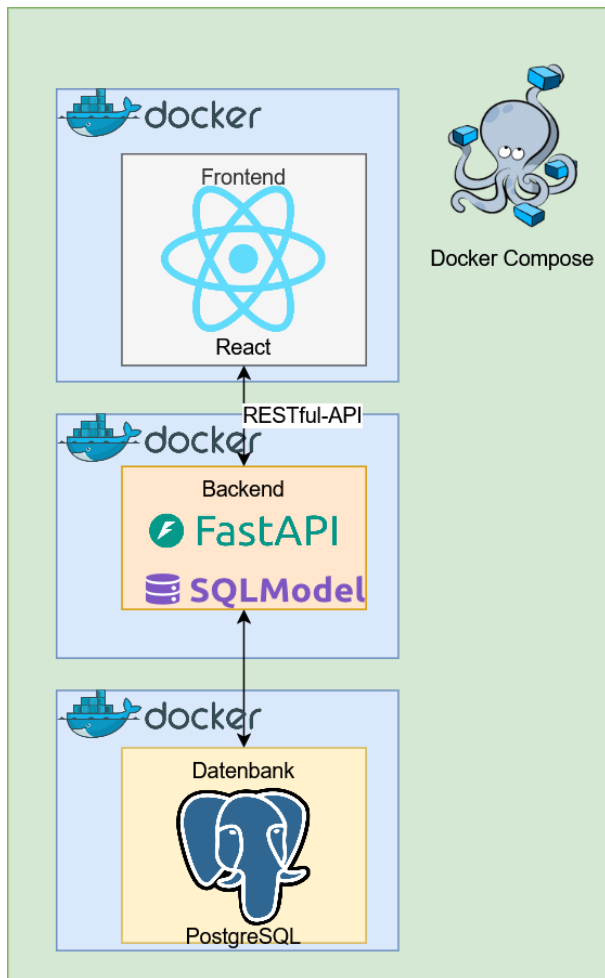


Abbildung 1. Bausteinsicht über den Technologie-Stack und die verwendeten Technologien und deren Zusammenspiel.

damit dem Frontend eine RESTful-Schnittstelle [restful] zum Datenaustausch zur Verfügung. Zur Datenhaltung wird die relationale Datenbank PostgreSQL [postgresql] genutzt. Die Kommunikation zwischen Datenbank und Backend übernimmt das Framework SQLAlchemy [sqlmodel], welches Mittels Object-Relational Mapping (ORM) die Datenbanktabellen als Python-Objekte zur Verfügung stellt.

1) *Vorlage Hr. Neumann:* Provides (1) design decisions based on the previously defined requirements and (2) a visualization of the functional structure at top level including relationships („Grobe Zerlegung“), thus, gives an overview on modules, frameworks, and middleware.

In discussions of multi-tier architecture, layer is often used interchangeably – and mistakenly – for tier. They aren't the same. A „layer“ refers to a functional division of the software, but a „tier“ refers to a functional division of the software that runs on infrastructure separate from the other divisions. The Contacts app on your phone, for example, is a three-layer application, but a single-tier application, because all three layers run on your phone.

In discussions concerning multi-tier architecture, the term

„layer“ is frequently misused interchangeably with „tier“, despite their distinct meanings. A layer denotes a functional partition within the software, whereas a tier signifies a functional division that operates on separate infrastructure from other divisions/tiers. For instance, the Camera app or Settings app on your phone exemplifies a three-layer application but remains a single-tier application since all three layers run on your phone.

## B. Frontend

Das Frontend der Anwendung wurde mit dem Framework **React** umgesetzt. Zusammen mit dem Tool **Vite** wird eine schnelle Projektinitialisierung als auch ein sehr performantes Hot Module Replacement bereitgestellt. Dadurch können Änderungen am Code durch einfaches Speichern in Echtzeit im Browser dargestellt werden. Dies führt zu einer deutlich angenehmeren und zeiteffizienteren Entwicklung.

Als Programmiersprache wurde **TypeScript** gewählt. Im Vergleich zu JavaScript bietet TypeScript eine statische Typisierung von Variablen, Funktionen und anderen Komponenten. Dies ermöglicht eine frühzeitige Fehlererkennung - was sich positiv auf die Entwicklungszeit auswirkt - als auch bessere Lesbarkeit des Codes und vereinfachte Wartungen bei größeren und komplexeren Projekten.

Für die bessere Gestaltung der Benutzeroberfläche wird **Inline-CSS** in den React-Komponenten verwendet. Das heißt CSS-Stile liegen nicht in eigenen Dateien sondern werden mithilfe der `style={{}}`-Syntax direkt im JSX/TSX-Code eingebunden. Auf separate Dateien und externe Styling-Frameworks wie **TailwindCSS** wurde bewusst verzichtet um die Syntax möglichst komponentennah und unkompliziert zu halten.

Die Kommunikation mit dem Backend erfolgt ausschließlich über eine Rest-API. Hierfür wird die Bibliothek **Axios** verwendet. Diese bietet eine einfache Anbindung an diverse Endpunkte. Die empfangenen Daten liegen im JSON-Format vor und werden von den einzelnen Komponenten individuell verarbeitet und angezeigt.

Für die zentrale Zustandsverwaltung wird die Bibliothek **Redux** verwendet. Dadurch lassen sich globale Zustände und Daten einfach effizient verwalten. Im Falle des Logins bzw. der Registrierung sorgt ein globaler Zustand dafür, dass nicht eingeloggte Benutzer keinen Zugriff auf geschützte Routen haben um unerlaubten Zugriff zu vermeiden. Die Nutzerrollen des angemeldeten Users werden auf die gleiche Art und Weise gespeichert. So hat der Nutzer zudem nur Zugriff auf die für seine Rolle vorgesehenen Seiten und Funktionen. Die Integration erfolgt mithilfe des **react-redux**-Bindings, wodurch der State über das gesamte Frontend hinweg einheitlich zugänglich ist.

## C. Struktur und Komponenten

Der Sourcecode ist in verschiedene Ordner unterteilt. Zusammengehörige Funktionalitäten sind dabei gruppiert um die Wartbarkeit zu verbessern. Im Folgenden werden die wichtigsten Ordner / Dateien und deren Aufgaben beschrieben.

**api** Beinhaltet die zentrale `axiosInstance` über welche alle Http-Requests abgewickelt werden.

**components** Enthält Komponenten die im Verlauf der Anwendung öfter verwendet werden. Dazu wählen: `AuthSidebar` `HomeSidebar` und `ProtectedLayout`

#### D. Backend

Das Backend von GetraenkeIO ist als Python Applikation geschrieben. Mithilfe der Frameworks FastAPI wird eine REST-API für das Frontend zur Verfügung gestellt.

FastAPI ist ein Framework, das speziell zum Entwickeln von REST-APIs erstellt wurde und sehr intuitiv zu bedienen ist. Es ist damit möglich mit relativ wenigen Zeilen Code einen funktionierenden Endpunkt inklusive Validierung der Werte zu erzeugen. Ebenfalls unterstützt es verschiedene Middlewares. Konkret wurde die CORS-Middleware genutzt. Diese ermöglicht es Cross-Origin Resource Sharing (CORS) Anfragen vom Frontend entgegenzunehmen. Eine Besonderheit von FastAPI ist, dass eine API-Dokumentation in Form eines OpenAPI-Dokuments [`openapi`] automatisiert erstellt wird. Diese wird auf der Route `"/docs"` als interaktive Oberfläche mittels SwaggerUI [`swagger`] zur Verfügung gestellt. Dies erleichtert die Abstimmung zwischen den Entwicklerteams von Front- und Backend.

Für die Verwaltung der Datenbankverbindung wurde das Framework SQLAlchemy gewählt. Dieses wurde vom gleichen Entwickler wie FastAPI entwickelt und ist speziell. Deshalb ist eine Integration dieser beiden Technologien sehr gut möglich. SQLAlchemy ist ein ORM-Framework. Das Datenmodell für die Datenbank wird von SQLAlchemy auf Basis von den im Python-Code erzeugten Datenklassen (Models) erstellt. Ebenfalls müssen für die gängigen Create, Read, Update und Delete (CRUD) Operationen keine direkten SQL-Abfragen erzeugt werden, es genügt das zugehörige Model zu ändern und an die Datenbanksession weiterzugeben.

1) *Struktur und Komponenten:* Der Quellcode der Backend-App gliedert sich in mehrere Komponenten (Ordner/Dateien), deren Funktion im Folgenden erklärt wird.

**app/api/routes** Logik für alle Routen, der RESTful-Schnittstelle.

**app/api/dependencies.py** Funktionen/Abhängigkeiten, welche von mehreren Endpunkt-Implementierungen aus `app/api/routes` verwendet werden (Benutzer-Authentifizierung, Zugriff auf Datenbank-Session, Überprüfung ob aktueller Benutzer Admin-Rechte hat).

**app/core** Sämtliche Logik zur Konfiguration der Anwendung (Datenbank, Passwort-Hash-Funktion, Lesen der Einstellungen aus Umgebungsvariablen).

**app/crud** Logik zur Interaktion mit der Datenbank (Erstellen, Lesen, Schreiben und Löschen von den Datenmodellen aus `/app/models`).

**app/models** Datenmodelle für die in der Datenbank gespeicherten Daten. Ebenfalls die zugehörigen Datenmodelle, die in den Endpunkten (GET/POST) verwendet werden und die Logik zum Validieren der Daten.

**app/tests** Komponenten-Tests für die Anwendung, welche mithilfe von pytest [`pytest`] automatisch bei einem `push` ins Git-Repository ausgeführt werden.

**main.py** Initialisierung und Start-Up der Anwendung.

**requirements.txt** Enthält alle genutzten Pakete um diese automatisiert installieren zu können.

**.env.example** Beispielhafte `.env`-Datei, welche die zur Konfiguration nutzbaren Umgebungsvariablen und Beispielwerte für diese enthält.

2) *REST-Schnittstelle:* Die einzige Schnittstelle zwischen Front- und Backend ist die REST-Schnittstelle. Diese stellt alle relevanten Daten und Informationen für das Frontend im JSON-Format bereit. Im folgenden werden alle wichtigen Endpunkte genauer beschrieben:

**GET /users** Gibt eine Liste mit Details über alle Benutzer zurück. Ist nur für den Admin nutzbar.

**GET /users/me** Gibt Informationen über den aktuell eingeloggt Benutzer zurück.

**GET /users/{user\_name}** Gibt Informationen über den Benutzer mit dem Name `user_name` zurück. Ist nur für den Admin nutzbar.

**POST /users/** Endpunkt zum Erstellen eines neuen Benutzers bei seiner Registrierung.

**POST /users/{user\_id}/recharges** Endpunkt zum Aufladen des Guthabens eines Benutzers um einen bestimmten Betrag. Ist nur für den Admin nutzbar.

**GET /users/{user\_id}/recharges** Gibt alle Aufladungen eines Benutzers zurück. Ist für den Benutzer selbst und den Admin nutzbar. Ist nur für den Admin nutzbar.

**GET /drinks** Gibt eine Liste mit allen Daten der aktuell vorhandenen Getränken zurück.

**GET /drinks/{drink\_id}** Gibt eine Liste mit allen Daten der aktuell vorhandenen Getränken zurück.

**POST /drinks** Erlaubt es dem Admin ein neues Produkt erstellen. Dazu müssen alle Attribute des neuen Getränks in einem JSON-Objekt mitgegeben werden. Ausgenommen davon ist die ID, welche bei der Erstellung des Eintrags in der Getränketabelle automatisch mithilfe eines UUID-Generators erstellt wird.

**PUT /drinks/{drink\_id}** Lässt Administratoren die Attribute der Getränke ändern. Dabei können beliebige Kombinationen an Attributen eines Getränks gleichzeitig geändert werden. Lediglich die ID ist hierüber nicht veränderbar. Zum Ändern der gewünschten Attribute werden diese Entsprechend mit neuen Werten als JSON-Objekt an die API übergeben. Ist ein Attribut nicht im Objekt vorhanden, so bleibt dort der ursprüngliche Wert erhalten.

**DELETE /drinks/{drink\_id}** Lässt Administratoren ein Getränk löschen. Dabei wird nach erfolgreicher Löschung eine Kopie des Objekts dem Nutzer zurückgegeben.

**POST /transactions** Ermöglicht es Nutzern ein Getränk zu kaufen. Dabei wird das Guthaben des Nutzers um den Gesamtpreis verringert und die Stückzahl des Getränks entsprechend angepasst. Zudem wird die Transaktion in einer separaten Tabelle protokolliert.

**GET /transactions** Gibt einem Admin alle getätigten Transaktionen als Liste zurück.

**GET /transactions/me** Gibt Nutzern alle von ihnen durchgeführte Transaktionen als Liste zurück

### E. Datenhaltung

Zur Datenhaltung in der Produktionsumgebung wird eine relationale PostgreSQL-Datenbank verwendet. Die Kommunikation mit dieser findet ausschließlich über das SQLModel-Framework statt. Da ein ORM-Framework verwendet wird, könnte die Datenbank relativ einfach gegen alle anderen von diesem Framework ausgetauscht werden. In der Anfangsphase der Entwicklung wurde diese Möglichkeit genutzt, um SQLite [sqlite] als Entwicklungsdatenbank verwenden zu können und durch die einfache Konfiguration schnell ein lauffähiges System erzeugen zu können.

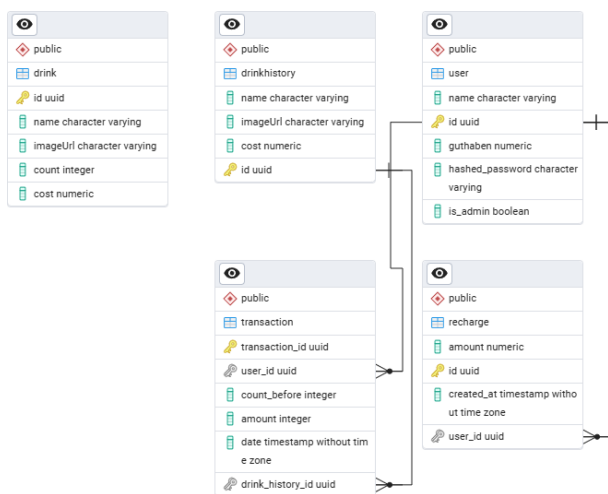


Abbildung 2. ER-Diagramm zur PostgreSQL Datenbank

Die Daten der Getränkeverwaltung werden in unserer Datenbank, wie in Abbildung 2 beschrieben, auf fünf verschiedene Tabellen aufgeteilt. So werden aktuelle Nutzer und Getränke in jeweils einzelnen Tabellen abgespeichert. Abseits davon sind die weiteren Tabellen zur Überwachung der getätigten Aktionen verantwortlich. Somit wird das Aufladen von Guthaben in der recharge Tabelle und jeder Kauf in der transaction Tabelle abgespeichert. Um jedoch sowohl die ursprünglichen Preise, als auch weitere möglicherweise veränderte Attribute der Getränke für spätere Analysezwecke korrekt vorliegen zu haben wird für jedes veränderte Getränk, das gekauft wird, ebenfalls ein Eintrag in der drinkhistory Tabelle erstellt.

## V. DISCUSSION | EVALUATION | LESSONS LEARNED | IMPEDIMENTS

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat

sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

## VI. FAZIT UND AUSBLICK

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.