🍽️ Kochen = viel Zeit & Aufwand

→ **Alternative: Mealprep**

- Was soll ich an **Zubehör** kaufen?
- Welche **Zutaten** brauche ich wirklich?
- Gibt es passende **Mealprep-Rezepte** zum Kochen?
- Wie erstelle ich den **Essensplan** – von Hand oder in einer App?
- Gibt es eine **All-in-One App**, die alles kann?

Lösung: GetIntoMealPrep

🥙 Nur erprobte, alltagstaugliche **Mealprep-Rezepte**

📖 Eigene **digitale Rezeptsammlung** erstellen

🗓️ Wöchentliche **Essenspläne per Klick** zusammenstellen

🛒 **Automatische Einkaufsliste** für alle Mahlzeiten

📊 Überblick über Nährstoffe & Portionsgrößen

Anforderungen

Nutzerfreundliche MealPrep-Webapp

Schönes, intuitives UI

Sicherer Login & JWT-geschützte API

Rezeptübersicht & Favoriten

Wochenplaner für Mahlzeiten

Ernährungs-Statistiken

Leistungsstarkes Backend mit Datenbank

Gehostet mit Domain & SSL-Zertifikat (HTTPS)

Vorgehensweise

🎨 **1. Frontend – Design und Aufbau**

- Auswahl eines passenden Farbschemas und Hintergrunddesigns
- Evaluierung und ggf. Integration geeigneter UI-Bibliotheken
- Erstellung eines ersten groben Layouts ohne Funktionalität

🔧 **2. Backend – Grundgerüst**

- Aufsetzen des Backend-Projekts (Initialisierung, Projektstruktur)
- Definition erster Routen und Controller-Setup ohne Business-Logik

Vorgehensweise

⚙️ 3. **DevOps – Entwicklungsumgebung**

- Einrichtung von **Docker-Containern** mit:
- **dotnet watch** für das Backend
- **vite dev** für das Frontend
- Setup von Nginx als **Reverse Proxy**

🔐 **4. Authentifizierung**

- Integration von **Logto IDP** zur Nutzerverwaltung und Authentifizierung

🗃️ **5. Datenbank – Entwurf und Initialisierung**

- Modellierung der Datenbank mittels ORM (z. B. Entity Framework)
- Definition der Datenmodelle und Beziehungen

Vorgehensweise

🔄 **6. API und Logik**

- Implementierung der Backend-Controller
- Anbindung der Frontend-Komponenten an die API (API Calls, State Handling)
- Integration der Anwendungslogik in Frontend und Backend

🌱 **7. Seed-Daten**

- Erstellung und Einbindung von Seed-Daten im Backend für Tests und Initialdaten

Vorgehensweise

🗓️ **8. Wochenplan-Feature**

- Erweiterung des Backends und Frontends zur Unterstützung eines Wochenplans
- UI/UX-Komponenten zur Darstellung und Bearbeitung

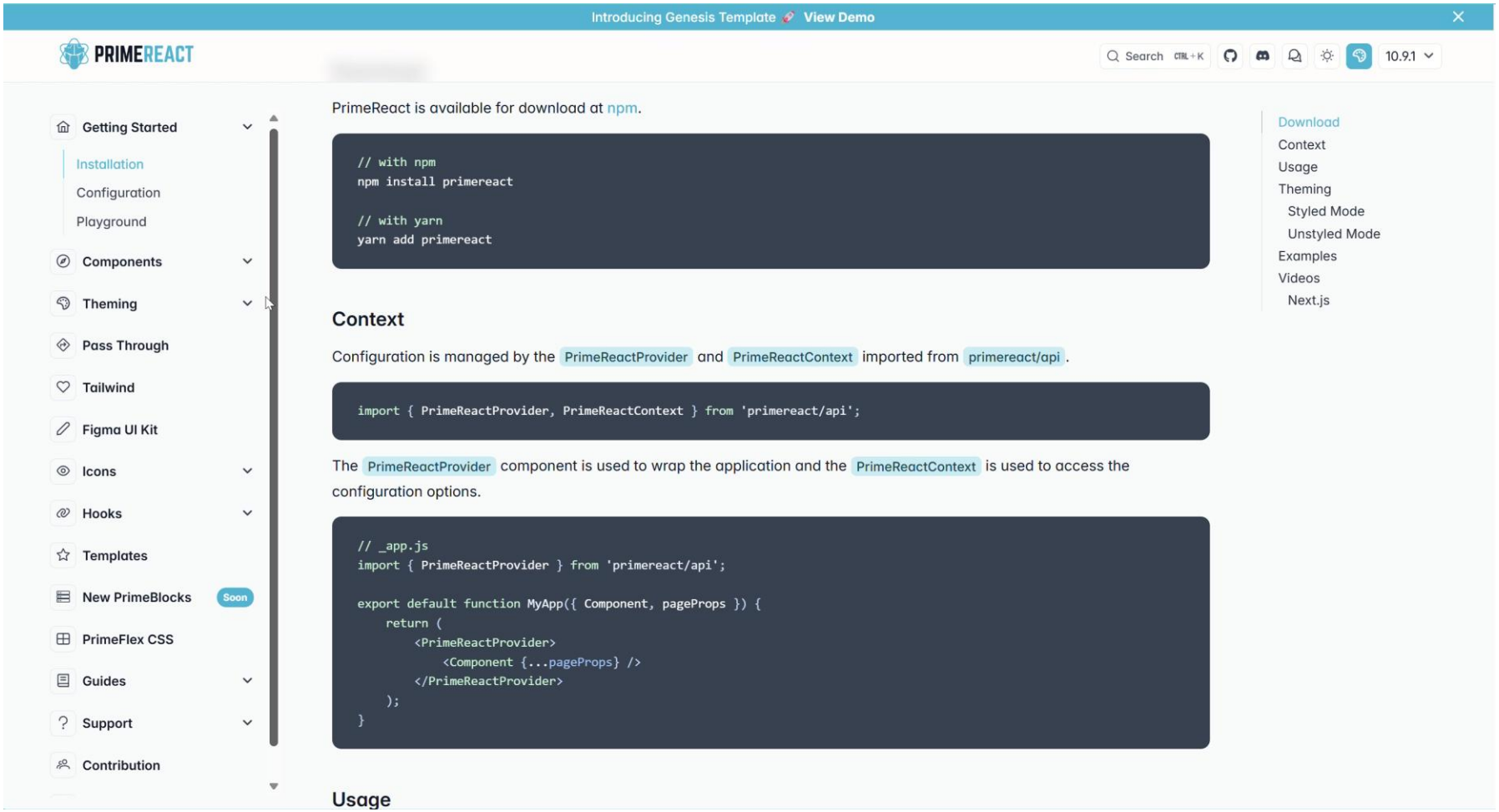
🚀 **9. Produktiv-Setup (DevOps)**

- Finales Setup für Produktion mit **Nginx**
- Konfiguration eines **Ubuntu vServers**:
  - Domain-Konfiguration
  - SSL-Zertifikate mit **Let's Encrypt**
  - Zugang via **SSH** und **FTP**
  - Startskript für Container-Deployment (Docker)

Techstack Frontend

React	TypeScript	Vite	Logto	Primereact
				

Primereact


Introducing Genesis Template 🚀 View Demo
PRIMEREACT
Search CTRL + K
10.9.1
Getting Started
Installation
Configuration
Playground
Components
Theming
Pass Through
Tailwind
Figma UI Kit
Icons
Hooks
Templates
New PrimeBlocks
Soon
PrimeFlex CSS
Guides
Support
Contribution
PrimeReact is available for download at npm.
// with npm
npm install primereact
// with yarn
yarn add primereact
Context
Configuration is managed by the PrimeReactProvider and PrimeReactContext imported from primereact/api .
import { PrimeReactProvider, PrimeReactContext } from 'primereact/api';
The PrimeReactProvider component is used to wrap the application and the PrimeReactContext is used to access the configuration options.
// _app.js
import { PrimeReactProvider } from 'primereact/api';
export default function MyApp({ Component, pageProps }) {
return (
<PrimeReactProvider>
<Component {...pageProps} />
</PrimeReactProvider>
);
}
Usage
Download
Context
Usage
Theming
Styled Mode
Unstyled Mode
Examples
Videos
Next.js

Frontend Architektur & Struktur

Wiederverwendbare Komponenten → `/components`

Seiten hierarchisch in `/pages`, spezifische Komponenten direkt dort

API-Logik & Token-Handling in eigene Hooks ausgelagert

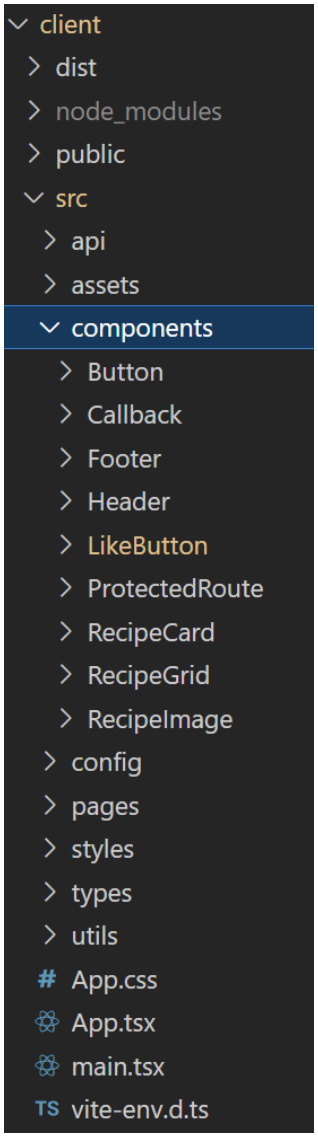
- > Jetzt schauen wir uns den Code an

Routenübersicht

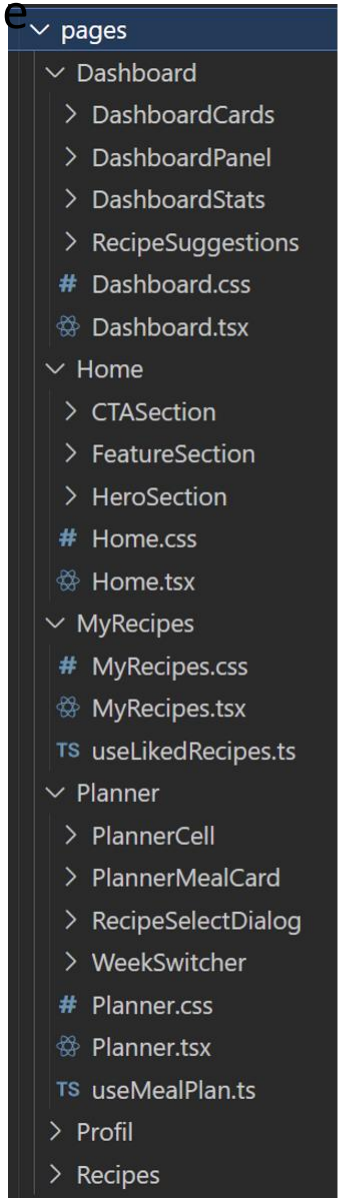
App.tsx

```tsx
function App() {
  return (
    <DndProvider backend={HTML5Backend}>
      <LogtoProvider config={config}>
        <Router>
        <Header />
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/callback" element={<Callback />} />
            <Route
              path="/dashboard"
              element={
                <ProtectedRoute>
                  <Dashboard />
                </ProtectedRoute>
              }
            />
            <Route
              path="/recipes"
              element={
                <ProtectedRoute>
                  <AllRecipes />
                </ProtectedRoute>
              }
            />
            <Route
              path="/recipes/:id"
              element={
                <ProtectedRoute>
                  <RecipeDetails />
                </ProtectedRoute>
              }
            />
            <Route path="/profil" element={<Profil />} />
            <Route path="/planner" element={<Planner />} />
            <Route path="/my-recipes" element={<MyRecipes />} />
          </Routes>
          <Footer />
        </Router>
      </LogtoProvider>
    </DndProvider>
  );
}
```

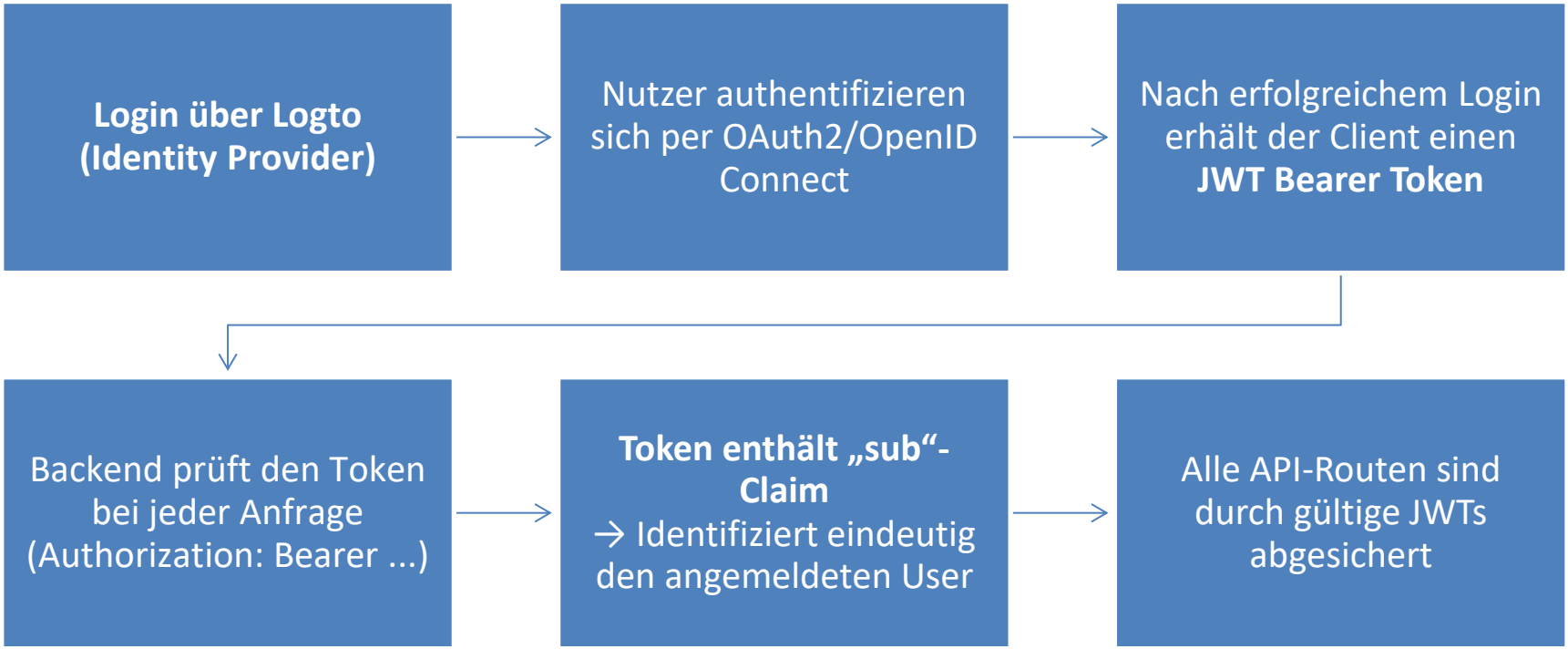
Komponenten



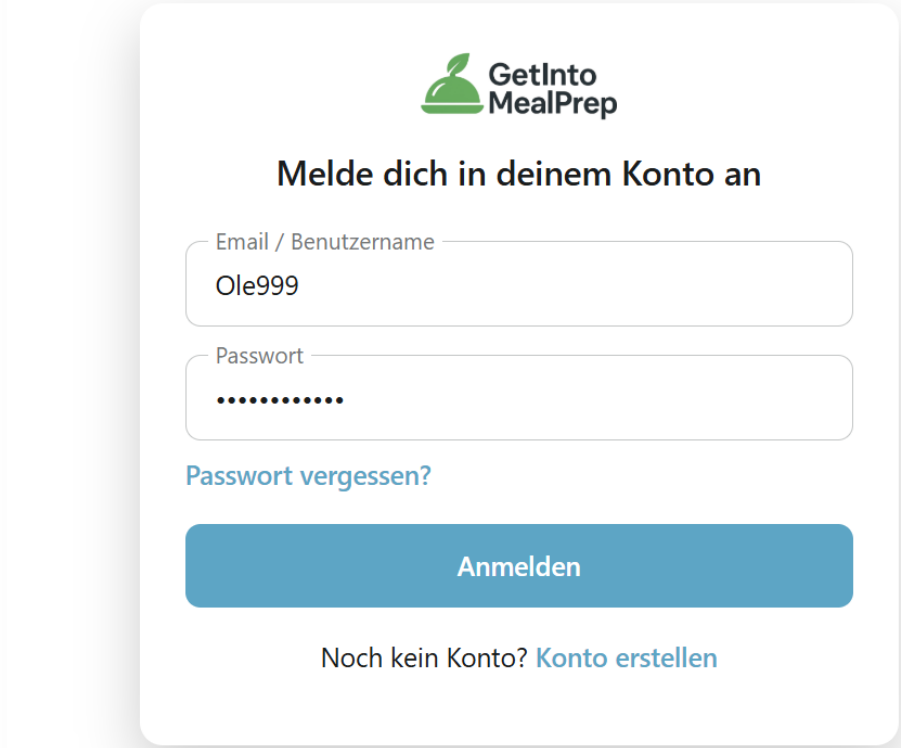
Seitenhierarchie

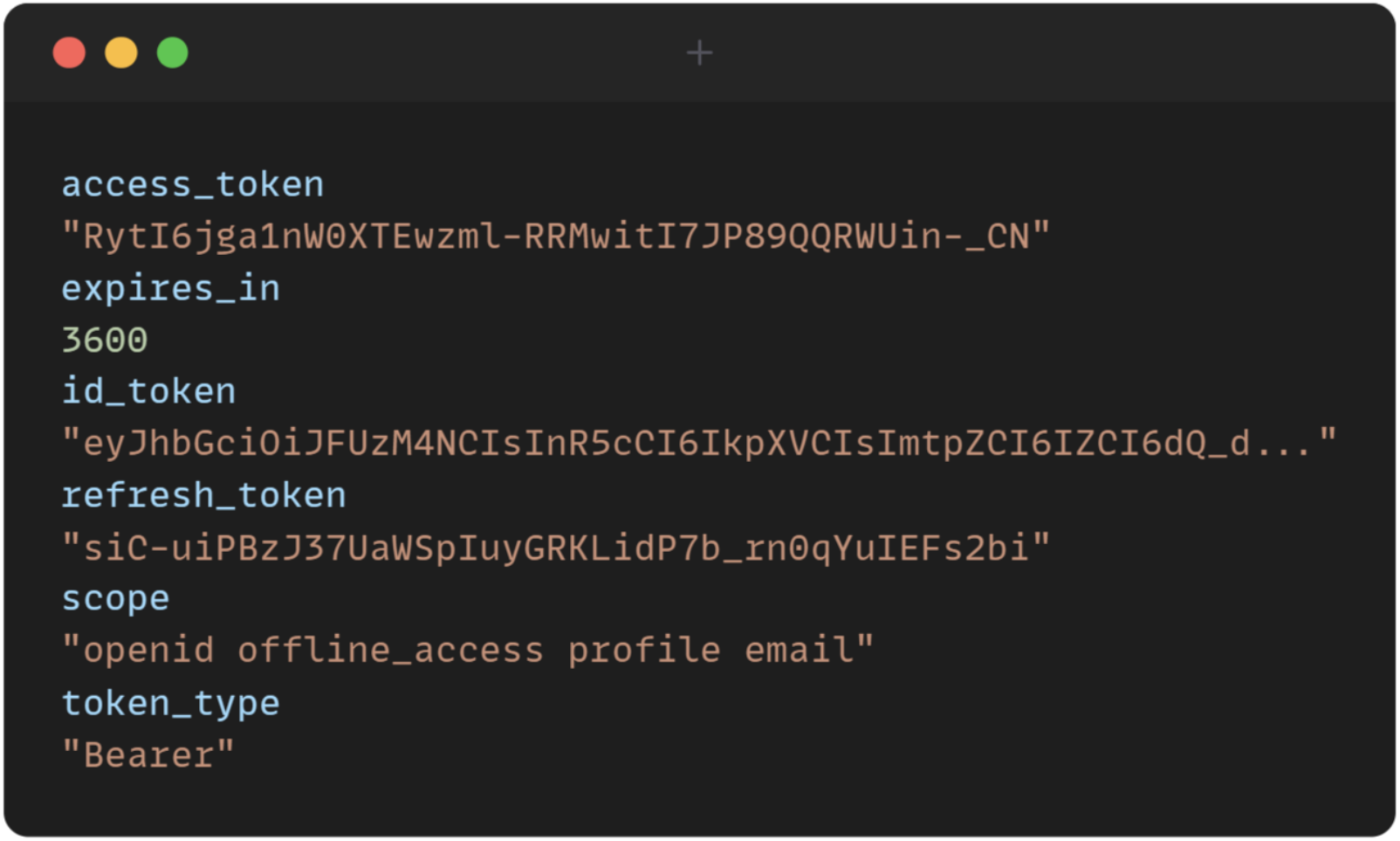


Authentifizierung mit Logto (IDP)



Weiterleitung zur Login Seite




access_token
"RytI6jga1nW0XTEwzml-RRMwitI7JP89QQRWUin-_CN"
expires_in
3600
id_token
"eyJhbGciOiJFUzM4NCIsInR5cCI6IkpXVCIsImtpZCI6IZCI6dQ_d..."
refresh_token
"siC-uiPBzJ37UaWSpIuyGRKLidP7b_rn0qYuIEFs2bi"
scope
"openid offline_access profile email"
token_type
"Bearer"

JSON CLAIMS TABLE COPY


```
{
  "sub": "qu2g4fps5ba4",
  "name": null,
  "picture": null,
  "updated_at": 1747889550681,
  "username": "test_user",
  "created_at": 1745498758541,
  "email": "test_user@gmail.com",
  "email_verified": true,
  "at_hash": "_Dnuk-RRs69O9IgFH6WVmJTJzDUTxzxh",
  "aud": "bjnuh81fqwtn5ghf7zrd8",
  "exp": 1747893156,
  "iat": 1747889556,
  "iss": "https://1vvqmn.logto.app/oidc"
}
```

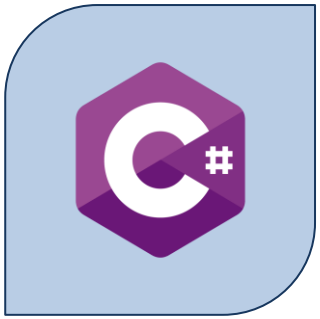
Callback.tsx

```tsx
import { useHandleSignInCallback } from '@logto/react';
import { useNavigate, useLocation } from 'react-router-dom';

const Callback = () => {
  const navigate = useNavigate();
  const location = useLocation();

  const { isLoading, error } = useHandleSignInCallback(() => {
    const returnTo = (location.state as any)?.from?.pathname || '/dashboard';
    navigate(returnTo, { replace: true });
  });

  if (isLoading) {
    return <div>Redirecting...</div>;
  }

  if (error) {
    return <div>Login failed. Please try again.</div>;
  }

  return null;
};

export default Callback;
```

useRecipeSuggestions.ts

```ts
import { useLogto } from "@logto/react";
import axios from "axios";

export async function fetchRecipes(count: number) {
  const { getIdToken } = useLogto();
  const token = await getIdToken(); // 1. Token von Logto holen

  // 2. Request ans Backend mit Bearer Token im Header
  const res = await axios.get(
    `/api/recipe/random?count=${count}`,
    {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    }
  );
  return res.data;
}
```

Backend Tech Stack



C# MIT EF CORE MIGRATIONS

DB MIT POSTGRESQL

DURCH JWT-TOKEN GESCHÜTZT

Benutzerauthentifizierung mit JWT

Program.cs

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.Authority = "https://1vvqmn.logto.app/oidc";
        options.Audience = "bjnuh81fqwtn5ghf7zrd8";
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true
        };
    });

builder.Services.AddControllers(options =>
{
    var policy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build();

    options.Filters.Add(new AuthorizeFilter(policy));
});
```

Object-Relational Mapping (ORM)

C#-Klassen ↔ Datenbanktabellen (automatisch abgebildet)

DbContext als zentrale Verbindung zur Datenbank

Relationen (1:n, n:m) per Fluent API oder Data Annotations

Datenbankabfragen via LINQ

🧭 API Controller in ASP.NET Core

🧱 **Aufbau & Funktion**

- Jeder Controller verarbeitet Anfragen zu bestimmten API-Routen
- Erbt von einer gemeinsamen **BaseController**-Klasse
- Authentifizierte Nutzer werden per sub-Claim aus dem JWT identifiziert

📂

RecipeController→ Suche, Favoriten, Zutaten, Zufallsrezepte…

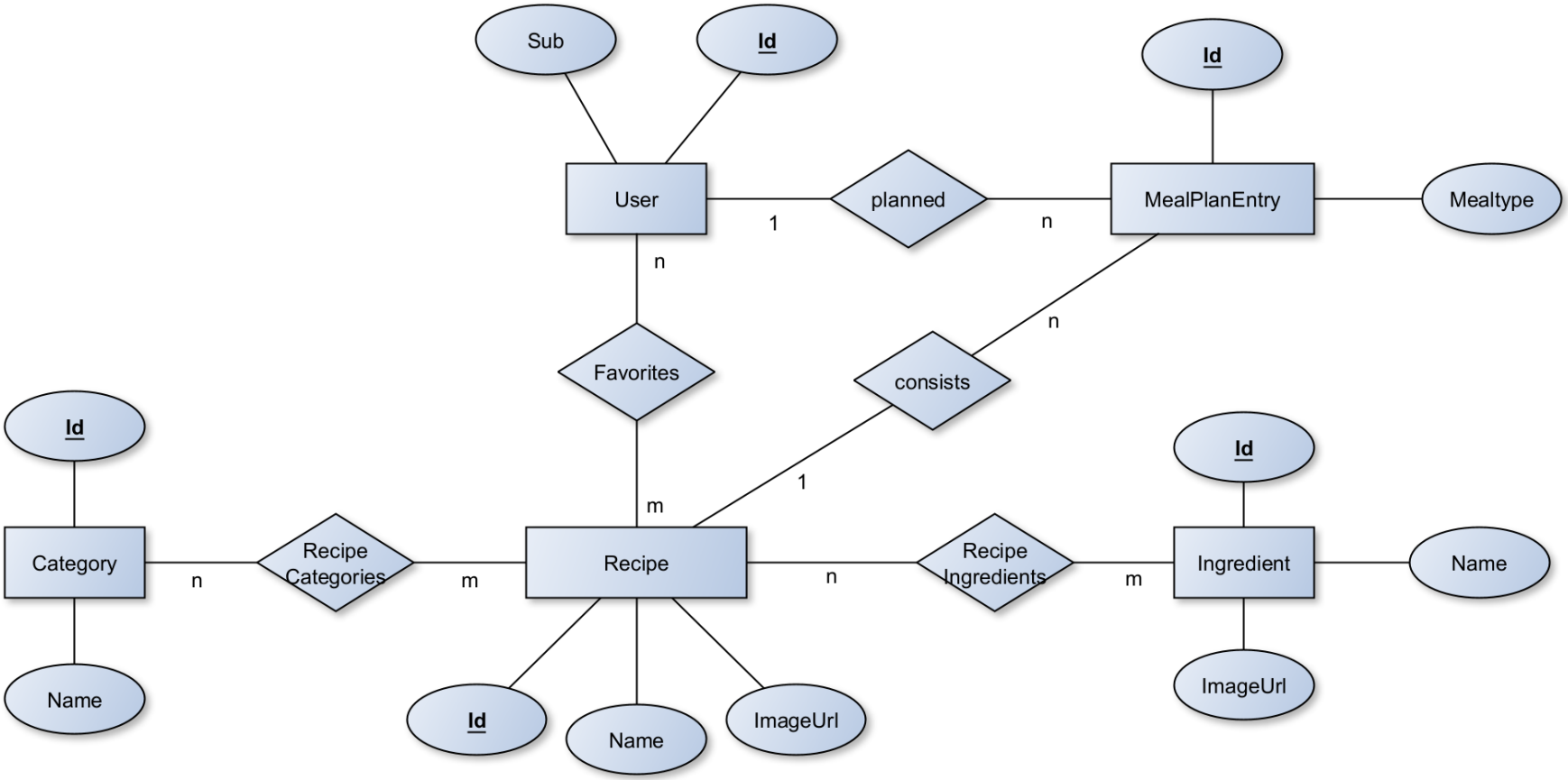
MealPlanController→ Wochenplan verwalten (CRUD), Drag & Drop, Zeitlogik

User Handling in der API

BaseController.cs

```
[Authorize]
public abstract class BaseController : ControllerBase
{
    protected readonly AppDbContext _context;

    protected BaseController(AppDbContext context) ⇒ _context = context;

    // Holt User-ID aus JWT ('sub'-Claim)
    protected string GetUserId() ⇒
        User.FindFirst("sub")?.Value ?? throw new UnauthorizedAccessException();

    // Findet oder erstellt User
    protected async Task<User> GetOrCreateUserAsync()
    {
        var id = GetUserId();
        var user = await _context.Users.FirstOrDefaultAsync(u ⇒ u.Sub == id);
        return user ?? await CreateUserAsync(id);
    }

    private async Task<User> CreateUserAsync(string id)
    {
        var user = new User { Sub = id };
        _context.Users.Add(user);
        await _context.SaveChangesAsync();
        return user;
    }
}
```

Datenbank ER-Modell (Auszug)



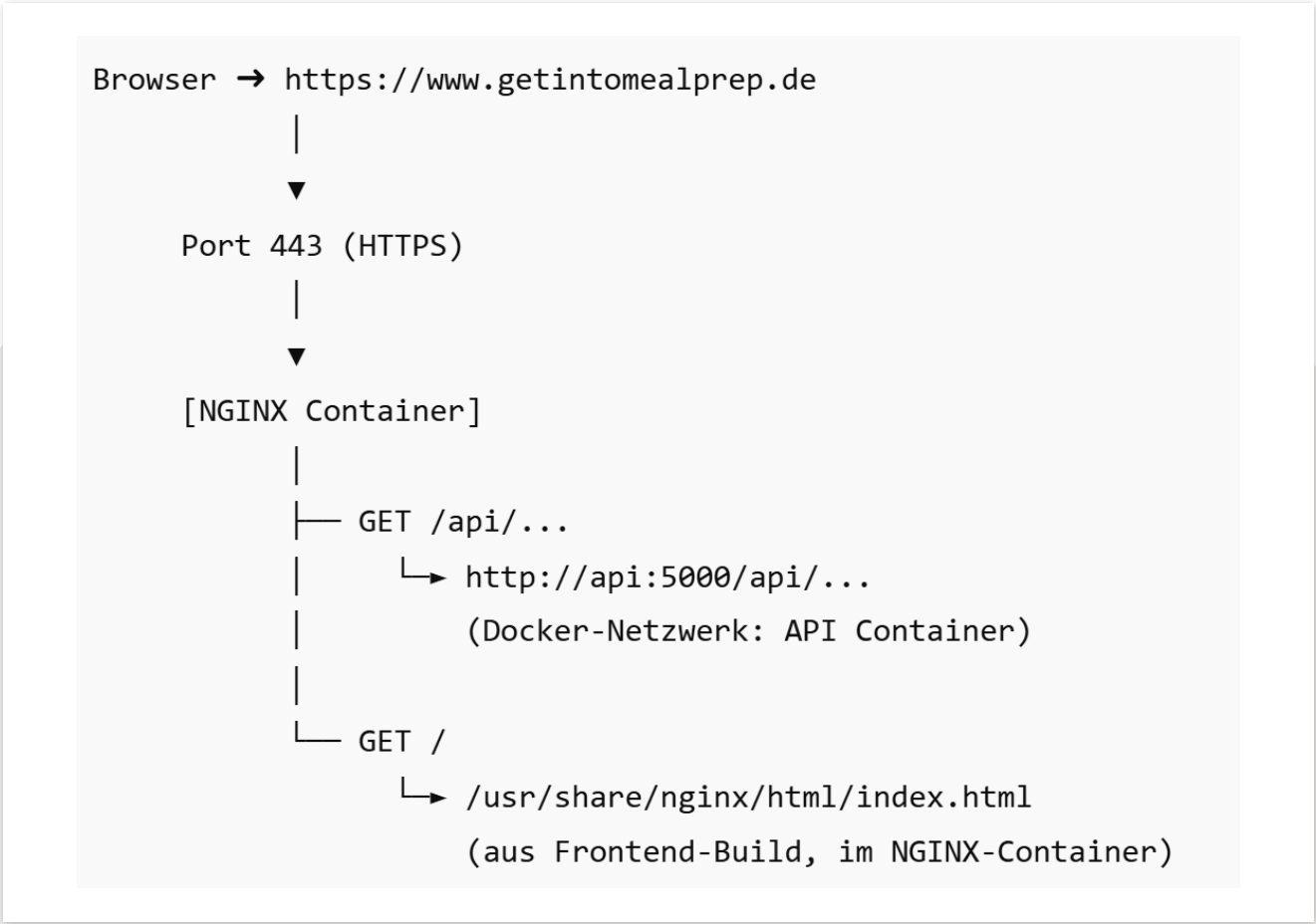
Attribute


Users
Id integer
Sub text NN
Recipes
Id integer
Name text NN
Description text
Instructions text NN
PortionCount integer NN
Difficulty integer NN
CaloriesPerServing real NN
ImageUrl text
Ingredients
Id integer
Name text NN
Price numeric NN
CaloriesPer100g integer NN
Protein real NN
Fat real NN
Carbs real NN
ImageUrl text
Categories
Id integer
Name text NN
RecipeCategories
RecipeId integer NN
CategoryId integer NN
primary key(RecipeId,CategoryId)
UserFavoriteRecipes
FavoriteRecipesId integer NN
UserId integer NN
primary key(FavoriteRecipesId,UserId)
RecipeIngredients
RecipeId integer NN
IngredientId integer NN
Quantity real NN
Unit text NN
primary key(RecipeId,IngredientId)
MealPlanEntries
Id integer
UserId integer NN
RecipeId integer NN
Date timestamp NN
MealType text NN
Position integer NN
CreatedAt timestamp NN
UpdatedAt timestamp NN

🚀 DevOps & Deployment mit Docker + Nginx

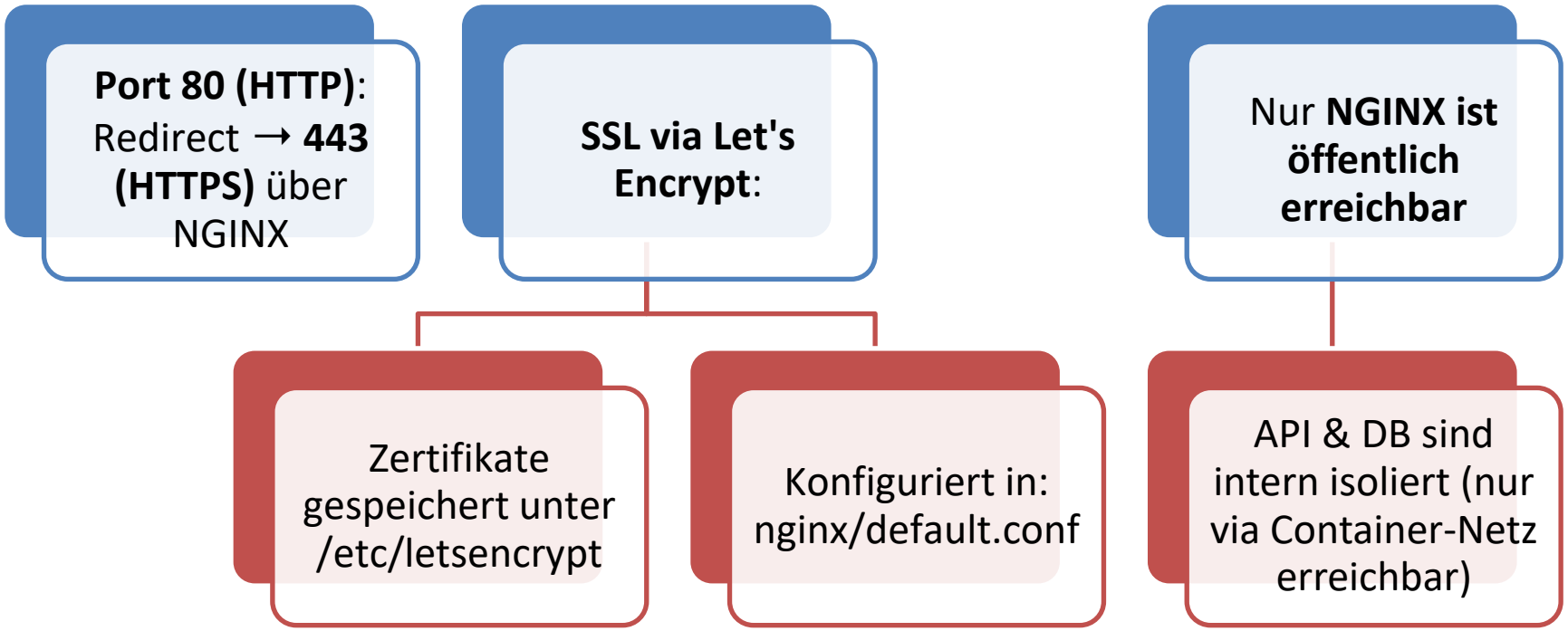
🧱 Container-Übersicht

Service	Container-Name	Port (intern → extern)	Aufgabe
`api`	`getintomealprep_api`	5000 → *intern only*	ASP.NET Core Backend (.NET 9)
`frontend`	`getintomealprep_frontend`	80 → *intern only*	Produktionsbuild (HTML/JS via Nginx)
`db`	`getintomealprep_db`	5432 → 5433	PostgreSQL-Datenbank
`nginx`	`getintomealprep_nginx`	**80 → 80, 443 → 443**	Reverse Proxy + HTTPS

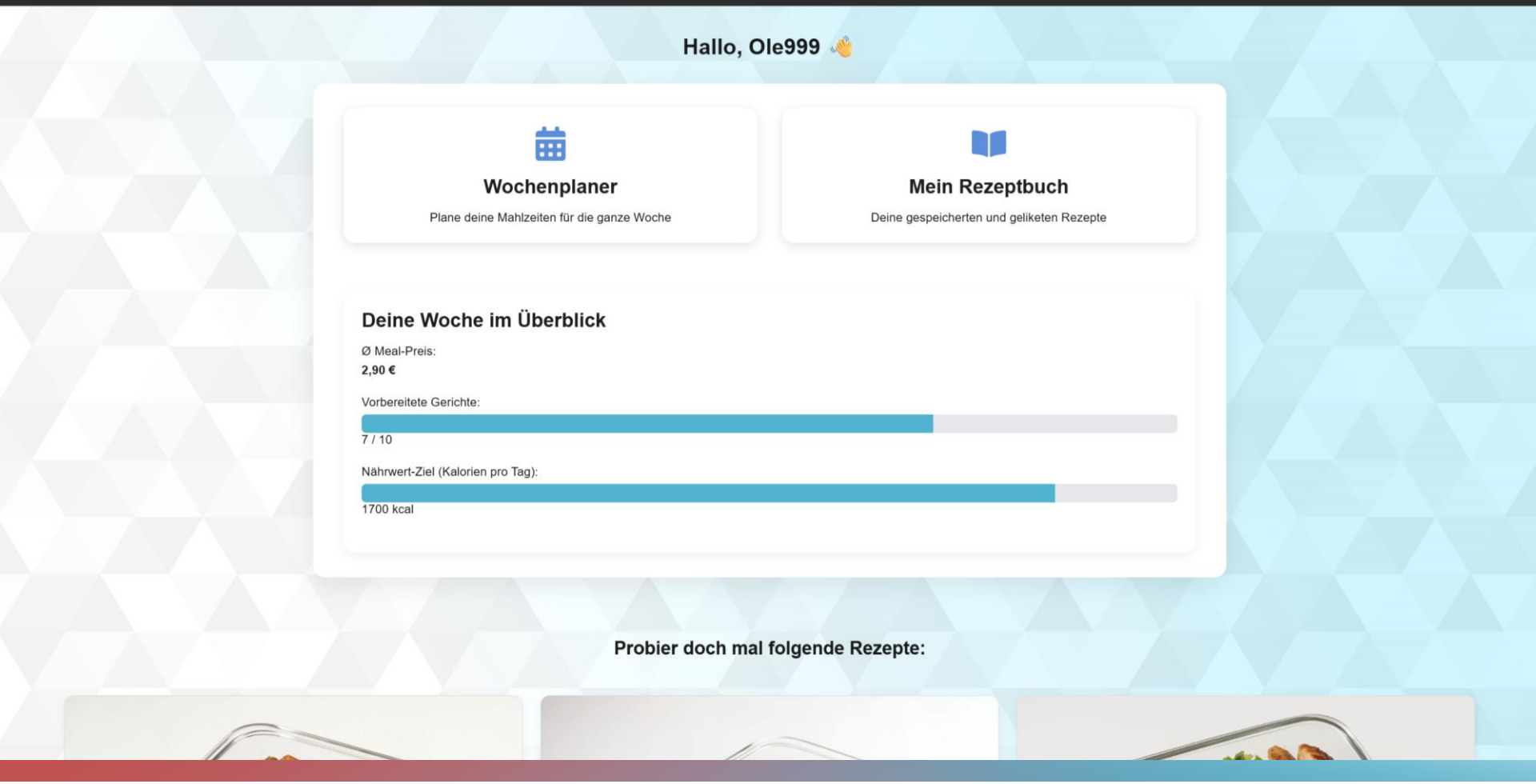


🌐 Domain-Routing mit NGINX

🔒 Sicherheit & HTTPS



Deployment-Architektur (Docker Compose)



Demo von GetIntoMealPrep

Herausforderungen & Learnings

- Integration von Logto & Tokenmanagement
- Dev-Ops: Docker-Deployment auf vServer
- Entwurf und Entwicklung von Full Stack Webanwendungen
- UI/UX-Optimierung für Nutzerfreundlichkeit

Fazit & Ausblick

Ziel erreicht:

Moderne, sichere MealPrep-App im Fullstack-Szenario

Potenzial:

Community-Funktionen, Rezepte teilen, Nährstoffanalyse, Rezepte in der UI hinzufügen etc, Social Media Plattform mit Kommentaren, Cloudflare etc

Fragen & Diskussion