Hochschule Zittau/Görlitz
Fakultät Elektrotechnik und Informatik

Erstellung einer Todo App mit React und AWS CDK

Beleg

Modul / Lehrveranstaltung: Web Engineering 3
Dozent: Christopher Hilgner

Niklas Kaulfers

Matrikelnummer: 1064032

Abgabedatum: 18. Januar 2026

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein durchaus häufig erstelltes Projekt in der Softwareentwicklung ist eine Todo-App. Anhand solcher können grundlegende Kenntnisse der Entwickler zur jeweiligen Umgebung gezeigt werden. Im Verlauf dieser Arbeit wird der Entwicklungsprozess einer solchen App im Serverless Kontext mit einem React Frontend erläutert. Hierzu entstehen Einblicke in den technischen Aufbau der Anwendung. Auch die genaue Komposition der Endpunkte und einige Codeabschnitte werden betrachtet.

1.1 Serverless

Um den genauen Aufbau dieser Anwendung zu verstehen, muss der Leser vorkenntnisse im Bereich des *Cloud Computings* und *Serverless* haben. Serverless bezieht sich auf Softwareanwendungen, welche in der Cloud veröffentlicht werden, der Entwickler sich jedoch nicht um den tieferen Aufbau des Servers kümmern muss. So ist der Begriff Serverless durchaus irreführend. Server existieren, werden jedoch von dem Hyperscaler, in diesem Fall AWS, verwaltet. Viele Service wie AWS Lambda oder AWS DynamoDB fallen in die Kategorie von Serverless.

Kapitel 2

Tech Stack

2.1 Backend

Das Backend besteht vollständig aus AWS Servicen. Um genau zu sein ApiGateway, Lambda, DynamoDB und Cognito. Diese haben jeweils eigene Aufgaben. So ist ApiGateway verantwortlich dafür, eine RestAPI zur Verfügung zu stellen. Alle Endpunkte der Anwendung sind innerhalb dieser API zu finden. Der Service Cognito schützt die Endpunkte, in dem er nur verifizierten Nutzern den Zugriff auf die API gestattet. Ohne Verifizierung hat der Nutzer des Frontends keinen Zugriff auf die API Endpunkte. Lambdas stellen Funktionen im Serverless Kontext da. In diesen kann die Logik der Anwendung mit eigenem Code definiert werden. Innerhalb der hier behandelten Anwendung wird die gesamte Logik in Typescript und mittels Nodejs geschrieben. Für Persistenz sorgt DynamoDB, die NoSQL Datenbank von AWS.

2.2 Frontend

Um mit dem Backend zu interagieren wird ein React Frontend verwendet. Mit React können Single Page Applications (SPAs) erstellt werden. Eine solche SPA besteht aus mehreren Komponenten, welche individuell, voneinander unabhängig ausgeführt werden können. Dieses nutzt für styling *Tailwindcss* und *MUI*. Tailwind stellt eine Sammlung an css Code da, welcher im Code einfach durch HTML Klassen genutzt werden kann. MUI hingegen ist eine Komponenten Bibliothek für React. Zusammen sorgen MUI und Tailwind für ein anschauliches Erscheinungsbild.
Für Nutzerverifizierung im Frontend wird *react oidc* genutzt. Gemeinsam mit einem von AWS bereitgestellten Login-Bildschirm sorgt diese Bibliothek für eine Absicherung der Anwendung.

2.3 Entwicklungsumgebung

Die gesamte Anwendung wurde im Umfeld von *IntelliJ IDEA* erstellt. Als Programmiersprache wird ausschließlich *Typescript* verwendet, für Frontend, Backend und Infrastruktur. Um die Anwendung ordentlich zur Verfügung zu stellen wird das Backend über AWS CDK erstellt. Dies ist ein Infrastructure as Code (IaC) Tool für AWS. In AWS CDK kann ein *Cloudformation Stack* mittels einer Programmiersprache definiert werden. Dieser Stack stellt ein Abbild der zu erwartenden Infrastruktur innerhalb der AWS Cloud da. Mittels dieses Stacks werden die einzelnen Serverless Service miteinander verknüpft. Auch werden hier Zugriffsrechte mit AWS IAM definiert.

Kapitel 3

Aufbau und Architektur

Die Anwendung stellt eine Todo-App da. In dieser kann ein Nutzer eigene Todos erstellen. Ein Todo besteht aus einem Titel, einer Beschreibung, einem Boolean-Wert und einer eindeutigen ID. Zudem einen Verweis auf in welche Listen sich dieses Todo befindet. Auch die Listen haben einen Verweis auf alle Todos, welche sich in dieser Liste befinden. Listen und Todos haben Methoden, um diese mit neuen Werten zu überschreiben. Auch besteht die Möglichkeit sie zu löschen. Man kann einzelne Parameter Updaten, hier ist der Verweis auf die jeweilige andere Liste jedoch nicht funktional. Das liegt daran, dass es sich bei der Datenbank nicht um eine relationale Datenbank handelt. Entsprechende Logik zu den Relationen muss somit in der Logik eingebaut werden.

Kapitel 4

Implementierung

Die Entwicklung fand in zwei alleinstehenden Projekten statt – dem Backend und dem Frontend. Vor allem von Bedeutung ist das Backend. Dort wird die Logik definiert. Das Backend besteht aus einem AWS CDK Projekt. In diesem werden verschiedene *Constructs* verwendet. Ein Construct definiert den Rahmen in welchem eine Komponente verwendet wird und ermöglicht so Objektorientiert Programmierung (OOP). In Aws gibt es 3 Level von Constructs, nämlich L1, L2 und L3. Diese definieren die Herkunft und Komplexität des Constructs. Die grundlegende Version sind L1 Constructs, diese stellen eins zu eins ein Abbild der Konfiguration für Cloudformation da. Hingegen sind L2 Ressourcen bereits angepasst von AWS, um sinnvoll und ohne zu viel Arbeit verwendet werden zu können. Mit L3 Ressourcen werden mehrere Ressourcen erstellt, welche sinnvoll miteinander interagieren[1]. Das untenstehende Construct für ApiGateway ist ein in diesem Projekt verwendetes L3 Construct, in dem die verschiedenen Endpunkte, Definitionen für SwaggerIO und Integrationen definiert sind.
Auch werden in diesen Constructs die Trust-Relationen erstellt. Diese werden vom Service IAM genutzt, um Servicen Zugriff zu anderen Servicen zu erlauben. So hat ApiGateway ohne IAM Rollen nicht die nötigen Recht, um die Lambdafunktion mit der Entsprechenden Logik auszuführen.

```
1  export class ApiGatewayConstruct extends Construct {
2      private readonly props: ApiGatewayProps;
3      private readonly _restApi: RestApi;
4
5      constructor(scope: Construct, id: string, props: ApiGatewayProps) {
6          super(scope, id);
7          this.props = props;
8          // ...
9      }
10 }
```

4.1 Endpunkte

Alle Funktionen dieser Anwendung sind mit eigenen Endpunkten im Backend ausführbar.

4.2 Beispiel: attach

[1] Amazon Web Services, *AWS CDK Constructs*, `https://docs.aws.amazon.com/cdk/v2/guide/constructs.html`, Accessed: 2026-12-03, 2026.

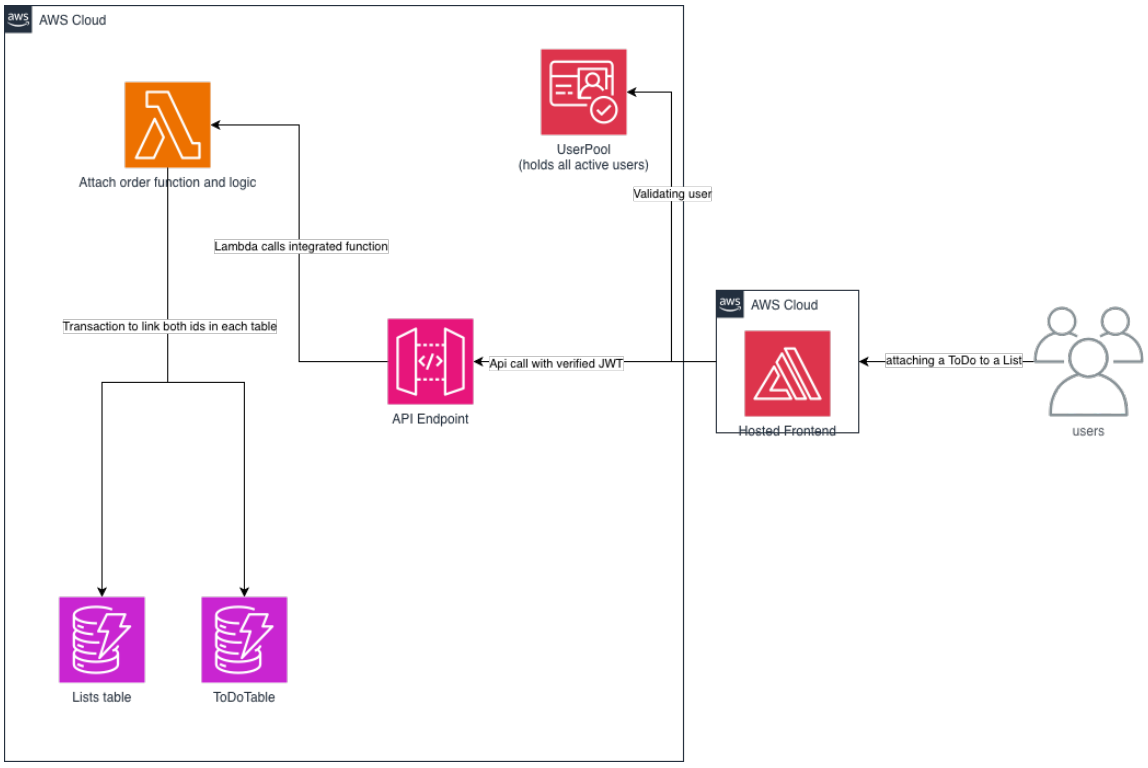

Abbildung 4.1: Ablauf des Hinzufügens eines Todos zu einer Liste

Kapitel 5

Fazit

Appendices

Literatur

[1] Amazon Web Services, *AWS CDK Constructs*, `https://docs.aws.amazon.com/cdk/v2/guide/constructs.html`, Accessed: 2026-12-03, 2026.