

Projekt Cool-Cars

Funktionale Programmierung

Beschreibung

Dieses Dokument beinhaltet Aufgaben zur Cool-Cars Anwendung. Auf der Basis dieses Projektes, sollen die Aspekte der Funktionalen Programmierung in JavaScript/React angewendet werden.



Inhaltsverzeichnis

1	COOL CARS	2
1.1	EINLEITUNG	2
1.2	ANFORDERUNGEN - VERSTÄNDNIS DER FUNKTIONALEN PROGRAMMIERUNG	2
1.3	ERWEITERUNGEN	2
1.4	AUSGANGSLAGE	2
1.5	AUFGABE - DOKUMENTATION (KOMPETENZNACHWEIS)	3
1.6	ABGABE DER SOFTWAREABLAG	3
1.7	BEWERTUNGSKRITERIEN	3
2	ANHANG	4
2.1	WAS IST MARKDOWN?	4
2.2	KOMPETENZMATRIX	4
3	ANHANG	6
3.1	HIGHER ORDER FUNCTIONS – HOF	6

1 Cool Cars

1.1 Einleitung

Im Rahmen dieses Projekts sollen Sie eine App mit einem Frontend und einem Backend weiterentwickeln. Dabei sollen Sie die Konzepte der funktionalen Programmierung in JavaScript/React anwenden.

Zielsetzung

Erweitern Sie die App, sodass sie ein benutzerfreundliches Frontend als auch ein effizientes Backend aufweist. Demonstrieren Sie dabei die Prinzipien der funktionalen Programmierung, wie sie in der Kompetenzmatrix beschrieben sind.

Zeitvorgaben

4 + 3 x 2 Lektionen

27.11.2024	Projektstart, Anwendung (Git-Repo-Link)	(4 Lektionen)
04.12.2024	Bearbeitung des Projektes (Update Git-Repos)	(2 Lektionen)
11.12.2024	Bearbeitung des Projektes (Update Git-Repos), Zwischenabgabe	(2 Lektionen)
18.12.2024	Bearbeitung des Projektes (Update Git-Repos), Endabgabe	(2 Lektionen)

Form

Partnerarbeit

1.2 Anforderungen - Verständnis der funktionalen Programmierung

Wenden Sie die Konzepte der funktionalen Programmierung (z.B. pure functions, immutable values, etc.) an.

Zeigen Sie den Unterschied zu anderen Programmierparadigmen auf, die Sie eingesetzt haben.

Integrieren Sie die funktionalen Programmierungskonzepte im Frontend.

1.3 Erweiterungen

Die folgenden Erweiterungen könnten mit Funktionaler Programmierung umgesetzt werden. Selbstverständlich gibt es viele Möglichkeiten Lösungen einzubauen, welche mit Funktionaler Programmierung umgesetzt werden können.

Die Funktionalität der Anwendung kann durch verschiedene Erweiterungen erweitern werden, die die Prinzipien der funktionalen Programmierung verdeutlichen und anwenden.

Hier ein paar Beispiele für eine erweiterte Funktionalität:

- Sortierung der einzelnen Spalten (ASC/DESC)
- Suche über alle Einträge
- Paging bei vielen Einträgen
- Filterfunktionen und /oder Dropdowns



1.4 Ausgangslage

Cool-Cars Frontend: <https://github.com/bbwrl/m450-cool-cars-fe.git>




Cool-Cars Backend: <https://github.com/bbwrl/m450-cool-cars-be.git>

1.5 Aufgabe - Dokumentation (Kompetenznachweis)

Erstellen Sie eine Dokumentation (Kompetenznachweis), der die Anwendung der funktionalen Programmierung in Ihrem Projekt beschreibt. Beziehen Sie sich dabei auf die relevanten Kompetenzen aus der beiliegenden Kompetenzmatrix. Deklarieren Sie die Stufe, auf der Sie sich befinden (die einzelnen Partner der Gruppe).

Hier sind einige konkrete Vorschläge:



- Implementieren Sie eine Sortierfunktion der einzelnen Spalten (ASC/DESC), die es ermöglicht, die Einträge in der Passwort-Safe-App sowohl aufsteigend als auch absteigend zu sortieren. Nutzen Sie dafür pure functions, um sicherzustellen, dass die Sortieroperationen keine Seiteneffekte haben und die Daten unverändert bleiben. Dies steht im Einklang mit der Kompetenz AG1 AE1 und AF1, bei der die Eigenschaften von Funktionen beschrieben und von anderen Programmierstrukturen abgegrenzt werden. 
- Entwickeln Sie eine Suchfunktion, die über alle gespeicherten Einträge hinweg nach spezifischen Begriffen sucht. Verwenden Sie dafür Lambda-Ausdrücke und Higher-Order Functions, um eine flexible und erweiterbare Suche zu ermöglichen. Diese Erweiterung unterstützt die Kompetenz C3E und C3F, die den Einsatz von Lambda-Ausdrücken zur Steuerung des Programmflusses beschreibt. 
- Fügen Sie eine Paging-Funktion hinzu, die es ermöglicht, grosse Mengen von Daten in überschaubaren Seiten zu navigieren. Implementieren Sie diese Funktion so, dass die Daten in kleinen, unveränderlichen Chunks geladen und angezeigt werden. Dies entspricht der Kompetenz BE1 und BF1, bei der der Endzustand als Anforderung im Sinne der deklarativen Programmierung beschrieben wird. 
- Implementieren Sie Filterfunktionen, die es den Benutzern ermöglichen, die Einträge nach bestimmten Kriterien zu filtern. Verwenden Sie hierbei Immutable Data Structures, um sicherzustellen, dass die Originaldaten unverändert bleiben. Diese Funktionalität greift die Kompetenz AE1 und AF1 auf, bei der das Konzept von immutable values erläutert und angewendet wird.

Durch die Umsetzung in diesem Projekt demonstrieren Sie nicht nur die Anwendung der funktionalen Programmierung, sondern erfüllen auch die Anforderungen und Kompetenzen aus der Kompetenzmatrix.

1.6 Abgabe der Softwareablage

Das Projekt ist mit und in einem Softwareverwaltungssystem zu führen (Git). Die beiden Partnerinnen und Partner, wie auch die Lehrperson haben Zugriff auf das Repository.

Die Entwicklung ist von beiden Partnerinnen und Partner durchzuführen. Diese checken ihren Beitrag zur Erweiterung der Applikation regelmässig ein. Die Aufgaben sind so zu planen, dass beide Partner am Projekt arbeiten können. Eine komplexe Branch Struktur wird aber nicht erwartet.

→ Repository, Lehrperson einladen und Zugriffsrechte einstellen (BBWRL)

→ Dokumentation und Reflexion, Kompetenznachweis

1.7 Bewertungskriterien

2 Punkte Form und Zusammenarbeit in der Gruppe

2 Punkte Einhaltung der Termine



10 Punkte Dokumentation und Reflexion, bzw. Kompetenznachweis

10 Punkte Umsetzung in der APP, Code Beispiele

24 Punkte Total

2 Anhang

2.1 Was ist Markdown?

Markdown ist eine einfache Auszeichnungssprache, bei der, ähnlich wie bei dem bekannten Wiki-Text, Dokumente mit Markdown über spezielle Sonderzeichen formatiert werden können. Markdown kann mit jedem simplen Text-Editor oder mit speziellen Editoren erstellt werden.

<https://de.wikipedia.org/wiki/Markdown>

<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>



2.2 Kompetenzmatrix

Kompetenzband:	HZ	Grundlagen	Fortgeschritten	Erweitert
Unterschiede zwischen funktionaler Programmierung und anderen Programmierparadigmen aufzeigen.	1	AG1: Ich kann die Eigenschaften von Funktionen beschreiben (z.Bsp. pure function) und den Unterschied zu anderen Programmier-Strukturen erläutern (z.Bsp. zu Prozedur).	AF1: Ich kann das Konzept von <i>immutable values</i> erläutern und dazu Beispiele anwenden. Somit kann ich dieses Konzept funktionaler Programmierung im Unterschied zu anderen Programmiersprachen erklären (z.Bsp. im Vergleich zu referenzierten Objekten)	AE1: Ich kann aufzeigen wie Probleme in den verschiedenen Konzepten (OO, prozedural und funktional) gelöst werden und diese miteinander vergleichen.
Anforderungen und Design beschreiben	1	BG1: Ich kann den Unterschied zwischen Anforderungen der imperativen Programmierung (definierte Folge von Handlungsanweisungen) und der deklarativen Programmierung (Beschreibung des Endzustandes) erklären. BG2: Ich kann Elemente des Functional Design erklären. (z.Bsp. Immutable data types, model, solution, domain of interest, constructors, composable operators)	BF1: Ich kann den Endzustand als Anforderung im Sinne der deklarativen Programmierung beschreiben. (Das gewünschte Ergebnis wird beschrieben statt die Arbeitsschritte.) BF2: Ich kann für eine Problemstellung ein Functional-Design entwerfen und dabei die Elemente des Functional Designs anwenden.	BE1: Ich kann Anforderungen aus der imperativen Programmierung in Anforderungen der deklarativen Programmierung transferieren. ("klar definierte Abfolge" transformieren zu "Endergebnis beschreiben") BE2: Ich kann ein Design einer imperativen Programmierung in ein Design der deklarativen Programmierung transferieren.
Funktionale Programmierung umsetzen	2	C1G: Ich kann ein Algorithmus erklären C2G: Ich kann Funktionen als Objekte behandeln und diese in Variablen speichern und weitergeben. C3G: Ich kann einfache Lambda-Ausdrücke schreiben, die eine einzelne Operation durchführen, z.B. das Quadrieren einer Zahl oder das Konvertieren eines Strings in Großbuchstaben.	C1F: Ich kann Algorithmen in funktionale Teilstücke aufteilen C2F: Ich kann Funktionen als Argumente für andere Funktionen verwenden und dadurch höherwertige Funktionen erstellen. C3F: Ich kann Lambda-Ausdrücke schreiben, die mehrere Argumente verarbeiten können.	C1E: Ich kann Funktionen in zusammenhängende Algorithmen implementieren C2E: Ich kann Funktionen als Objekte und Argumente verwenden, um komplexe Aufgaben zu lösen und den Code sauberer und effizienter zu gestalten. C3E: Ich kann Lambda-Ausdrücke verwenden, um den Programmfluss zu steuern, z.B. durch Sortieren von Listen basierend auf benutzerdefinierten Kriterien.



Kompetenzband:	HZ	Grundlagen	Fortgeschritten	Erweitert
Refactoring und bestehenden Code optimieren		C4G: Ich kann die Funktionen Map, Filter und Reduce einzeln auf Listen anwenden.	C4F: 1. Ich kann Map, Filter und Reduce kombiniert verwenden, um Daten zu verarbeiten und zu manipulieren, die komplexere Transformationen erfordern.	C4E: 1. Ich kann Map, Filter und Reduce verwenden, um komplexe Datenverarbeitungsaufgaben zu lösen, wie z.B. die Aggregation von Daten oder die Transformation von Datenstrukturen.
	3,4	DG1: Ich kann einige Refactoring-Techniken aufzählen, die einen Code lesbarer und verständlicher machen.	DF1: Ich kann mit Refactoring-Techniken einen Code lesbarer und verständlicher machen.	DE1: Ich kann die Auswirkungen des Refactorings auf das Verhalten des Codes einschätzen und sicherstellen, dass das Refactoring keine unerwünschten Nebeneffekte hat.
		DG2: Ich kann allgemeine Massnahmen zur Verbesserung der Leistung von Code aufzählen.	DF2: Ich kann vorgegebene Massnahmen zur Verbesserung der Leistung von Code umsetzen.	DE2: Ich kann effiziente Algorithmen, Techniken oder Datenstrukturen auswählen und einsetzen, um die Leistung von Code zu verbessern.

3 Anhang

3.1 Higher Order Functions – HOF

Higher-Order Functions (HOFs) sind Funktionen, die entweder eine oder mehrere Funktionen als Argumente entgegennehmen oder als Ergebnis zurückgeben. In der funktionalen Programmierung sind sie ein zentrales Konzept, da sie es ermöglichen, abstrakte und wiederverwendbare Funktionsblöcke zu erstellen.

Hier sind einige wichtige Eigenschaften und Beispiele von Higher-Order Functions:

Eigenschaften von Higher-Order Functions

- Akzeptieren von Funktionen als Argumente: Eine HOF kann eine oder mehrere Funktionen als Eingabeparameter akzeptieren.
- Zurückgeben von Funktionen: Eine HOF kann eine Funktion als Rückgabewert haben.
- Kombinierbarkeit: HOFs können kombiniert werden, um komplexe Operationen in einfache, modulare Teile zu zerlegen.

Beispiele von Higher-Order Functions

map:

```
const numbers = [1, 2, 3, 4];  
const doubled = numbers.map(num => num * 2);  
console.log(doubled); // [2, 4, 6, 8]
```

In diesem Beispiel ist map eine HOF, die eine Funktion (`num => num * 2`) als Argument nimmt und diese Funktion auf jedes Element des Arrays numbers anwendet.

filter:

```
const numbers = [1, 2, 3, 4];  
const evens = numbers.filter(num => num % 2 === 0);  
console.log(evens); // [2, 4]
```

Hier ist filter eine HOF, die eine Funktion (`num => num % 2 === 0`) als Argument nimmt und ein neues Array mit allen Elementen erstellt, für die die Funktion true zurückgibt.

reduce:

```
const numbers = [1, 2, 3, 4];  
const sum = numbers.reduce((total, num) => total + num, 0);  
console.log(sum); // 10
```

reduce ist eine HOF, die eine Funktion (`((total, num) => total + num)`) und einen Startwert (in diesem Fall 0) als Argumente nimmt. Sie wendet die Funktion auf jedes Element des Arrays an und akkumuliert einen einzigen Wert.

sort:

```
const numbers = [4, 2, 3, 1];  
const sorted = numbers.sort((a, b) => a - b);  
console.log(sorted); // [1, 2, 3, 4]
```

In diesem Beispiel nimmt sort eine Vergleichsfunktion (`((a, b) => a - b)`) als Argument, die die Reihenfolge der Elemente im Array definiert.

Nutzen von Higher-Order Functions

- Abstraktion und Wiederverwendbarkeit: HOFs ermöglichen es, häufig verwendete Muster in wiederverwendbare Funktionen zu abstrahieren.
- Klarheit und Kürze: Code, der HOFs verwendet, ist oft klarer und prägnanter.
- Modularität: HOFs fördern die Trennung von Bedenken, indem sie komplexe Operationen in kleinere, verständliche Teile zerlegen.

Insgesamt sind Higher-Order Functions ein mächtiges Werkzeug in der funktionalen Programmierung, das dabei hilft, sauberen, modularen und wiederverwendbaren Code zu schreiben.