

Name: _____

Matrikelnummer: _____

Deine Note in der Endklausur sagt nichts über dein Talent als Fullstack-EntwicklerIn aus. Viele wesentliche Fähigkeiten können nicht in einer schriftlichen Prüfung bewertet werden, z.B. selbstständiges Lernen, Wissensvermittlung, Einfühlungsvermögen und Hilfsbereitschaft innerhalb des Teams.

Der primäre Zweck der Endklausur ist es, eine formelle Anforderung der Universität zu erfüllen, nämlich zu einer quantifizierbare Metrik pro Teilnehmer zu finden.

Abgesehen davon dient die Klausur als Motivation: Wenn du diesen Satz liest, dann bereitest du dich wahrscheinlich auf die Prüfung vor. In diesem Fall hat die Endklausur und die Probeklausur ihren Zweck erfüllt.

Viel Erfolg!
Robert Schäfer

[illegible]

1. Project Management

- 1 (a) Vervollständige den folgenden Satz: Das Motto unseres Kurses ist: “Le__n and S____e”
- 1 (b) Was ist der Busfaktor? Schreibe eine Definition auf.

.....

.....

- 1 (c) Was ist besser? Ein möglichst hoher Busfaktor oder ein niedriger Busfaktor?
- A. Ein hoher Busfaktor ist besser.
- B. Ein niedriger Busfaktor ist besser.
- 3 (d) Wie kann der Wissenstransfer im Team gewährleistet werden? Nenne mindestens drei unterscheidbare Methodiken, die wir im Kurs kennen gelernt haben:

.....

.....

.....

- 4 (e) Wobei helfen automatisierte Software-Tests?

Software-Tests helfen dabei:

- ☐ Regressionen zu vermeiden
- ☐ das Verhaltens der Anwendung zu dokumentieren
- ☐ Abhängigkeiten im Quellcode zu erkennen und zu vermeiden
- ☐ bei der Software-Architektur, etwa bei beim Design von Schnittstellen
- ☐ bei der Wartung des Quellcodes, z.B. beim Refactoring
- ☐ die Korrektheit des Quellcodes zu beweisen
- ☐ sicherzustellen, dass die Anforderungen des Kunden bzw. Endbenutzers erfüllt sind
- ☐ Aussagen über das Laufzeitverhalten zu treffen, z.B. Komplexitätsklassen von Algorithmen

- (f) Wir haben in unserem Kurs einen Software-Test als “positiv” definiert, wenn er fehlschlägt. (Also ähnlich wie ein “positiver” medizinischer Test, welcher aussagt, dass eine Testperson mit einer Krankheit infiziert sei.)

Fehlerhafte Software-Tests können zu folgenden Problemen führen:

- A. Das Team verliert an Disziplin und das Vertrauen in Software-Testing. Es beginnt Änderungen am Quellcode zu akzeptieren, obwohl der Build-Server fehlgeschlagen ist.
- B. Es kommt zu Verzögerungen beim Ausrollen der Anwendung (Deployment).
- C. Die Endbenutzer können betroffene Funktionen der Anwendung nicht mehr nutzen.
- D. Es können Sicherheitslücken entstehen.

- 2 i. Welche dieser Probleme entstehen durch “falsch-positive” Software-Tests? Auswahl: _____
- 2 ii. Welche dieser Probleme entstehen durch “falsch-negative” Software-Tests? Auswahl: _____

- 2 (g) Mit welcher Technik kann man “falsch-negative” Tests vorbeugen? Anders gefragt: Wie kann man sicherstellen, dass ein Software-Test überhaupt einen Fehler aufzeigen würde?

.....

2. Git

- (a) Folgendes Szenario: Wir haben vergessen, rechtzeitig einen “Feature-branch” zu erstellen und jetzt zeigt der **master** branch auf unseren letzten commit mit der ID “G”. Abbildung 1 visualisiert den **git** Graph zu diesem Zeitpunkt.

Unser Ziel ist es, nachträglich einen Feature-Branch zu erstellen damit wir einen Pull Request mit dem gewünschten Commit-Log erstellen können. Außerdem wollen wir unseren lokalen **master** branch auf den Stand des **origin/master** zurücksetzen. Als letztes strukturieren wir die commits mit einem interaktiven **rebase** um, mit dem Ziel, unserem Team das “Code-Review” erleichtern.

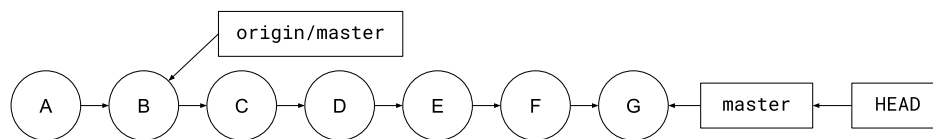


Abbildung 1: Visualisierung des initialen **git**-Graphen.

- 1 i. Im ersten Schritt führen wir den folgenden Befehl aus:

```
git checkout -b feature-branch
```

Wie verändert sich der **git**-Graph? Skizziere den Graphen:

- 1 ii. Im nächsten Schritt führen wir den folgenden Befehl aus:

```
git checkout master
```

Wie verändert sich der **git**-Graph? Skizziere den Graphen:

1

- iii. Wir setzen den **master** auf den Stand des **origin/master** zurück mit diesem Befehl:

```
git reset origin/master --hard
```

Skizziere den **git**-Graphen:

1

- iv. Wir wechseln zurück auf den branch **feature-branch** mit diesem Befehl:

```
git checkout feature-branch
```

Schließlich beginnen wir ein interaktives **rebase** mit diesem Befehl:

```
git rebase origin/master --interactive
```

Danach öffnet sich mein Editor mit einer Liste von commits. Welche commits befinden sich in dieser Liste?

.....

1

- v. Die Liste der commits sortiere ich neu, entferne einige commits und verschmelze commits mit **squash**. Das sieht so aus:

```
pick E      commit message of E
pick D      squash commit message of D
squash F    with commit message of F
pick C      commit message of C
```

Wenn ich jetzt diese Änderung am “rebase-log” abschließe, dann löse ich das **rebase** aus. Dieser Prozess wird aber einmal unterbrochen. Warum?

.....

- 1 vi. Wie sieht der **git**-Graph letztendlich aus? Skizziere den finalen Graphen:

3. Funktionale Programmierung

- 2 (a) Was sind “higher order functions”?

.....

- 2 (b) Im folgenden Code-Beispiel wird eine Funktion namens **higherOrderFunction** deklariert, die Argumente der Funktion heißen **some**, **arguments**, **you**, **can** und **choose**. Bitte schreibe eine Implementierung, sodass die Funktion *ganz offensichtlich* zu einer “higher-order function” wird.

```
function higherOrderFunction(some, arguments, you, can, choose) {
  // Write down any implementation, which obviously turns this function into a
  // higher order function.
  // ...
```

```
  // return ?
  // If you want, return something
}
```

- (c)

Lerne folgende Methoden auf **Array.prototype**: **forEach**, **map**, **find**, **filter**, **reduce**

Die folgenden Code-Beispiele enthalten unnötige temporäre Zustände. Zustände sind immer schlecht, weil sie mögliche Fehlerquellen sind. Wie können die folgenden Code-Beispiele umgeschrieben (refactored) werden, um die unnötigen Zustände zu eliminieren? Das Verhalten des Codes darf sich nicht verändern.

```
function findNextId (){
  let lastId = 0;
  for (i = 0; i < data.todos.length; i++) {
    if (data.todos[i].id > lastId) {
      lastId = data.todos[i].id;
    }
  }
  lastId += 1;
  return lastId;
}
```

```
function filterTodos(mok, userAuth) {  
  var retArray = []  
  
  for (var i = 0; i < mok.length; i++) {  
    if (mok[i].userAuth == userAuth) {  
      retArray.push(mok[i])  
    }  
  }  
  return retArray  
}
```

- 3 i. Es gibt für jedes Code-Beispiel einen Punkt, wenn die richtige Funktion von **Array.prototype** ausgewählt wird.
- 3 ii. Für jedes korrekte Refactoring eines Code-Beispiels gibt es einen weiteren Punkt. Geringfügige Syntax-Fehler führen zu keinem Punktabzug.

4. Authorization and Authentication

- 2 (a) Was bedeuten die Begriffe “Authorization” und “Authentication”? Erkläre, welche Aufgaben mit den Begriffen gemeint sind und grenze sie voneinander ab.
-
-
-
- 1 (b) Werden die Nutzdaten (der Payload) eines JWT (JSON-Web-Token) verschlüsselt übertragen?
- A. Ja
- B. Nein

- 1 (c) Wenn der Schlüssel zur Überprüfung der Signatur der JSON-Web-Token erneuert wird, werden daraufhin alle bis dahin ausgelieferten Token ungültig?
- A. Ja
- B. Nein

5. GraphQL and Apollo-Server

- 7 (a) Welche Probleme löst GraphQL in Bezug auf REST?
- ☐ HTTP-Caching Mechanismen werden besser ausgenutzt .
 - ☐ Auf dem Client kann die Benutzeroberfläche automatisch aktualisiert werden.
 - ☐ Die Anzahl der Anfragen wird minimiert.
 - ☐ Die Menge der ausgetauschten Daten pro Anfrage (der Traffic) wird minimiert.
 - ☐ Die Anzahl der Datenbank-Abfragen wird minimiert.
 - ☐ Daten können in einer Graph-Datenbank gespeichert werden.
 - ☐ Der Server wird robuster gegenüber Denial-of-Service Angriffen.
- (b) Die Methodensignatur eines resolvers enthält vier Argumente, nämlich **parent**, **args**, **context** und **resolveInfo**.

- 1 i. Sollte das Argument **context** in allen resollern gleich sein?
- A. Ja
- B. Nein

- 1 ii. Wozu wird das Argument **resolveInfo** genutzt?

.....

.....

- 2 (c) Schau auf die Implementierung eines Apollo-Servers in Abbildung 9. Wenn wir die GraphQL Anfrage in Abbildung 2 an den Server schicken, mit welcher Antwort reagiert der Server?

```
query {  
  allStudents(limit: 2) {  
    id  
    fullname(reverse: true)  
  }  
}
```

Abbildung 2: Graphql query



(d) Was schreibt der Server als Ausgabe aufs Terminal? Schreibe die `console.log` Ausgaben auf. Folgende korrekte Angaben werden bewertet:

1

i. Korrekte Auswahl, welche Log-Ausgaben überhaupt erreicht werden

1

ii. Korrekte Anzahl aller Ausgaben

1

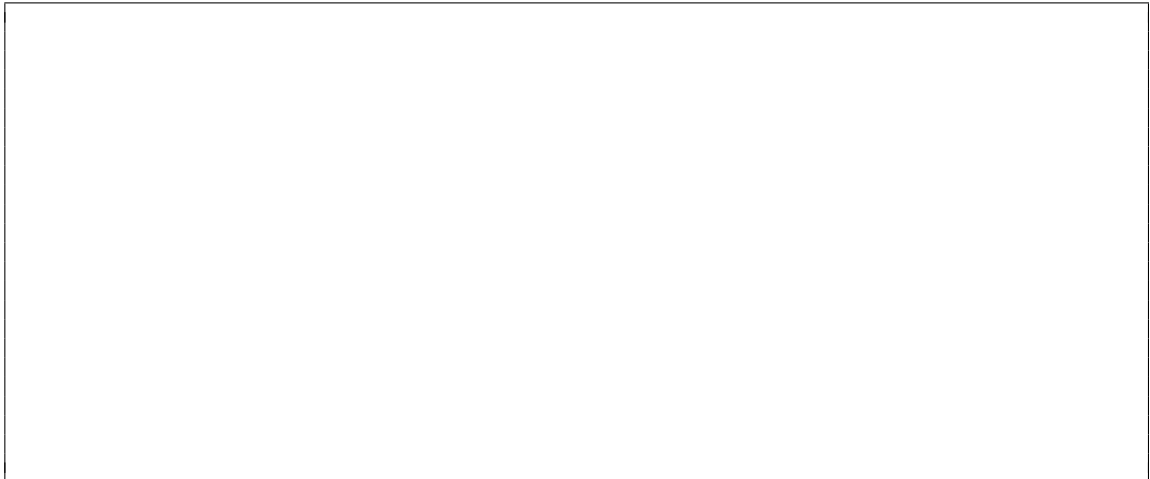
iii. Korrekte Reihenfolge der Ausgaben

1

iv. Korrekter Wert des Arguments **parent**

1

v. Korrekter Wert des Arguments **args**




```
1  const { applyMiddleware } = require('graphql-middleware')
2  const { ApolloServer, gql } = require('apollo-server')
3  const { makeExecutableSchema } = require('graphql-tools')
4
5  const typeDefs = `
6  type Student {
7    id: ID!
8    firstname: String!
9    lastname: String!
10   fullname(reverse: Boolean!): String!
11 }
12
13 type Query {
14   student(id: ID!): Student!
15   allStudents(limit: Int!): [Student!]
16 }
17 `
18 const students = [
19   { id: 1, firstname: 'Alice', lastname: 'Wonderland' },
20   { id: 2, firstname: 'Bob', lastname: 'Builder' },
21   { id: 3, firstname: 'Mallory', lastname: 'Malicious' },
22 ]
23
24 const middleware = {
25   Student: {
26     fullname: (resolve, parent, args, context, resolveInfo) => {
27       const result = resolve(parent, args, context, resolveInfo)
28       console.log('Student.fullname resolved:', result)
29       return result
30     }
31   }
32 }
33
34 const resolvers = {
35   Query: {
36     student: (parent, args, context, resolveInfo) => {
37       console.log('Query.student:', parent, args)
38       return students.find(student => student.id === args.id)
39     },
40     allStudents: (parent, args, context, resolveInfo) => {
41       console.log('Query.allStudents:', parent, args)
42       return students.slice(0, args.limit || students.length)
43     },
44   },
45   Student: {
46     fullname: (parent, args, context, resolveInfo) => {
47       console.log('Student.fullname:', parent, args)
48       if (args.reverse) return [parent.lastname, parent.firstname].join(' ')
49       return [parent.firstname, parent.lastname].join(' ')
50     }
51   }
52 }
53
54 let schema = makeExecutableSchema({ typeDefs, resolvers })
55 schema = applyMiddleware(schema, middleware)
56 const server = new ApolloServer({ schema })
57 server.listen().then(({ url }) => {
58   console.log(`Server ready at ${url}`)
59 })
```

Abbildung 3: Apollo server implementation

6. Neo4J

- (a) Wie wird das relationale Datenmodell in ein graph-basiertes Datenmodell übertragen? Nimm als Beispiel eine Neo4J-Datenbank.

1

i. Der Name einer Tabelle wird zu: _____

1

ii. Die Spalten in einer Tabelle werden zu: _____

1

iii. Eine Zeile in einer Tabelle wird zu: _____

1

iv. Fremdschlüssel zu anderen Tabellen werden zu: _____

- (b) Betrachte die **cypher** Anfrage:

```
MATCH (node)-[*]->(yetAnotherNode) RETURN *
```

- i. Was bedeutet der Stern in den eckigen Klammern im “pattern” dieser query?

.....

- ii. Ist so eine Anfrage auch in einer relationalen Datenbank mit SQL möglich?

- A. Ja
 B. Nein

2

- (c) Können in einer Neo4J Datenbank Fremdschlüssel auf fehlende Einträge zeigen? Solche Zeiger ins Leere heißen auch “dangling pointer”. Begründe deine Antwort:

.....

1

- (d) Wieviele “Labels” darf ein Knoten in einer Neo4J Datenbank haben? Antwort: _____

1

- (e) Wieviele “Typen” darf eine Beziehung in einer Neo4J Datenbank haben? Antwort: _____

- (f) Wir arbeiten auf einer leeren Neo4J Datenbank. In die Neo4J-Webkonsole geben wir folgende **cypher** Befehle ein:

```
CREATE(romeo:Person {name: 'Romeo'})
CREATE(juliet:Person {name: 'Juliet'})
```

Abbildung 4 (a) zeigt eine Graph-Visualisierung nachdem wir die Befehle ausgeführt haben.

Nun geben wir die folgende **cypher** Befehle ein:

```
MERGE(:Person {name: 'Romeo'})-[:LOVES]->(:Person {name: 'Juliet'})
```

Jetzt sieht der Graph wie auf Abbildung 4 (b) aus.

1

- i. Wie können die **cypher** Befehle so verändert werden, dass keine Duplikate mehr entstehen?

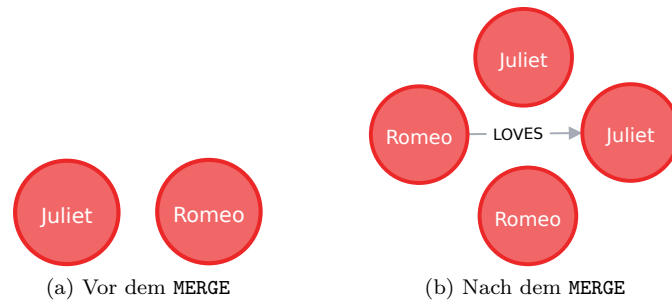


Abbildung 4: Warum entstehen Duplikate?

2

- ii. Warum entstehen überhaupt Duplikate? Wie verhält sich MERGE hier in diesem Beispiel und warum entstehen daraufhin die zwei zusätzliche Knoten?

.....

.....

.....

.....

7. VueJS and NuxtJS

- 4 (a) Zu den Vorteilen von client-seitig gerenderten “Single-page-applications” gegenüber traditionellen server-seitig gerenderten Web-Anwendungen gehören unter anderem:
- ☐ Weniger Last auf dem Server
 - ☐ Desktop-ähnliches Nutzererlebnis
 - ☐ Größerer Anteil Nutzdaten in der Kommunikation zwischen Client- und Server
 - ☐ Bessere Suchmaschinen-Optimierung
 - ☐ Kompatibilität mit älteren Browsern
- 1 (b) Was ist isomorpher Quellcode?
-
-
- 3 (c) Welche dieser sog. “JavaScript globals” sollte in Quellcode vermieden werden, welcher auch von server-seitig vorgerenderten Web-Anwendungen ausgeführt wird:
- ☐ `document`
 - ☐ `console`
 - ☐ `window`
- 1 (d) Zu welcher Art von Fehler kommt es, wenn man trotzdem eines dieser “globals” in einem NodeJS Prozess aufruft?
-
- (e) Wirf einen Blick auf die Vue-Komponente im Code-Beispiel 5. Wenn in den folgenden Teilaufgaben Änderung im Quelltext vorgenommen werden sollen, können diese direkt in die Abbildung eingepflegt werden.
- 1 i. Offenbar hat sich ein Fehler eingeschlichen. Eigentlich soll die Methode `handleClick` beim Klick auf den Knopf ausgelöst werden. Durch welche Änderung kann der Fehler behoben werden?
- 1 ii. Wenn die Methode `handleClick` ausgelöst wird, welchen Wert hat dann `message`?
- Antwort: _____
- 2 iii. Füge ein `<input>` Feld hinzu, dessen Texteingabe mit dem Zustand `message` verknüpft ist. Bei jeder Text-Eingabe soll sich also der Wert von `message` verändern.
- 2 iv. Implementiere eine computed property namens `reversedMessage`, welche den String `message` umdreht. Tipp:
- ```
someString.split('').reverse().join('')
```
- Dreht eine lokale Variable `someString` herum.
- 1 v. Warum ist es eigentlich sinnvoll, `reversedMessage` als computed property zu implementieren?
- .....
- .....

- 1 vi. Was spricht dagegen, **reversedMessage** als Zustand in **data** zu initialisieren und zur Laufzeit zu verändern?

.....  
 .....

- 1 vii. Was spricht dagegen, es als Methode zu implementieren und als **reversedMessage()** im “template” aufzurufen?

.....  
 .....

```

1 <template>
2 <div>
3 <!-- Hier kann das <input> Feld eingefügt werden -->
4
5
6 <p>
7 {{ message }}
8 </p>
9 <button>
10 Click me
11 </button>
12 <p>
13 {{ reversedMessage }}
14 </p>
15 </div>
16 </template>
17
18 <script>
19 export default {
20 data() {
21 return {
22 message: 'Hello!'
23 }
24 },
25 computed: {
26 // Hier kann die computed property namens `reversedMessage` implementiert
27 // werden
28
29
30
31 },
32 methods: {
33 handleClick() {
34 this.message = `${this.message}!`
35 },
36 },
37 }
38 </script>

```

Abbildung 5: Eine kleine VueJS-Komponente

## 8. Fullstack testing and Requirements Engineering

- 1 (a) Im Kurs haben wir die “Testing-Pyramide” und den “Testing-Diamanten” kennengelernt. Beides sind Modelle, die eine Hilfestellung bieten sollen, um zu einem ausgewogenen Verhältnis der jeweiligen Test-Klassen zu finden. Welche Klasse von Tests hat laut dem “Testing-Diamanten” den größten Wert und sollte deshalb besonders oft implementiert werden?

Antwort: \_\_\_\_\_

- 2 (b) Für welchen Zweck nutzt man Cucumber Testing?

.....  
 .....

- 2 (c) Die Schablone eine User-Story sieht wie folgt aus:

```
As a <role>
I want to do <feature>
In order to <benefit>
```

Diese Schablone soll dazu dienen, dass man bei der Erfassung von Anforderungen bestimmte Informationen nicht vergisst, z.B. die Zielgruppe und den Grund für eine gewünschte Funktion. Warum ist es so wichtig, diese Informationen stets zu dokumentieren?

.....  
 .....  
 .....  
 .....

## 9. CSS

```
1 <form class="login-form">
2 <input type="text">
3 <input type="password">
4 <button class="submit-button">
5 Login
6 - please
7 </button>
8 </form>
```

Abbildung 6: Ein beispielhaftes HTML Dokument

- 3 (a) Welche HTML-Elemente werden bzw. welches HTML-Element wird in Abbildung 6 mit den folgenden CSS Selektoren selektiert? Gib als Antwort Zeilennummern an:

i.

```
.login-form:first-child
```

Antwort: \_\_\_\_\_

ii.

`.login-form :first-child`

Antwort: \_\_\_\_\_

iii.

`.login-form > :first-child`

Antwort: \_\_\_\_\_

1

(b) Mit welchen CSS-Selektor können alle Eingabefelder selektiert werden?

Antwort: \_\_\_\_\_

## 10. Code Review

10

(a) Die folgenden Code-Beispiele sind Auszüge aus den Abgaben der Hausaufgaben und enthalten Schwächen. Finde diese Schwächen und schlage eine Verbesserung vor. Wenn du eine Schwäche finden kannst und eine sinnvolle Verbesserung vorschlägst, gibt es dafür einen Punkt. Wenn ein schwerwiegendes Problem vorliegt und du es erklären kannst, gibt es dafür einen weiteren Punkt. Es spielt keine Rolle, welches Code-Beispiel du dir aussuchst, es werden alle Verbesserungen und Erklärungen für diese Aufgabe summiert.

```
1 import { mount } from '@vue/test-utils'
2 import ListItem from './ListItem'
3 const wrapper = mount(ListItem)
4
5 describe('given a `todo`', () => {
6 it('renders todo text', () => {
7 expect(wrapper.text()).toContain('Save');
8 })
9
10 it('should show an input field', () => {
11 expect(wrapper.emitted('edit', () => {
12 wrapper.contains('input').toBe(false)
13 })))
14 })
15
16 describe('click on delete button', () => {
17 expect(wrapper.emitted('delete'))
18 })
19 })
```

Abbildung 7: Auszug aus einem VueJS Komponenten Test

```
1 import { mount } from '@vue/test-utils'
2 import ListItem from './ListItem.vue'
3
4 describe('ListItem', () => {
5 describe('given an `item`', () => {
6 const dummy = { id: "1", message: "Foo" };
7 const wrapper = mount(ListItem, { propsData: { item: dummy } });
8
9 it('init with dummy parameter', () => {
10 expect(wrapper.vm.item).toEqual(dummy);
11 });
12
13 it('renders item', () => {
14 var foundItem = wrapper.find('#item-description');
15 var itemDescription = wrapper.vm.item.id + ' ' + wrapper.vm.item.message;
16 expect(foundItem.text()).toEqual(itemDescription);
17 });
18
19 describe('testing `Delete` button', () => {
20 it('click on button emits delete event', () => {
21 //wrapper.vm.deleteItem();
22 wrapper.find('#button-delete').trigger('click');
23 //console.log(wrapper.emitted());
24 var itemToDelete = wrapper.emitted('delete-item')[0][0];
25 //console.log(itemToDelete);
26 expect(itemToDelete.message).toEqual(wrapper.vm.item.message);
27 expect(itemToDelete.message).toEqual(dummy.message);
28 });
29 });
30 });
31 });
```

Abbildung 8: Auszug aus einem VueJS Komponenten Test



```
1 export default const typeDefs = `
2 type todos {
3 title: String
4 }
5 type Query {
6 todos: [todos]
7 users: [User]
8 }
9 type User {
10 name: String!
11 password: String!
12 id: Int
13 }
14 type Mutation {
15 addToDo(title: String!, token: String!): [todos]
16 deleteToDo(index: Int!, token: String!): [todos]
17 updateToDo(title: String!, index: Int!, token: String!): [todos]
18 loginUser(username: String!, password: String!): AuthPayload!
19 }
20 input UserLoginInput {
21 username: String!
22 password: String!
23 }
24 type AuthPayload {
25 token: String!
26 }
27 `
```

Abbildung 9: Die GraphQL Typendefinitionen eines Apollo-Servers. Mit **token** ist ein JWT Bearer token gemeint. Das Feld **password** ist tatsächlich ein Benutzerpasswort und ist entsprechend sensibel.

Hier können Probleme ausführlicher erklärt werden, wenn der Platz neben den Code nicht ausreichen sollte. Bitte die Zeilennummer und die Abbildung angeben um mir die Zuordnung zu erleichtern

