Hochschule Hof

University of Applied Sciences

Titel:

# Brain-Check

Vorgelegt von:

Osama Aldalati

Betreut von:

Prof. Dr. Walter Kern

# Inhaltsverzeichnis:

# 1. Einleitung

## 1.1 Was ist Brain-Check

Brain-Check ist ein Quiz-App, der die Users die Möglichkeit geben, ihre Ideen in Form ein Quiz mit anderen Users zu teilen, sowie einer genaueren Analyse ihren Quizze zu erhalten.

## 1.2 Roll

Ich bin der Entwickler im Team gewiesen, und in folgenden Studienarbeit wird unser Projekt in Perspektive der Entwickler beschrieben.

# 2. Überblick über die Entwicklung

Brain-Check ist einer Rest-Architektur basiertes Webanwendung.

## 2.1 Frontend:

In der Frontend-Entwicklung unserer Anwendung haben wir uns für die Verwendung von JavaScript als Programmiersprache und das React-Framework entschieden. Zur Erweiterung der Funktionalität haben wir verschiedene externe Bibliotheken integriert, darunter:

- React Router (react-router und react-router-dom): Diese Bibliotheken ermöglichen das Routing in unserer React-Anwendung. Da React ist Framework für SPA, ist React Router eingesetzt, um User das Gefühl klassische Webanwendungen zu ermöglichen, um User Experience zu schaffen.

- Chart.js und React Chartjs-2 (chart.js und react-chartjs-2): Zur Erstellung von Analyse-Diagrammen von Quizze ist die JavaScript-Bibliothek Chart.js eingesetzt. react-chartjs-2 wiederum bietet eine React-Bindung für die nahtlose Integration von Chart.js in unsere React-Anwendung, Analyse-Diagrammen von Quizze grafisch darzustellen.

- Bootstrap und React Bootstrap (bootstrap und react-bootstrap): Um das Design und Layout unserer Anwendung zu verbessern, verwenden wir Bootstrap. react-bootstrap liefert React-Komponenten, die auf den Bootstrap-Komponenten basieren und eine Integration in React-Anwendungen erleichtern.

(Siehe Projekt\UI)

## 2.2 Backend:

Im Backend, das auf Node.js basiert, verwenden wir eine Vielzahl von Modulen und Bibliotheken, um die Kernfunktionalitäten sowie die Sicherheit unserer Anwendung zu erweitern, sowie Bibliotheken um bessere Dev-Umgebung zu ermöglichen. Jedes dieser Module spielt eine entscheidende Rolle bei der Entwicklung, der Datenverarbeitung und der Sicherstellung der Leistungsfähigkeit unseres Systems. Hier sind die genutzten Abhängigkeiten im Detail:

- express: Als Hauptgrundlage unserer RESTful API bietet express ein leistungsstarkes und flexibles Framework für den Aufbau von Webanwendungen und APIs in Node.js. Es ermöglicht uns, Routen zu definieren, Anfragen zu bearbeiten und Middleware zu implementieren.

- express-validator: Als Middleware für Express dient express-validator der Validierung von Eingabedaten. Es stellt sicher, dass eingehende Benutzerdaten unseren definierten Kriterien entsprechen und säubert die Daten entsprechend.

- jsonwebtoken: jsonwebtoken ist ein Modul für die Erzeugung und Verifikation von JSON Web Tokens (JWT). Diese Tokens werden für die Authentifizierung und Autorisierung von Benutzern in unserer Anwendung verwendet und ermöglichen eine sichere Kommunikation zwischen Client und Server.

- body-parser: Als Middleware für Express, unser Haupt-Framework für die Handhabung von HTTP-Anfragen, erlaubt es uns das Parsen von Daten im Anfrage-Body in verschiedenen Formaten wie JSON oder URL-Encodings. Dies ist wichtig für die Verarbeitung und Interpretation von Anfragen, die an unsere API gesendet werden.

- sequelize: sequelize dient als ORM (Object-Relational Mapping) für die Interaktion mit relationalen Datenbanken in Node.js. Es ermöglicht uns, Datenbankmodelle zu definieren, Datenbankabfragen auszuführen und Beziehungen zwischen Datenobjekten zu verwalten.

- mysql2: Als MySQL-Client für Node.js bietet mysql2 eine Schnittstelle zur Interaktion mit einer MySQL-Datenbank. Es wird von sequelize Technologie erfordert um Access auf Datenbank zu erhalten.

- uuid: uuid ist eine Bibliothek zur Generierung von UUIDs (Universally Unique Identifier). Diese eindeutigen Identifikatoren werden in unserer Anwendung für verschiedene Zwecke verwendet, wie beispielsweise die eindeutige Identifizierung von Datenobjekten.

- nodemailer: Mit nodemailer können wir E-Mails über unsere Node.js-Anwendung versenden. nodemailer ist in unser Projekt in Verknüpfung mit einer Online-Mail Server „Brevo“ und Eine Gmail-Konto „braincheck2023@gmail.com“ eingesetzt

- bcryptjs: Dieses Modul ermöglicht uns das sichere Hashing von Passwörtern, wodurch die Passwortspeicherung in unserer Datenbank geschützt und sicherer wird. Es ist ein essentieller Bestandteil unserer Authentifizierungsstrategie.

- crypto: Als integriertes Node.js-Modul unterstützt uns crypto bei der Implementierung random Code für Email-Verifikation

- nodemon: ist eine devDependencies, welche für Entwickler bessere Dev-Umgebung ermöglicht, durch Server automatisch nach Änderung neuzustarten

- prettier: ist eine devDependencies, welche der Code Format nach festen Schema „.prettierrc“ reformatiert, um Code in Projekt einheitlich zu behalten

(Siehe Projekt\Server)

### 2.3 Datenbank:

Zu unsre Datenspeicherung in das Projekt Quiz-App wurde eine MySQL lokale Server mit folgenden Modellierung aufgesetzt. (Siehe ERR-Diagramm)

## 3. Design Artefakt

### 3.1 Allgemeine Verteilung:

Brain Check ist in drei Hauptsubsysteme unterteilt, welche in nebenstehende Komponentendigramm beschrieben. Die Implementation das System bzw. die drei Subsysteme in fünf Iterationen verteilt.
(Siehe systemKomponentendigramm )

### 3.2 Project <BrainCheck> Iteration <0>:

Start Iteration mit Ziel als Entwickler das Infrastruktur für das Projekt aufzustellen.

Erledigte Tasks:

- NodeJS Projekt zu erstellen und alle nötigen Abhängigkeiten in Package.json File zu konfigurieren

- React Projekt zu erstellen und alle nötige Abhängigkeiten in Package.json File zu konfigurieren

- Datenbank modellieren.
- Projekte auf GitHub Hochladen

## 3.3 Project <BrainCheck> Iteration <1>:

In Iteration 1 war unser Ziel das erste Subsystem Authentifizierung zu implementieren.

In folgenden Abschnitte wird die Implementierung in Frontend und Backend beschrieben, mit Unterstützung von entsprechenden Classen – Sequenzdiagramme, welche Aufbau und Ablauf verdeutlichen.Zum Authentifizierung ist JWT „JSON Web Tokens“ zum Authentifizierung eingesetzt, mit Hilfe von Middleware-Konzept in Express.js. Daher war der Einsatz als eine Middleware, welche Weiterleitung der Request an Controller den Token aus Request Header kontrolliert hat, dann wird der Request an Controller weitergeleitet. In JWT können Daten gespeichert werden, daher ist dieser Prinzip in BrainCheck eingesetzt, um User-ID in Middleware aus Token abzulesen, dann in Request zu speichern, somit in dem Controller festgestellt, von welcher User der Request ist.
(Siehe \Projekt\UI\src\views\auth, \Projekt\Server\routes\auth.js , \Projekt\Server\middleware\is-auth.js )

```
const token = jwt.sign(
  {
    email: user.dataValues.email,
    _id: user.dataValues._id,
  },
  process.env.API_KEY,
  { expiresIn: "1h" },
);
```

*Abbildung 1 create JWT Token*

```
let decodedToken;
try {
  const token = req.get("Authorization");
  decodedToken = jwt.verify(token, process.env.API_KEY);

  if (!decodedToken) {
    throw error;
  }

} catch (err) {
  err.statusCode = 401;
  err.message = "not authenticated";
  throw err;
}


req.userId = decodedToken._id;
next();
```

*Abbildung 2 verify JWT Token*

Erledigte Tasks:

- authController im Backend
  „login, signup, getEmailverification, getEmailverificationAgain, PostEmailverification“
- Auth Route
- AuthLayOut React Component
- SignupPage, LoginPage, actionLogik und Token-management
- Mail Server connection aufstllen

### 3.3.1 Classendiagramm

In folgenden ist die Classendiagramm, welches der Aufbau der Authentifizierung in Frontend beschreibt.
(Siehe Klassen-Diagramm-Frontend)

### 3.3.2 Sequenzdiagramm:

In folgenden PDF ist der Sequenzdiagramme, welche der Ablauf beim Login und Registrieren beschreiben.
(Siehe Login, Signup)

## 3.4 Project <BrainCheck> Iteration <2>:

In Iteration 2 war unser Ziel das zweite Subsystem Create & Answer Quiz zu implementieren.

Das Quiz besteht aus mehrere Fragenarten, welche Einfache Auswahl, Mehrfache Auswahl, Richtig oder Falsch und richtige Text sind. Bei Implementierung das zweite Subsystem wurden express, express-validation eingesetzt. Durch das Prinzipe Validation-Schema von express-validation ist einer Feste Schema zum Validieren von Requeste bei erstellen (Siehe Create Schema)

und antworten ein Quiz. Das Schema entspricht der Folgenden JSON-Schemen:
(Siehe Antworten Schema)

Alle Requeste, welche nicht an den Schemen orientieren, werden von Server nicht bearbeitet.
(Siehe \Projekt\UI\src\views\quiz, \Projekt\Server\routes\quiz.js )

Erledigte Tasks:

- Erstellen von Validation Schemen für Erstellen und Antworten eines Quiz
- Komponente für verschiede Fragenarten implementiert
- Erstellen einer Einheitliche Logik in React für Antworten und Erstellen von Quiz, um die Wiederverwendbarkeit der React Komponente auszunutzen
- Action Logik zum Senden Requeste beim Erstellen und Antworten
- Loader Logik zum Quiz von Server holen und Darstellung
- Controller quiz mit erstellen, löschen, antworten und holen Funktionen

### 3.4.1 Classendiagramm:

In folgenden PDF ist die Classendiagramm, welches der Aufbau der Subsystem Create & Answer Quiz in Frontend beschreibt
(Siehe Erstellen -und Antworten Klassen Diagramm)

### 3.4.2 Sequenzdiagramm:

In folgenden PDF ist der Sequenzdiagramme, welche der Ablauf beim Erstellen eines Quiz beschreiben.
(Siehe Erstellung Sequenzdiagramme)

## 3.5 Project <BrainCheck> Iteration <3>:

In Iteration 3 war unser Ziel das dritte Subsystem Statistik Analyse von Quiz zu implementieren und Rollenmanagement einzustellen.

Für die Darstellung der Statistiken mit Chart.js wurden zwei Diagrammtypen verwendet: der Balken-Chart (Bar-Chart) und das Doughnut-Chart. Der Balken-Chart zeigt, wie viele Teilnehmer jede Frage richtig oder falsch beantwortet haben. Das Doughnut-Chart hingegen stellt die Ergebnisse der Teilnehmer in verschiedenen Bereichen dar, nämlich "0% - 25%", "25% - 50%", "50% - 75%", und "75% - 100%". Beim Bar-Chart wird vermutlich die Anzahl der Teilnehmer dargestellt, die jede Frage richtig oder falsch beantwortet haben. Für jede Frage gibt es zwei Balken im Chart, einer für die richtigen und einer für die falschen Antworten. Das Doughnut-Chart dient wahrscheinlich dazu, die Verteilung der Ergebnisse der Teilnehmer in verschiedenen Prozentbereichen zu zeigen. Dies kann helfen, zu visualisieren, wie viele Teilnehmer beispielsweise weniger als 25%, zwischen 25% und 50%, zwischen 50% und 75%, oder über 75% der Fragen richtig beantwortet haben.
Diese Visualisierungen helfen, einen schnellen Überblick über die Leistung der Teilnehmer in Bezug auf die Fragen des Quiz zu erhalten und ermöglichen eine einfache Interpretation der Verteilung der Antworten.

Nach jeder Quizteilnahme werden die Antworten mit den Benutzer-IDs in der Datenbank gespeichert – eine persönliche 'Antwortliste' für jeden Teilnehmer. Clever automatisiert durch 'Hooks' in Sequelize, aktualisiert das System die Statistik direkt bei neuen Einträgen. Dadurch sind die Diagramme immer topaktuell, ohne manuelle Updates. Der Server liefert diese Daten im richtigen Format ans Frontend für die dynamischen Diagramme.
Dieses System garantiert aktuelle Infos und ermöglicht eine transparente, Echtzeit-Darstellung der Quizleistungen.

**Teilnehmerantworten speichern:** Jedes Mal, wenn jemand am Quiz teilnimmt, werden die Antworten zusammen mit der Benutzer-ID in der Datenbank gespeichert. Das ist wie das Anlegen einer persönlichen "Antwortliste" für jeden Teilnehmer.Automatische. **Aktualisierung der Statistik:** Hier kommt die Cleverness des Systems ins Spiel. Dank eines speziellen Mechanismus namens "Hooks" in Sequelize wird die Statistik -automatisch aktualisiert, sobald neue Antworten in die Datenbank eingetragen werden. Das bedeutet, dass die Diagrammdaten immer aktuell sind – keine manuellen Updates nötig.
**Datenbereitstellung für die Darstellung:** Der Server reicht die aktuellen Statistikwerte in einem bestimmten Format an die Frontend-Anwendung weiter. Diese Daten dienen als Grundlage für die Erstellung der Diagramme, die den Benutzern zeigen, wie gut sie und andere Teilnehmer das Quiz absolviert haben.

Im Grunde genommen sorgt dieses System dafür, dass alle Daten immer auf dem neuesten Stand sind. Es ist wie ein unsichtbarer Helfer im Hintergrund, der sicherstellt, dass die Statistiken in Echtzeit aktualisiert werden, damit die Benutzer die aktuellen Informationen über die Quizleistung haben. Das ermöglicht nicht nur ein genaues Bild der

Ergebnisse, sondern auch eine transparente und aktuelle Darstellung der Teilnehmerleistungen.
(Siehe \Projekt\UI\src\views\profielPage, \Projekt\UI\src\views\statisticPage,
\Projekt\Server\routes\statistic.js, \Projekt\Server\routes\user.js)

Erledigte Tasks:

- Dashboard Komponente
- Aftersave-Hook Logik
- Abholung von Data Logik
- ChartJs einsetzen
- Profile Page
- Logik zum Aufruf von Statistik in Profilepage
- Admin Roll
- Control Panel für Admin
- Admin Controller

### 3.5.1 Classendiagramm:

In folgenden PDF ist die Classendiagramm, welches Das Darstellung des Statistikwerte darstellt
(Siehe Statistik  Klassen Diagramm)

### 3.5.2 Sequenzdiagramm:

In folgenden PDF ist der Sequenzdiagramme, welche der Ablauf beim Updaten einer Statistik beschreiben.
(Siehe Update Sequenzdiagramme)

## 3.5 Project <BrainCheck> Iteration <4>:

Die 4. Iteration ist zum Refactoring von Code in Backend ausgenutzt. Ziel Wiederverwendbarkeit in Code zu maximieren.

# 4. Developer Test:

die Unit-Tests wurden mithilfe des Vitest-Frameworks geschrieben, um wichtige Funktionen des Servers zu prüfen, bevor dieser gestartet wird. Hier sind die verschiedenen Funktionalitäten, die durch diese Tests abgedeckt werden:

**Preparation:** Vorbereitung für die Unit Tests durch Datenbank Verbindung aufbauen und Test Users in Datenbank erzeugen
(Siehe Projekt\Server\tests\preparation.js)

**Benutzer-Authentifizierungstests:** Diese überprüfen, ob das System korrekt auf fehlerhafte Anmeldeinformationen reagiert und ob erfolgreiche Anmeldungen korrekt verarbeitet werden, einschließlich der Generierung von Tokens.
( Siehe Projekt\Server\tests\login.test.js)

**Util-Funktionen-Tests:** Sie stellen sicher, dass die Hilfsfunktionen, die zur Validierung von Quiz-Schemas, Antworten, Ergebnissen und Durchschnittsberechnungen verwendet werden, ordnungsgemäß funktionieren.
(Siehe Projekt\Server\tests\utils.test.js)

**Quiz Erstellung- und Antwortfunktionen-Tests:** Diese Tests decken die Erstellung von Quizfragen und die Verarbeitung von Antworten auf diese Quizfragen ab, um sicherzustellen, dass sie korrekt und gemäß den Anforderungen funktionieren.
(Siehe Projekt\Server\tests\create.test.js, Projekt\Server\tests\answer.test.js)

**Allgemeine Serverfunktionalitäts-Tests:** Sie prüfen grundlegende Funktionen wie den Aufbau der Datenbankverbindung, das ordnungsgemäße Request-Handling und die allgemeine Integrität des Servers.
(Siehe Projekt\Server\tests\app.test.js)

**Datenbankoperationen-Tests:** Diese überprüfen CRUD-Operationen (Create, Read, Update, Delete) in der Datenbank sowie die Ausführung von Abfragen, um sicherzustellen, dass die Datenbankzugriffe korrekt und zuverlässig sind.
(Siehe Projekt\Server\tests\app.test.js)

Diese Unit-Tests sind entscheidend, um sicherzustellen, dass der Server reibungslos und fehlerfrei funktioniert, bevor er live geschaltet wird. Sie gewährleisten, dass alle wesentlichen Funktionen des Servers korrekt implementiert und funktionsfähig sind.

## 5. Schluss

Die Mitwirkung an der Entwicklung der 'Brain-Check'-App war eine bedeutende Erfahrung für mich. Der erste Kontakt mit Node.js und React war aufregend, aber herausfordernd. Die Gestaltung einer neuen Quiz-App war eine lehrreiche Erfahrung.
Das Arbeiten mit neuen Technologien stellte eine echte Herausforderung dar. Es gab viele Ups und Downs, aber jeder Rückschlag brachte mir neue Einsichten und half, meine Kreativität zu fördern.
Die Zusammenarbeit im Team war entscheidend, und die Anleitung von Prof. Dr. Walter Kern war sehr hilfreich.
Diese Erfahrung hat mir nicht nur gezeigt, wie man eine Quiz-Plattform entwickelt, sondern auch, wie man Probleme löst und sich kontinuierlich weiterentwickelt. Sie hat meine Denkweise verändert und mein Verständnis für die Softwareentwicklung vertieft.
Insgesamt war es eine anspruchsvolle, aber äußerst lohnende Erfahrung. Es ging über die Entwicklung einer Anwendung hinaus und war eine Reise des persönlichen und beruflichen Wachstums. Ich bin dankbar für diese Chance und freue mich darauf, weiterhin in diesem Bereich zu lernen und zu wachsen.

## 6. Quellen


Express: https://expressjs.com/de/api.html

NodeJS: https://nodejs.org/docs/latest/api/

React: https://react.dev/reference/react/apis

Express-validator: https://express-validator.github.io/docs/api/check/

React-Router-Dom: https://reactrouter.com/en/main

ChartJs: https://www.chartjs.org/docs/latest/getting-started/

Sequelize: https://sequelize.org/api/v6/

JWT: https://jwt.io/introduction

Bootstrap: https://react-bootstrap.netlify.app/

Quell Code und Ressourcen „Frontend, Backend, Diagramms usw.“ : https://github.com/OSALD2000/Projekt.git

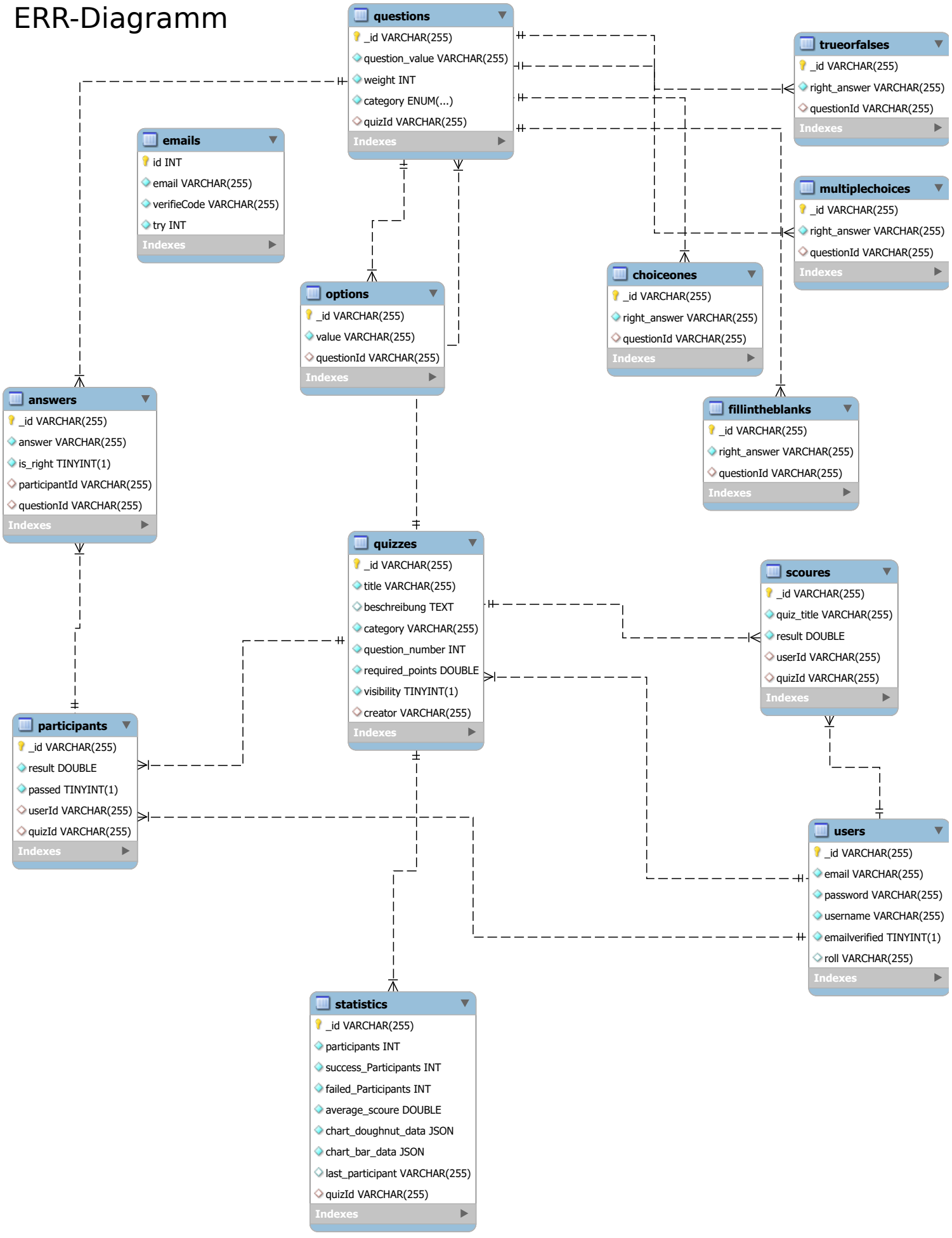
Vorlesung Software-System-Entwicklung

# Selbstständigkeitserklärung Studienarbeit

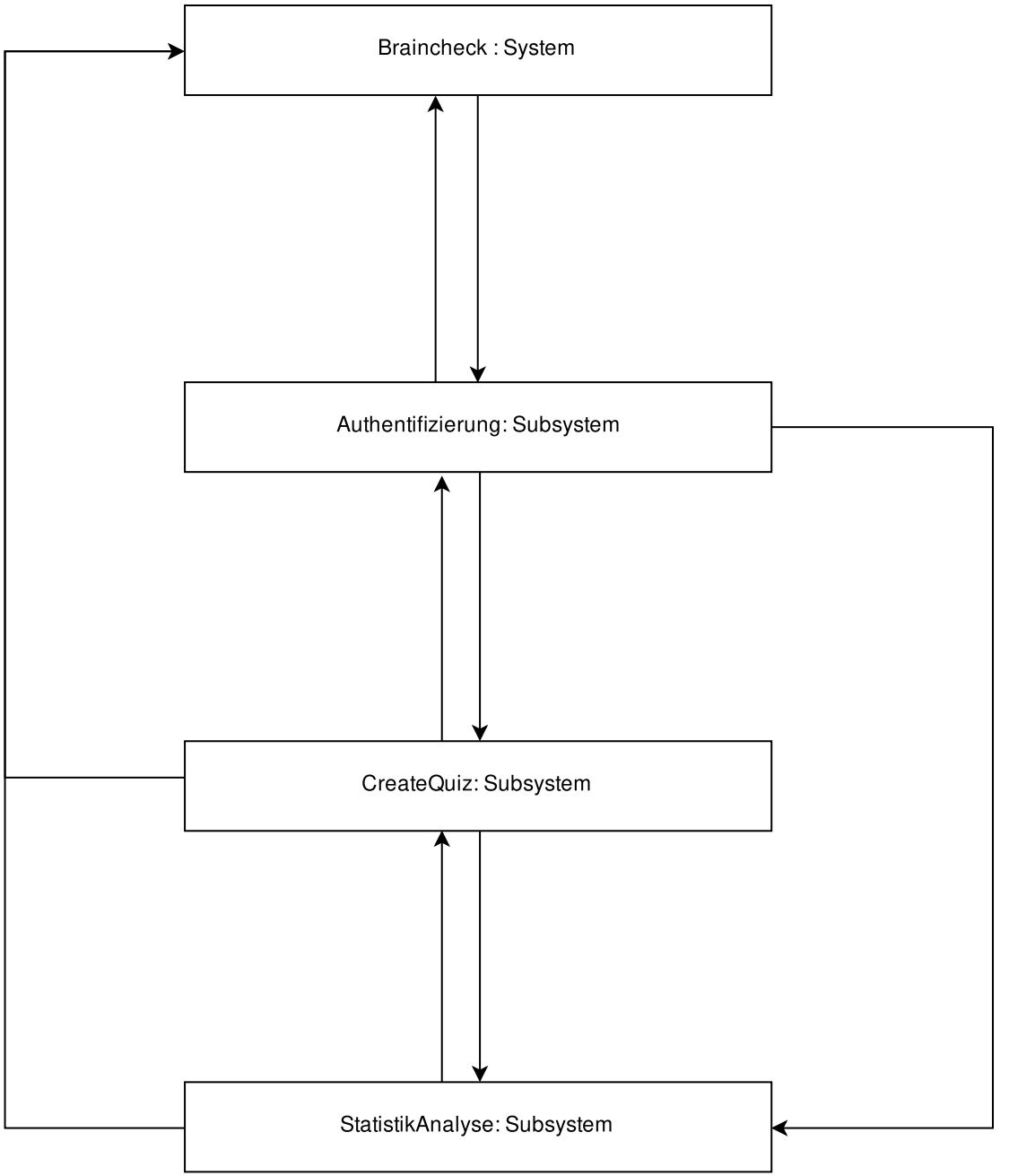
Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.
Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken – dazu gehören auch Internetquellen – als solche kenntlich gemacht habe.

_____ _____

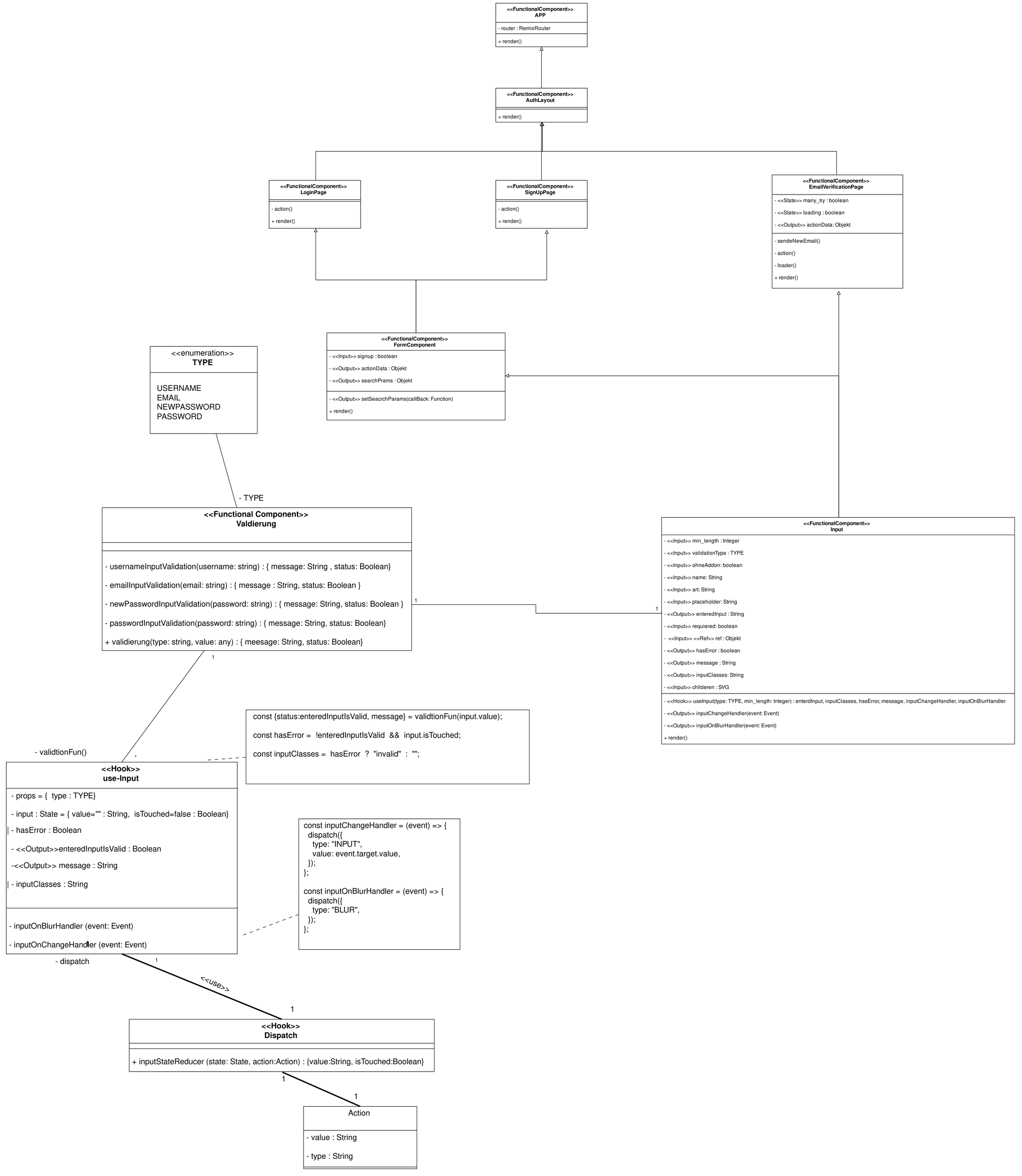
# ERR-Diagramm



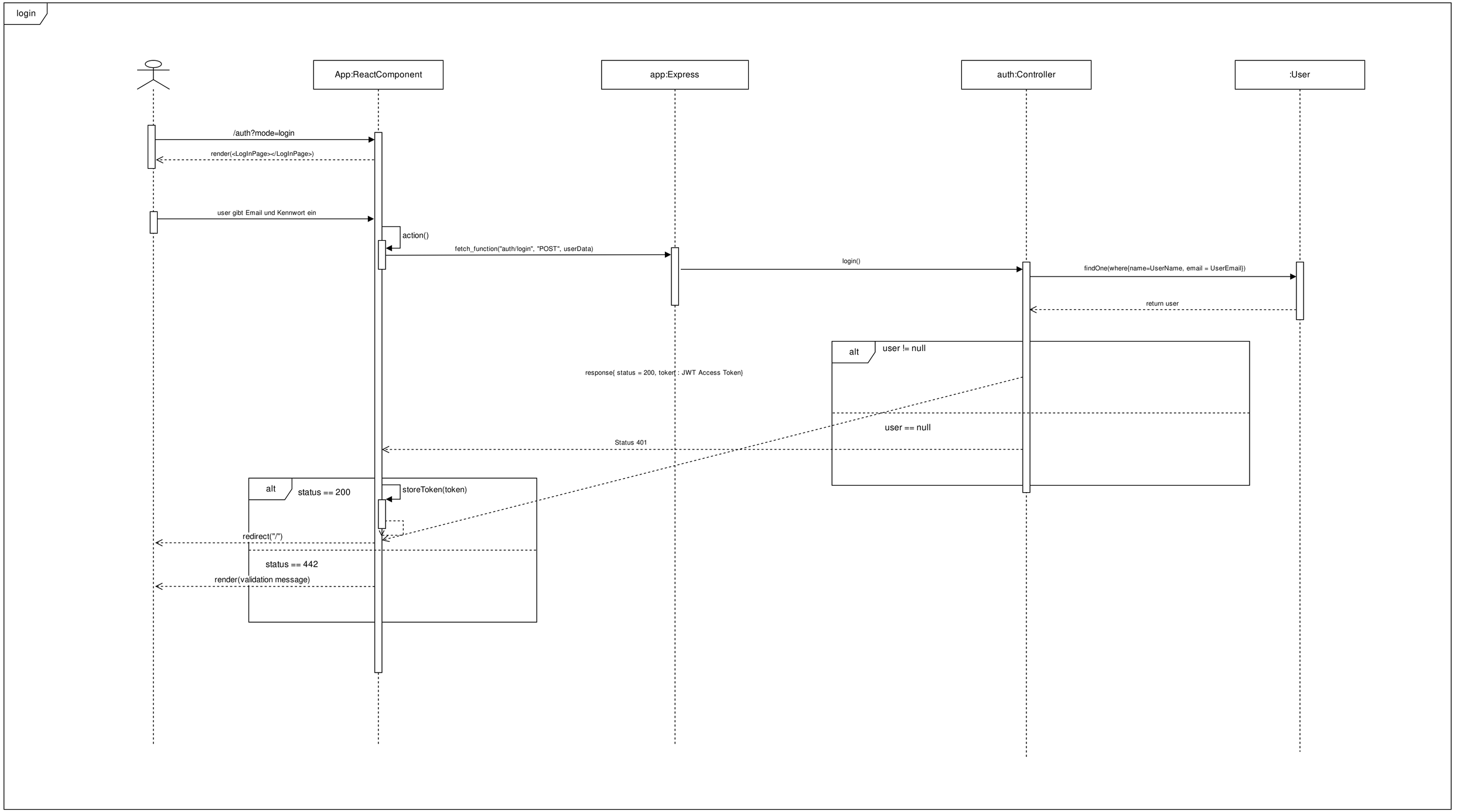
# systemKomponentendigramm



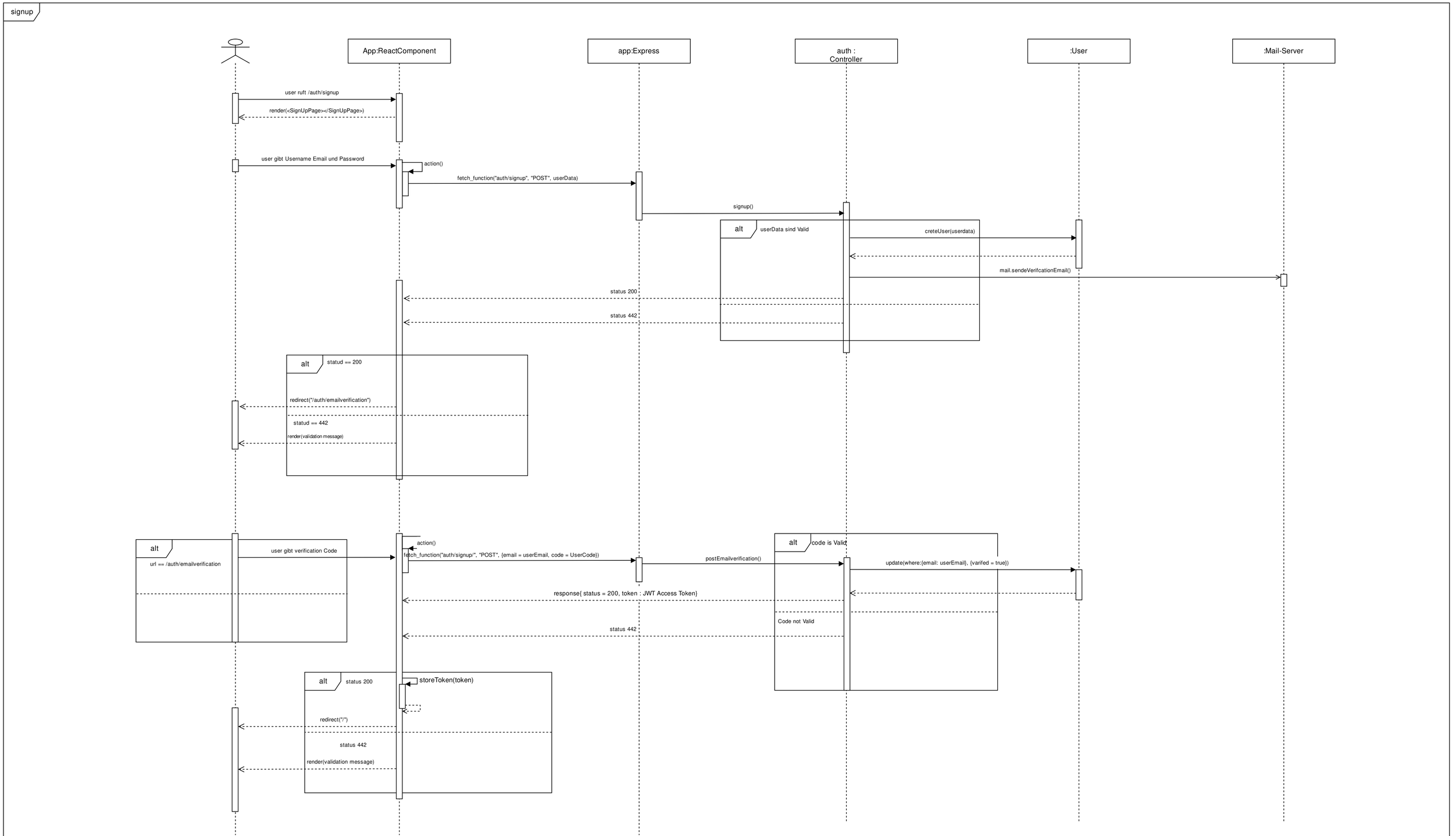
# Klassen-Diagramm-Frontend



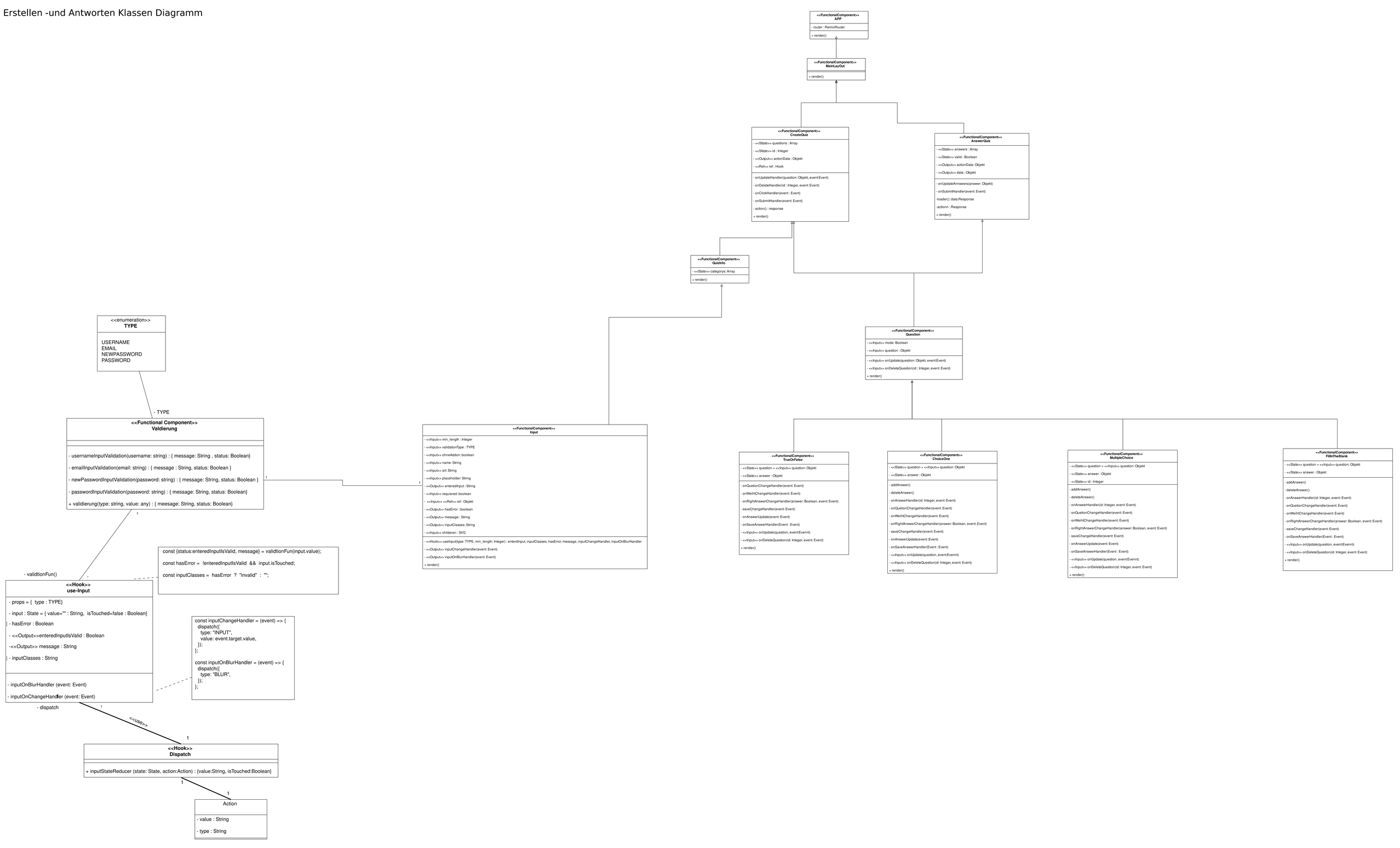
# Login


login
App:ReactComponent
app:Express
auth:Controller
:User
/auth?mode=login
render(<LogInPage></LogInPage>)
user gibt Email und Kennwort ein
action()
fetch_function("auth/login", "POST", userData)
login()
findOne(where{name=UserName, email = UserEmail})
return user
alt
user != null
response{ status = 200, token : JWT Access Token}
user == null
Status 401
alt
status == 200
storeToken(token)
redirect("/")
status == 442
render(validation message)

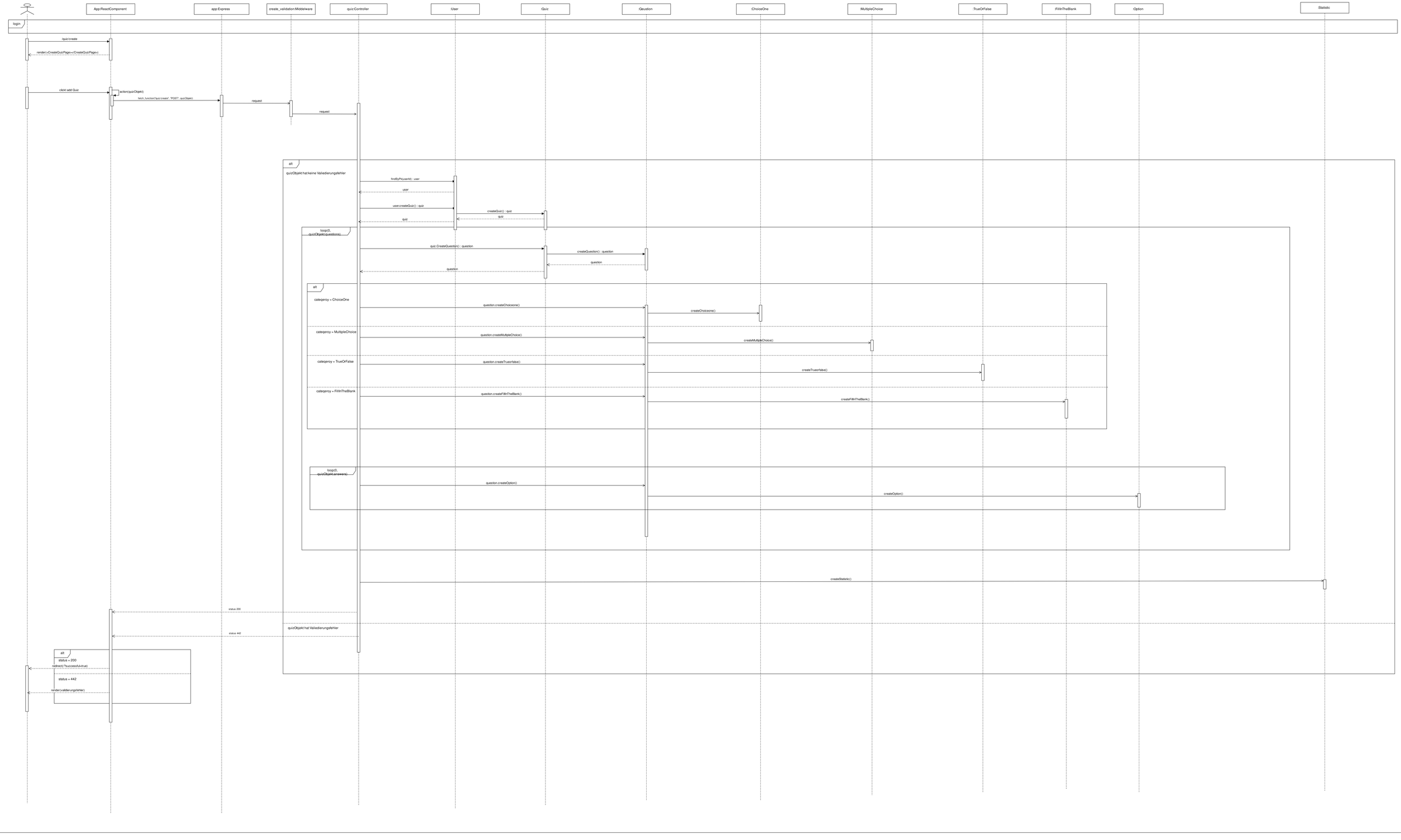
# Signup

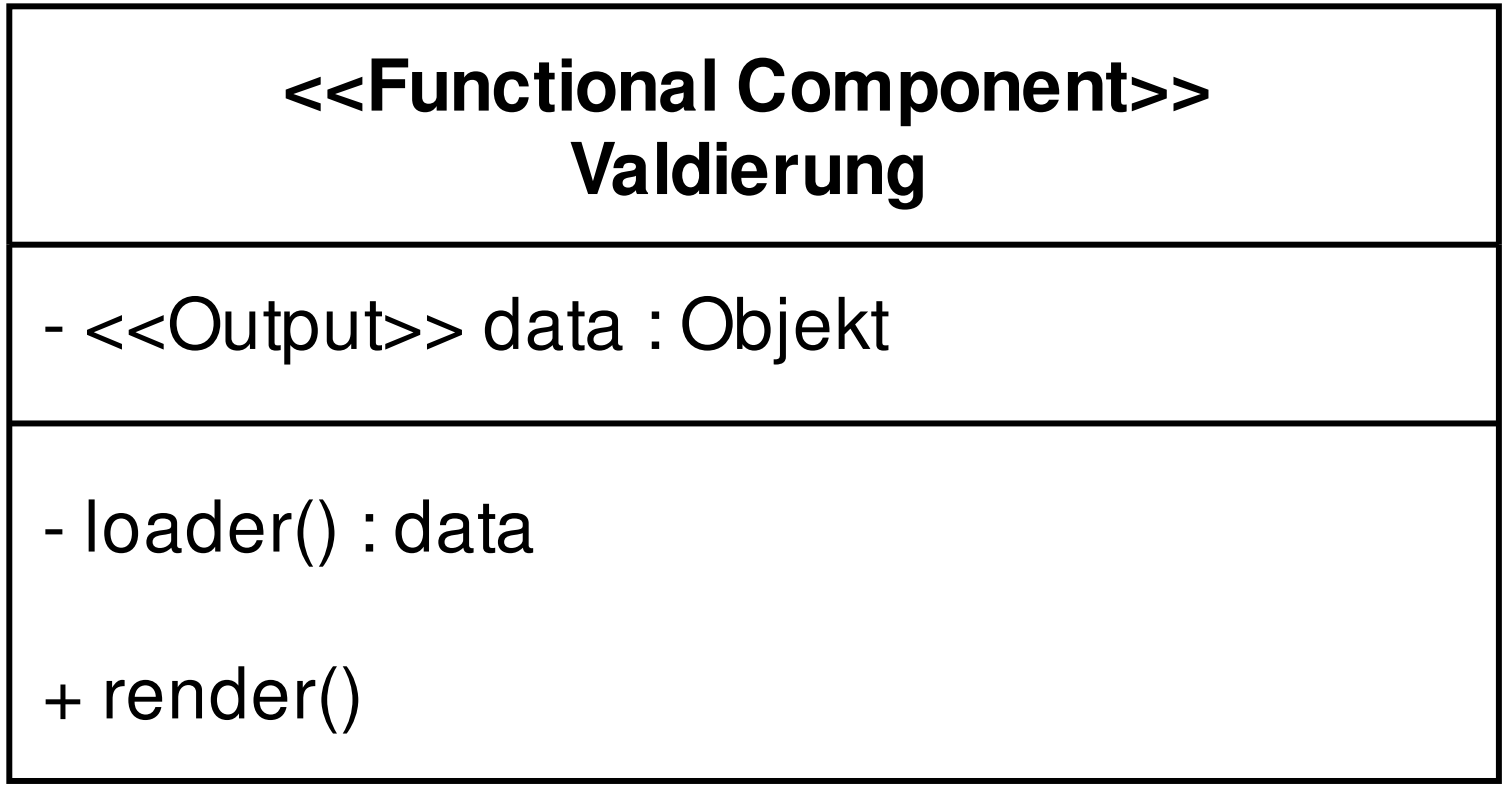

signup
App:ReactComponent
app:Express
auth :
Controller
:User
:Mail-Server
user ruft /auth/signup
render(<SignUpPage></SignUpPage>)
user gibt Username Email und Password
action()
fetch_function("auth/signup", "POST", userData)
signup()
alt
userData sind Valid
creteUser(userdata)
mail.sendeVerifcationEmail()
status 200
status 442
alt
statud == 200
redirect("/auth/emailverification")
statud == 442
render(validation message)
alt
url == /auth/emailverification
user gibt verification Code
action()
fetch_function("auth/signup/", "POST", {email = userEmail, code = UserCode})
postEmailverification()
alt
code is Valid
update(where:{email: userEmail}, {varifed = true})
response{ status = 200, token : JWT Access Token}
Code not Valid
status 442
alt
status 200
storeToken(token)
redirect("/")
status 442
render(validation message)

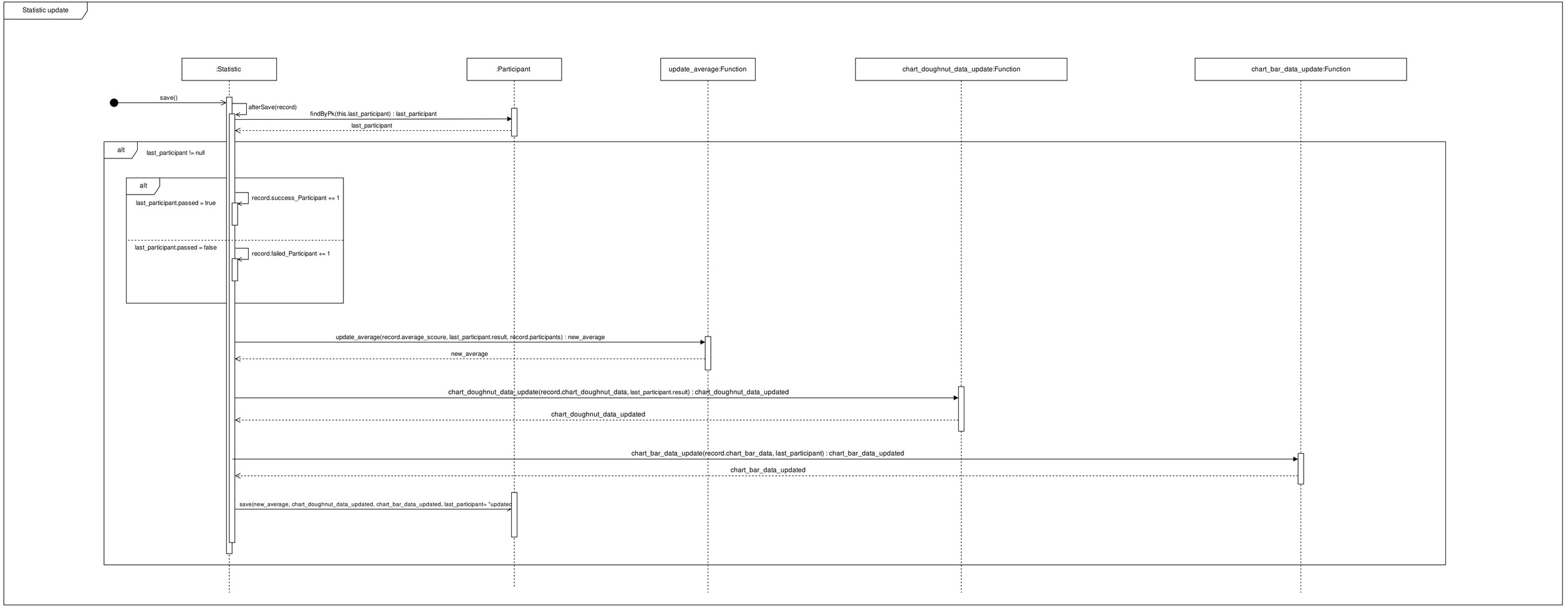
Erstellen -und Antworten Klassen Diagramm



create Quiz


:App.ReactComponent
app.Express
create_validation.Middleware
quiz.Controller
:User
:Quiz
:Qeustion
:ChoiceOne
:MultipleChoice
:TrueOrFalse
:FillInTheBlank
:Option
:Statistic
login
:quiz.create
render(<CreateQuizPage/>,<CreateQuizPage/>)
clickt add Quiz
action(quizObjekt)
fetch_function("quiz/create", "POST", quizObjekt)
request
request
alt
quizObjekt hat keine Validierungsfehler
findByPk(userId) : user
user
user.createQuiz() : quiz
createQuiz() : quiz
quiz
quiz
loop(0, quizObjekt.questions)
quiz.CreateQuestion() : question
createQuestion() : question
question
question
alt
categeroy = ChoiceOne
question.createChoiceone()
createChoiceone()
categeroy = MultipleChoice
question.createMultipleChoice()
createMultipleChoice()
categeroy = TrueOrFalse
question.createTrueorfalse()
createTrueorfalse()
categeroy = FillInTheBlank
question.createFillInTheBlank()
createFillInTheBlank()
loop(0, quizObjekt.answers)
question.createOption()
createOption()
createStatistic()
status 200
quizObjekt hat Valiedierungsfehler
status 442
alt
status = 200
redirect(/?success/full=true)
status = 442
render(validierungsfehler)


<<Functional Component>>
Valdierung
- <<Output>> data : Objekt
- loader() : data
+ render()


Statistic update
:Statistic
:Participant
update_average:Function
chart_doughnut_data_update:Function
chart_bar_data_update:Function
save()
afterSave(record)
findByPk(this.last_participant) : last_participant
last_participant
alt
last_participant != null
alt
last_participant.passed = true
record.success_Participant += 1
last_participant.passed = false
record.failed_Participant += 1
update_average(record.average_scoure, last_participant.result, record.participants) : new_average
new_average
chart_doughnut_data_update(record.chart_doughnut_data, last_participant.result) : chart_doughnut_data_updated
chart_doughnut_data_updated
chart_bar_data_update(record.chart_bar_data, last_participant) : chart_bar_data_updated
chart_bar_data_updated
save(new_average, chart_doughnut_data_updated, chart_bar_data_updated, last_participant= "updated

# Antworten Schema

```
{
    "quizId": "Quiz Id wird von Server geliefert",
    "questions": [
        {
            "questionId": "Frage Id wird von Server geliefert",
            "category": "Category wird von Server geliefert",
            "answer": "in dem Fall ist eine Text-Eingabe Frage daher ist einer String, welche  der Eingabe des Users ist. Muss mind 2 Zeichen lang
sein"
        },
        {
            "questionId": "Frage Id wird von Server geliefert",
            "category": "Category wird von Server geliefert",
            "answer": "in dem Fall Mehrfach Auswahl muss eine Array von Objekten jeder Objekt hat eine Value, welche einer der Antworten ist "
        },
        {
            "questionId": "Frage Id wird von Server geliefert",
            "category": "Category wird von Server geliefert",
            "answer": "in dem Fall Einfache Auswahl ist eine String welche einer der Möglichkeiten entspricht"
        },
        {
            "questionId": "Frage Id wird von Server geliefert",
            "category": "Category wird von Server geliefert",
            "answer": "in dem Fall Richtig oder Falsch muss einer Boolean sein"
        }
    ]
}
```

# Create Schema

```
{
    "title": "Ein Title muss mind. 6 Zeichen lang sein",
    "quizCategory": "Kategorie der Quiz",
    "beschreibung": "nicht requiered",
    "required_points": "Ein Float zwischen 0.01 und 1",
    "questions": [
        {
            "id": 1,
            "question_value": "Frage inhalt muss mind. 3 Zeichen lang sein",
            "category": "TRUEORFALSE category falls eine Richtig oder Falsch Frage ist",
            "weight": "eine Integer großer 1",
            "right_answer": "muss Boolean sein"
        },
        {
            "id": 3,
            "question_value": "Frage inhalt muss mind. 3 Zeichen lang sein",
            "category": "CHOICEONE  category falls eine Enfache Auswahl ist",
            "weight": "eine Integer großer 1",
            "right_answer": {
                "value": "Objekt mit value in dem Richtige Antwort ist, welche in answers array sein muss"
            },
            "answers": [
                {
                    "id": 1,
                    "value": "Objekt mit value in dem Möglcihkeit"
                },
```

```
            {
                "id": 2,
                "value": "Objekt mit value in dem Möglcihkeit"
            },
            {
                "id": 3,
                "value": "Objekt mit value in dem Möglcihkeit"
            }
        ]
    },
    {
        "id": 5,
        "question_value": "Frage inhalt muss mind. 3 Zeichen lang sein",
        "category": "FILLINTHEBLANK falls eine Text Eingabe ist",
        "weight": "eine Integer großer 1",
        "right_answer": "Eine String der mind. zwei Zeichen lang sein muss"
    },
    {
        "id": 7,
        "question_value": "Frage inhalt muss mind. 3 Zeichen lang sein",
        "category": "MULTIPLECHOICE falls eine Mehrfache Auswahl ist",
        "weight": "eine Integer großer 1",
        "right_answer": [
            {
                "id": 1,
                "value": "Objekt mit value in dem eine der Antworten"
            },
            {
                "id": 3,
                "value": "Objekt mit value in dem eine der Antworten"
            }
        ],
        "answers": [
            {
                "id": 1,
                "value": "Objekt mit value in dem Möglcihkeit"
            },
            {
                "id": 2,
                "value": "Objekt mit value in dem Möglcihkeit"
            },
            {
                "id": 3,
                "value": "Objekt mit value in dem Möglcihkeit"
            },
            {
                "id": 4,
                "value": "Objekt mit value in dem Möglcihkeit"
            }
        ]
    }
]
}
```