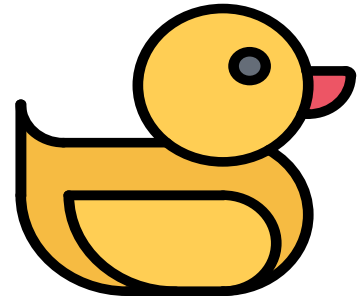


# QuackR Projekt-Report



**Das ist quackR - BBQ & PartY Web-app!**

## **Einleitung**

1. quack- who?!
2. Wie sind wir auf die Idee gekommen?
3. Aufbau der Anwendung.
4. Benutzte Techniken, Frameworks und Spezifikationen.
5. Design Entscheidungen.
6. Entwicklungsverlauf und Schwierigkeiten.
7. Erfahrungen mit dem Tool ALEX.
8. Eine kurze Tour durch das Webinterface.

## **1. Was ist quackR?**

quackR ist eine App, die Menschen verbindet, die zusammen Zeit verbringen möchten. Ob es um Grillen oder um eine Party geht ist egal. Jeder Mensch kann sich registrieren, eine Veranstaltung erstellen oder sich für andere Veranstaltungen anmelden. Das Ziel von quackR ist einfacher, kostenlos und Zielorientiert zu sein.

## 2. Wie sind wir auf die Idee gekommen?

Es ist Sommer und in dieser Jahreszeit möchten Leute mehr Zeit draußen oder mit Freunden verbringen! Noch besser ist es, wenn Essen und kaltes Bier dabei sind. Ob eine spontane Party oder Grill Veranstaltung einfacher organisieren zu können oder schon organisierte zu finden, ist quackR genau dafür gedacht.

## 3. Aufbau der Anwendung.

Die Anwendung besteht aus zwei Maven-Module. Diese wurden „*quackr-app*“ und „*quackr-server*“ benannt und enthalten dementsprechend das Front-End und das Back-End.

### 3.1. Backend

Das Backend enthält die gesamte Geschäftslogik, verbindet sich mit einem Datenbankserver und bedient HTTP-Requests. Es besteht aus drei verschiedenen Schichten: einer Persistenzschicht, einer Dienstschicht (Business-Logik) und REST-Controllern.

Die Persistenzschicht enthält die Domänenentitäten, verwaltet den Datenbankzugriff und führt einfache CRUD-Operationen durch. Die Serviceebene bildet Domänenentitäten auf Ressourcenklassen ab und verarbeitet sie. Wir haben uns aus Gründen der Einfachheit für die Trennung der Persistenz- und Dienstschicht Klassen entschieden, da die Domänenklassen nicht für die Interaktion mit der API geeignet sind. Wenn beispielsweise eine Anfrage nach allgemeinen Benutzerinformationen gestellt wird, wollen wir nicht das gehashte Benutzerpasswort oder jeden Kommentar, den der Benutzer jemals abgegeben hat, weitergeben, sondern nur eine Teilmenge der Daten. Die REST-Controller haben dann einfach die Aufgabe, die HTTP-Endpunkte bereitzustellen, die JSON sowie XML unterstützen, und mit den Diensten zu interagieren.

Ein einfaches tokenbasiertes Authentifizierungsmodell sowie die Berechtigungsverwaltung sind ebenfalls im Backend implementiert.

### 3.2. Front-End

Das Front-End ist ähnlich wie das Backend aufgebaut, indem es aus einem Dienst- und Ansichtsschicht, die mit dem Benutzer und untereinander interagieren muss.

Die Dienste sind für das Senden und Empfangen von HTTP-Requests verantwortlich. Das Verzeichnis "*model*" enthält einfache TypeScript-Klassen, die direkt auf die JSON-

Antworten aus unserem Backend verweisen. Dies erleichtert die Arbeit mit den Daten erheblich und war ein weiterer Grund, die Persistence-Domain-Entitäten in mehrere verschiedene Ressourcenklassen aufzuteilen.

Die View-Komponenten wiederum sind dafür verantwortlich, dem Endbenutzer alle verarbeiteten Informationen übersichtlich und ansprechend darzustellen sowie Eingaben entgegenzunehmen und mit dem Benutzer so zu interagieren, dass ein durchgängiger Zustand gewährleistet ist. Beispielsweise sollte der Benutzer im Idealfall das Ergebnis seiner/ihrer Aktionen sofort sehen können oder er/sie sollte sich nicht auf einer Seite befinden, auf der er/sie keine Zugriffsberechtigung hat. Die UI muss daher immer eine korrekte Darstellung des Backends widerspiegeln.

### **3.3. Authentifizierungs-/Autorisierungsmechanismus**

Der Authentifizierungsmechanismus wurde mit Zugriffstoken (JWT) und Benutzerrollen implementiert. Dies gibt einem Benutzer die Möglichkeit, sich zu authentifizieren und berechtigt zu sein, eine bestimmte Aktion auszuführen oder auf bestimmte Ressourcen zuzugreifen.

Die verfügbaren Rollen sind *"user"* und *"admin"*. Die Rolle *"user"* ermöglicht es dem Endbenutzer, sich anzumelden, seine eigenen Daten zu bearbeiten, Veranstaltungen zu erstellen, zu bearbeiten sowie Veranstaltungen, die von anderen Benutzern erstellt wurden, zu besuchen oder zu verlassen. Es erlaubt dem Benutzer jedoch nicht, andere Benutzer zu löschen, ihre Konten zu bearbeiten, ihr Passwort zu ändern, ihre Veranstaltungen zu löschen etc. Diese Funktionalität ist nur einem Benutzer mit der Rolle *"admin"* verfügbar. Das *"admin-menu"* Seite in der Web-App spiegelt diese Fähigkeiten wider. Der erste Benutzer, der sich anmeldet, erhält die Administratorrolle, er hat dann die Möglichkeit, auch andere Benutzer zu Admins zu machen.

Zugriffstoken dienen als Authentifizierungsmechanismus, sie ermöglichen es dem Backend zu wissen, wer eine bestimmte Anfrage sendet und welche Berechtigungen er hat. Die einzigen Aktionen, die ohne Angabe eines Zugriffstokens durchgeführt werden können, sind die Benutzerregistrierung und Anmeldung.

## **4. Benutzte Techniken, Frameworks und Spezifikationen.**

### **4.1 Frameworks und Bibliotheken im Backend**

Wir haben uns entschieden, unseren Backend-Code auf das sehr beliebte Spring-Framework und genauer gesagt auf die Spring-Boot-Variante zu stützen. Wir finden, dass Spring-Boot ein sehr schneller Weg ist, um einen RESTful Webservice von Grund auf

neu zu erstellen. Es bietet eine schnelle Möglichkeit, die Anwendung zu starten und zu testen, ohne sie auf einem Webserver deployen zu müssen. Spring-Boot beinhaltet auch das Hibernate ORM und ermöglicht es uns, eine Datenquelle einfach zu konfigurieren. Wir haben die bestehende CRUD-Repository-Implementierung von Spring-Data vermieden und uns stattdessen dafür entschieden, die Funktionalität selbst transparenter zu implementieren.

Für unsere Datenbank haben wir uns für PostgreSQL entschieden, da es eine gute Open-Source-Alternative zu gängigen proprietären relationalen Datenbanksoftware wie OracleDB und IBM DB2 ist. Um die Anwendung ohne laufenden PostgreSQL-Server zu starten und Integrationstests durchzuführen, verwenden wir die einfache H2-In-Memory-Datenbank.

Für unsere REST-Controller haben wir die JAX-RS-Spezifikation mit der Implementierung von Jersey sowie die Standard-Validierungstechniken in Java zur Validierung eingehender JSON/XML verwendet.

Wir haben auch die beliebte Lombok-Bibliothek genutzt, die es uns ermöglicht, auf einfache Weise Getter/Setter und Konstruktoren mit Annotationen zu erstellen. Wir haben festgestellt, dass Lombok durch die automatische Aktualisierung der Getter/Setter Methodendefinitionen wertvolle Zeit spart, wenn unser Modell geändert werden muss.

Um die Anwendung zu schützen, haben wir Apache-Shiro und die BCrypt-Bibliothek verwendet, um Benutzerpasswörter zu hashen.

## **4.2 Frameworks und Bibliotheken im Frontend**

Für unseren Frontend-Code benutzen wir Angular 7, indem wir Angular-CLI verwendet haben, um das Projekt und die verschiedenen Komponenten darin zu generieren. Für unser Styling haben wir uns für das Bootstrap 4 CSS Framework entschieden, das mit wenig Aufwand einen angenehmen und modernen Aussehen anbietet.

## **5. Design Entscheidungen im UI**

Für die Benutzeroberfläche der Web-App haben wir uns entschieden, die intuitiven Bootstrap-Controls und -Formulare sowie ein Farbschema mit drei Hauptfarben zu verwenden. Schaltflächen, die negative Aktionen ausführen, befinden sich auf der linken Seite von Formularen und Schaltflächen, die positive Aktionen ausführen, auf der rechten Seite.

Die Hauptübersicht besteht aus einer oberen Kopfzeile, dem Hauptinhalt, in der Regel entweder einem Formular oder einer Liste von Veranstaltungen, die als "Karten" formatiert sind, und einer Fußzeile. Dieses grundlegende Layout sollte den meisten Benutzern vertraut und intuitiv sein.

## **6. Entwicklungsverlauf und Schwierigkeiten.**

Unser Entwicklungsprozess bestand aus drei Teilen. Eine Planungsphase, eine Backend-Entwicklungsphase und eine Frontend-Entwicklungsphase.

In der Planungsphase mussten wir klare Anforderungen definieren. Das bedeutete, ein paar verschiedene Ideen durchzugehen, diejenige auszuwählen, die uns am besten gefällt, und sie zu konkretisieren. Wie aus dem Anwendungscode ersichtlich ist, haben es einige unserer ursprünglichen Ideen nicht in die finale Version der App geschafft. Zum Beispiel ein System zur Bewertung von Benutzern, die Möglichkeit, Veranstaltungen als privat einzustellen und die Möglichkeit für Benutzer, Kommentare zu Veranstaltungen zu hinterlassen. Diese Funktionen sind im Backend implementiert, aber nicht im Frontend. Dies ist vor allem auf Zeitdruck, aber auch auf die Komplexität dieser Funktionen zurückzuführen. Sie können jedoch zu einem späteren Zeitpunkt problemlos implementiert werden.

Nachdem wir definiert haben, was unsere Anwendung sein sollte, sind wir zur Entwicklungsphase des Prozesses übergegangen und haben mit dem Backend angefangen. Hier haben wir zunächst unsere Domain-Entitäten definiert sowie die Beziehungen, die sie untereinander haben müssen. Anschließend haben wir die HTTP-Endpunkte definiert, die gebraucht sind, und die dahinter stehende Geschäftslogik. Der Rest des Prozesses bestand aus der Implementierung und dem Testen des Codes. Einige der Schwierigkeiten, auf die wir stießen, waren die Interaktion zwischen Spring-Boot, Jersey und Apache Shiro, da diese Kombination in Online-Tutorials nicht sehr häufig vorkommt. Auch das Testen von Service- und Controllermethoden erwies sich nach der Integration von Apache Shiro als schwierig, da die Access-Token auf irgendeine Weise gemocked werden mussten. Diese Probleme wurden jedoch gelöst, nachdem man etwas mehr Zeit mit den Frameworks verbracht hatte.

Der nächste Schritt war die Erstellung eines Front-Ends, das mit der REST-API interagiert. Wir haben wieder damit begonnen, das Aussehen der Website zu definieren. Wir haben das gemacht, indem wir es auf Papier skizzierten. Danach sind wir mit der Implementierung angefangen. Dies war mit Angular-CLI problemlos möglich und es sind bei diesem Schritt keine größeren Schwierigkeiten aufgetreten.

Als VCS-Tool haben wir Git verwendet. Das hat uns geholfen, die Arbeit im Team leichter aufzuteilen und Code-Konflikte zu vermeiden.

## **7. Erfahrungen mit dem Tool ALEX.**

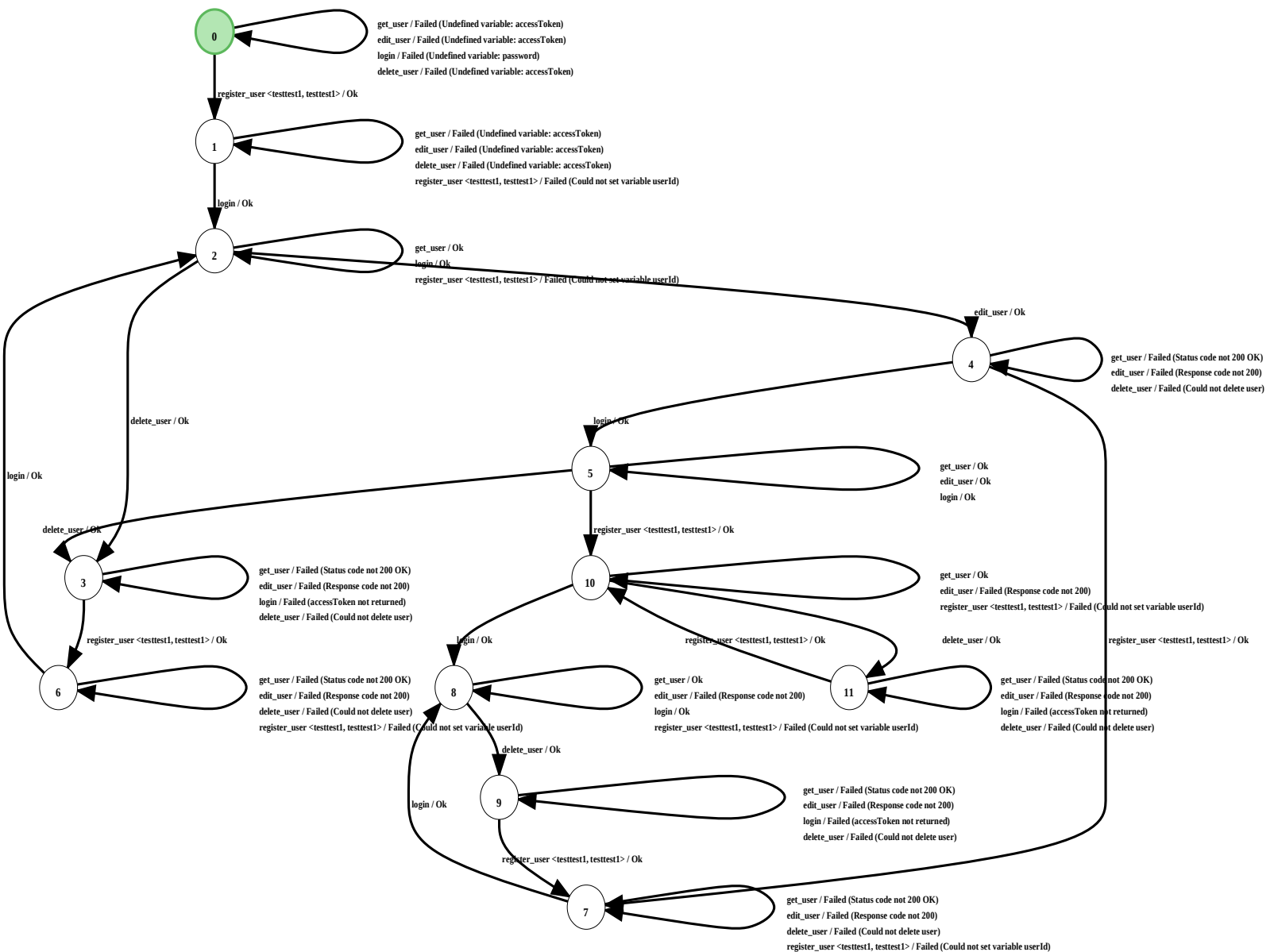
Für unsere App haben wir uns entschieden, ALEX unsere REST-Endpunkte lernen zu lassen und eine Hypothese für sie zu erstellen. Dazu haben wir zwei Mengen von Eingangssymbolen erstellt. Eine Menge für Aktionen, die mit Benutzern verknüpft sind, und eine andere Menge für Aktionen, die mit Veranstaltungen verknüpft sind.

Wir haben den TTT-Algorithmus mit dem Random Word EQ Oracle mit Min Length, Max Length und Random Words verwendet, die alle auf 100 gesetzt wurden. Der Seed wurde auf 42 und die Batch-Größe auf 20 gesetzt.

Das folgende Diagramm stellt die erste Hypothese dar, die für diese Symbolmenge erstellt wurde:

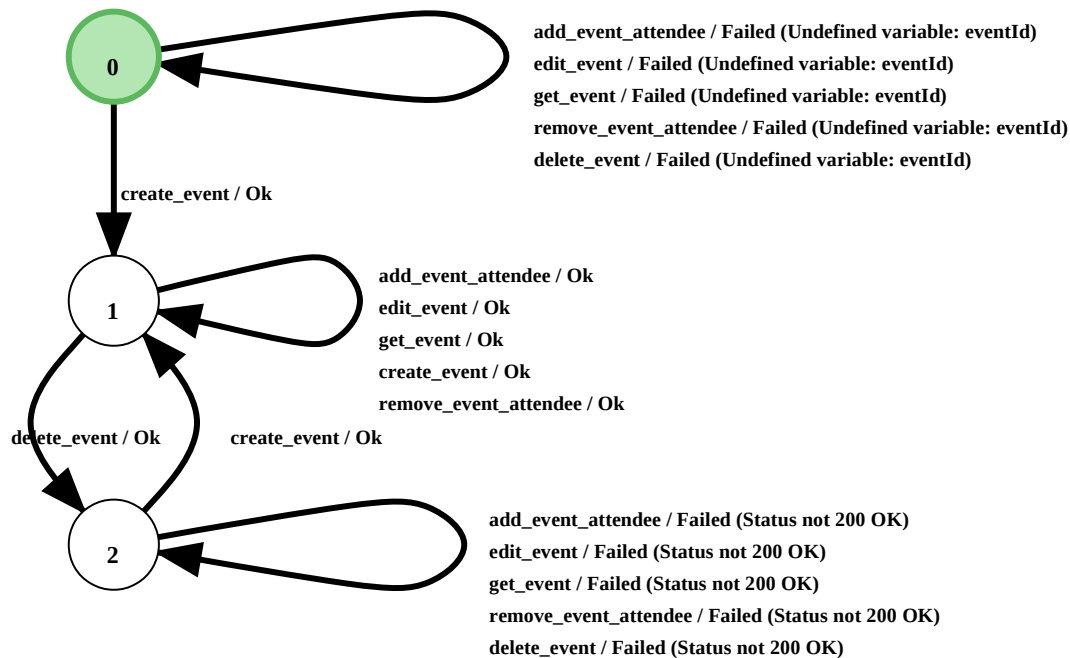
$$\Sigma_1 = \{register\_user, delete\_user, login, get\_user, edit\_user\}$$

(Wenn Sie das Diagramm nicht lesen können, verwenden Sie bitte die Zoomfunktion in Ihrem Software.)



Das folgende Diagramm stellt das Ergebnis unserer zweite Menge dar:

$\Sigma_2 = \{create\_event, edit\_event, get\_event, delete\_event, remove\_event\_attendee, add\_event\_attendee\}$




Unsere REST-API wurde mit einem `/reset`-Endpunkt ausgestattet, der den Inhalt der Datenbank löscht. Die beiden Symbole hatten auch die entsprechenden "Reset"-Symbole und wurden von ALEX erfolgreich gelernt, und wir konnten kein inkonsistentes Verhalten in der Anwendung feststellen.

## 8. Eine kurze Tour durch das Webinterface.

Beim Start der Anwendung wird der Benutzer mit dem Anmeldebildschirm begrüßt. Er/Sie kann dann auf die Anmeldeseite gehen und ein Konto erstellen.

The screenshot shows the quackR web interface. At the top, there is a teal header bar with the quackR logo on the left and "Log in" and "Sign up" buttons on the right. The main content area is white and features a "Log in to quackR" heading. Below the heading are two input fields: "Username" and "Password". A "Login" button is positioned below the password field. At the bottom of the page, there is a teal footer bar with the quackR logo and the text "About quackR".





[Log in](#)
[Sign up](#)

### What is quackR?

quackR was created to help with the organisation of all types of activities. Whether it's organizing small barbeques or massive parties, there's a place for everyone.

Your ideas and suggestions help us to constantly improve quackR's features. Let us know how we can improve your experience at [quackr.gmbh@gmail.com](mailto:quackr.gmbh@gmail.com)

### Sign up for quackR

Username


Password

Confirm password

[Sign up](#)

[About quackR](#)

Danach wird der Benutzer auf die Homepage gebracht, wo er/sie alle Veranstaltungen sieht, die andere Benutzer erstellt haben.



[My events](#)
[Events I'm going to](#)

max

Post your own event

#### Watching football

Germany - Someone??

Who's going:

Organizer: cveti

Location: The library??

Date: 2019-08-14

Attendance: 0 / 50

[Attend](#)

#### Summer party

It's a party! Come in a good mood!

Who's going:

cveti  
max

Organizer: mitko

Location: OH14

Date: 2019-07-17

Attendance: 2 / 30

[Unattend](#)

#### BBQ

Just a BBQ, bring beer, meat and/or veggies

Organizer: cveti

[About quackR](#)

Der Benutzer kann nun durch Anklicken der Schaltfläche "Post your own event" eine Veranstaltung erstellen, die von ihm/ihr unter "My events" erstellten Veranstaltungen und die von ihm/ihr besuchten Veranstaltungen unter "Events I'm going to" einsehen. Nach dem Anklicken des Buttons in der oberen rechten Ecke wird ein Dropdown-Menü angezeigt, in dem der Benutzer sich abmelden, sein/ihr Konto bearbeiten oder, wenn er/sie ein Admin ist, auf das Administrationsmenü zugreifen kann.



## All registered users:

**max**

Role: ADMIN

Delete

Revoke permissions

**cveti**

Role: USER

Delete

Make admin

**mitko**

Role: USER

Delete

Make admin

## All events:

**Watching football**

Germany - Someone??

Organizer: cveti

Who's going:

Location: The library??

mitko

Date: 2019-08-14

Attendance: 1 / 50

Delete

**Summer party**

It's a party! Come in a good mood!

Organizer: mitko

Who's going:

Location: OH14

max

Date: 2019-07-17

cveti

Attendance: 2 / 30

Delete

**BBQ**Just a BBQ, bring beer, meat and/or  
veggies.

Organizer: cveti

