

**Projekt-Report zur Studienleistung für die Vorlesung
Webtechnologien 2**

QuackR

Alexander Hahn (192947) Lebatt Mdeih (190572)
Yifeng He (200032) Xiaoshi Wang (197843)

2. Juli 2019

1 Projektidee

1.1 Anwendungsgebiet

Wir wollen eine Web App wie Jodel bauen. Man kann eine Quacks mit anderen Nutzern teilen. Dann haben wir diskutiert, welche Funktionalitäten soll es haben.

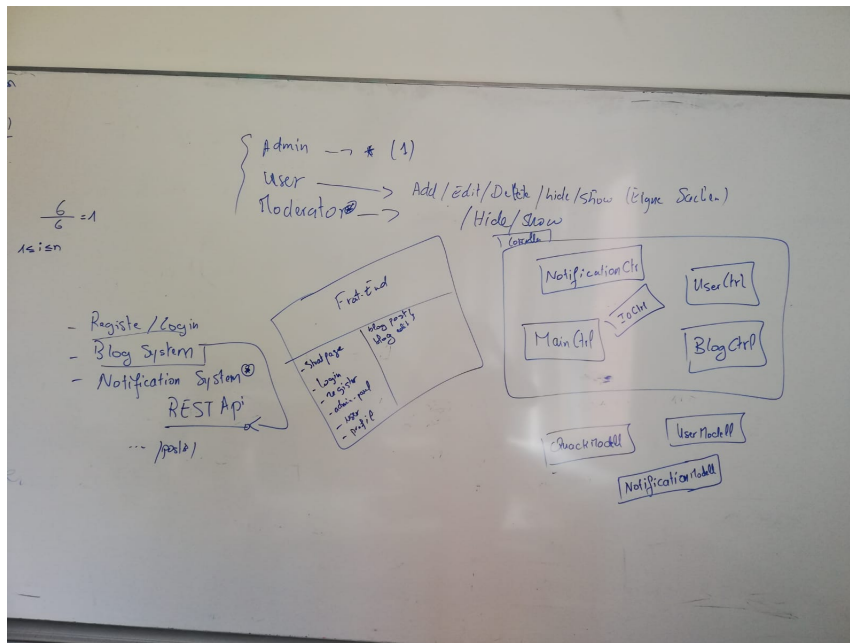


Abbildung 1.1: Ideen am Anfang

1.2 Anforderungen

Dann berlegen wir, was man mit Quack machen kann. Man muss zuerst Nutzer erstellen können, und mit diesem Konto einloggen. Außerdem kann man neue Quacks mit Titel und Text erstellen. Die werden durch viele Angular Komponenten implementiert und auch mit Bootstrap verschnert. Um es zu bauen brauchen wir auch eine Datenbank. Und alle APIs sollen wie Aufgabenstellung RESTful sein. Shiro muss auch für Authentifizierungen implementiert werden. Am Ende muss es auch mit ALEX getestet werden. Dann haben wir über unsere Datenbank diskutiert, welche Tabellen und welche Spalten sollen enthalten werden. Am Ende entschieden wir uns, zwei Tabellen zu machen. Eine ist Users, die

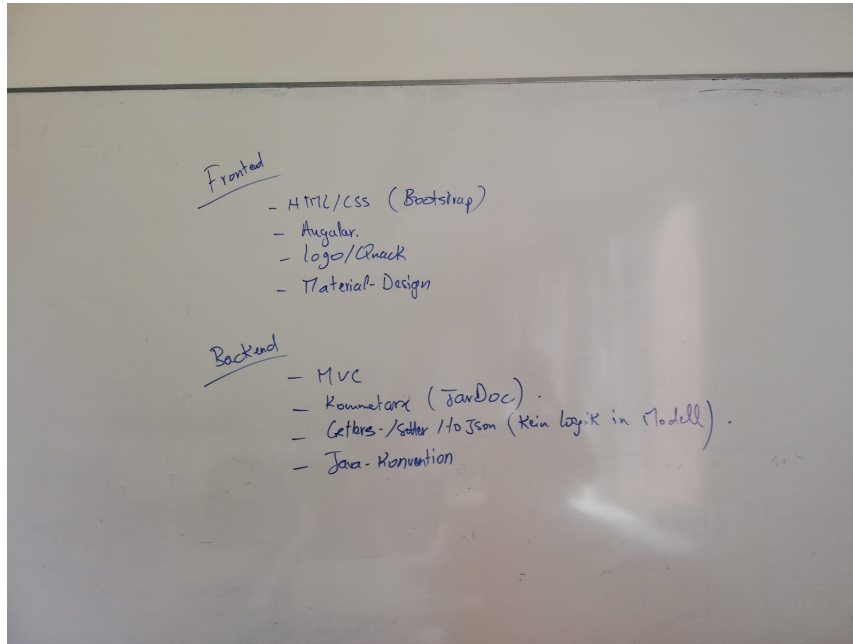


Abbildung 1.2: Ideen der Quacks

andere ist Quacks. Wir haben auch gedacht, Notification Systeme zu machen. Aber wegen Zeit Beschrnkungen haben wir am Ende die weggelassen.

2 Entwicklungsprozess

Wir haben zuerst eine Repository auf Bitbucket erstellt und arbeiten wir mit Git. Dann haben wir das Frontend Mockups erstellt, damit man schon sehen kann, wie das am Ende aussieht. Wir haben hier Maven, Angular und Bootstrap genutzt.

Nachdem fangen wir mit Backend an. Zuerst haben wir das persistence Layer mit JPA criteria geschrieben. Und zwei Entities werden erstellt, womit man mit Datenbank kommunizieren kann. Dann haben wir einen IO Controller, der alle IO Aktionen auf persistence layer arbeitet. Fr Users und Quacks haben wir auch zwei Controllers , die mit RESTful APIs sind. Dann kann man das Frontend mit Backend verbinden. Dafr haben wir eine QuackRServices Klass hinzugefgt. Jetzt kann man schon die CRUDs fr beide Users und Quacks bedienen.

Mit fertigen Methoden knnen wir jetzt Shiro und ALEX implementieren. Wir haben bei Login, Senden und viele Sessions Authentifizierung hinzugefgt, damit es richtig laufen knnte. Dann nutzen wir ALEX, um zu schauen, ob unseres Backend die Requests richtig bearbeitet und richtige Responses zurckgibt.

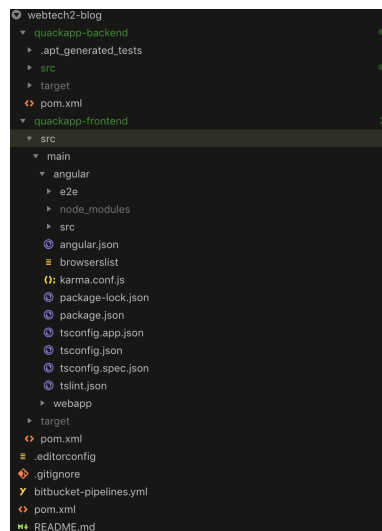


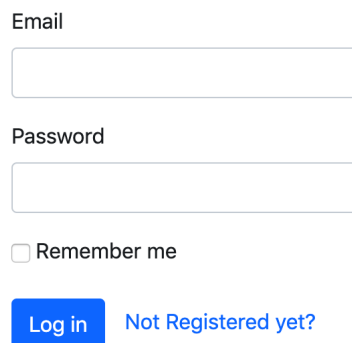
Abbildung 2.1: Quack Directoty

3 Ergebnis

3.1 Frontend

3.1.1 Log-in und Sign-up Component

Um unsere Quack App zu nutzen, braucht man zuerst einen Account. Wir haben oben rechts von der Seite eine Button hinzugefgt und die routing davon ist mit Log-in Angular Component verbunden.



The image shows a login form with the following elements:

- An input field labeled "Email".
- An input field labeled "Password".
- A checkbox labeled "Remember me".
- A blue button labeled "Log in".
- A blue link labeled "Not Registered yet?".

Abbildung 3.1: Log-in Komponente

Auch wenn der Nutzer noch kein Konto hat, kann man durch Ausfüllen des Formulars erstellen. Man braucht die Email Adresse, Passwort, Nickname und Geschlecht um einen Konto zu erstellen.

Die beide Seiten hier sind auch mit Hilfe von Bootstrap ein gewisses Ma responsive. Aber wir haben momentan nicht auf Gerte getestet.

[Already registered? Log in](#)

Email address

Password

Re-enter Password

Nickname

Gender ☒ Male ☐ Female

☐ I agree to the Terms and Conditions

[Sign up](#)

Abbildung 3.2: Sign-up Komponente

3.1.2 Post/Edit, Start, Profile Seite

Alle drei Seiten beruhen sich auf folgende Techniken: **Angular**, **CSS**, **HTML**, **JavaScript**, **Bootstrap**.

Home Profile Dashboard

Quacksapp Hier ein Banner

Quacks of the day

Quack Quack Quack

Text:

☐ only me

Quacken

Made with ❤️

Abbildung 3.3: Post/Edit Seite

Der Schwerpunkt liegt an der Einsetzung der Bausteine der Webseite. Beispielsweise sollte Beitragfenster in der Mitte der Webseite liegen. Aber es ist nicht anpassend wenn das Beitragfenster lnger als obere Banner ist. Damit muss man es immer aufpassen.

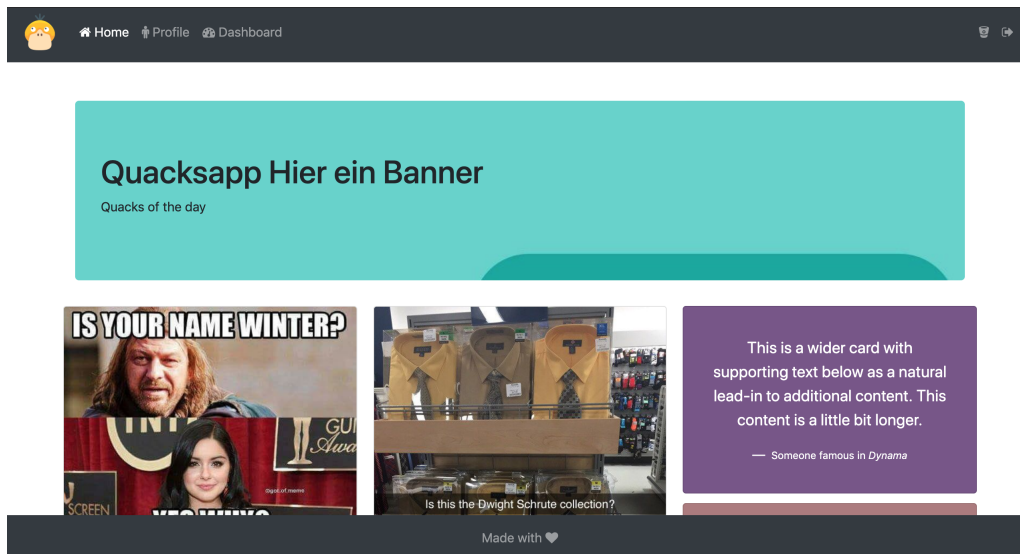


Abbildung 3.4: Start Seite

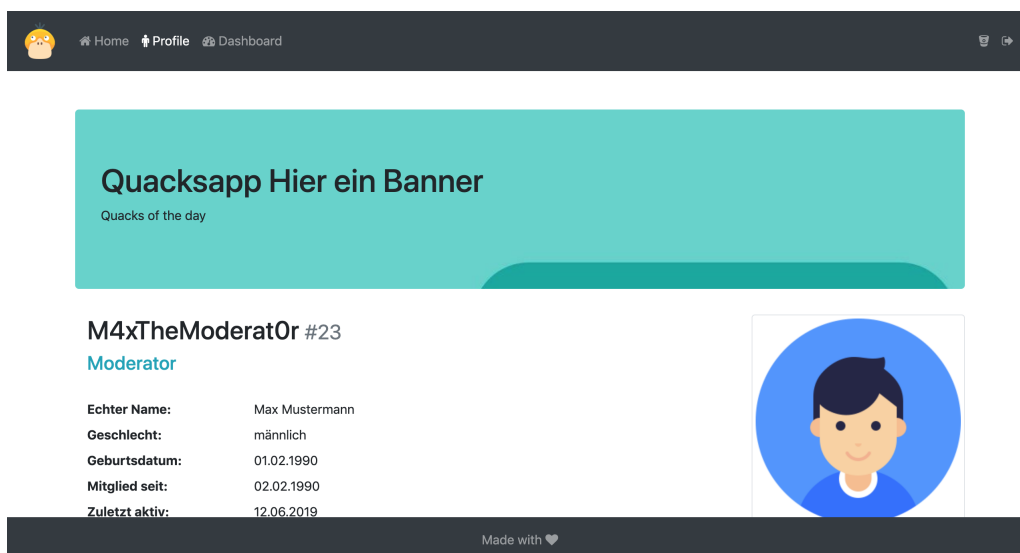


Abbildung 3.5: Profile Seite

Eine gute Wahl ist dass Entwicklertools bei Chrome zur Verfgung steht. Mit diesem Tool wird die Gre des Objektes ganz genau bezeichnet, indem Entwickler die Bausteine der Webseite an passendem Bereich einsetzen kann.

3.2 Backend

3.2.1 REST-API

Wie die Aufgabestellung gesagt, müssen das Frontend mit RESTful APIs die Request zu Backend schicken. Bei unsere Quack App haben wir zwei große Entities: Users und Quacks. Dann haben wir für die beide viele RESTful APIs durch Quack Controller und User Controller erstellt.

- Bestimmte Quack lesen: `/api/quacks/{id}`
Dann ruft unser QuackController den IOController, und gibt am Ende die gesuchte Quack zurück.
- Alle bestehenden Quacks lesen: `/api/quacks`
Dann ruft unser QuackController den IOController, und gibt am Ende alle Quacks zurück, die schon in DB liegen
- Bestimmte User lesen: `/api/users/{id}`
Dann ruft unser UserController den IOController, und gibt am Ende die gesuchte User zurück.
- Alle bestehenden Users lesen: `/api/users`
Dann ruft unser UserController den IOController, und gibt am Ende alle Users zurück, die schon in DB liegen.

Durch verschieden Typen von HTTP-Requests, werden verschiedene Aktionen ausgeführt. Die werden durch Annotations vor jeder Methode bestimmt. Dann ist unsere APIs mit JAX-RS erstellt.

- GET: Gewünschte JSON Objekte ausgeben.
- POST: Neu erstellen.
- PUT: Bestehende aktualisieren.
- DELETE: Löschen.

Bei der User Controller und Quack Controller haben wir auch zusätzlich ein paar reguläre Ausdrücke geschrieben. Sie überprüfen die Korrektheit und Gültigkeit von der Eingaben bei POST- oder PUT-Requests.

3.2.2 JPA-Criteria

Kriterien ist eine vordefinierte API, die verwendet wird, um Abfragen für Entitäten definieren. Es ist eine alternative Möglichkeit der Definition einer JPQL Abfrage. Diese Abfragen sind typischer, tragbar und einfach zu modifizieren durch wechselnden, die

Syntax. hnlich zu JPQL, es folgt eine abstrakte Schema (leicht zu bearbeiten Schema) und eingebettete Objekte. Die Metadaten-API wird mit Kriterien API vermischt zu persistente modell Entitt fr Kriterien Abfragen.

Der groe Vorteil ist von Kriterien API ist dass Fehler knnen frher whrend der Kompilierung festgestellt werden. String-basierte JPQL Abfragen und JPA Kriterien basierte Abfragen sind gleich in Leistung und Effizienz.

Die Kriterien und die JPQL sind eng miteinander verbunden und drfen zu entwerfen mit hnlichen Betreiber in ihre Abfragen. Daraus folgt, **javax.persistence.criteria** -Paket, um eine Abfrage zu entwerfen. Die Abfrage Struktur Mittel die Syntax Kriterien Abfrage.

Die Abfrage demonstriert die grundlegenden Schritte, um eine Kriterien erstellen.

- **EntityManager** -Instanz wird verwendet, um ein Objekt zu erstellen Criteria-Builder.
- **CriteriaQuery** -Instanz wird verwendet, um eine Abfrage-Objekt erstellen. Attribute dieses Query-Objekt wird mit den Details der Abfrage gendert werden.
- **CriteriaQuery.form** -Methode aufgerufen wird, um die Abfrage root.
- **CriteriaQuery.select** wird aufgerufen, um die Ergebnisliste Typ festzulegen.
- **TypedQuery<T>** -Instanz wird verwendet, um eine Abfrage fr die Ausfhrung vorzubereiten und die den Typ des Abfrageergebnisses.
- **getResultList** Methode auf dem TypedQuery<T> Aufgabe, eine Abfrage auszufhren. Diese Abfrage gibt eine Auflistung von Einheiten, wird das Ergebnis in einer Liste gespeichert.

Bei unsere Projekt hat IO-Controller wesentlich mit JPA-Criteria zu tun. Andere Controllers verwenden Methoden von IO-Controller um die Daten mit persistence-layer austauschen zu knnen. Ein Beispiel stellt wie folgendes dar.

```

public void updateUser(long id, User user) {
    CriteriaBuilder cb = this.entityManager.getCriteriaBuilder();
    CriteriaUpdate<User> update = cb.createCriteriaUpdate(User.class);

    Root<User> e = update.from(User.class);

    update.set(User_.username, user.getUsername());
    update.set(User_.email, user.getEmail());
    update.set(User_.passwordHash, user.getPasswordHash());
    update.set(User_.birthday, user.getBirthday());
    update.set(User_.signUpTimestamp, user.getSignUpTimestamp());
    update.set(User_.lastActiveTimestamp, user.getLastActiveTimestamp());
    update.set(User_.admin, user.isAdmin());
    update.set(User_.moderator, user.isModerator());
    update.where(cb.equal(e.get(User_.id), id));

    this.entityManager.createQuery(update).executeUpdate();
}

```

Abbildung 3.6: Codeabschnitt

3.3 ALEX

Mit Automata Learning Experience (ALEX) bieten wir ein Tool, mit dem Sie auf der Basis von LearnLib auf Automatisierungsmodelle von Webanwendungen und JSON-basierten Webdiensten schließen können.

Wir haben für alle Fälle die Tests geschrieben, und gucken, ob wir die richtige Antworten bekommen können. Dann erstellt ALEX für uns eine Automat. Mit den REST APIs von unseren Controllern kann man beispielsweise Anfragen durchlaufen und testen.

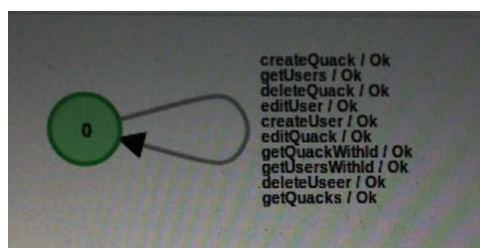


Abbildung 3.7: ALEX

3.4 Apache Shiro

Die Grundsteine für Security-Maßnahmen gilt es möglichst früh in einem Projekt zu legen. Das gilt besonders für die Verwaltung von Nutzern und Rollen. Das Apache-Projekt Shiro hilft dabei Webanwendungen abzusichern.

Shiro bietet zwei Konzepte für die Autorisierung an: **Rollen** und **Permissions**. Rollenbasierte Berechtigungsprüfungen eignen sich für einfache, berschaubare Szenarien. Permissions hingegen bieten die Möglichkeit Berechtigungen flexibel und beliebig feingranular zu definieren und zu vergeben. Dabei gilt alles als verboten, was nicht explizit erlaubt ist.

Einige wichtige Bausteine wie folgende darstellen:

1. **shiro.ini** : Dies ist eine Datei um **SecurityManager** zu konfigurieren.
2. **web.xml** : Standard-Webanwendungen initialisieren Shiro, indem sie die folgenden XML-Chunks zu web.xml hinzufügen
3. **Realm**: Es spielt eine Rolle als 'Bridge' zwischen shiro und Daten der Web-App. Wenn es an der Zeit ist, mit sicherheitsrelevanten Daten wie Benutzerkonten zu interagieren, um eine Authentifizierung (Anmeldung) und Autorisierung (Zugriffskontrolle) durchzuführen, sucht Shiro viele dieser Dinge in einem oder mehreren für eine Anwendung konfigurierten Bereichen.