Dokumentation „Die Mannschaft EM 2024“

Im Rahmen eines Hochschulprojektes, sollte eine sinnvolle Website erstellt werden. In diesem Rahmen haben wir, Louis, Michel und ich, Lucas uns dazu entschieden, eine Fan-Website für die deutsche Mannschaft bei der EM 2024 im eigenen Land zu kreieren. Das Ziel sollte es sein, neben den für die Prüfungsleistung erwarteten Features, eine Seite zu erschaffen, die jeder von uns privat nutzen würde.
Somit haben wir uns zu Beginn auf folgende Ziele geeinigt:

- Multi-View-Website mit unterschiedlichen Ansichten für diverse Funktionalitäten
- Nutzung und Webservices im Rahmen eines Newsletters und einer Verknüpfung zu Navigationsinformationen für die Stadien der EM
- Ein mit einem Back-End verknüpftes Forum, dass von angemeldeten Usern genutzt werden kann, um somit allen Nutzern die Möglichkeit zu geben, sich über aktuelle Themen auszutauschen. Das beinhaltet einen Foto-upload.
- Einen Liveergebnisfeed, der alle Livespiele anzeigt und vergangenen Spiele auflistet.
- Ein responsives und ansprechendes Design für das Front-End
- Design auf Figma und nachfolgende Realisierung

Um diese Ziele zu erreichen, haben wir uns teils täglich und wöchentlich über die anstehenden Themen ausgetauscht und zusammen die Website gebaut. Dabei übernahm Michel den Backend-Teil, während Louis und ich uns mit dem Frontend beschäftigten.

Damit diese beiden Codeteile zusammen funktionieren und die genutzte Datenbank, sowie die Api´s eingeführt werden konnten, haben wir den das Backend folgendermaßen an unser css, html und js Frontend anlehnend aufgebaut:

Spring boot 3.2.5 – Java 17 - Maven 4.0.0

Maven Dependencies:

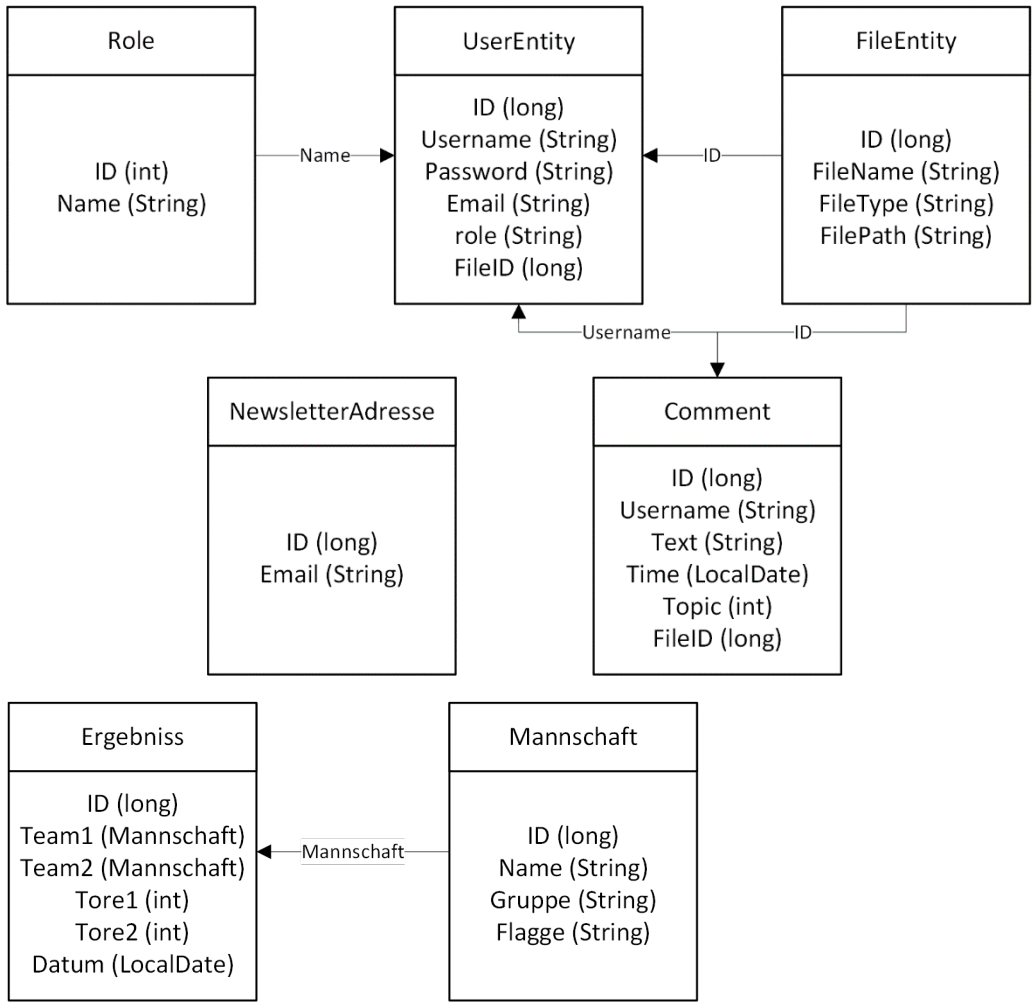
- spring-boot-starter-parent (version 3.2.5)
- spring-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-test
- spring-boot-starter-security
- spring-boot-starter-thymeleaf
- spring-boot-starter-data-rest
- spring-boot-starter-json
- lombok
- mysql-connector-j

Damit das Backend läuft, müssen folgende System installiert werden:
MySQL Server/ MySQL Workbench/ IDE/ Browser
Eine Installationsanleitung befindet sich auf Git.

(Unterstrichende Entitys befinden sich in der Datenbank, haben aber im Produkt keine Bedeutung)

Datenbank Entitys (Tabellen):

UserEntity
FileEntity
Comment
Role
NewsletterAdresse
Mannschaft
Ergebniss



Alle 7 Entitys besitzen eine entsprechende Repository Klasse mit den die Datenbank Zugriffe durchgeführt werden.
Das heißt Create/Read/Update/Delete, also „finde alle Userdaten vom User mit Namen xyz“ oder „Finde alle Kommentare aus Topic 4“ etc.

Services (Logiken die Ausgeführt werden können)

UserService:

- Existiert Username?
- User anhand von Namen laden

- UserInfo für Profile-Seite anhand von Namen laden
- Profilbild ändern bei User mit Namen xyz

FileService:
- File im System und Referenz in Datenbank speichern

CommentService:
- Kommentar mit Bild speichern
- Kommentar ohne Bild speichern
- Alle Kommentare aus einem Topic laden
- Alle Kommentare aus einem Topic und zugeordneten Bilder und PFPs laden
- Für alle Kommentare mit Username xyz den Username ändern

NewsletterService:
- Adresse dem Newsletter hinzufügen
- Adresse dem Newsletter entfernen

CustomUserDetailsService:
- Spring-User-Session erstellen durch UserEntity
- Rollen Namen zu Rechten matchen

ErgebnissService:
- *Ergebniss speichern*

MannschaftService:
- *Mannschaft speichern*

LiveScoreService:
- *Live/Past/Upcoming-Scores laden*

Controllers (Verarbeiten Frontend Requests POST/GET/PUT):
AuthController (/api/auth):
- POST (/register) – Registrierung eines Users
- *GET (/status) – Sollte Status über Authentication im Frontend wiedergeben*

FileController (/api/files/):
- GET (/{fileId}) – File anhand von FileID laden

CommentController (/api/comments):
- POST – Kommentar posten
- GET (/list) – Kommentarliste für Topic erstellen

NewsletterController (/api/newsletter):
- POST – Adresse dem Newsletter hinzufügen
- DELETE – Adresse dem Newsletter entfernen

UserProfileController (api/profile):
- GET – UserProfile-Infos laden
- PUT – UserInfos updaten
- POST (/picture) – User-PFP updaten

ResouceController (stellt die htmls bereit):

- GET (/login) – login.html
- GET (/kader) – kader.html
- GET (/stadien) – stadien.html
- GET (/forum) – forum.html
- GET (/ergebnisse) – ergebnisse.html
- GET (/profile) – profile.html

LiveScoreController (/api):

- *GET (/live-scores) – Live-Scores*
- *GET (/past-scores) – Vergangene Scores*
- *GET (/upcoming-scores) – Zukünftige Matches*

DTOs (Data Transfer Objects):

Werden genutzt, um jsons (mehrere Daten als eine Einheit) mit dem Frontend auszutauschen.

CommentWithFileDTO (Kommentar für die Forum Anzeige):

- CommentID
- Username
- Text
- Zeitpunkt
- Topic
- FileName (Komment Attachment)
- FileNameUserPFP

ProfileDTO (Daten für UserProfil):

- Username
- Email
- Password
- ProfilePictureURL

RegisterDTO (Daten zum Erstellen eines Users):

- Username
- Email
- Password

Security Config

- BCrypt Passwordencoder
- filterChain (welche Requests dürfen unter welchem Level von Authentifizierung gemacht werden)
- AccesLevels momentan:
    - Profile und Kommentare posten nur für eingeloggte Nutzer
    - Alles andere öffentlich zugänglich

Known Issues:

- Im Forum ist der Posten Button verfügbar, selbst wenn der User nicht eingeloggt ist. Der Versuch, dennoch zu Posten, führt dazu, dass die gesamte Html Datei als Fehler ausgegeben wird, anstelle, dass zum Login umgeleitet wird

- Profilbild Dimensionen sind nicht festgesetzt, sondern passen sich dem ausgewählten Bild an.

- Favicon wird nicht geladen, obwohl es an der richtigen Stelle hinterlegt ist.

- Vertikale Bilder im Forum hochzuladen, stellt eine Herausforderung dar, weil die Vorschau vom vertikalen Bild immer erst beim 2. Versuch ordnungsgemäß angezeigt wird.

Ausstehende Features:

- Erstellung eines Buttons zum Öffnen des Profils, da dieser gerade nur über URL-Eingabe localhost:8080/profile erreichbar ist .
- Login/Logout Button je nach Authentifizierungsstatus
- Benutzerprofilbild wird außerhalb vom Forum noch durch einen Standardplatzhalter dargestellt, anstelle das Profilbild des Benutzers zu fetchen.
- LiveScore API
- Ergebnisse aus Datenbank laden, anstelle sie in den html-code zu schreiben.

Frontend

Unser Frontend ist mit html, css und Js erstellt. Wir nutzen css, um den html Code zu stylen und den JS-Code, um zbs. die E-Mail-Api anzusprechen. Hier wird ein externer Anbieter genutzt, der die User-Emailadresse erhält und eine von uns erstellte Mail versendet.
Das Frontend ist auf mehreren Seiten responsiv und kann auch auf kleineren Monitoren problemlos genutzt werden. Im Allgemeinen haben wir versucht, die gesamte Website responsiv und möglichst benutzerfreundlich zu gestalten. Durch Pop-Ups, wie bei den Nutzungserklärungen, oder beim Posting im Forum, bekommt der Nutzer eine visuell ansprechende Möglichkeit, sich auf der Seite umzusehen.
Aktuell sind einige Texte und die Ergebnisse im Frontend eingebettet. Genauer gesagt im html Code mit <p> Absätzen. Im Allgemeinen ist das Fronend auf jeder Seite gleich aufgebaut. Der Header spielt eine wichtige Rolle bei der Navigation und ist daher auch auf jeder Seite gleich implementiert.

Learnings der Gruppe

Im Allgemeinen haben wir erkannt, dass wir besonders in der frühen Phase mehr zusammen hätten arbeiten müssen. Die Arbeitsteilung war grundlegend gut, jedoch hätte man sie engmaschiger beaufsichtigen müssen. Das späte Zusammenfügen der Codeteile hat zu oft zu Problemen geführt. Gleichzeitig haben wir alle eine Menge an Erfahrungen sammeln können.