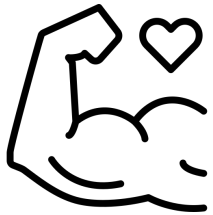
FH MÜNSTER
University of Applied Sciences

MSB FB Wirtschaft
Münster School of Business

Gym+ Hybride App



Web & Mobile Engineering

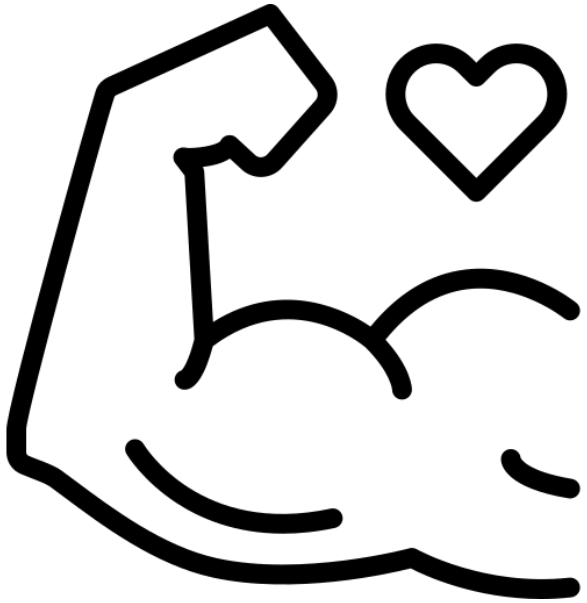
Victor Corbet (MatrNr.: 1066911)

Pascal Thesing (MatrNr.: 1231337)

Henrik Bruns (MatrNr.: 1231324)

Inhalt

- Einführung
  - Motivation
  - Rollen
  - Merkmale der App
  - Vorgehen
  - Technische Besonderheiten
- Demo
  - Walk-Through
  - Mobile + Browser
- Rückblick
  - Design Entscheidungen
  - Kritische Würdigung



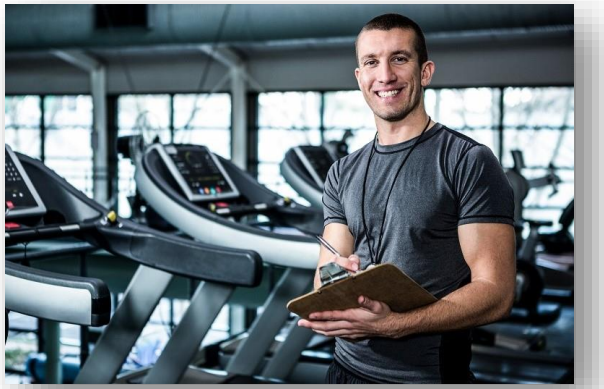
Motivation

- **Probleme**
  - Kein individueller Trainingsplan, sondern „einfach drauf los trainieren“
  - Welche Übungen gibt es?
  - Welches Training ist für mich optimal?
  - Wie benutze ich die Geräte?
  - Welche Kurse gibt es?
  - Wie kontaktiere ich den Kursleiter?

- **Lösung: Gym-App**
  - Professionelle, von Trainern erstellte, individuelle Trainingspläne, immer griffbereit
  - Alle Übungen mit Beispielfoto
  - Einfaches Kontaktieren eines Trainers
  - Socializing durch In-App-Chat
  - Einfaches Kursmanagement

Rollen

- **Trainer**
  - Sehr kompetent, sozial und oft relativ jung
  - Erstellt individuelle Trainingspläne
  - Hält Kurse
  - Antwortet auf Fragen von Trainierenden



- **Mitglied**
  - Vorwissen variiert stark, alter gemischt
  - Hat Motivation, Wissen und Ziele
  - Nimmt an Kursen teil
  - Stellt Fragen an Trainer
  - Führt Trainingspläne durch, um Ziele zu erreichen

Merkmale der App

Verfügbar für IOS, Android und Browser

Accountbasierte Multi User & Multi Rollen Anwendung

Offlinefähig

Realtime-Messaging

Vorgehen

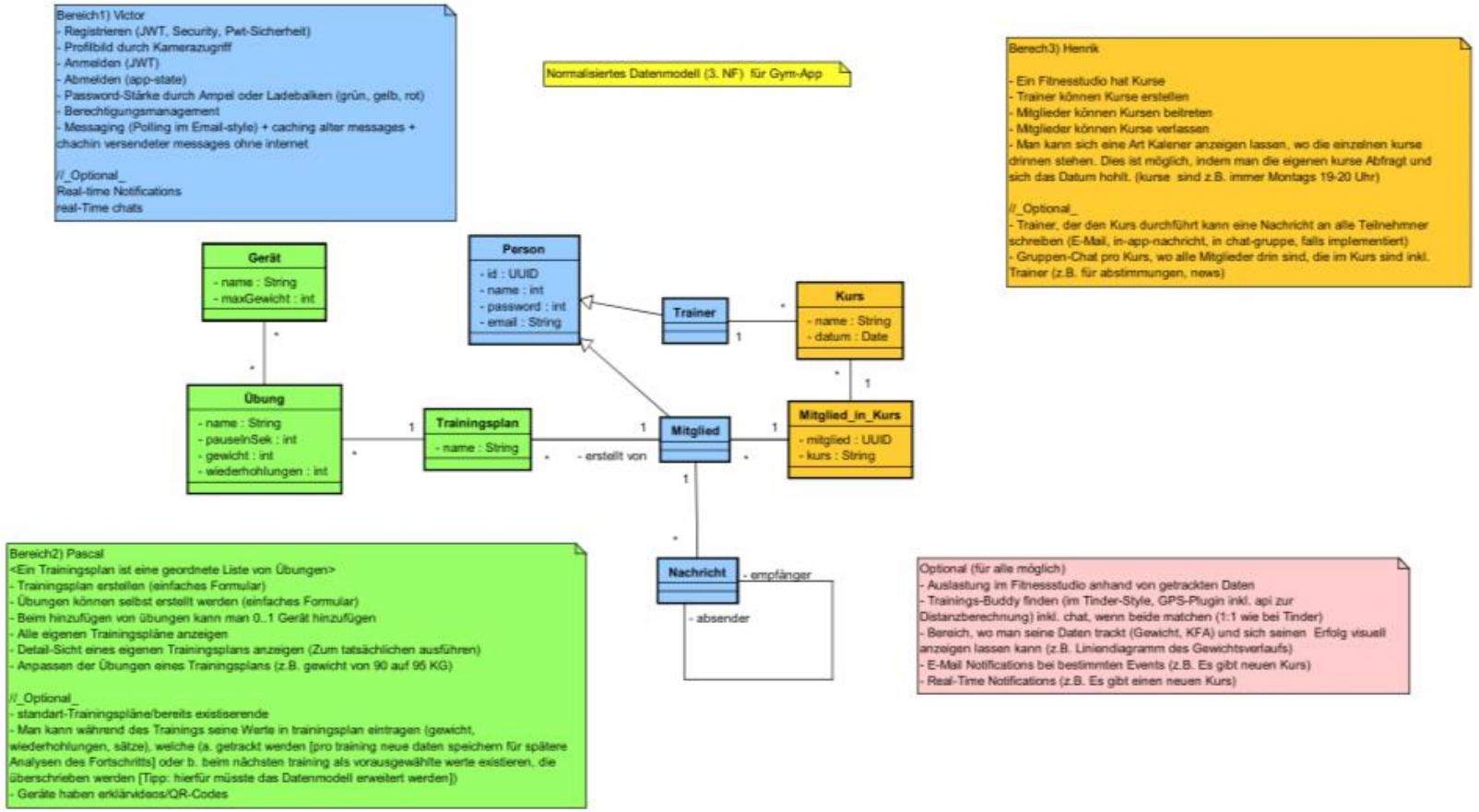
Teams-Projekt: Dateien, Videokonferenzen, Pair-Programming

WhatsApp: Kommunikation

Gitlab: Kollaboratives Entwickeln, CI/CD-Pipeline, Code-Review

Iterationen: Ist-Zustand besprechen & nächsten Schritte planen

Divide and Conquer

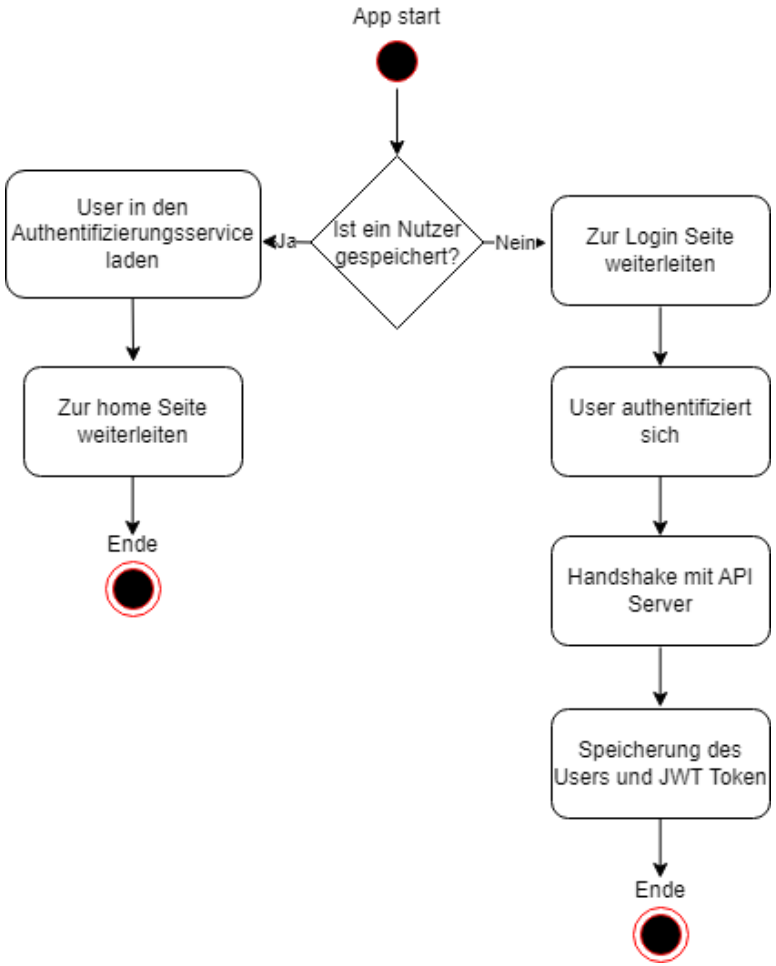


Technisches

Technisches

Authentifizierung und Rollenmanagement

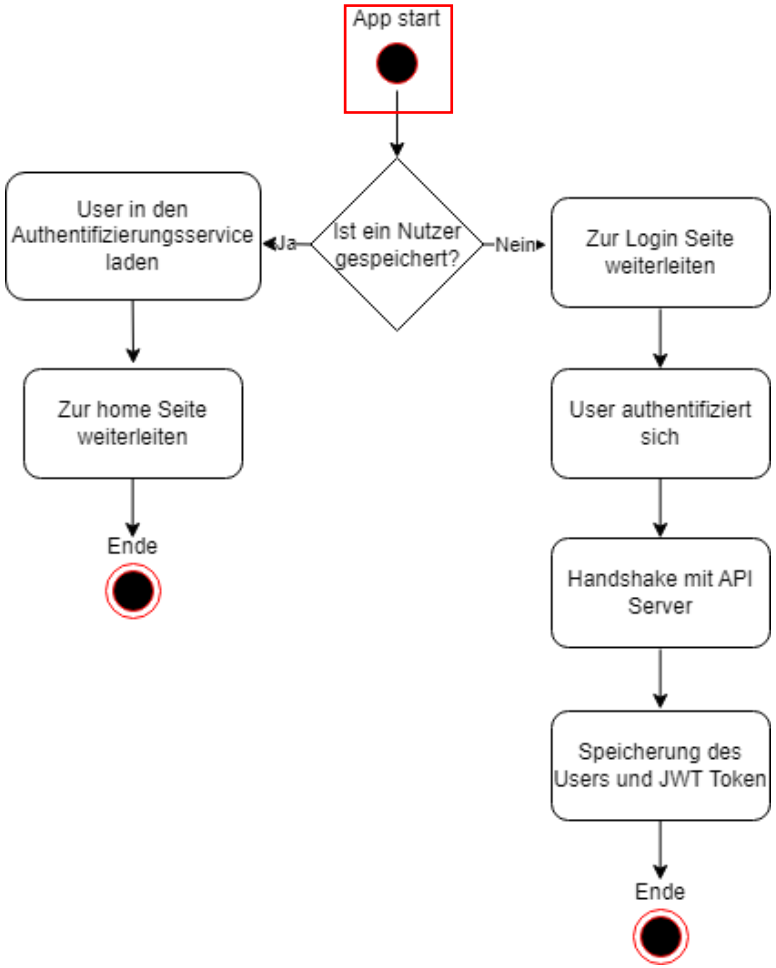
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

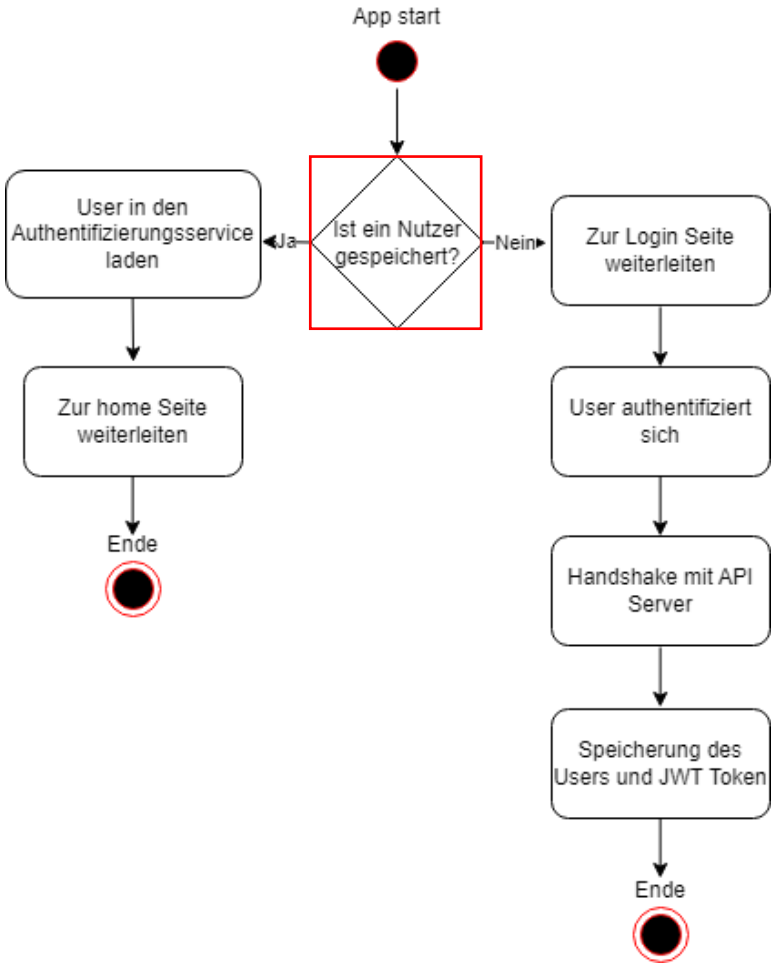
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

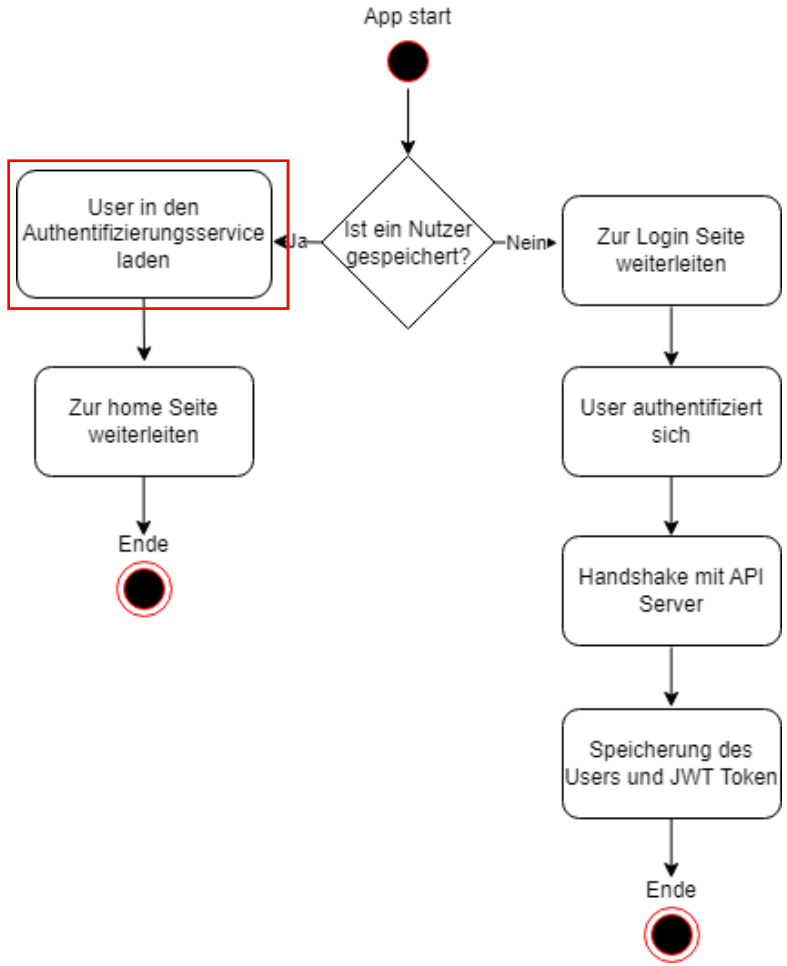
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

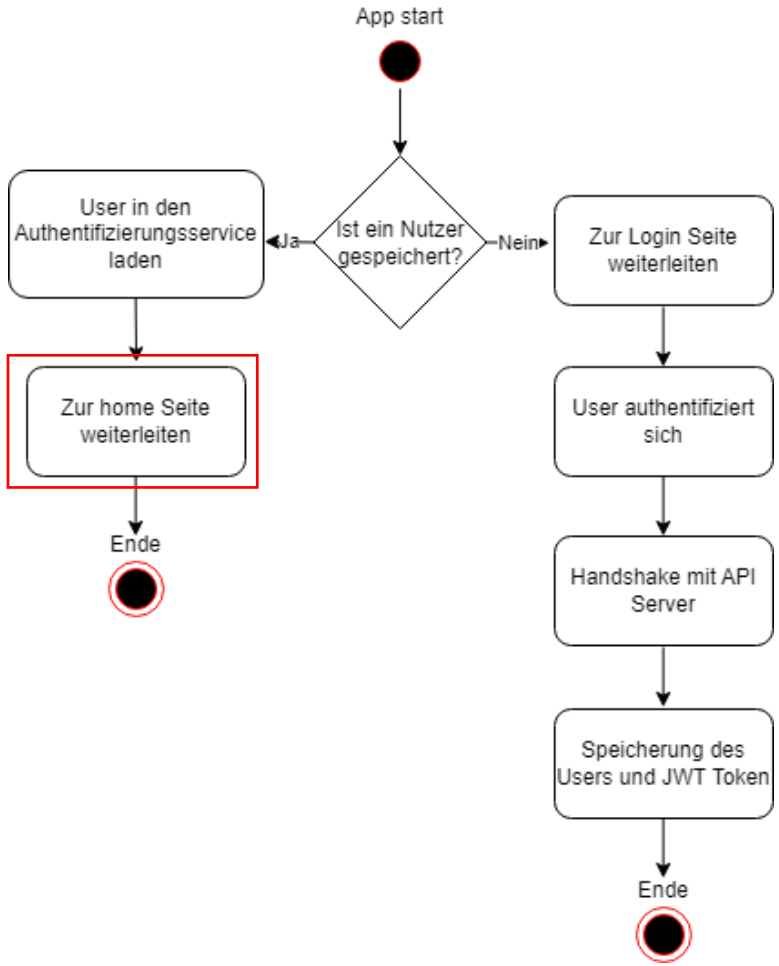
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

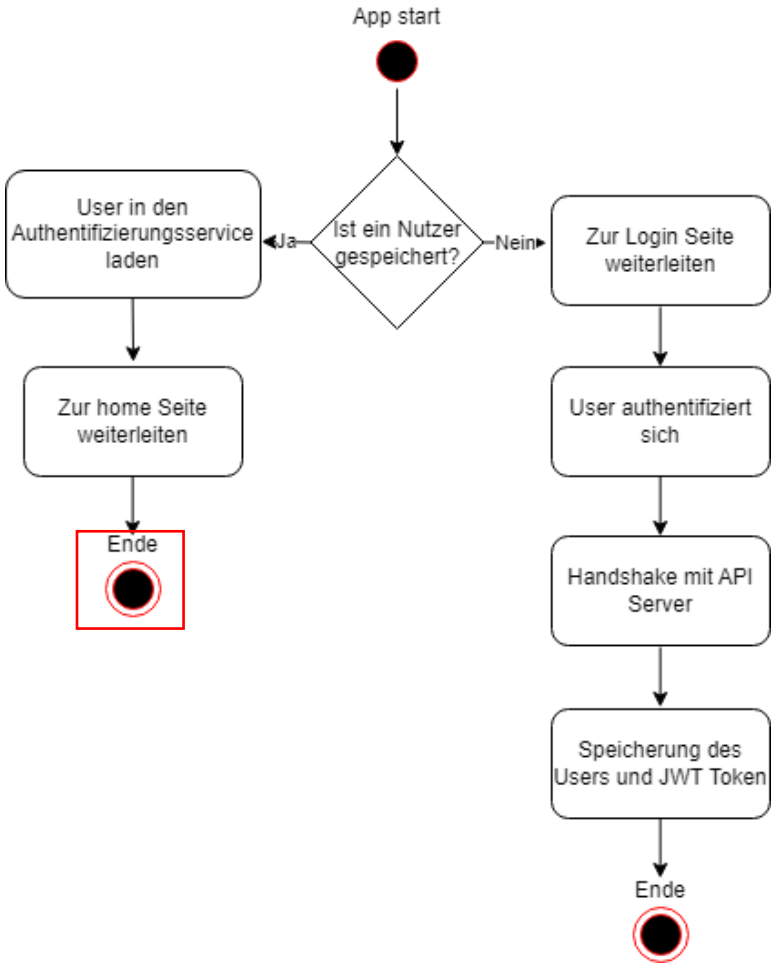
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

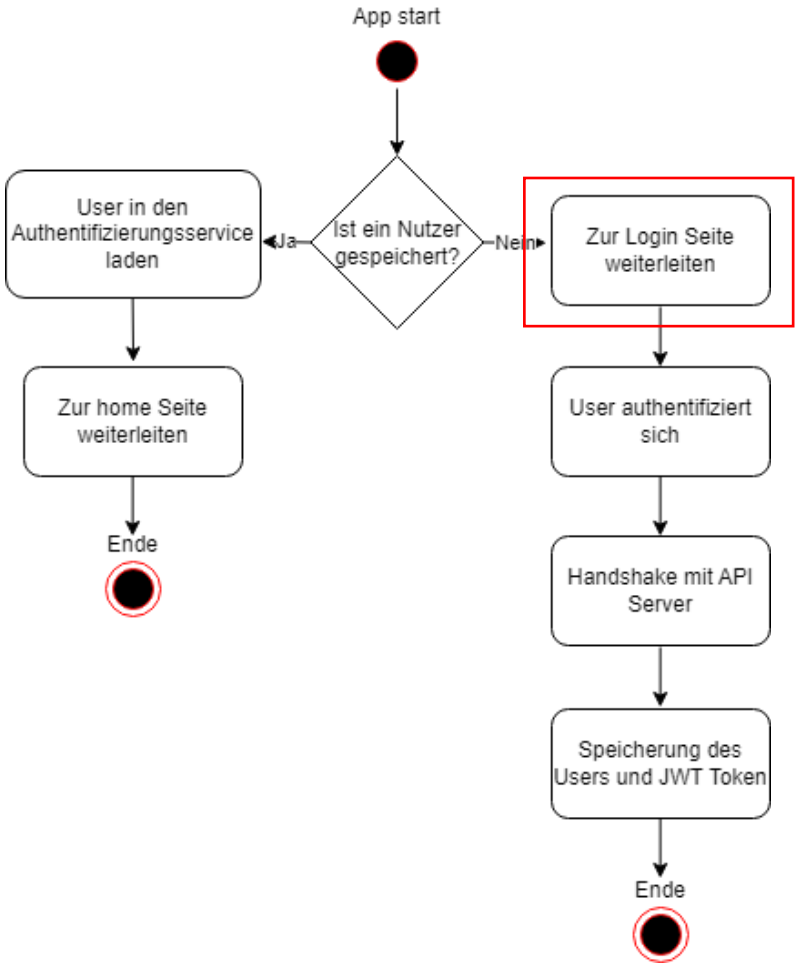
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

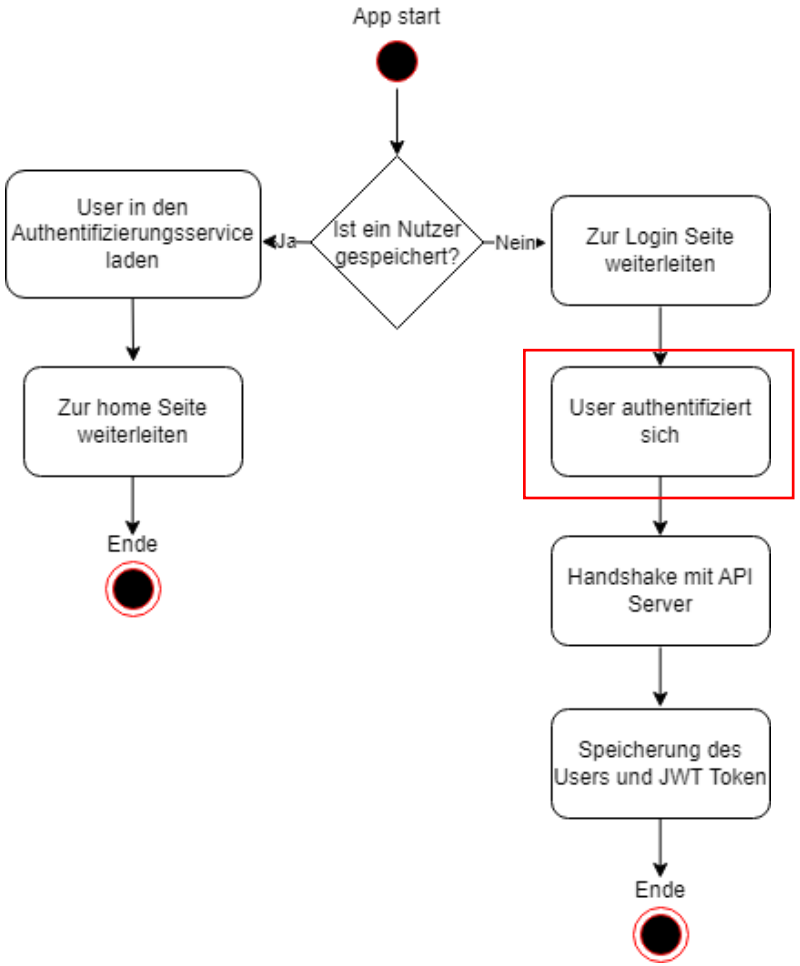
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

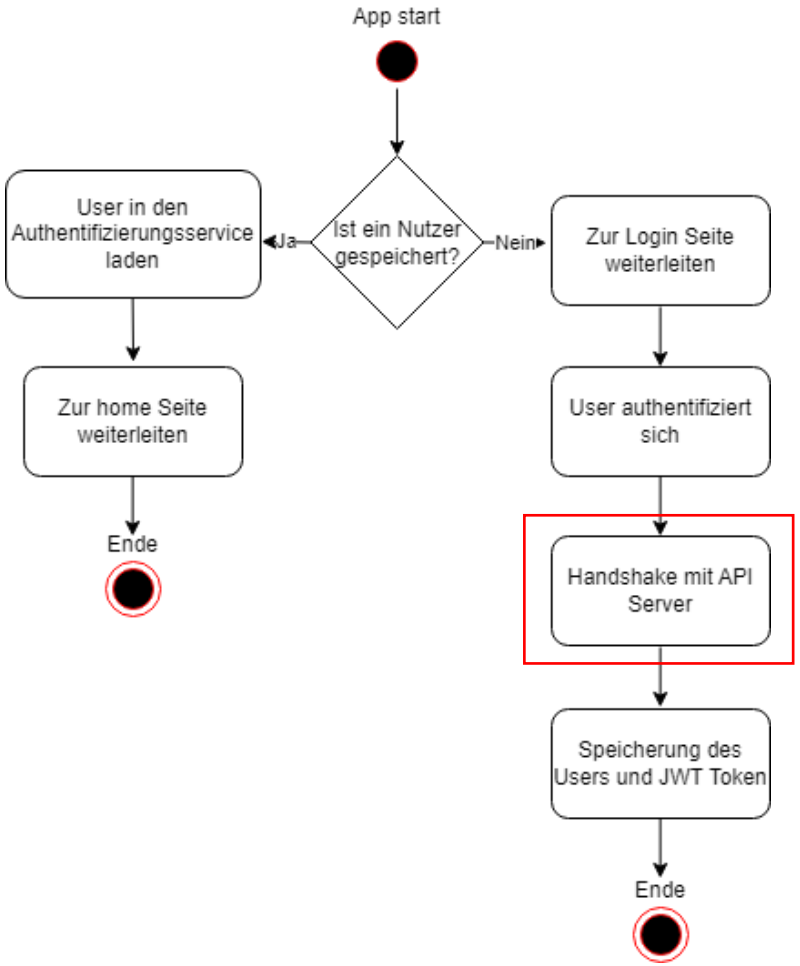
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

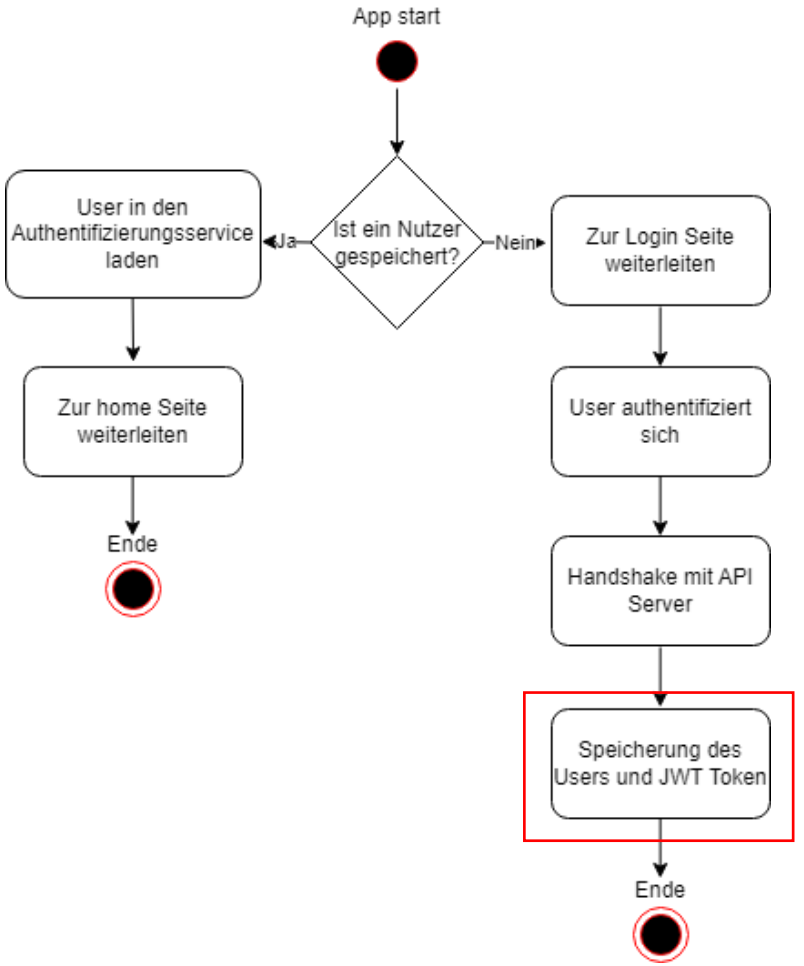
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

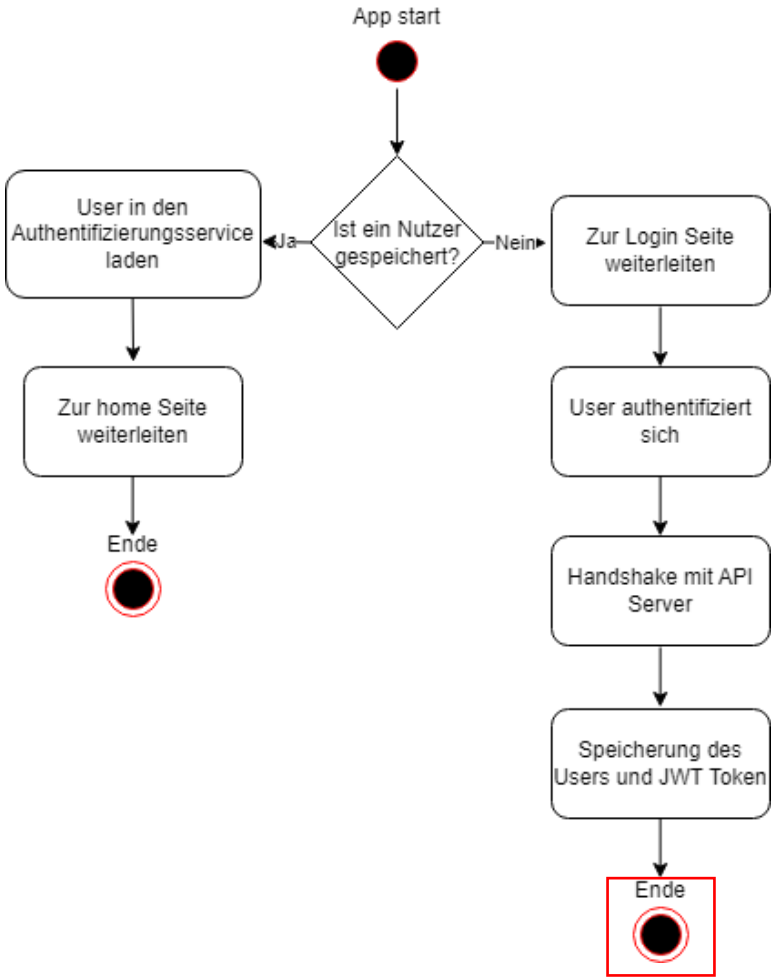
- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

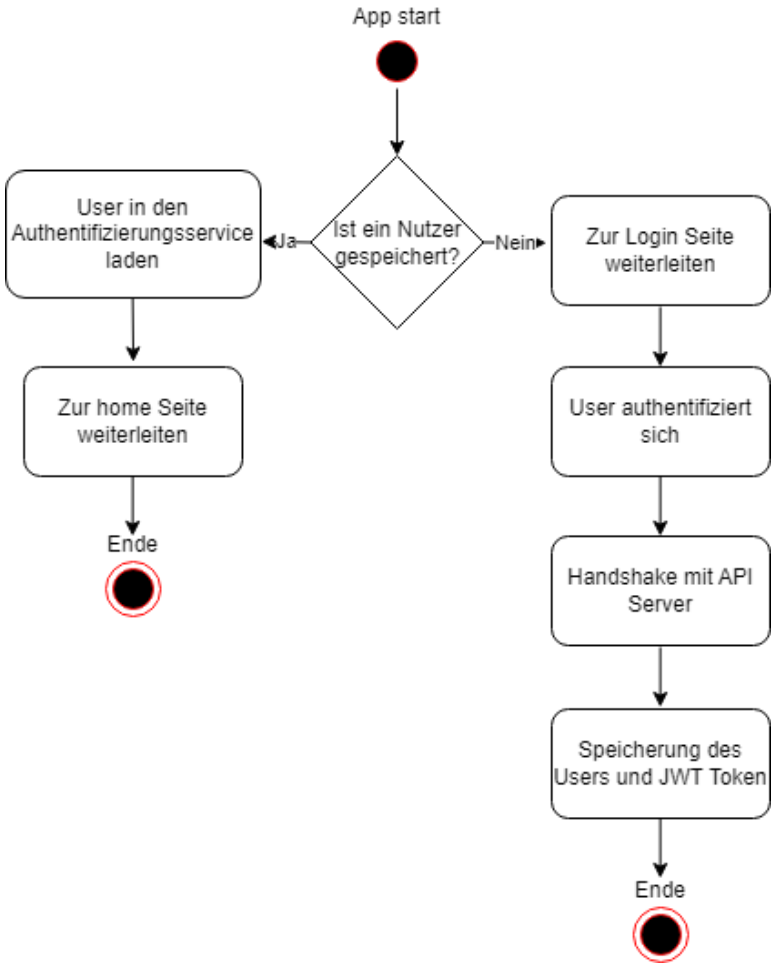
Authentifizierung und Rollenmanagement

- Bei starten der App wird der gespeicherte User und JWT-Token aus dem Storage geladen und der User muss ggf. nicht mehr einloggen
- Nach der Authentifizierung mittels JWT Token, Speicherung des Token und des Users im ionic storage
- AuthenticationService als zentraler Ort zum Managen und Abfragen des states.
- Neu rendern von globalen Komponenten bei Änderungen.
- Speicherung im Storage: Keine Cors Cookie Probleme für Lokale Entwicklung mit Abweichenden Ips und Domains



Technisches

Authentifizierung und Rollenmanagement

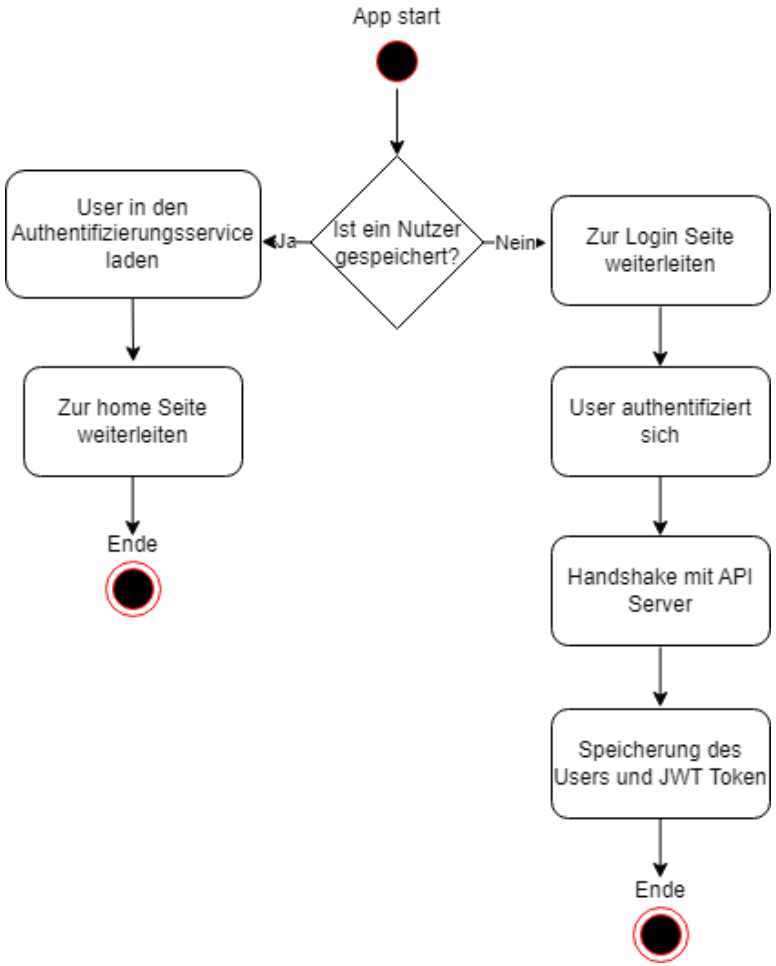


Technisches

Authentifizierung und Rollenmanagement

- Nutzer Vorteile:
  - Nutzer haben keine Lust ein Password beim Sport einzutippen
  - Schneller Start ins Training

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat



Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
You, 2 weeks ago | 2 authors (You and others)
class AuthenticationService {
  public storage!: Storage;

  private user!: string|null;

  async init(): Promise<void> {
      this.storage = new Storage();
      await this.storage.create();
      this.user = await this.storage.get(storageKey);
  }

  isAuthenticated(): boolean {
    return !!this.user;
  }

  isTrainer() {
    try {
        return this.getUser().isTrainer;
    } catch (error) {
        return false;
    }
  }

  async resetUserStorage() {
    await this.storage.remove(storageKey);
    this.user = null;
  }

  getUser(): Promise<User> {
    const user = this.user;

    if(!user) {
      throw Error('User was not set');
    }

    return JSON.parse(user);
  }

  async storeUser(user: User) {
    this.storage.set(storageKey, JSON.stringify(user));
    this.user = JSON.stringify(user);

    await WorkoutSyncDao.sync();
    await TaskSyncDao.sync();
    await ExerciseSyncDao.sync();
    await CourseSyncDao.sync();
    await ChatSyncDao.sync();
    await MemberInCourseSyncDao.sync();
    await UserSyncDao.sync();
    await MassageSyncDao.sync();
  }
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
You, 2 weeks ago | 2 authors (You and others)
class AuthenticationService {
  public storage!: Storage;

  private user!: string|null;

  async init(): Promise<void> {
      this.storage = new Storage();
      await this.storage.create();
      this.user = await this.storage.get(storageKey);
  }

  isAuthenticated(): boolean {
    return !!this.user;
  }

  isTrainer() {
    try {
        return this.getUser().isTrainer;
    } catch (error) {
        return false;
    }
  }

  async resetUserStorage() {
    await this.storage.remove(storageKey);
    this.user = null;
  }

  getUser(): Promise<User> {
    const user = this.user;

    if(!user) {
      throw Error('User was not set');
    }

    return JSON.parse(user);
  }

  async storeUser(user: User) {
    this.storage.set(storageKey, JSON.stringify(user));
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
isAuthenticated(): boolean {
    return !!this.user;
}

isTrainer() {
    try {
        return this.getUser().isTrainer;
    } catch (error) {
        return false;
    }
}

async resetUserStorage() {
    await this.storage.remove(storageKey);
    this.user = null;
}

getUser(): Promise<User> {
    const user = this.user;

    if(!user) {
        throw Error('User was not set');
    }

    return JSON.parse(user);
}

async storeUser(user: User) {
    this.storage.set(storageKey, JSON.stringify(user));
    this.user = JSON.stringify(user);

    await WorkoutSyncDao.sync();
    await TaskSyncDao.sync();
    await ExerciseSyncDao.sync();
    await CourseSyncDao.sync();
    await ChatSyncDao.sync();
    await MemberInCourseSyncDao.sync();
    await UserSyncDao.sync();
    await MassageSyncDao.sync();
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
You, 2 weeks ago | 2 authors (You and others)
class AuthenticationService {
  public storage!: Storage;

  private user!: string|null;

  async init(): Promise<void> {
      this.storage = new Storage();
      await this.storage.create();
      this.user = await this.storage.get(storageKey);
  }

  isAuthenticated(): boolean {
    return !!this.user;
  }

  isTrainer() {
    try {
      return this.getUser().isTrainer;
    } catch (error) {
      return false;
    }
  }

  async resetUserStorage() {
    await this.storage.remove(storageKey);
    this.user = null;
  }

  getUser(): Promise<User> {
    const user = this.user;

    if(!user) {
      throw Error('User was not set');
    }

    return JSON.parse(user);
  }

  async storeUser(user: User) {
    this.storage.set(storageKey, JSON.stringify(user));
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
You, 2 weeks ago | 2 authors (You and others)
class AuthenticationService {
  public storage!: Storage;

  private user!: string|null;

  async init(): Promise<void> {
      this.storage = new Storage();
      await this.storage.create();
      this.user = await this.storage.get(storageKey);
  }

  isAuthenticated(): boolean {
    return !!this.user;
  }

  isTrainer() {
    try {
        return this.getUser().isTrainer;
    } catch (error) {
        return false;
    }
  }

  async resetUserStorage() {
    await this.storage.remove(storageKey);
    this.user = null;
  }

  getUser(): Promise<User> {
    const user = this.user;

    if(!user) {
      throw Error('User was not set');
    }

    return JSON.parse(user);
  }

  async storeUser(user: User) {
    this.storage.set(storageKey, JSON.stringify(user));
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
You, 2 weeks ago | 2 authors (You and others)
class AuthenticationService {
  public storage!: Storage;

  private user!: string|null;

  async init(): Promise<void> {
      this.storage = new Storage();
      await this.storage.create();
      this.user = await this.storage.get(storageKey);
  }

  isAuthenticated(): boolean {
    return !!this.user;
  }

  isTrainer() {
    try {
        return this.getUser().isTrainer;
    } catch (error) {
        return false;
    }
  }

  async resetUserStorage() {
    await this.storage.remove(storageKey);
    this.user = null;
  }

  getUser(): Promise<User> {
    const user = this.user;

    if(!user) {
      throw Error('User was not set');
    }

    return JSON.parse(user);
  }

  async storeUser(user: User) {
    this.storage.set(storageKey, JSON.stringify(user));
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
${this.routeItems.map(route => {
    return html`
    <ion-route url="${route.routePath}" component="${route.component}" .componentProps="${route.props}" .beforeEnter=${() => this.beforeEnter(route)}></ion-route>
    `;
})}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
${this.routeItems.map(route => {
    return html`
    <ion-route url="${route.routePath}" component="${route.component}" .componentProps="${route.props}" .beforeEnter=${() => this.beforeEnter(route)}></ion-route>
    `;
})}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
.componentProps="${route.props}" .beforeEnter=${() => this.beforeEnter(route)}></io
```

```
async beforeEnter(route: RouteItem) {
    if(!route.authRequired) {
        return true;
    }

    if(!authenticationService.isAuthenticated()) {
        notificationService.showNotification('Bitte loggen Sie sich ein.')
        return { redirect: '/users/sign-in' };
    }

    if(route.trainerRequired) {
        if(!authenticationService.isTrainer()) {
            notificationService.showNotification('Route ist nur für Trainer verfügbar.')
            return { redirect: '/home' };
        }
    }

    return true;
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
.componentProps="${route.props}" .beforeEnter=${() => this.beforeEnter(route)}></io
```

```
async beforeEnter(route: RouteItem) {
    if(!route.authRequired) {
        return true;
    }

    if(!authenticationService.isAuthenticated()) {
        notificationService.showNotification('Bitte loggen Sie sich ein.')
        return { redirect: '/users/sign-in' };
    }

    if(route.trainerRequired) {
        if(!authenticationService.isTrainer()) {
            notificationService.showNotification('Route ist nur für Trainer verfügbar.')
            return { redirect: '/home' };
        }
    }

    return true;
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
.componentProps="${route.props}" .beforeEnter=${() => this.beforeEnter(route)}></io
```

```
async beforeEnter(route: RouteItem) {
    if(!route.authRequired) {
        return true;
    }

    if(!authenticationService.isAuthenticated()) {
        notificationService.showNotification('Bitte loggen Sie sich ein.')
        return { redirect: '/users/sign-in' };
    }

    if(route.trainerRequired) {
        if(!authenticationService.isTrainer()) {
            notificationService.showNotification('Route ist nur für Trainer verfügbar.')
            return { redirect: '/home' };
        }
    }

    return true;
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
.componentProps="${route.props}" .beforeEnter=${() => this.beforeEnter(route)}></io
```

```
async beforeEnter(route: RouteItem) {
    if(!route.authRequired) {
        return true;
    }

    if(!authenticationService.isAuthenticated()) {
        notificationService.showNotification('Bitte loggen Sie sich ein.')
        return { redirect: '/users/sign-in' };
    }

    if(route.trainerRequired) {
        if(!authenticationService.isTrainer()) {
            notificationService.showNotification('Route ist nur für Trainer verfügbar.')
            return { redirect: '/home' };
        }
    }

    return true;
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
@customElement('app-header')
// eslint-disable-next-line @typescript-eslint/no-unused-vars
class HeaderComponent extends LitElement {
  static styles = componentStyle;

  @property() title = '';

  @property({ type: Array }) routeItems: RouteItem[] = [];

  @property() private currentRoute!: RouteItem;

  @state() menuOpen = false;

  render() {
    return html`
      <a class="title">${this.title}</a>
      <span class="menu-button" @click="${this.toggleMenu}"></span>
      <ol ?open=${this.menuOpen}>
        ${this.routeItems.filter(routeItem => {
          if(routeItem.authRequired !== authenticationService.isAuthenticated()) {
            return false;
          }

          if(routeItem.trainerRequired == true && !authenticationService.isTrainer()) {
            return false;
          }

          return routeItem.inBrowserHeader;
        })
        .map(
          routeItem => html`<li><a @click="${() => this.navigate(routeItem.routePath)}">${routeItem.title}</a></li>`
        )}
      </ol>
    `;
  }

  navigate(route) {
    this.menuOpen = false;
    router.navigate(route);
  }

  toggleMenu() {
    this.menuOpen = !this.menuOpen;
  }
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
@property() title = '';

@property({ type: Array }) routeItems: RouteItem[] = [];

@property() private currentRoute!: RouteItem;

@state() menuOpen = false;

render() {
    return html`
        <a class="title">${this.title}</a>
        <span class="menu-button" @click="${this.toggleMenu}"></span>
        <ol ?open=${this.menuOpen}>
            ${this.routeItems.filter(routeItem => {
                if(routeItem.authRequired !== authenticationService.isAuthenticated()) {
                    return false;
                }

                if(routeItem.trainerRequired == true && !authenticationService.isTrainer()) {
                    return false;
                }

                return routeItem.inBrowserHeader;
            })
            .map(
                routeItem => html`<li><a @click="${() => this.navigate(routeItem.routePath)}">${routeItem
            )}
        </ol>
    `;
}

navigate(route) {
    this.menuOpen = false;
    router.navigate(route);
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
@property() title = '';

@property({ type: Array }) routeItems: RouteItem[] = [];

@property() private currentRoute!: RouteItem;

@state() menuOpen = false;

render() {
  return html`
    <a class="title">${this.title}</a>
    <span class="menu-button" @click="${this.toggleMenu}"></span>
    <ol ?open=${this.menuOpen}>
      ${this.routeItems.filter(routeItem => {
        if(routeItem.authRequired !== authenticationService.isAuthenticated()) {
          return false;
        }

        if(routeItem.trainerRequired == true && !authenticationService.isTrainer()) {
          return false;
        }

        return routeItem.inBrowserHeader;
      })
      .map(
        routeItem => html`<li><a @click="${() => this.navigate(routeItem.routePath)}">${routeItem
      )}
    </ol>
  `;
}

navigate(route) {
  this.menuOpen = false;
  router.navigate(route);
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
@property() title = '';

@property({ type: Array }) routeItems: RouteItem[] = [];

@property() private currentRoute!: RouteItem;

@state() menuOpen = false;

render() {
  return html`
    <a class="title">${this.title}</a>
    <span class="menu-button" @click="${this.toggleMenu}"></span>
    <ol ?open=${this.menuOpen}>
      ${this.routeItems.filter(routeItem => {
        if(routeItem.authRequired !== authenticationService.isAuthenticated()) {
          return false;
        }

        if(routeItem.trainerRequired == true && !authenticationService.isTrainer()) {
          return false;
        }

        return routeItem.inBrowserHeader;
      })
      .map(
        routeItem => html`<li><a @click="${() => this.navigate(routeItem.routePath)}">${routeItem
      )}
    </ol>
  `;
}

navigate(route) {
  this.menuOpen = false;
  router.navigate(route);
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
@property() title = '';

@property({ type: Array }) routeItems: RouteItem[] = [];

@property() private currentRoute!: RouteItem;

@state() menuOpen = false;

render() {
  return html`
    <a class="title">${this.title}</a>
    <span class="menu-button" @click="${this.toggleMenu}"></span>
    <ol ?open=${this.menuOpen}>
      ${this.routeItems.filter(routeItem => {
        if(routeItem.authRequired !== authenticationService.isAuthenticated()) {
          return false;
        }

        if(routeItem.trainerRequired == true && !authenticationService.isTrainer()) {
          return false;
        }

        return routeItem.inBrowserHeader;
      })
      .map(
        routeItem => html`<li><a @click="${() => this.navigate(routeItem.routePath)}">${routeItem
      )}
    </ol>
  `;
}

navigate(route) {
  this.menuOpen = false;
  router.navigate(route);
}
```

Technisches

Authentifizierung und Rollenmanagement

- Technische Vorteile:
  - Keine Cors Probleme bei lokaler Entwicklung
  - Leichtes Abfragen ob ein User eingeloggt ist und welche Rolle er hat

```
const child = document.querySelector('app-header') as LitElement;
if(child) {
    child.requestUpdate();
}
```

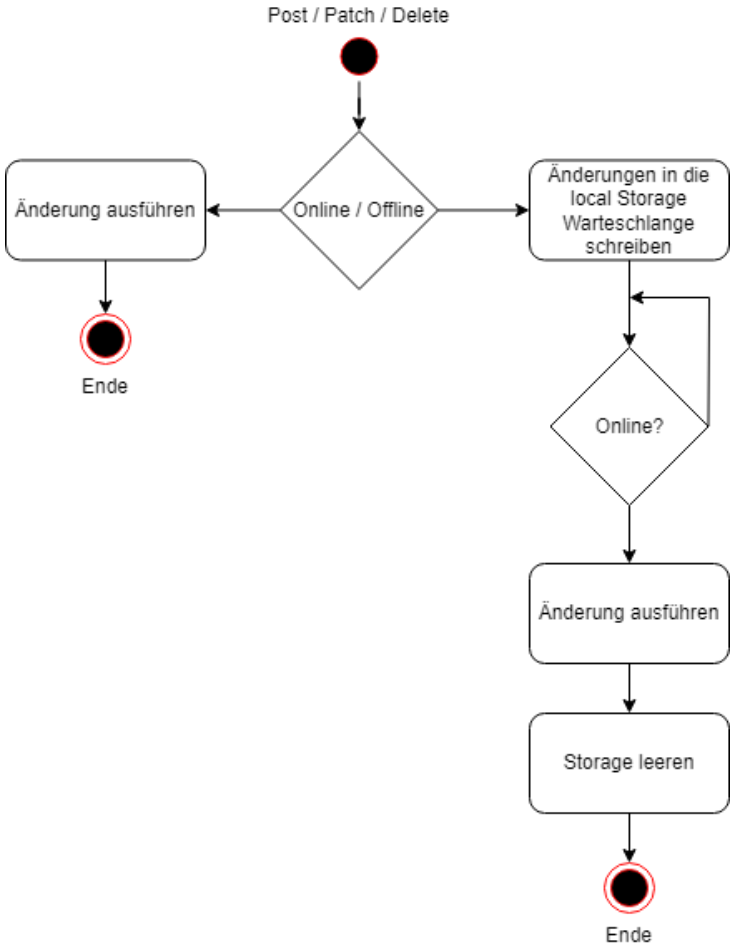
```
@property() title = '';

@property({ type: Array }) routeItems: RouteItem[] = [];

@property() private currentRoute!: RouteItem;

@state() menuOpen = false;

render() {
    return html`
        <a class="title">${this.title}</a>
        <span class="menu-button" @click="${this.toggleMenu}"></span>
        <ol ?open=${this.menuOpen}>
            ${this.routeItems.filter(routeItem => {
                if(routeItem.authRequired !== authenticationService.isAuthenticated()) {
                    return false;
                }

                if(routeItem.trainerRequired == true && !authenticationService.isTrainer()) {
                    return false;
                }

                return routeItem.inBrowserHeader;
            })
            .map(
                routeItem => html`<li><a @click="${() => this.navigate(routeItem.routePath)}">${routeItem
            )}
        </ol>
    `;
}

navigate(route) {
    this.menuOpen = false;
    router.navigate(route);
}
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

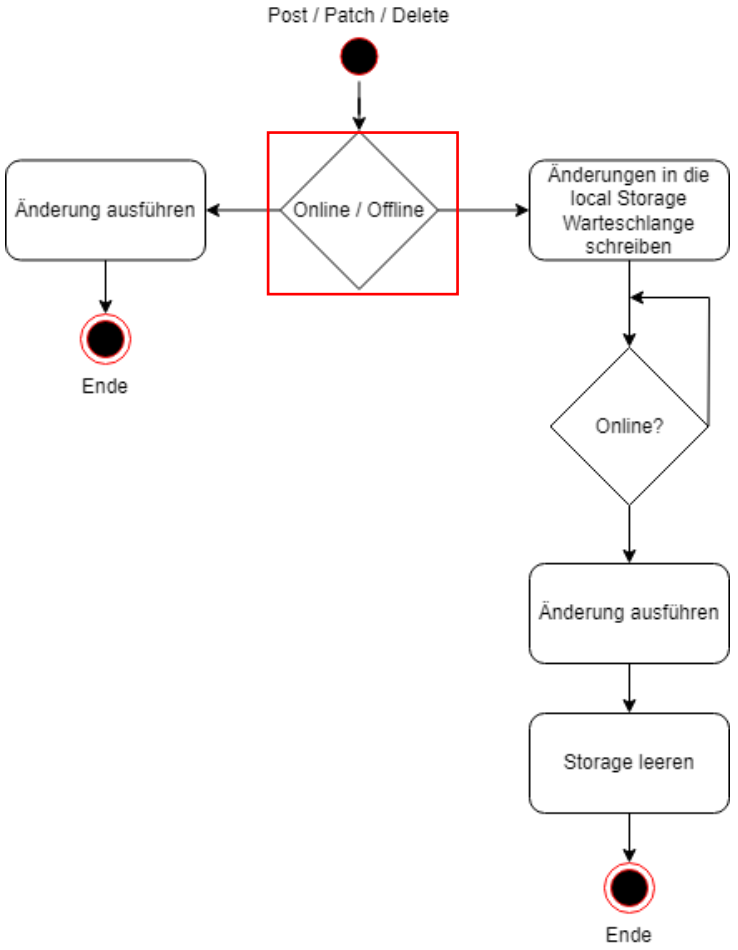
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



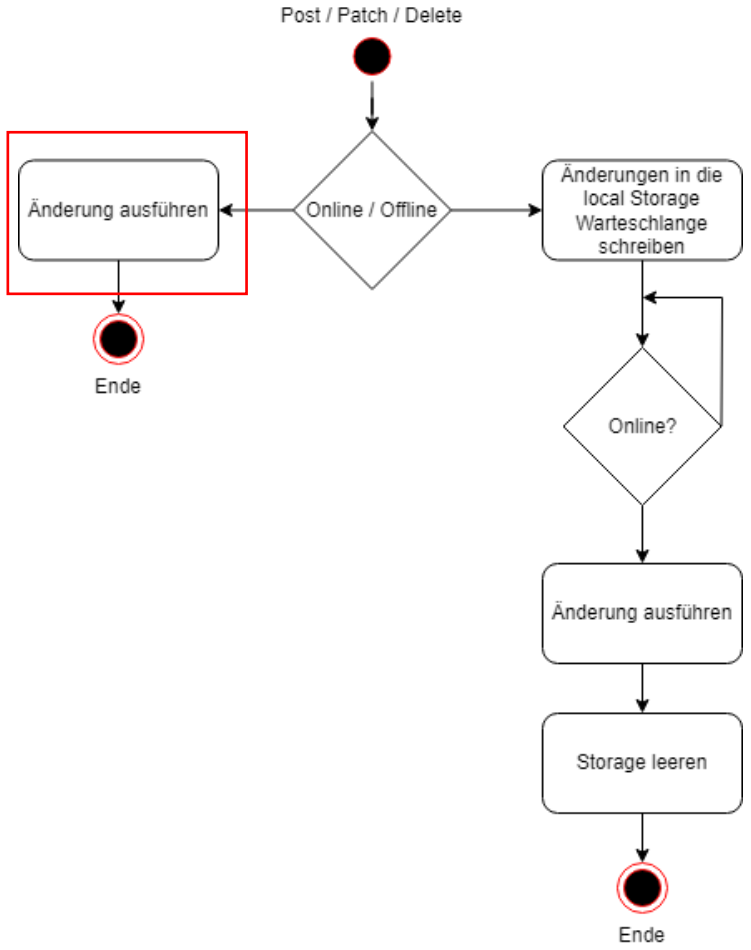
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



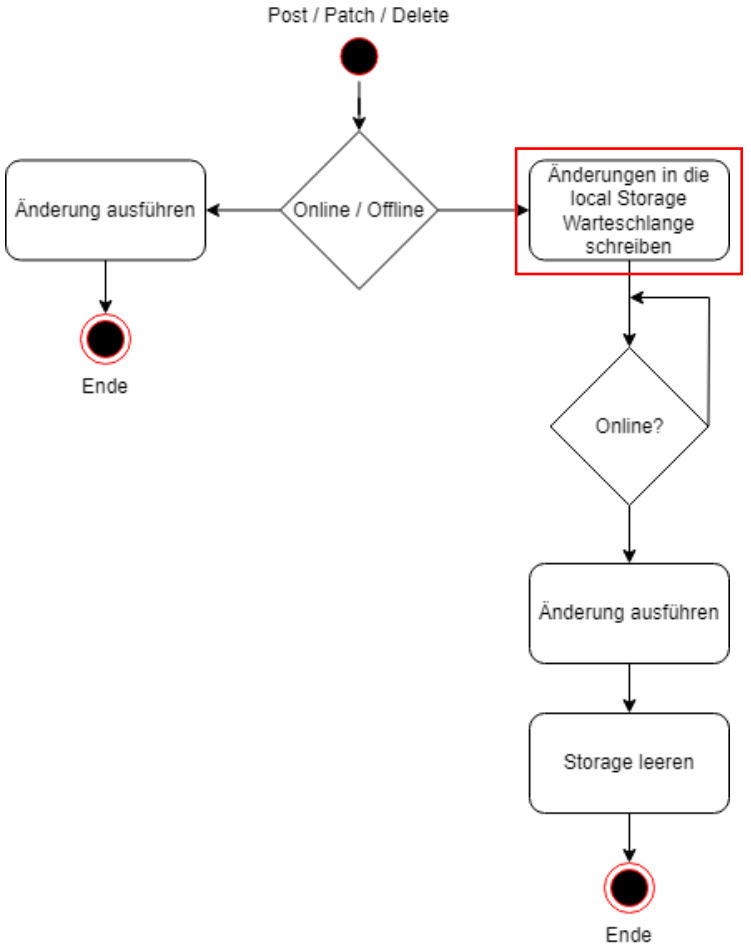
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



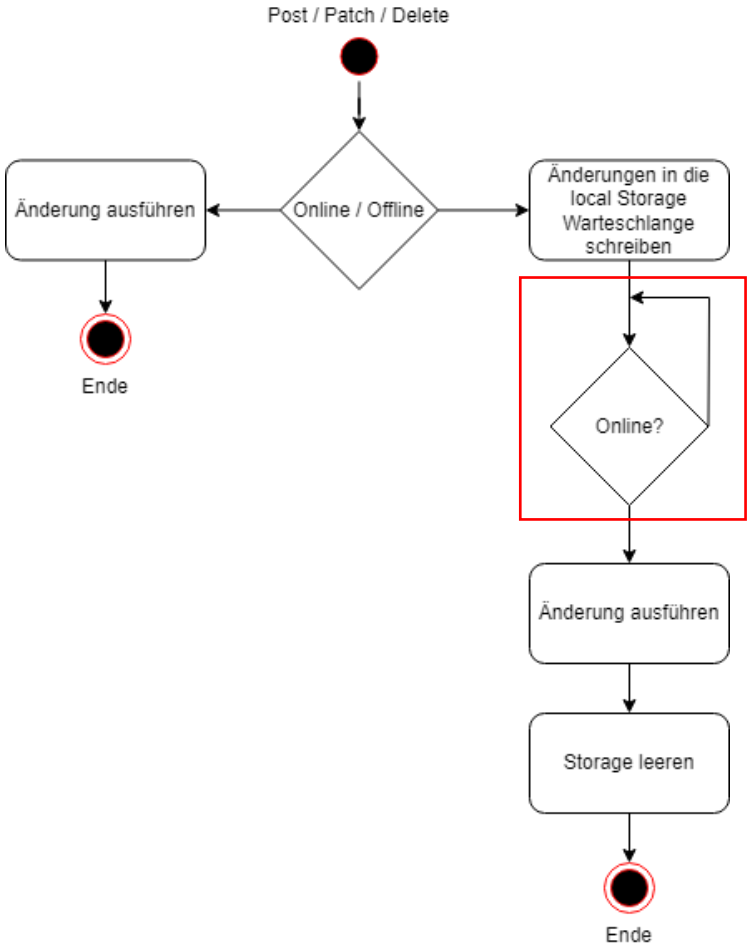
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



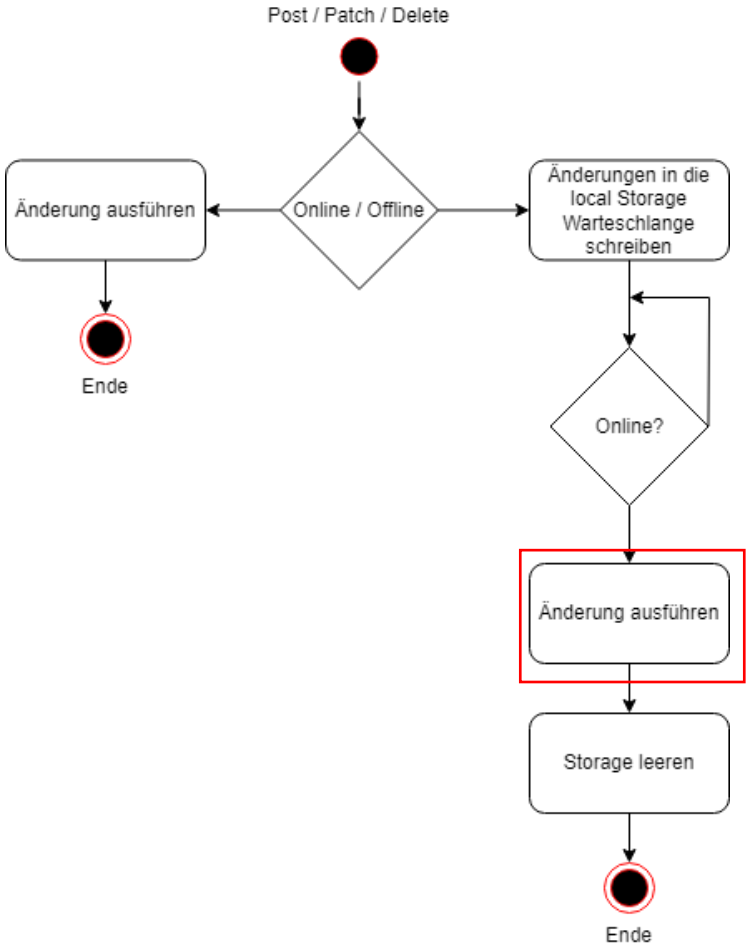
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



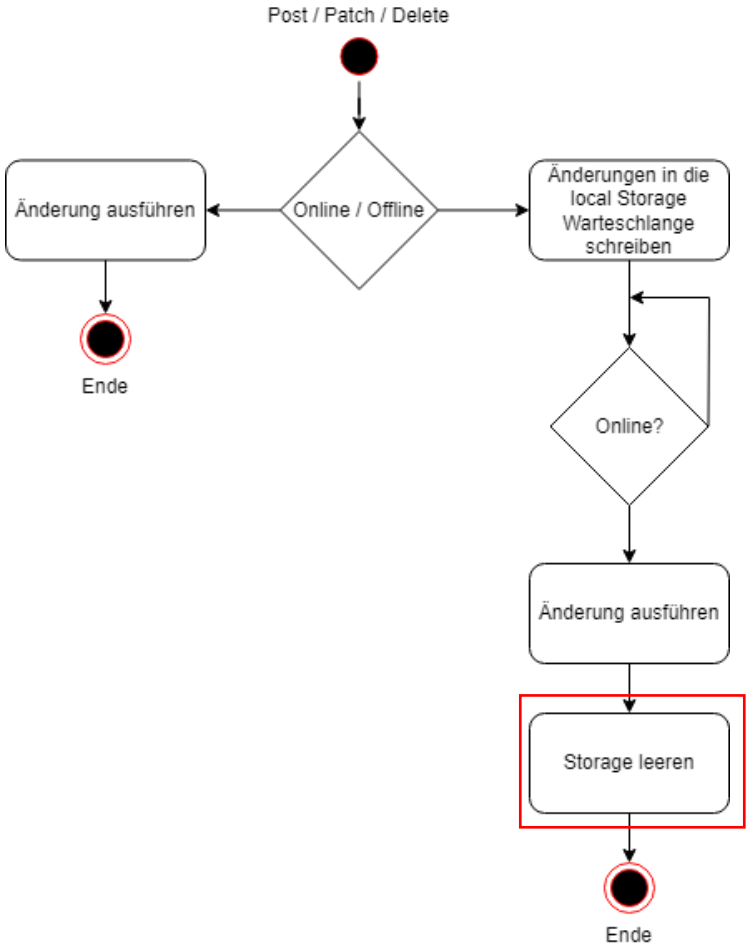
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



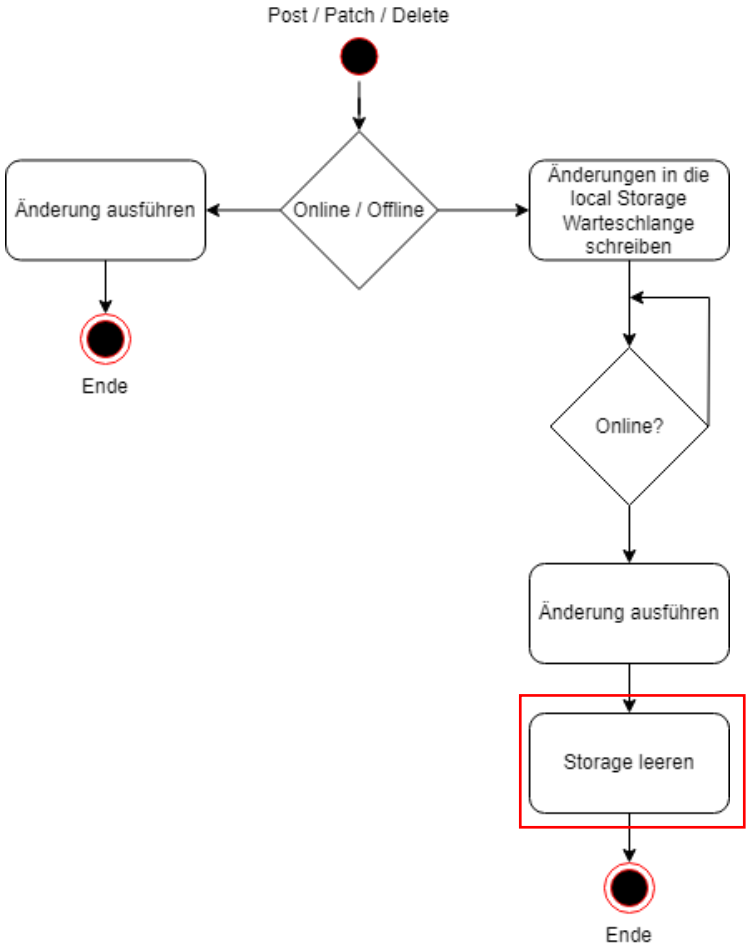
Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb



Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

```
export class HttpClient {
  private config!: HttpClientConfig;
  private storage!: Storage;
  public isOffline: boolean = false;

  init(config: HttpClientConfig) {
    this.config = config;

    Network.getStatus().then(async status => {
      await this.onNetworkStatusChanged(status, false);
    })

    Network.addListener('networkStatusChange', async status => {
      await this.onNetworkStatusChanged(status, true);
    })
  }

  public get(url: string) {
    return this.createFetch('GET', url);
  }

  public post(url: string, body: unknown) {
    return this.createFetch('POST', url, body);
  }

  public put(url: string, body: unknown) {
    return this.createFetch('PUT', url, body);
  }

  public patch(url: string, body: unknown) {
    return this.createFetch('PATCH', url, body);
  }

  public delete(url: string) {
    return this.createFetch('DELETE', url);
  }

  public async getJwt(): Promise<string|null> {
    if(!this.storage) {
      this.storage = new Storage();
      await this.storage.create();
    }

    return (await this.storage.get(JWT_KEY)) ?? null
  }

  private async createFetch(method: string, url: string, body?: unknown) {
    const jwt = (await this.storage.get(JWT_KEY)) ?? null;
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

```
export class HttpClient {
  private config!: HttpClientConfig;
  private storage!: Storage;
  public isOffline: boolean = false;

  init(config: HttpClientConfig) {
    this.config = config;

    Network.getStatus().then(async status => {
      await this.onNetworkStatusChanged(status, false);
    })

    Network.addListener('networkStatusChange', async status => {
      await this.onNetworkStatusChanged(status, true);
    })
  }

  public get(url: string) {
    return this.createFetch('GET', url);
  }

  public post(url: string, body: unknown) {
    return this.createFetch('POST', url, body);
  }

  public put(url: string, body: unknown) {
    return this.createFetch('PUT', url, body);
  }

  public patch(url: string, body: unknown) {
    return this.createFetch('PATCH', url, body);
  }

  public delete(url: string) {
    return this.createFetch('DELETE', url);
  }

  public async getJwt(): Promise<string|null> {
    if(!this.storage) {
      this.storage = new Storage();
      await this.storage.create();
    }

    return (await this.storage.get(JWT_KEY)) ?? null
  }

  private async createFetch(method: string, url: string, body?: unknown) {
    const jwt = (await this.storage.get(JWT_KEY)) ?? null;
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

```
export class HttpClient {
  private config!: HttpClientConfig;
  private storage!: Storage;
  public isOffline: boolean = false;

  init(config: HttpClientConfig) {
    this.config = config;

    Network.getStatus().then(async status => {
      await this.onNetworkStatusChanged(status, false);
    })

    Network.addListener('networkStatusChange', async status => {
      await this.onNetworkStatusChanged(status, true);
    })
  }

  public get(url: string) {
    return this.createFetch('GET', url);
  }

  public post(url: string, body: unknown) {
    return this.createFetch('POST', url, body);
  }

  public put(url: string, body: unknown) {
    return this.createFetch('PUT', url, body);
  }

  public patch(url: string, body: unknown) {
    return this.createFetch('PATCH', url, body);
  }

  public delete(url: string) {
    return this.createFetch('DELETE', url);
  }

  public async getJwt(): Promise<string|null> {
    if(!this.storage) {
      this.storage = new Storage();
      await this.storage.create();
    }

    return (await this.storage.get(JWT_KEY)) ?? null
  }

  private async createFetch(method: string, url: string, body?: unknown) {
    const jwt = (await this.storage.get(JWT_KEY)) ?? null;
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

```
private async createFetch(method: string, url: string, body?: unknown) {
  const jwt = (await this.storage.get(JWT_KEY)) ?? null;

  let requestOptions: RequestInit = {
    headers: { 'Content-Type': 'application/json; charset=utf-8' },
    method: method,
    };

  if(jwt) {
    requestOptions.headers['jwt'] = (await this.storage.get(JWT_KEY)) ?? null;
  }

  if (body) {
    requestOptions.body = JSON.stringify(body);
  }

  const path = this.config.baseURL + (url.startsWith('/') ? url.substring(1) : url);

  if(this.isOffline) {
    let requests: Request[] = await this.getRequestArray() ?? [];
    requests.push({
      path: path,
      requestOptions: requestOptions
    });

    await this.setRequestArray(requests);

    return;
  }

  return this.doFetch(path, requestOptions);
}

async doFetch(path: string, requestOptions: string) {
  const response = await fetch(path, requestOptions);
  if (response.ok) {
    if(response.headers.has('jwt')) {
      await this.storage.set(JWT_KEY, response.headers.get('jwt'));
    }

    return response;
  } else {
    let message = await response.text();
    try {
      message = JSON.parse(message).message;
    } catch (e) {
      message = (e as Error).message;
    }
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

```
    return this.doFetch(path, requestOptions);
  }

  async doFetch(path: string, requestOptions: string) {
    const response = await fetch(path, requestOptions);
    if (response.ok) {
      if(response.headers.has('jwt')) {
        await this.storage.set(JWT_KEY, response.headers.get('jwt'));
      }

      return response;
    } else {
      let message = await response.text();
      try {
        message = JSON.parse(message).message;
      } catch (e) {
        message = (e as Error).message;
      }
      message = message || response.statusText;
      return Promise.reject({ message, statusCode: response.status });
    }
  }

  private async onNetworkStatusChanged(status: ConnectionStatus, showNotification = false) {
    if(!this.storage) {
      this.storage = new Storage();
      await this.storage.create();
    }

    this.isOffline = !status.connected;

    if(!status.connected) {
      notificationService.showNotification('Sie sind nun offline!', 'info')
      return;
    }

    if(showNotification) {
      notificationService.showNotification('Sie sind wieder online!', 'info')
    }

    const requests: Request[] = [];

    ((await this.getRequestArray()) ?? []).forEach(async (value: Request) => {
      await this.doFetch(value.path, value.requestOptions)
    })

    await this.clearStorage();
  }
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

```
    return this.doFetch(path, requestOptions);
}

async doFetch(path: string, requestOptions: string) {
    const response = await fetch(path, requestOptions);
    if (response.ok) {
        if(response.headers.has('jwt')) {
            await this.storage.set(JWT_KEY, response.headers.get('jwt'));
        }

        return response;
    } else {
        let message = await response.text();
        try {
            message = JSON.parse(message).message;
        } catch (e) {
            message = (e as Error).message;
        }
        message = message || response.statusText;
        return Promise.reject({ message, statusCode: response.status });
    }
}

private async onNetworkStatusChanged(status: ConnectionStatus, showNotification = false) {
    if(!this.storage) {
        this.storage = new Storage();
        await this.storage.create();
    }

    this.isOffline = !status.connected;

    if(!status.connected) {
        notificationService.showNotification('Sie sind nun offline!', 'info')
        return;
    }

    if(showNotification) {
        notificationService.showNotification('Sie sind wieder online!', 'info')
    }

    const requests: Request[] = [];

    ((await this.getRequestArray()) ?? []).forEach(async (value: Request) => {
        await this.doFetch(value.path, value.requestOptions)
    })

    await this.clearStorage();
}
```

Technisches
Offlinefähigkeit

- Aktuelle Entitäten für den User werden im ionic Storage (IndexedDB / localStorage) gespeichert
- Optimistischen updaten bei Änderungen
- Sofortiges Feedback für die Nutzer für ein flüssiges Usererlebnis auch im Offline betrieb

Aber wie wird das genutzt?

```
  return this.doFetch(path, requestOptions);
}

async doFetch(path: string, requestOptions: string) {
  const response = await fetch(path, requestOptions);
  if (response.ok) {
    if(response.headers.has('jwt')) {
      await this.storage.set(JWT_KEY, response.headers.get('jwt'));
    }

    return response;
  } else {
    let message = await response.text();
    try {
      message = JSON.parse(message).message;
    } catch (e) {
      message = (e as Error).message;
    }
    message = message || response.statusText;
    return Promise.reject({ message, statusCode: response.status });
  }
}

private async onNetworkStatusChanged(status: ConnectionStatus, showNotification = false) {
  if(!this.storage) {
    this.storage = new Storage();
    await this.storage.create();
  }

  this.isOffline = !status.connected;

  if(!status.connected) {
    notificationService.showNotification('Sie sind nun offline!', 'info')
    return;
  }

  if(showNotification) {
    notificationService.showNotification('Sie sind wieder online!', 'info')
  }

  const requests: Request[] = [];

  ((await this.getRequestArray()) ?? []).forEach(async (value: Request) => {
    await this.doFetch(value.path, value.requestOptions)
  })

  await this.clearStorage();
}
```

Technisches
Offlinefähigkeit

- Spieglung der Daten des Nutzers im Ionic Storage

- Einfachste Lösung!
- (Verbesserungspotential, bspw. partielles Laden von Updates, Handling von Fehlern, Weniger Requests)

```
export class SyncDAO<T extends Entity> implements GenericDAO<T> {
  private route!: string
  private storage!: Storage

  constructor(route: string) {
    this.route = route;
  }

  async init(){
    if(!this.storage) {
    this.storage = new Storage();
    await this.storage.create();
  }
  }

  async sync() {
    await this.init();

    if(httpClient.isOffline) {
      return;
    }

    const response = await httpClient.get(this.route);
    const responseData = (await response.json());

    const map: Map<string, T> = new Map();

    responseData.forEach(async (responseDatum) => {
      map.set(responseDatum.id, responseDatum)
    })

    this.storage.set(this.route, map)
  }

  public async create(partEntity: Omit<T, keyof Entity>) {
    await this.init();

    const entity = { ...partEntity, id: uuidv4(), createdAt: new Date().getTime() };

    const map = await this.getMap();

    await map.set(entity.id, entity as T);
    this.setMap(map);

    await httpClient.post(this.route, entity);

    return Promise.resolve(entity as T);
  }

  public async findAll(entityFilter?: Partial<T>) {
    await this.sync();

    const result = [] as T[];

    for (const entity of (await this.getMap()).values()) {
      if (!entityFilter || this._matches(entity, entityFilter)) {
        result.push(entity);
      }
    }
    return Promise.resolve(result);
```

Technisches
Offlinefähigkeit

- Spieglung der Daten des Nutzers im Ionic Storage

- Einfachste Lösung!
- (Verbesserungspotential, bspw. partielles Laden von Updates, Handling von Fehlern, Weniger Requests)

```
export class SyncDAO<T extends Entity> implements GenericDAO<T> {
  private route!: string
  private storage!: Storage

  constructor(route: string) {
    this.route = route;
  }

  async init(){
    if(!this.storage) {
    this.storage = new Storage();
    await this.storage.create();
  }
  }

  async sync() {
    await this.init();

    if(httpClient.isOffline) {
      return;
    }

    const response = await httpClient.get(this.route);
    const responseData = (await response.json());

    const map: Map<string, T> = new Map();

    responseData.forEach(async (responseDatum) => {
      map.set(responseDatum.id, responseDatum)
    })

    this.storage.set(this.route, map)
  }

  public async create(partEntity: Omit<T, keyof Entity>) {
    await this.init();

    const entity = { ...partEntity, id: uuidv4(), createdAt: new Date().getTime() };

    const map = await this.getMap();

    await map.set(entity.id, entity as T);
    this.setMap(map);

    await httpClient.post(this.route, entity);

    return Promise.resolve(entity as T);
  }

  public async findAll(entityFilter?: Partial<T>) {
    await this.sync();

    const result = [] as T[];

    for (const entity of (await this.getMap()).values()) {
      if (!entityFilter || this._matches(entity, entityFilter)) {
        result.push(entity);
      }
    }
    return Promise.resolve(result);
```

Technisches
Offlinefähigkeit

- Spieglung der Daten des Nutzers im Ionic Storage
- Ids werden vom Client vergeben

- Einfachste Lösung!
- (Verbesserungspotential, bspw. partielles Laden von Updates, Handling von Fehlern, Weniger Requests)

```
export class SyncDAO<T extends Entity> implements GenericDAO<T> {
  private route!: string
  private storage!: Storage

  constructor(route: string) {
    this.route = route;
  }

  async init(){
    if(!this.storage) {
    this.storage = new Storage();
    await this.storage.create();
  }
  }

  async sync() {
    await this.init();

    if(httpClient.isOffline) {
      return;
    }

    const response = await httpClient.get(this.route);
    const responseData = (await response.json());

    const map: Map<string, T> = new Map();

    responseData.forEach(async (responseDatum) => {
      map.set(responseDatum.id, responseDatum)
    })

    this.storage.set(this.route, map)
  }

  public async create(partEntity: Omit<T, keyof Entity>) {
    await this.init();

    const entity = { ...partEntity, id: uuidv4(), createdAt: new Date().getTime() };

    const map = await this.getMap();

    await map.set(entity.id, entity as T);
    this.setMap(map);

    await httpClient.post(this.route, entity);

    return Promise.resolve(entity as T);
  }

  public async findAll(entityFilter?: Partial<T>) {
    await this.sync();

    const result = [] as T[];

    for (const entity of (await this.getMap()).values()) {
      if (!entityFilter || this._matches(entity, entityFilter)) {
        result.push(entity);
      }
    }
    return Promise.resolve(result);
```

Technisches
Offlinefähigkeit

- Spieglung der Daten des Nutzers im Ionic Storage
- Ids werden vom Client vergeben

- Einfachste Lösung!
- (Verbesserungspotential, bspw. partielles Laden von Updates, Handling von Fehlern, Weniger Requests)

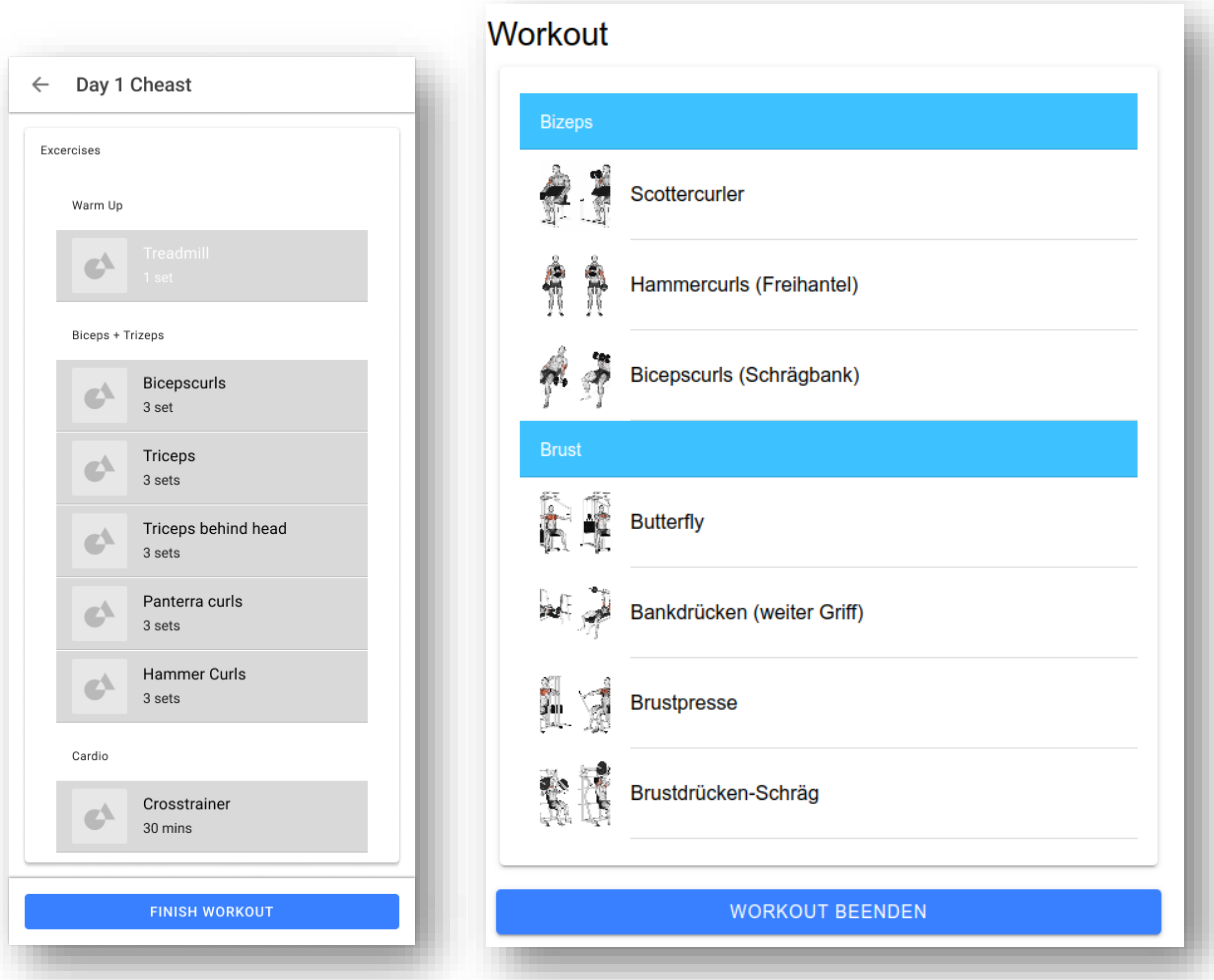
```
async sync() {
  await this.init();

  if(httpClient.isOffline) {
    return;
  }

  const response = await httpClient.get(this.route);
  const responseData = (await response.json());

  const map: Map<string, T> = new Map();

  responseData.forEach(async (responseDatum) => {
    map.set(responseDatum.id, responseDatum)
  })

  this.storage.set(this.route, map)
}

public async create(partEntity: Omit<T, keyof Entity>) {
  await this.init();

  const entity = { ...partEntity, id: uuidv4(), createdAt: new Date().getTime() };

  const map = await this.getMap();

  await map.set(entity.id, entity as T);
  this.setMap(map);

  await httpClient.post(this.route, entity);

  return Promise.resolve(entity as T);
}

public async findAll(entityFilter?: Partial<T>) {
  await this.sync();

  const result = [] as T[];

  for (const entity of (await this.getMap()).values()) {
    if (!entityFilter || this._matches(entity, entityFilter)) {
      result.push(entity);
    }
  }
  return Promise.resolve(result);
}

public async findOne(entityFilter: Partial<T>) {
  await this.sync();

  for (const entity of (await this.getMap()).values()) {
    if (this._matches(entity, entityFilter)) {
      return Promise.resolve(entity);
    }
  }
  return Promise.resolve(null);
}

public async update(entity: Partial<T> & Pick<Entity, 'id'>) {
  await this.init();
```

Live Demo

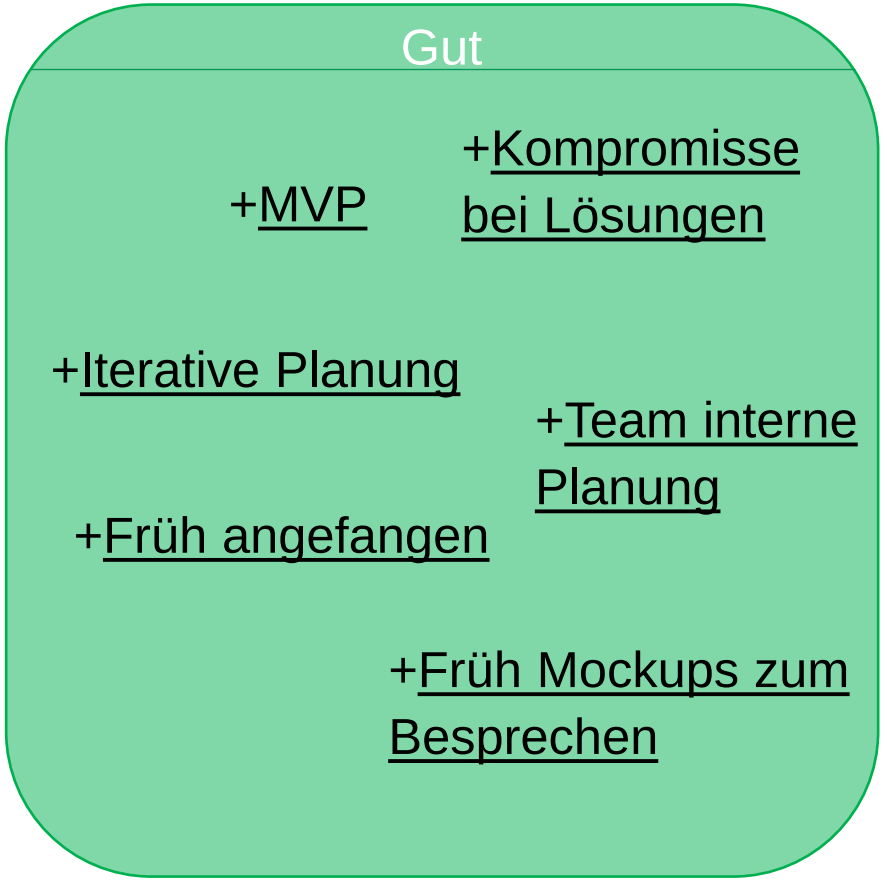
Rückblick & Fazit

Design

- Design als grobe Richtlinie
- Halfen bei der Feature-Besprechung und Erstellung des MVP
- Viele Übereinstimmungen lassen sich finden



Gut


Gut
+MVP
+Kompromisse bei Lösungen
+Iterative Planung
+Team interne Planung
+Früh angefangen
+Früh Mockups zum Besprechen

Gut / Schlecht

Schlecht

-Ionic Komponenten

-Taskman und Ionic mischen

-Ionic Dokumentation

Gut

+MVP

+Kompromisse bei Lösungen

+Iterative Planung

+Team interne Planung

+Früh angefangen

+Früh Mockups zum Besprechen

Danke!