Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitenskap
Institutt for datateknologi, elektroteknologi og realfag

Måltidsplanlegger
Systemdokumentasjon

Versjon 1.0

Dokumentet er basert på Systemdokumentasjon utarbeidet ved NTNU. Revisjon og tilpasninger til bruk ved IDER, DATA-INF utført av Carsten Gunnar Helgesen, Svein-Ivar Lillehaug og Per Christian Engdal. Dokumentet finnes også i engelsk utgave.

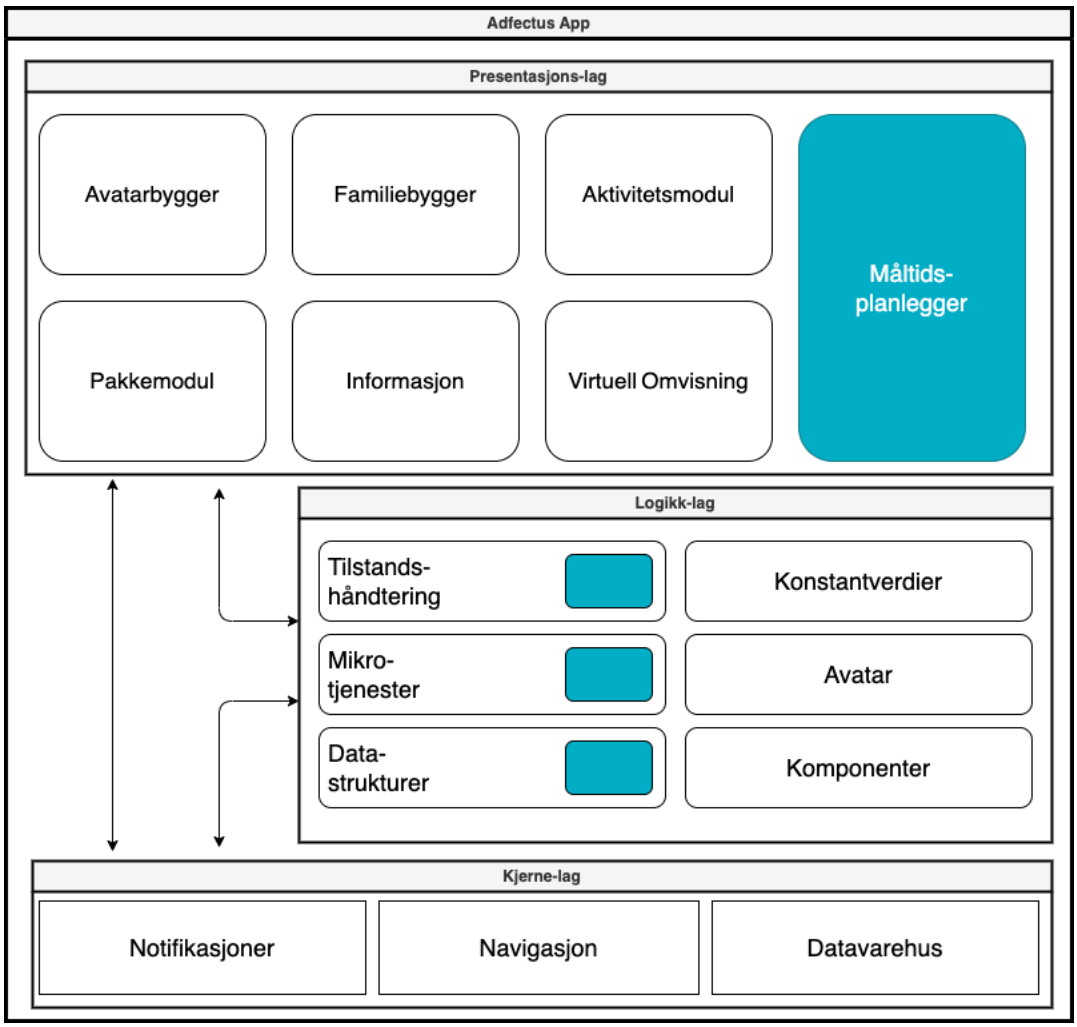
INNHOLDSFORTEGNELSE

1 INNLEDNING

Denne systemdokumentasjonen gir en oversikt over prosjektet Måltidsplanleggeren skrevet i JavaScript-rammeverket React Native. Den inkluderer overordnet arkitektur, prosjektstruktur, klassediagram, sikkerhet, installasjon og kjøring av prosjektet, kodens dokumentasjon og enhetstesting. Dokumentasjonen er utarbeidet for å gi et strukturert bilde av systemets oppbygging, funksjonalitet og driftsmessige aspekter. Formålet er å gi en strukturert forklaring av systemet til utviklere, vedlikeholdspersonell og brukere.

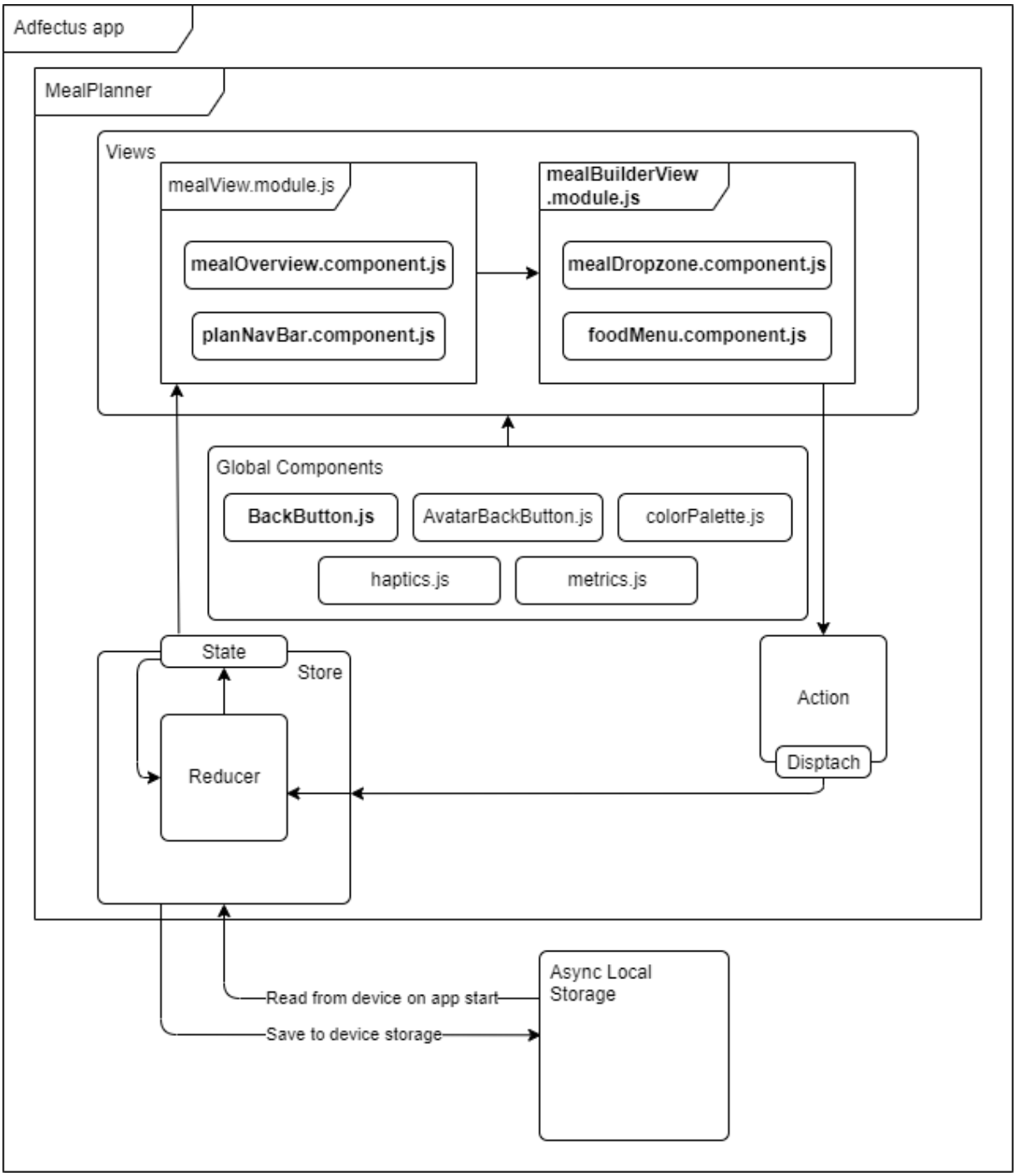
2 ARKITEKTUR

Oppbygningen av applikasjon er inspirert av den trelags-arkitektur. Øverst er presentasjonslaget med alle sidene som brukerne ser. I midten er logikklaget med gjenbrukbare komponenter og tilstandshåndtering, og på bunnen er notifikasjoner, navigering og datalagring. Det er brukt databaser for lagring av tilstandsdokumenter, men måltidsplanleggeren bruker ikke databasehåndtering utover dette.



Figur 1: Adfectus nåværende systemarkitektur. Felt markert i blått er en del av prosjektet.

Modulen som er laget i dette prosjektet er tilsvarende delt inn. I view-ene er det komponenter som utfører det meste av logikken til modulen, mens selve view-ene (.module.js) setter dem sammen til en visning for brukeren. Det er er brukt globale komponenter som både er laget av Adfectus-teamet før prosjektstart og gruppen underveis. Videre er Redux-biblioteket brukt for tilstandshåndtering, for å bevare endringer ved oppretting og endring av måltider. Når applikasjonen lukkes blir alt lagret lokalt på mobilenheten.



Figur 2: Arkitekturdiagram som viser flyten av data i modulen og hvordan komponentene er koblet sammen. Inspirasjon hentet fra: (Abramov, u.å., avsnitt 5).

3 PROSJEKTSTRUKTUR

Figurene under viser hvordan Adfectus-appen er strukturert. Filer som er i mapper som ikke er relevant for dette prosjektet er ikke vist. Det følger forklaringer til noen av mappene gruppen har jobbet med

```
AdfectusUser/
├ .expo/
├ .yarn/
├ .vscode/
├ ios/
├ android/
├ node_modules/
├ src/
│  ├ assets/
│  │  ├ images/
│  │  │  ├ meal-planner/
│  │  │  │  ├ drinks/
│  │  │  │  ├ foods/
│  │  │  │  ├ time-picker/
│  │  │  │  ├ apple-empty.png
│  │  │  │  ├ glass-empty.png
│  │  │  │  ├ overlay-dropzone.png
│  │  │  │  ├ plate-empty-old.png
│  │  │  │  └ plate-empty.png
│  │  ├ previews/
│  │  ├ puzzle/
│  │  └ sounds/
│  ├ core/
│  │  ├ async-storage/
│  │  ├ environment/
│  │  ├ notifications/
│  │  ├ router/
│  │  └ store/
│  ├ global/
│  │  ├ avatar/
│  │  ├ components/
│  │  ├ constants/
│  │  ├ mock-data/
│  │  │  ├ meal-plan/
│  │  │  │  └ meal-plan.template.js
│  │  │  ├ puzzle/
│  │  │  └ meal.mock.js
│  │  ├ reducers/
│  │  │  ├ active-you/
│  │  │  ├ avatar/
```

Figur 3: Mappe- og filstruktur (1/2)

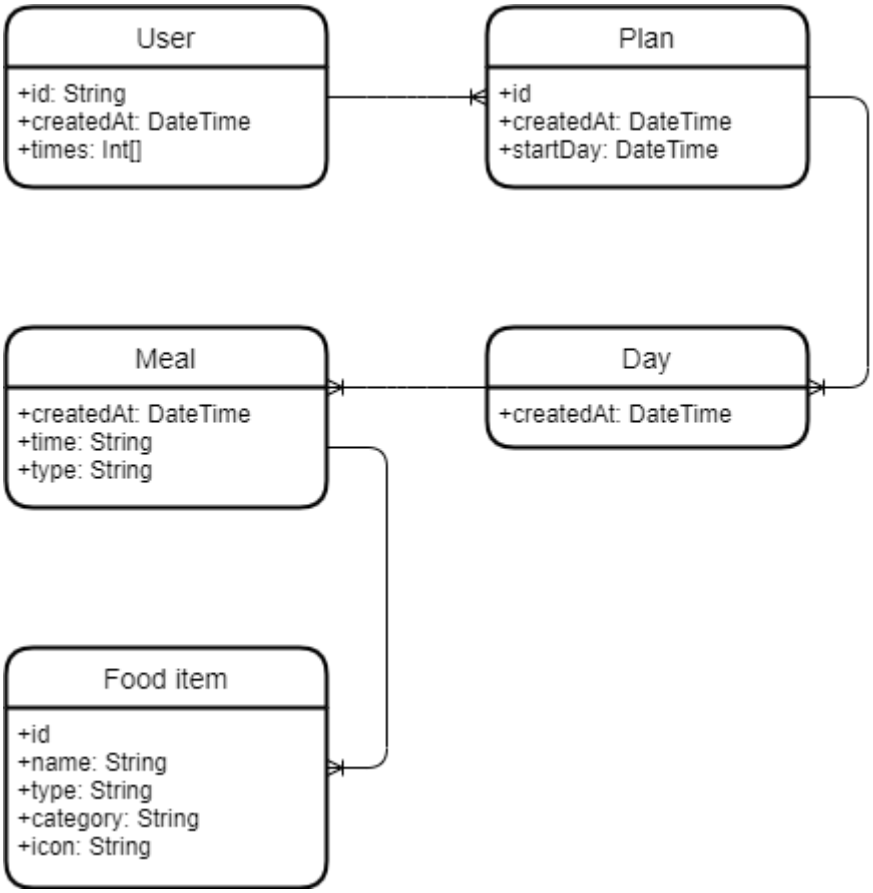
```
│  │  │  ├ closet/
│  │  │  ├ how-to/
│  │  │  ├ meal/
│  │  │  │  └ meal.slice.js
│  │  │  ├ packing/
│  │  │  └ puzzle/
│  │  └ services/
│  └ views/
│     ├ active-you/
│     ├ avatar-builder/
│     ├ door-sign/
│     ├ family-builder/
│     ├ how-to/
│     ├ info/
│     ├ intro/
│     ├ meal-planner/
│     │  ├ mealModule/
│     │  │  ├ dailyview.component.js
│     │  │  ├ food-menu.component.js
│     │  │  ├ mealCard.component.js
│     │  │  ├ mealDropzone.component.js
│     │  │  ├ planview.component.js
│     │  │  └ timePicker.component.js
│     │  ├ MealBuilder.module.js
│     │  ├ meal.module.js
│     │  └ mealModuleSettings.module.js
│     ├ packing/
│     ├ puzzle/
│     ├ user-input/
│     └ virtual-tour/
├ .gitignore
├ .node-version
├ 3rdPartyLibraries.txt
├ CHANGELOG.md
├ CODE-GUIDELINES.md
├ GIT-GUIDELINES.md
├ README.md
├ app.js
└ app.json
```

Figur 4: Mappe- filstruktur (2/2)

- **src/assets/images/meal-planner**: Inneholder png-filer av alle bildene for mat- og drikkevarer som brukes i Måltidsplanleggeren
- **src/core/environment**: Inneholder en environment.js fil med nøkkel til databasen. Den ligger i .gitignore for at den ikke skal være tilgjengelig for andre enn Adfectus teamet. Det medfører at prosjektet som blir levert ikke kan kjøres.
  * Dette prosjektet forholder seg kun til databasen gjennom Redux, så dette dokumentet inneholder ikke et kapittel for databasemodell.
- **src/global/mock-data**: Inneholder data for matobjektene, skisse for oppretting av ny plan og et plan-objekt med delvis utfylt data som ble brukt til testing av modulen.
- **src/global/reducers/meal/meal.slice.js**: Inneholder Redux-konfigurering og funksjoner for manipulering av planer.
- **src/views/meal-planner**: Inneholder det meste av arbeidet gruppen har gjort. Her er alle view-ene, samt ikke-globale komponenter til view-ene.

4 KLASSEDIAGRAM

Datastrukturen er i JSON-format, og derfor er entitetene under ikke typiske klasser. Klassediagrammet under er en tilnærmet representasjon. Hvert objekt inneholder en liste av neste objektet. En bruker kan ha mange planer, der planene representerer uker. Hver plan har syv dager (mandag til søndag). Hver dag har fire måltider (frokost, lunsj, middag og kveldsmat). Hvert måltid består av opptil fire matgjenstander (en drikke, en frukt eller snacks og to større objekter, for eksempel skiver og frokostblanding).



Figur 5: Klassediagram som viser strukturen for dataen i Måltidsplanleggeren

Forklaringer for ikke-åpenbare attributter:

- *User, times*: Er de fire tidspunktene i løpet av en dag brukeren skal spise. Ved innlevering av dette prosjektet er det ikke lagt inn funksjonalitet for å sette dette i appen, og den er dermed ikke i bruk som planlagt.
- *Plan, startDay*: Er datoen for mandagen den uken planen er ment for. I Dagsoversikten vises én uke av gangen, og startDay brukes for å sette første dag.
- *Meal, time*: Er tidspunktet for måltidet.
- *Food item, type*: Frokost, lunsj, middag eller kveldsmat
- *Food item, category*: Brukes til å bedømme hvor sunn maten er. Denne funksjonaliteten er en videreutvikling og ikke lagt inn enda, så foreløpig er den alltid en tom streng.
- *Food item, icon*: Adressen til en bilderepresentasjon av matobjektet.

5 SIKKERHET

Dette prosjektet inneholder ingen brukerinput i form av tekstfelt, ellers måtte det tas høyde for SQL-injections. Det eneste gruppen måtte passe på var at URL-en og nøkkelen til databasen ikke havner på GitHub.

I planleggingen av prosjektet var det tenkt at administratorsiden til Adfectus-appen skulle utvides med funksjonalitet for Måltidsplanleggeren om det var tilstrekkelig med tid. Dette ble aldri påbegynt, men i så fall måtte det sikres at innloggingspassord var hashet og lagret på en sikker måte.

6 INSTALLASJON OG KJØRING

For å kjøre programmet kreves det noen programmer og pakker som er gratis tilgjengelig på nettet.

Installasjonsrekkefølge:

1. Installer **Visual Studio Code** (andre tekstredigeringsverktøy og IDE-er fungerer også, men de må kunne brukes for å skrive, og eventuelt kjøre, React Native kode)
    a. Last ned fra: https://code.visualstudio.com/download
    b. Kjør klienten og følg instruksjonene

2. Installer **Git**
    a. Last ned fra: https://git-scm.com/downloads
    b. Følg instruksjonene ut ifra hvilket operativsystem den skal kjøres på
    c. Sjekk at installasjonen virket med kommandoen: `*git -version*`

3. Hent prosjektet fra **GitHub**
    a. Start et terminalvindu
    b. Naviger til ønsket mappe for plassering av prosjektet
    c. Kjør kommandoen: *git clone https://github.com/adfectus-as/AdfectusUser.git*

4. Installer **Node.js** (eller Yarn.js)
    a. Last ned fra: https://nodejs.org/en/download

5. Installer **Expo CLI**
    a. Kjør kommandoen: *npm install –global expo-cli*

6. Installer **Expo Go** på mobil eller nettbrett
    a. Android: https://play.google.com/store/apps/details?id=host.exp.exponent&pli=1
    b. IOS: https://apps.apple.com/app/expo-go/id982107779

7. Installer alle pakkene og avhengighetene i prosjektet:
    a. Kjør kommandoen: *npm install*

For å kjøre programmet Visual Studio Code:

1. Åpne prosjektet
    a. Start Visual Studio Code
    b. Trykk «File», «Open Folder» og velg plasseringen til prosjektet

2. Om terminalen ikke vises, trykk «Terminal» og «New Terminal»

3. I terminalen, kjør kommandoen: *npx expo start*
    a. Koden vil kjøres og det vil vises en QR-kode i terminalen

4. Start Expo Go appen på mobilen eller nettbrettet

5. Trykk «Scan QR code (eller «Enter URL manually»)

6. Bruk kamera for å skanne koden

7. Nettleseren på enheten vil åpnes med URL-en til programmet
    a. Herfra trykk «Expo Go»

8. Programmet blir nå bygget på enheten, og om vellykket skal appen kunne brukes

7 DOKUMENTASJON AV KILDEKODE

Kodedokumentasjon er en viktig del av alle utviklingsprosjekter, og den inneholder blant annet kodekommentarer og egne filer som forklarer hvordan kode skal kjøres og hva den gjør. Ulike bedrifter gjør det på forskjellige måter, men det er generelle retningslinjer for hvordan det skal gjøres. Tydelig dokumentasjon hjelper utviklere og dem som skal bruke koden til å øke forståelsen, og dermed minimere tidsbruken på å lese og forstå koden. (Svim, 2023).

I Adfectus er det brukt flere typer dokumentasjon:

1. **Readme filer**:
    - Forklarer hvordan prosjektet skal brukes og utformes.
    - **README.md**: Inneholder generelle retningslinjer for hvordan starte programmet, hvordan mapper og filer skal struktureres og navngis, og beskrivelse av ulike filtyper.
    - **GIT-GUIDELINES.md**: Forklarer hvordan Git skal brukes
    - **CODE-GUIDELINES.md**: Beskriver hvordan koden skal utformes

2. **Git commit meldinger**:
    - Det skal skrives tydelige meldinger om hva som endres eller legges til hver gang kode blir pushet til Git.

3. **JSDoc**:
    - Et forslag for å standardisere dokumentasjon for JavaScript. De forklarer hva funksjoner gjør, forventet inputverdi og -type, og forventet returverdi og -type.
    - I dette prosjektet er de brukt til å kommentere filer for Redux (et bibliotek for tilstandshåndtering), vist under

```
/**
* Creates a new plan/week,
* and adds it to the list of plans.
*
* @param {Object} state - The current state object.
* @param {object} action - The action object containing the payload
*                          to update the meal plan.
* @param {number} action.payload.id - The new ID of the meal plan to create.
* @returns {void}
*/
createPlan: (state, action) => {
    const indexOfPlan = state.plans
                            .findIndex(plan => plan.id === action.payload.planId)
    if (indexOfPlan !== -1) {
        console.log('Plan already exists');
        return;
    }
    const newWeek = newMealPlan(action.payload.planId);

    const lastplan = state.plans[0];
    const lastDay = state.plans[state.plans.length - 1].days[6];

    state.plans.push(newWeek);
},
```

Figur 6: Eksempel fra prosjektet for JSDoc

4. **Kodekommentarer**:
    - Det skal legges inn kommentarer der det ikke er selvforklarende hva som skjer

```
const addItems = (item) => {
    if (chosenItems.some(i => i.id === item.id)) return; // prevent duplicates

    setChosenItems( () => {
        return (
            chosenItems
            ? [...chosenItems, item]
            : [item]
        )
    })
};
```

Figur 7: Eksempel som viser en kommentar i prosjektet

5. **Selvforklarende kode**:
    - Selv med dokumentasjon, skal koden skrives mest mulig selvforklarende og lettlest. Det vil for eksempel si å bruke beskrivende variabelnavn og å unngå veldig komplekse og kompakte kodelinjer.
    - Eksempel på kode som ble endret for å gjøre den mer leselig:

```
const addItems = (item) => {
    if (chosenItems.some(i => i.id === item.id)) return; // prevent duplicates

    setChosenItems([...(chosenItems || []), item]);
};
```

Figur 8: Kodeblokk før endring

```
const addItems = (item) => {
    if (chosenItems.some(i => i.id === item.id)) return; // prevent duplicates

    setChosenItems( () => {
        return (
            chosenItems
            ? [...chosenItems, item]
            : [item]
        )
    })
};
```

Figur 9: Kodeblokk etter endring

8 TESTING

Enhetstesting er innført gjennom `console.log`-funksjonen. Grunnlaget for å ikke lage automatiserte enhetstester, er at dette er svært tidkrevende i React Native-miljøet, og garanterer heller ikke riktig testutførelse.

9 REFERANSER

Svim (2023) *Code documentation: benefits, challenges, and tips for success*. Tilgjengelig fra: https://swimm.io/learn/code-documentation/code-documentation-benefits-challenges-and-tips-for-success/ (Hentet: 21. mai 2023).