Indholdsfortegnelse

1 Design

Dette afsnit omhandler design delen af Sample Project 2024 hvor der vises skærmbilleder af forskellige dele af applikationen og reflektioner på udvalgte designvalg.

1.1 Den første oplevelse

Ved den første oplevelse af Sample Project 2024 mødes brugeren af en simpel start side der giver brugeren mulighed for enten at logge ind med en allerede eksisterende brugerprofil eller oprette en ny.

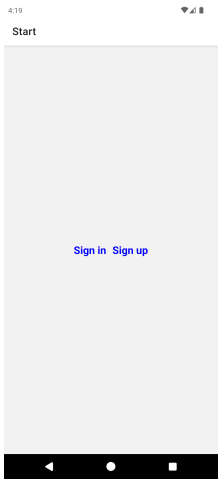

Figure 1: Start

1.2 Opret ny bruger/login

Ved oprettelse af ny bruger skal brugeren registrere sig med email og et selvvalgt kodeord. Der benyttes maskering i forhold til indtastning af kodeord og duplikat indtastning.

Ved login indtastes email og kodeord. Ligesom ved oprettelse af bruger er kodeordet maskeret af sikkerheds hensyn.

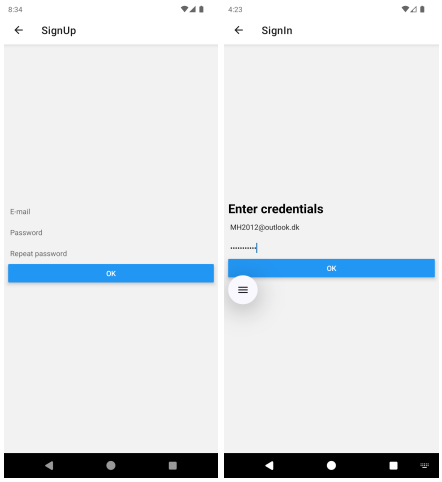

Figure 2: Opret/login

1.3 Hovedsiden

Ved fuldendt oprettelse eller login føres brugeren til applikationens centrale side hvor der er mulighed for at tilgå forskellige dele af applikationen. Selve hovedsiden er bygget op af fliser med titel og ikon for at give et hint om, hvad den enkelte flise gør ved interaktion.

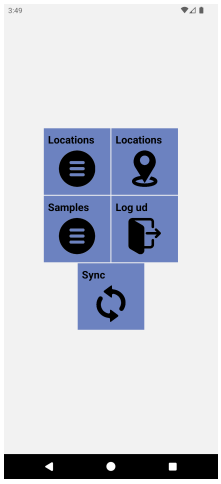

Figure 3: Hovedside

Ikoner kan udover at bidrage til det visuelle udtryk også bidrage til at højne brugeroplevelsen. Dette sætter dog krav til valg af ikoner som helst bør være

dækkende i forhold til flisens formål. Et eksempel kan være det populære *nåle ikon* som benyttes i diverse kort applikationer såsom Google Maps. Dette ikon efterlader meget lidt tvivl om kontekst og formål.

1.4 Visning af prøve lokationer

Brugeren vil ved interaktion med en af *locations* fliserne blive ført til en oversigt over prøve lokationer som kan benyttes til at oprette en ny prøve eller se en detaljeret visning over selvvalgt lokation.

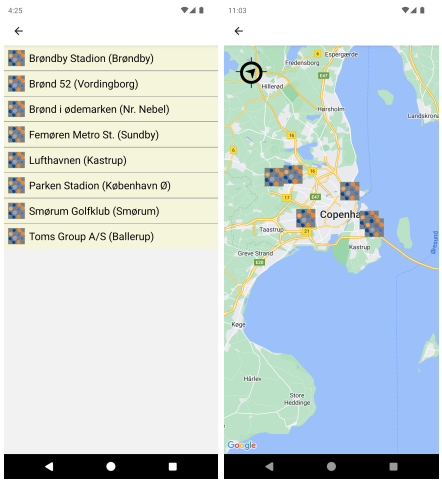

Figure 4: Lokationer

Hvert enkelt element er klikbart og fører brugeren til en side der giver en detaljeret visning af den valgte lokation.

Figure 5: Detaljeret visning af prøve lokation

Fra denne side kan brugeren vælge at interagere med den grønne knap og oprette en ny prøve tilknyttet denne lokation.

1.5 Oprettelse af prøver

Ved oprettelse af prøver er det obligatorisk at vælge lokation og prøve værdi samt type[1]. Der kan tilknyttes dokumentation i form af billeder ved interation med billed ikonerne i den øverste bjælke. Der kan tages billeder ved brug af kamera eller vælges fra enhedens fysiske lager.

[1] Der er ikke implementeret noget tjek på dette i nuværende version

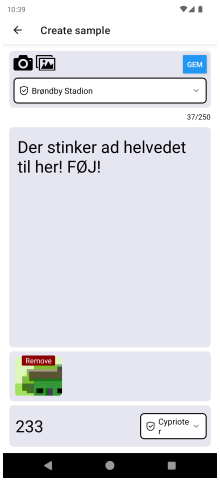

Figure 6: Oprettelse af prøve

2 Implementation

2.1 Opret prøve

Ved oprettelse af prøver bruges React hooks som *useState* til at gemme de indtastede værdier. Dette sikrer at værdierne ikke går tabt ved gentegning af brugerfladen.

```
1     const [loading, setLoading] = useState(false)
2     const [note, setNote] = useState("")
3     const [location, setLocation] = useState(route?.params?.
          location ?? "")
4     const [sampleType, setSampleType] = useState({})
5     const [sampleValue, setSampleValue] = useState(0)
6     const [images, setImages] = useState([])
7     const [count, setCount] = useState(0)
```

Når brugeren har indtastet sine værdier og interageret med gem knappen køres denne kode:

```
1     async function handleSaveClicked() {
2       if (!location)
3         return
4       setLoading(true)
5       const urls = await Storage.uploadObjects(images)
6       if (await Samples.save(createSample(urls)))
7         navigation.goBack()
8       setLoading(false)
9     }
```

Som det fremgår i linje 2 er det obligatorisk at vælge lokation.

Det næste er upload af valgte billeder til Firebase Storage. Dette ansvar har *uploadObjects()* funktionen i filen *StorageFirebase.js*:

```
async uploadObjects(dataObjects) {
  const blobs = await toStorageBlobs(dataObjects)
  const objectRefs = []
  let objectRef
  for (let i = 0; i < blobs.length; i++) {
    const blob = blobs[i]
    objectRef = await upload(blob)
    objectRefs.push(objectRef)
  }
  return objectRefs
}
```

Uddrag 1: Upload objekter

Denne funktion starter ud med at lave billederne om til såkaldte *blobs* som er et filformat specielt designet til repræsentation af multimedia filer:

```
async function toStorageBlobs(data) {
  let storageBlobs = []
  for (let i = 0; i < data.length; i++) {
    const blob = await asBlob(data[i])
    const storageBlob = { data: blob, path: createPath() }
    storageBlobs.push(storageBlob)
  }
  return storageBlobs
}
```

Uddrag 2: Upload blob

Den næste interessante funktion er *upload* som sørger for at uploade et blob til Firebase Storage:

```
async function upload(object) {
  const storageRef = getStorageRef(object.path)
  if (!storageRef)
    return undefined
  const imageUri = await uploadFile(storageRef, object.data)
  return { imageId: object.path, uri: imageUri }
}
```

2.2 Kort visning af lokationer

Lokationer bliver vist på et kort ved hjælp af *MapView* biblioteket:

```
<View style={styles.container}>
  <MapView
    region={mapCords}
    style={styles.map}>
    {locations.map(loc => (<MapMarker onPressed={toDetails} key
        ={loc.id} item={loc}   coordinates={loc.location}/>))}
  </MapView>
  <IconButton
    style={styles.posWrapper}
    width={64}
```

```
        height={64}
        uri={require("../assets/cPos.png")}
        onPress={goToCurrentLocation}
      />
    </View>
```

2.3 Services

Lokationer og prøver eksponeres gennem et service lag der håndterer forbindelsen mellem Firebase og brugerfladet. For at reducere indlæsningstider indlæses alt data i starten og gemmes i et array.

I følgende uddrag tages der udgangspunkt i prøver:

```
    async fetch() {
      if (fetchingRequired) {
        samples = await PersistenceProvider.fetchData()
        fetchingRequired = false
      }
    }
```

Der holdes styr på hvorvidt applikationen har indlæst data ved at sætte variablen *fetchingRequired* til *false* ved succesfuldt indlæsning. Data gemmes i et array så der ikke skal indlæses fra Firebase ved hvert kald.Der kan også indlæses en enkelt prøve ud fra et id:

```
    getById(id) {
      return samples.find(s => s.id == id)
    }
```