Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap

Institutt for datateknologi, elektroteknologi og realfag

Library of Things – En fullstack applikasjon for utlån av utstyr

Systemdokumentasjon

Versjon 3.0

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
02/02/2024	1.0	La til klassediagram og databasemodell	Jan-Petter Dåvøy
01/05/2024	2.0	La til API endepunkter	Fredrik Enes
06/05/2024	2.1	Endringer i api forklaring	Jan-Petter Dåvøy
12/05/2024	3.0	Lagt til README-fil	Fredrik Enes

Innhold

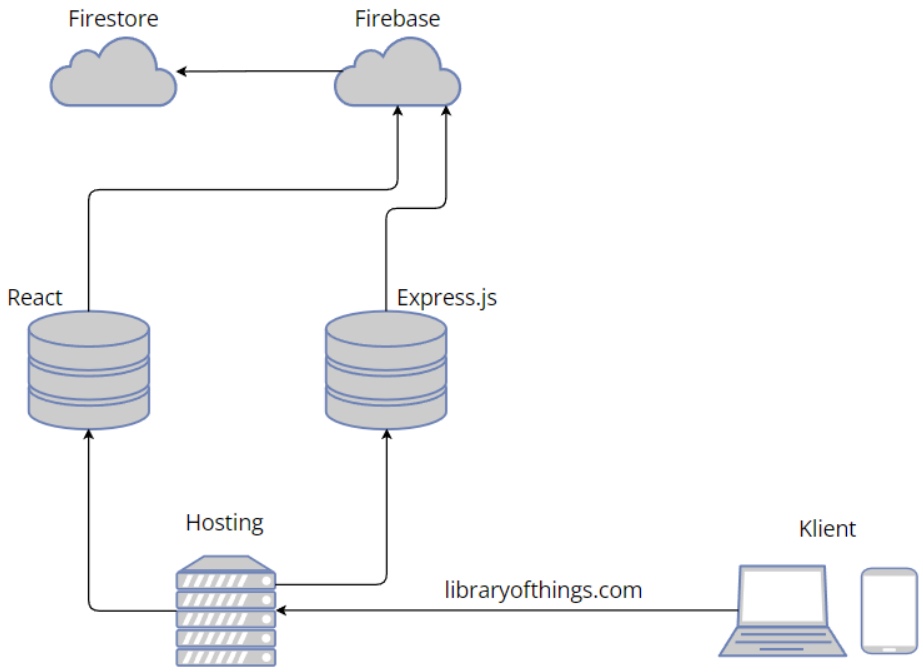
Figurliste

1 INNLEDNING

Hensikten med dette dokumentet er å få en oversikt over hvordan applikasjonen er strukturert. Dokumentet beskriver komponentene den er sammensatt av, databaseløsningen, og sikkerhetstiltakene nødvendig for å skape et akseptabelt sluttprodukt. Systemdokumentasjonen inneholder nødvendige instruksjoner for installasjon og kjøring av applikasjonen. Avslutningsvis beskrives dokumentasjon av kildekoden og testplanen for applikasjonen.

2 ARKITEKTUR

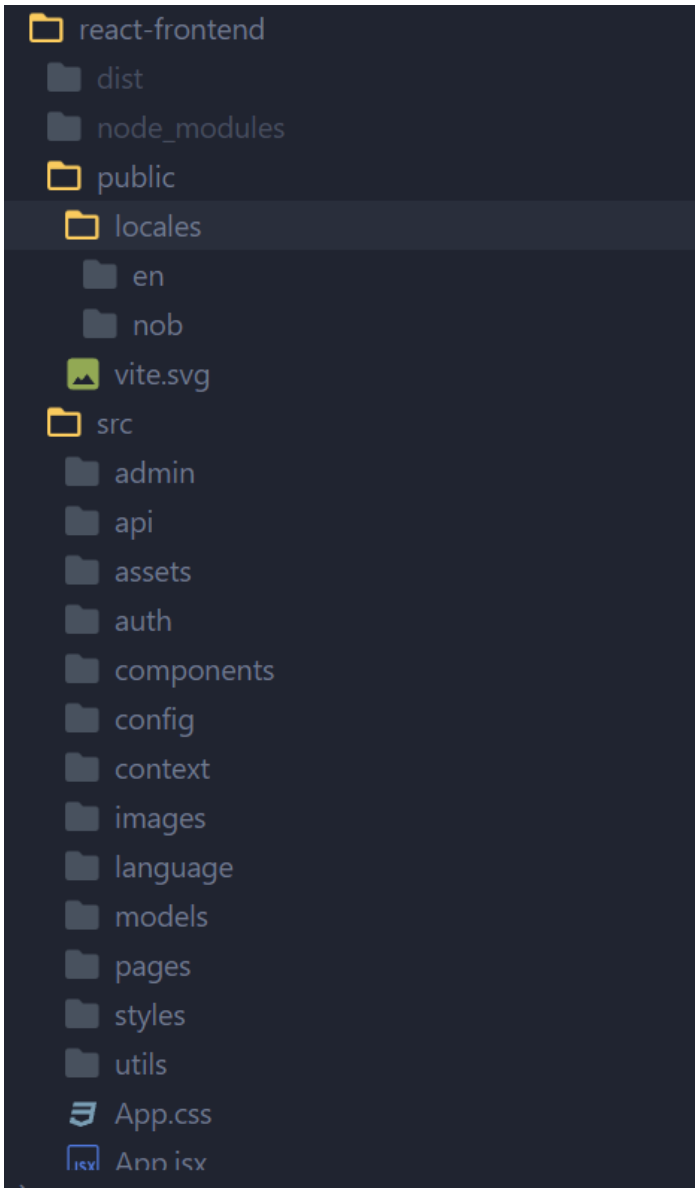
Arkitekturdiagrammet gir en oversikt over applikasjonens arkitektur. Klienten kobler seg opp mot en tjener (hosting) som gjør visuell fremvisning tilgjengelig for bruker gjennom frontend-løsningen. Tjeneren styrer også kommunikasjonen mot Express.js [1] hvor autentisering, kryptering, dekryptering og e-post tjenesten ligger. For tilgang til databaseløsningen i Firestore [2] benytter man Firebase [3] sine forhåndsdefinerte funksjoner. Skissen i figur 2.1 er laget i Visual Paradigm [4].



Figur 2.1: Arkitekturskisse for applikasjon. Skjermdump hentet fra: Eget prosjekt i Visual Paradigm.

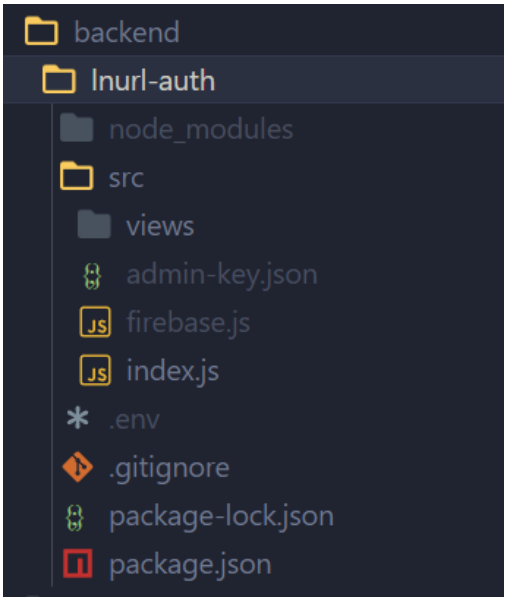
3 PROSJEKTSTRUKTUR

Dette kapittelet viser strukturen i kildekoden. Kildekoden er skrevet i Visual Studio Code [5] som ble benyttet av gruppemedlemmene til utviklingen. React-frontend prosjektmappen er den som inneholder store deler av kildekoden. De fleste interaksjonene med Firebase skjer direkte fra kildekoden i denne mappen og dette sikrer god ytelse uten flere mellomledd.



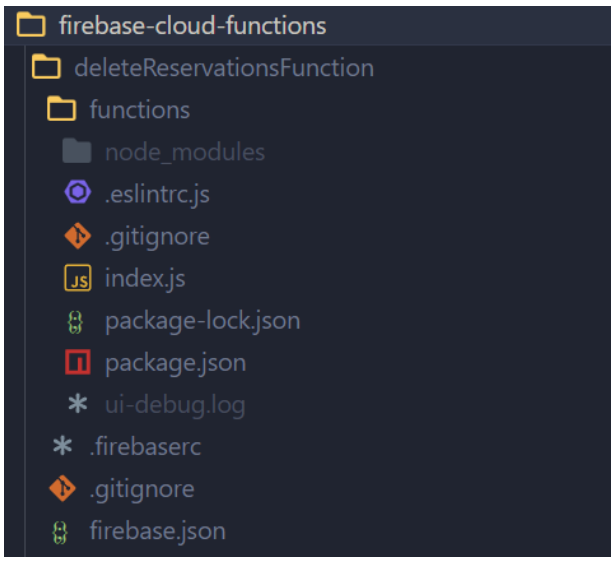
Figur 3.1: Frontend prosjektmappe. Skjermdump hentet fra: Egen kildekode.

I backend prosjektmappen ligger lnurl-auth mappen. Denne mappen inneholder kildekoden for autentiseringen, kryptering, dekryptering og e-post tjenesten.



Figur 3.2: Backend prosjektmappe. Skjermdump hentet fra: Egen kildekode.

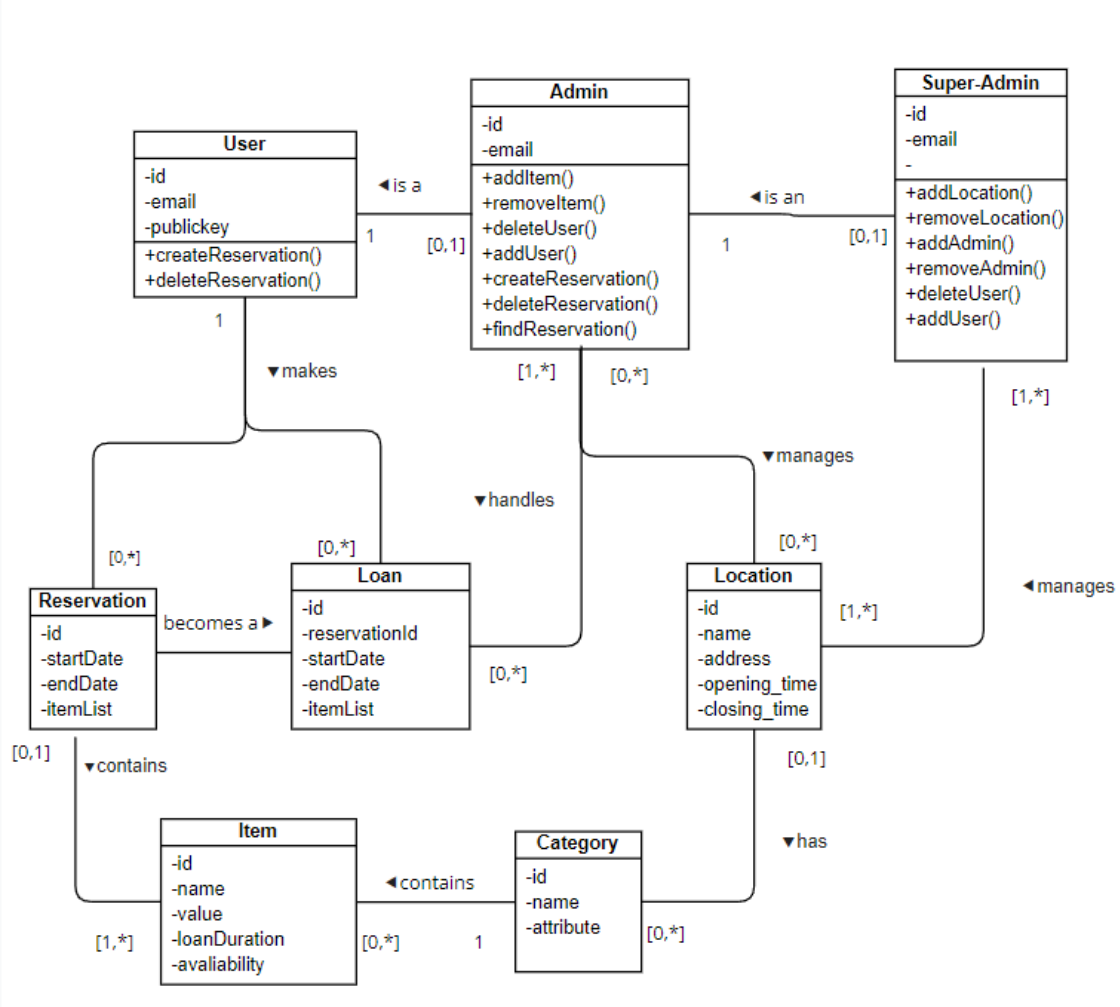
Cloud Scheduler [6] har ansvaret for å kjøre kildekoden i prosjektmappen *Cloud function*. Kildekoden i prosjektmappen brukes til å implementere funksjonen for å slette reservasjoner som har vært aktiv i over 2 timer.



Figur 3.3: Cloud Function prosjektmappe. Skjermdump hentet fra: Egen kildekode.

4 KLASSEDIAGRAM

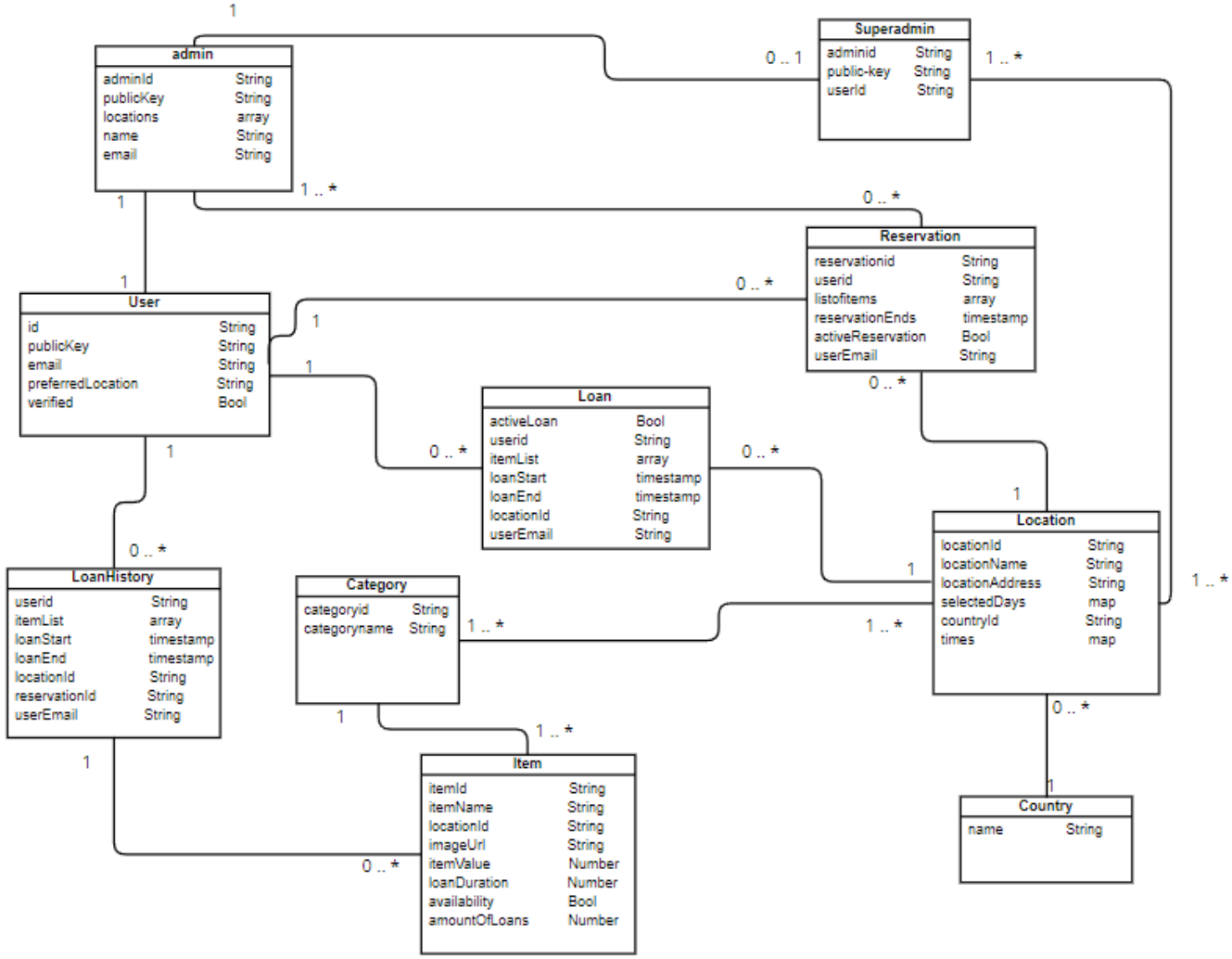
Klassediagrammet vil representer hvordan klassene i applikasjonen skal utformes, og hvordan de interagerer. Klassediagrammet er utformet ved hjelp av et UML(Unified modeling language)-diagram, og inkluderer de tenkte forholdene og multiplisitetene til klassene.



Figur 4.1: Første iterasjon av klassediagrammet. Skjermdump hentet fra: Eget prosjekt i Visual Paradigm.

5 DATABASEMODELL

Databasemodellen i figur 5.1 viser en overordnet modell over systemets database. Relasjonene er tatt med mellom de forskjellige kolleksjonene for å vise hvilket forhold disse har, og multiplisiteten mellom kolleksjonene er indikert med tall, eller * for å indikere «mange-forhold».



Figur 5.1: Databasemodell. Skjermdump hentet fra: Eget prosjekt i Visual Paradigm.

6 SERVER-TJENESTER

Dette kapittelet tar for seg løsningens server-tjenester, hvor Firebase fungerer som hoved backend-løsning.

Firebase er blitt benyttet som hoved backend-løsning og er en BaaS (Backend-as-a-Service)-tjeneste. Dette betyr at utviklerne kan dra nytte av forhåndsdefinerte funksjoner og tjenester uten å sette opp eller vedlikeholde sin egen backend-infrastruktur. Tjenesten blir gjort tilgjengelig i applikasjonen ved å legge inn en unik konfigurasjonsfil.

I tillegg til å benytte Firebase, er REST-ressursene laget ved hjelp av Express.js rammeverket.

REST-ressursene er blitt definert som følger:

- «/api»

  Hoved stien (main path) til APIet, inneholder en lenke som sender bruker videre til login.

- «/api/login»

  Viser en QR-kode som brukeren skanner med sin Lightning Wallet for å autentisere seg.

- «/api/user/

  Returnerer den offentlige nøkkelen til brukeren sin Lightning Wallet.

- «/api/logout/»

  Logger ut brukeren fra sesjonen sin på applikasjonen.

- «/api/send-loan-confirmation»
  Sender en bekreftelses e-post til brukeren når et lån har blitt opprettet.

- «/api/send-reservation-confirmation»
  Sender en bekreftelses e-post til brukeren når en reservasjon har blitt opprettet.

- «/api/send-email-verification»
  Sender en bekreftelses e-posst til brukeren, denne inneholder en lenke som brukeren må klikke på for å verifisere brukeren sin.

- «/api/email-verification/:id/:token »

  Mottar en http-request som skal inneholde en id og en token. Hvis begge parameterne er tilstede blir eposten til brukeren validert. Hvis ikke så blir feilmeldinger om hvilke parameter som ikke er tilstede gitt.

- «/api/get-email-token»

  Oppretter en unik kryptert token som brukes til å validere bruker-eposten.

- «/api/encrypt»

  Returnerer en kryptert streng fra en tekst input. Krypterer med «aes-256-cbc» kryptering.

- «/api/decrypt»

  Tar inn en kryptert streng og dekrypterer den. Returnerer den dekrypterte strengen.

7 SIKKERHET

Dette avsnittet går gjennom sikkerheten i applikasjonen og hvilke sikkerhetstiltak som er tatt underveis.

7.1 Sikring av QR-koder

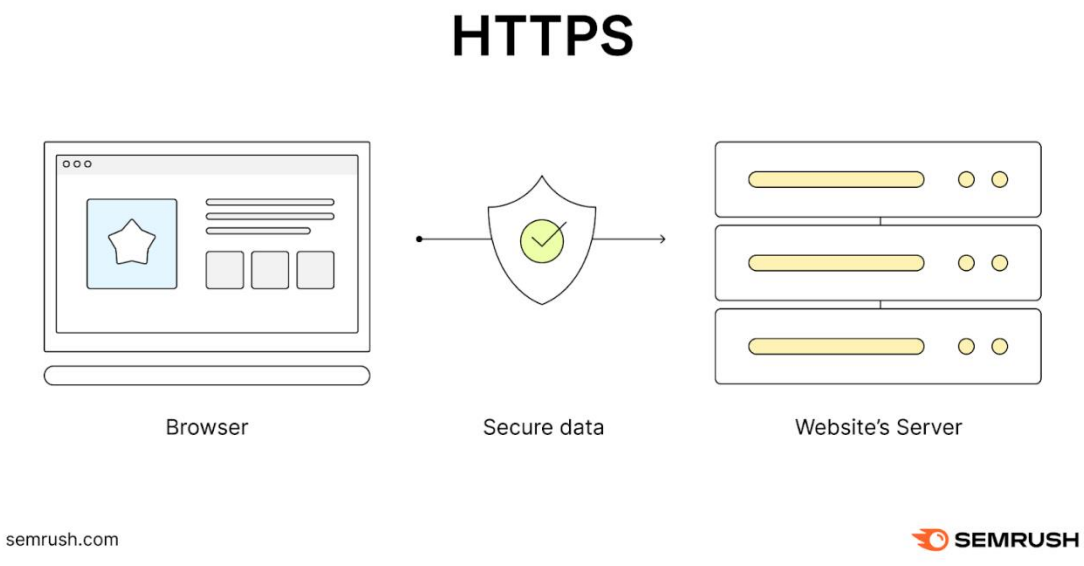
QR-koder benyttes i applikasjonen for å gi hurtigtilgang ved utlån og innhenting av lån. For å sikre oss mot at QR-koden ikke fremviser en sti i klartekst til bruker, er denne blitt kryptert med en «aes-256-cbc» algoritme. Kun en administrator sin QR-leser kan dekryptere QR-koden.

7.2 Mellomlagring

Mellomlagring (cache) av data i applikasjonen benyttes for å redusere antall spørringer som sendes til Firebase og Firestore. For å mellomlagre data benyttes secure local storage [7]. Secure local storage gjør krypteringen og dekryptering av Local storage uten at det er nødvendig å implementere ekstra funksjonalitet i applikasjonen.

7.3 HTTPS-protokollen

Dataoverføringen over HyperText Transfer Protocol Secure (HTTPS) [8] er en protokoll hvor all data som sendes og mottas er kryptert. Dette forhindrer at uautoriserte personer kan lese eller manipulerer dataen som sendes gjennom protokollen. All interaksjon med prosjektets RESTful API foregår over HTTPS-protokollen.



Figur 7.1: Hvordan HTTPS fungerer. Hentet fra: [9]

7.4 Versjonskontroll

For å sikre at kildekoden ikke inneholder informasjon som avdekker konfigurasjoner i applikasjonen er det blitt benyttet en .gitignore fil. Dersom konfigurasjonsfiler blir lagt inn i versjonskontrollsystemet vil de bli liggende i historikken og være åpen for de med tilgang til prosjektet. Hvis dette skjer, må nye konfigurasjonsfiler genereres eller Github-prosjektet [10] må slettes og lages på nytt. Gruppen har lagt inn sikkerhetstiltak som bidrar til at kildekoden til prosjektet kan være offentlig, dette innebærte å bruke .gitignore filen til å skjule sensitive filer som .env og andre konfigurasjonsfiler.

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?

firebase.js
```

Figur 7.2: Bilde av .gitignore fil. Skjermdump hentet fra: Egen kildekode.

7.5 Begrensing av tilgang i applikasjonen

Deler av applikasjonen er låst dersom man ikke er autentisert. For å unngå at besøkende av applikasjonen kan hente ut hvilke gjenstander som er på lokasjon er det satt opp funksjonalitet som forhindrer dette. En besøker vil bli stoppet etter å ha valgt en kategori som de vil se gjenstander fra. Dette er gjort ved å sjekke om besøkeren har mellomlagret data eller er logget inn på tjenesten med sin offentlige nøkkel.

8 INSTALLASJON OG KJØRING

Det er ønskelig for gruppen å kjøre applikasjonen gjennom en tjener. Dersom prosjektet skal kjøres lokalt og ikke via tjeneren, så må kildekoden hentes ut fra Github og «npm install» kommandoen må kjøres for å få installere pakkene som kreves i programmet. Stiene som benyttes for å peke til andre sider i applikasjonen må også oppdateres til å peke på «localhost» stier.

Applikasjonen kjøres ved å navigere inn i «react-frontend» mappen og kjøre kommandoen «npm install» etterfulgt av «npm run dev». For å kjøre backend-løsningen må det navigeres inn i lnurl-auth mappen og kjøre kommandoen «npm install» etterfulgt av «npm start».

Dersom det er ønskelig å legge til mer funksjonalitet til en Cloud Function så kan dette gjøres under «firebase-cloud-functions». Når ønsket funksjonalitet er lagt inn så benyttes kommandoen «firebase deploy --only functions», dette forutsetter at alle pakker er installert og nødvendige konfigurasjoner er lagt inn.

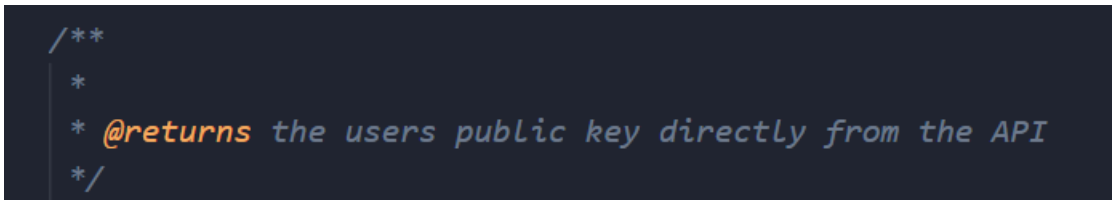
Tilgang til tjenester som kryptering, dekryptering og kommunikasjon med Firebase krever at konfigurasjonsfilene er lagt inn. Disse konfigurasjonsfilene er ikke tatt med under eksportering av prosjektet fordi de ikke skal overføres i klartekst.

9 DOKUMENTASJON AV KILDEKODE

Dette kapittelet inneholder hvordan kildekoden er dokumentert. Gruppen har benyttet JSDoc over funksjoner og en README-fil med informasjon om oversettelser, dokumentasjon og videre arbeid.

9.1 JSDoc

JSDoc [11] er brukt for å dokumentere kildekoden, det gjør det mulig å generere sider for å samle applikasjonens JSDoc i et dokument. Målet er å gjøre koden lettere å vedlikeholde for de som skal videreutvikle applikasjonen. JSDoc strengene er plassert over funksjonene, og gir en klar og strukturert oversikt over funksjonens formål.



Figur 9.1: JSDoc med musepeker over funksjon. Skjermdump hentet fra: Egen kildekode.

Figur 9.2: Fremvisning av JSDoc når musepekeren er på funksjonsnavnet. Skjermdump hentet fra: Egen kildekode

9.2 README.md

README-filen er dokumentert med hvordan nye språk legges til applikasjonen, hvilke dokumentering som er brukt og videre arbeid i prosjektet.

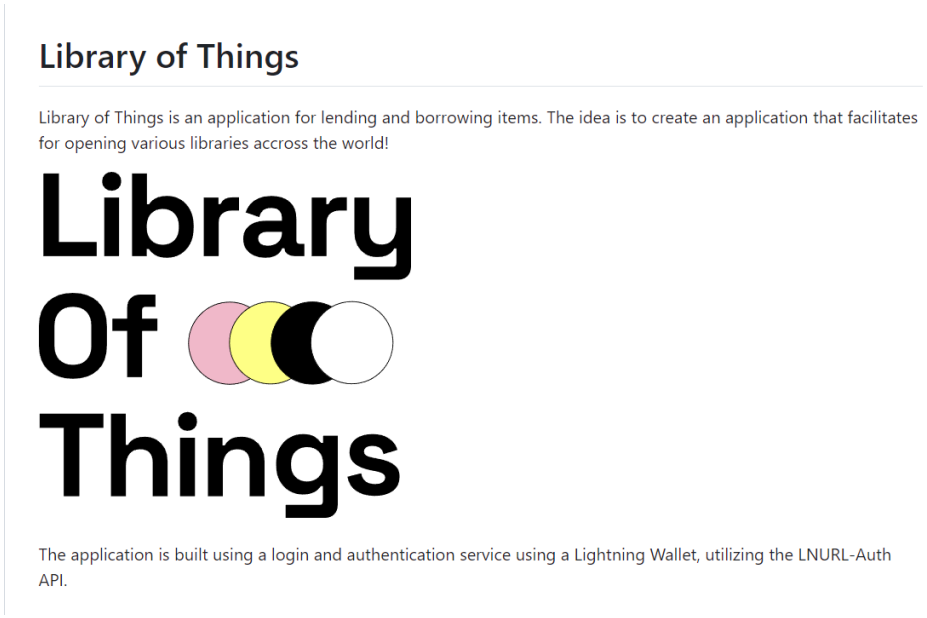


Table of contents

1 Introduction

This README file is meant to act as documentation for the project. It will contain instructions of how the code is structured, its documentation, and relevant guides on how to modify certain parts of the codebase.

2 Internationalization

One of the core ideas behind the project is that there should be a Library of Things avaliable to everyone, regardless of the language they speak, or their geographic location.

In order to facilitate for this avaliability, the project utilizes react-i18next in order to provide translations to different languages.

2.1 Adding new languages

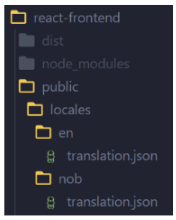


Figure 2.1: Translation folder structure
Figure 2.1 illustrates the folder strucure pertaining the access of language files. In order to add a new language, you first have to add a new folder to the react-frontend/public/locales/ folder. The new folder should correspond to the correct language code. After you have added the new folder, you need to create a translation.json file in the folder. That is, in: `react-frontent/public/locales/languagecode/translation.json`.

If you are currently in the `react-frontend` folder, you can run the command: `mkdir /public/locales/languagecode/` and then you can create a new `translation.json` file.

Figure 2.2: Translation folder-structure and example of translation.json files
Figure 2.2 illustrates how the translation.json file is structured. The file consists of key-value pairs, where the key indicates the page / component, and the value represents the intended place to insert the translation.

New translation files should follow the same structure as this file.

```
import { useTranslation } from 'react-i18next';
const { t } = useTranslation();
<div className="home-header">
    <h1>{t('welcome-message.header')}</h1>
    <img src={LOTLogo} className="home-logo"
        alt="image of the library of things logo" />
    <p className="home-welcome-message">{t('welcome-message.body')}</p>
</div>
```

Figure 2.3: Example of how useTranslation is used in the application
Figure 2.3 illustrates how react-i18next is used in the application, and can serve as an example of how you can use it.

2.2 Improvements to the existing code

There are two specific things which should be looked at to improve the functionality of the translations:

1. Fix naming convention
2. Automate adding new languages to language converter.

3. Documentation

The primary source of documentation in this project is done by using JSDoc. The idea, was to utilize docstrings as a way to increase the useability, and readability of the code during the project.

```
/**
 * This method checks if a user by a given publickey is an administrator or not.
 * @param {*} publicKey the publickey of the user
 * @returns true if user is admin, false otherwise
 */
```

Figure 3.1: Image displaying an example of how docstrings are used in the application
When adding additional functionality, and code to this project each global, or otherwise unclear function must include docstrings. Figure 3.1 gives an example of how this can be done.

4. Further improvements to the project

The project has utilized the inbuilt kanban-board that GitHub supports. The tasks listed in this GitHub project, can and should act as a guide for further improvements.

However, if by chance the board is not actively being updated here are some suggestions for functionalities that could be added:

1. NFC-tag functionality
2. Functionality to automatically add and remove languages
3. Implement payback funtionality using LNURLs-API
4. Optimizing the code:
    - Optimizing retrieval of images from firestore
    - Adding pagination when retrieving from the database
    - Optimizing database queries

10 REFERANSER

[1] «Express - Node.js web application framework». Åpnet: 25. februar 2024. [Online]. Tilgjengelig på: https://expressjs.com/

[2] «Firestore: NoSQL document database», Google Cloud. Åpnet: 6. mars 2024. [Online]. Tilgjengelig på: https://cloud.google.com/firestore/

[3] «Firebase | Google's Mobile and Web App Development Platform», Firebase. Åpnet: 2. mai 2024. [Online]. Tilgjengelig på: https://firebase.google.com/

[4] «Ideal Modeling & Diagramming Tool for Agile Team Collaboration». Åpnet: 15. april 2024. [Online]. Tilgjengelig på: https://www.visual-paradigm.com/

[5] «Visual Studio Code - Code Editing. Redefined». Åpnet: 2. mai 2024. [Online]. Tilgjengelig på: https://code.visualstudio.com/

[6] «Schedule a Cloud Function | Cloud Scheduler Documentation», Google Cloud. Åpnet: 2. mai 2024. [Online]. Tilgjengelig på: https://cloud.google.com/scheduler/docs/tut-gcf-pub-sub

[7] «react-secure-storage», npm. Åpnet: 29. april 2024. [Online]. Tilgjengelig på: https://www.npmjs.com/package/react-secure-storage

[8] «HTTPS», *Wikipedia*. 20. november 2023. Åpnet: 2. mai 2024. [Online]. Tilgjengelig på: https://no.wikipedia.org/w/index.php?title=HTTPS&oldid=23971119

[9] «What Is HTTPS & How Does It Work? [Explained]», Semrush Blog. Åpnet: 6. mai 2024. [Online]. Tilgjengelig på: https://www.semrush.com/blog/what-is-https/

[10] «Build software better, together», GitHub. Åpnet: 6. mars 2024. [Online]. Tilgjengelig på: https://github.com

[11] «Use JSDoc: Index». Åpnet: 2. mai 2024. [Online]. Tilgjengelig på: https://jsdoc.app/