Utveckling av underhållsmobilapplikation med React Native och Expo

Digitalisering av hemmets underhåll

Oscar Weber

Lärdomsprov

Informationsteknologi

2024

Lärdomsprov

Oscar Weber

Utveckling av underhållsmobilapplikation med React Native och Expo. Digitalisering av hemmets underhåll

Yrkeshögskolan Arcada: Informationsteknologi, 2024.

Sammandrag:

En underhållsbok används för att hantera hemförvaltningsinformation. Innehållet består av diverse uppgifter som användaren själv fyller i, till exempel: hemmets adress, postnummer, golvyta, byggmaterial och så vidare. Syftet är att undersöka hur en mobilapplikation kan underlätta, effektivisera och förenkla hanteringen av sin information. Målet är att erbjuda en digital lösning för skötseln av hemmet. Den digitala lösningen har tre krav för att nå målet och syftet: den ska vara användarvänlig, enkel och snabb att komma igång med samt lättillgänglig.

För programmeringsspråket JavaScript finns ramverken React Native och Expo, som är verktyg för att skapa mobilapplikationer. Typescript är ett annat programmeringsspråk som utökar Javascript med att tillägga till exempel typer, arbetet använder sig av Typescript istället för JavaScript. Ramverket Expo ger också tillgång till mobilapplikationen Expo Go. I det här arbetet beskrivs hur dessa verktyg fungerar och implementeras. Teoridelen behandlar även hur en välgjord applikation designas, med särskilt fokus på användarvänlighet.

Arbetet går genom produktionen av applikationen. Det förklaras hur de används för att nå syften och målen, inkluderar också vilka designval har gjorts och varför.

Resultatet är en applikation där användaren kan lätt mata in sin information och själv anpassa innehållet enligt sina egna behov. Applikationen ger också enkel åtkomst till informationen och gör det smidigt att börja använda underhållsboken

Följande steg för applikationen är att implementera någon form av backend, exempelvis en databas eller någon annan typ av molnlagring. Detta skulle göra det lättare att dela sin

information med andra, till exempel vid en husförsäljning. Sista steget är att publicera applikationen i Google Play Store.

Nyckelord:

React Native, Expo, Expo Go, Typescript, Underhållsbok

Degree Thesis

Oscar Weber

Development of a mobile maintenance application with React Native and Expo. Digitalisation of home maintenance

Arcada University of Applied Sciences: Information Technology, 2024.

Abstract:

A maintenance book is used to manage home management information. The book's content consists of diverse information that the user fills in themselves, such as home's address, postal code, floor area, building materials, and so on. The purpose is to research how a mobile application can facilitate, streamline and simplify the management of this information. The goal of the application is to offer a digital solution for maintaining one's home. The digital solution has three requirements to achieve its goal and purpose: it must be user-friendly, easy and quick to start using, and easily accessible.

For the JavaScript programming language, frameworks like React Native and Expo are tools designed to create mobile applications. Typescript is another programming language that extends JavaScript by adding types, this application uses Typescript instead of JavaScript. The Expo framework also provides access to the mobile application Expo Go. The paper theoretically reviews how these tools function and how they are implemented in the project. The theoretical section includes how a well-designed application is created, with extra emphasis on user-friendliness.

The paper goes through the production of the application and how the tools are utilized. It explains how the tools are used to achieve the purpose and goals, it also includes which design choices were made and why.

The result is an application where the user can easily input their information and configure the content according to their own needs. It also provides easy access to the information and makes it simple to start using the maintenance book.

The next steps for the application include implementing some form of backend, such as a database of another type of cloud storage. This would enable easier sharing of information

with another user, for example, when selling the house. The final step is to upload the application to the Google Play Store.

Keywords:

React Native, Expo, Expo Go, Typescript, Maintenance book

Innehåll

Förkortningar

API - Application programming interface
APK - Android Application Packages
EAS - Enterprise Application Software
IT - Informationsteknologi
JSON - JavaScript Object Notation
PDF - Portable document format - det kända formatet att spara dokument i
UI - User interface, användargränssnitt
UX - User experience
VSC - Visual studio code

1 Inledning

Att sköta och upprätthålla ett hem är inte alltid det lättaste. En underhållsbok är vad som används för att sköta hanteringen av hemförvaltningsinformation. Innehållet är diverse information man själv fyller i, som exempel: adress, postnummer, golvyta (kvadratmeter), tomtens storlek, byggmaterial, värmelösningar och så vidare. Målet är att flytta boken till användarens telefon. Detta arbete är inriktat på att underlätta och effektivisera hanteringen med hjälp av en mobilapplikation. (Energiatehokas koti, 2008)

Med tanke på att över 86% av jordens befolkning äger en smart telefon och enligt en undersökning i USA, spenderar deras befolkning mellan 5 och 6 timmar i dagen på telefonen. Enligt statistiken finns en smarttelefon oftast nära till hands. Därför är en mobilapplikation ett logiskt alternativ istället för en fysisk bok. (Press, 2024)

På grund av att målet var ett alternativ som alltid är nära till hands så måste själva mobilapplikationen också vara tillräckligt snabb och enkel att använda. Alltså resultatet är en användarvänlig, lätthanterad och enkel mobilapplikation.

1.1 Syftet

Applikationen görs för att kunna svara på hur en mobilapplikation kan effektivisera och underlätta hanteringen av hemförvaltningsinformation jämfört med en fysisk bok. Hur mycket enklare det blir för ägaren att kunna kolla upp sin information och ändra på den om

det krävs. Jämfört med att hämta boken och hitta rätt sida. Om man har skrivit med blyerts går det lätt att gumma men annars blir det svårare att göra ändringar. Det finns alltid risken att boken har varit förvarad på ett olämpligt ställe och skadats, eller till och med blivit av misstag bortkastad vid storstädning. Speciellt relevant när en underhållsbok har väldigt oregelbunden användning.

1.2 Målet

Målet bakom detta projekt är att kunna erbjuda en digital lösning åt användaren för att sköta sitt hem och potentiellt locka in nya användare som inte över huvud taget använt någon form av underhållsbok. Lathet är en av de större orsakerna man inte tagit i bruk en bok eller något annat format, det är därför simplicitet och snabbhet är en av huvudpunkterna med applikationen. Själva inmatningen av vilken information man vill att skall komma med blir den viktigaste delen och får inte bli klumpig. Eftersom det är en mobilapplikation i fråga kommer det vara en relativt rak väg att köra på när det gäller fortsatt utveckling och implementering av ny funktionalitet.

1.3 Avgränsningar

Arbetet kommer inte att behandla en potentiell backend, det kommer att vara gjort som en "offline applikation" vid den första iterationen. Själva designprocessen för potentiella logon, symboler och så vidare kommer inte att bli dokumenterat i arbetet, men varför komponenter är designade som de är kommer att tas upp samt en del User Experience (UX) och User Interface (UI) teori som relaterar till det.

Huvudsakligen kommer det att behandla verktygen och biblioteken som används, deras funktionalitet och hur man tar dem i bruk. Processen att utveckla en applikation med dessa verktyg som t.ex. fil struktureringen för applikationen och skapandet av sidor i React Native. Det är viktigt att notera att en direkt syn på hela källkoden kommer ej ges. Det finns en chans att detta blir en säljbar produkt, alltså det kommer inte att vara open-source.

2 Bakgrund och teori

2.1 React Native

React Native är ett ramverk för att bygga mobila applikationer med JavaScript och React. I grunden utnyttjar React Native det populära JavaScript-biblioteket React, som används för att kunna bygga en user interface (UI). Med React Native kan man sedan utveckla mobila UI's med hjälp av deklarativa komponenter. (React Native. 2024). Deklarativa syftar på att du definierar vad du vill ha och React Native tar hand om "hur" delen. (1800 INFORMATION, 2008).

React Native är en av flera broar mellan webbutveckling och mobilutveckling. React Native använder Javascript för att interagera med inbyggda komponenter och application programming interfaces (API) genom ett koncept som kallas "the Bridge" vilket tillåter kommunikation mellan Javascript-koden och den inbyggda plattformen. På detta vis får utvecklaren njuta av den inbyggda prestandan samt bibehålla flexibiliteten som React hämtar. (React Native Documentation, 2024).

En av de största styrkorna med React Native är "hot reloading". Denna funktion ger möjligheten att se ändringarna som görs till koden i realtid utan att måsta starta om hela applikationen. Funktionen snabbar drastiskt upp utvecklingsprocessen och gör justeringarna mera effektiva. React Native gör också det lätt att skriva kod en gång och sedan köra det på både iOS och Android plattformar, vilket förkortar utvecklingstiden och lättar på ansträngningen på utvecklaren. (Norbert, 2024)

React Native handlar om att vara ett mångsidigt och behändigt verktyg för att bygga sofistikerade mobila applikationer. Det finns också andra verktyg som Ionic, Cordova, Xamarin eller Flutter. Varken Flutter eller Xamarin tillåter Typescript, vilket är orsaken bakom varför de inte blev valda. Ionic och Cordova tillåter Typescript men React Native i kombination med Expo gav den optimala kombinationen av prestanda, distributionsprocess, utvecklingsprocess, tillgång till inbyggda API och programmeringsspråk för denna mobilapplikation. Därför används React Native i projektet. (Farid, 2024) (InApp, 2024)

2.2 TypeScript istället för JavaScript

Typescript är en så kallad “superset” av JavaScript. Alltså det inkluderar allting från JavaScript och har där på har extra funktioner. Typescript inför statiska typer vilket fångar felen vid utvecklingsfasen istället för när man kör applikationen. (TypeScript, 2024)

Med TypeScript behöver man definiera typer, props, states och andra variabler

```
// Example for screenshot
interface MyProps {
    message: string
}

const MyMessageComponent: React.FC<MyProps> = ({ message }) => {
    return (
        <View>
            <Text>{message}</Text>
        </View>
    )
}
```

Figur 1. Definierande av props för en komponent i TypeScript

Så här blir det omöjligt för variabeln “message” vara något annat än av typen “string”. En editor som Visual Studio Code (VSC) skulle ge felmeddelanden om man försöker ge något annat än en “string” till komponenten. På detta sätt hjälper TypeScript att använda rätta typer, minskar chansen för buggar och gör det lättare att upprätthålla koden. Det ger en visuell varning genast när man till exempel försöker ge en “string” när det borde vara ett nummer. (TypeScript, 2024)

2.3 Expo

Expo är ett open-source-ramverk och plattform för att bygga React Native applikationer. Expo simplifierar utvecklingsprocessen med att erbjuda en kombination av verktyg, bibliotek och tjänster som effektiviserar skapandet, testandet och även driftsättande av mobila applikationer. Med Expo kan man skriva Javascript eller Typescript-kod och använda sig av React Native-komponenter. Man får också tillgång till inbyggda enhets funktioner utan att måsta konfigurera plattform specifika miljöer som Android Studio eller Xcode (Xcode är som Android Studio för Apple). (Expo, 2024)

Expo tar bort mycket av det komplicerade som kommer med att utveckla mobila applikationer. Plattformen hanterar alla inbyggda konfigurationer och "dependencies" vilket ger möjligheten för utvecklaren att fokusera mera på att skriva frontend-kod. Expo Software Development Kit (SDK) ger tillgång till diverse inbyggda funktioner som kameror, GPS, sensorer, "push notifications" (notifikationer som skickas rakt till en användares telefon istället för inne i själva applikationen) och så vidare. Detta är möjligt på grund av unified Javascript API. (Expo Documentation, 2024)

2.3.1 Expo nyckelfunktioner och fördelar

Nyckelpunkter för Expo. (Expo Documentation, 2024):

1. **Ett hanterat arbetsflöde (Managed Workflow):** Expo tar hand om de inbyggda "dependencies" vilket gör det lättare för en utvecklare att skapa och upprätthålla plattformsoberoende applikationer.
2. **Expo SDK:** En uppsättning av färdigt konfigurerade APIs som ger tillgång till de inbyggda funktionerna, till exempel kameran eller filsystemet.
3. **Expo Go:** En mobilapplikation som gör att utvecklaren kan förhandsgranska och testa sina projekt på riktiga telefoner.
4. **OTA (over the air) uppdateringar:** Med Expo kan utvecklarna skuffa kod-uppdateringar rakt med att gå förbi processen för återgodkännande som App stores kräver.
5. **Flexibiliteten att kunna "ejecta":** Om utvecklaren behöver anpassade inbyggda moduler så ger Expo möjligheten att "ejecta" applikationen vilket ger tillgång till den inbyggda koden.

Fördelar jämfört med andra alternativ som Cordova eller Xamarin. (Farid, 2024):

1. **Snabb utveckling:** Expo tar hand som mycket av installationen så utvecklaren kan snabbt börja utveckla sin applikation.
2. **Konsistens över olika plattformar:** Expo säkerställer att samma kod fungerar både på Android och iOS.

3. **Simplifierad testning:** Med Expo Go mobilapplikationen kan man genast kolla genom sina ändringar på sin telefon vilket förbättrar arbetsflödet.

2.4 Expo Go

Expo Go är en mobilapplikation för både Android och iOS, designad för att effektivisera utvecklandet och testandet av React Native applikationer. Applikationen ger möjligheten att förhandsgranska och interagera med ett projekt utan att behöva kompilera inbyggd kod eller ställa in komplexa byggmiljöer.

Expo Go fungerar som en körtidsmiljö (runtime environment) för React Native applikationer byggda med Expo ramverket. All Javascript/Typescript kod utvecklaren skriver körs på enheten inom Expo Go applikationen vilket redan innehåller alla inbyggda bibliotek och APIs som behövs. Applikationen ansluter till en lokal utvecklingsserver (Metro bundler) genom en QR-kod eller nätverks Uniform Resource Locator (URL) för att ladda och köra den senaste koden i realtid. (Expo Go, 2024)

2.4.1 Expo Go nyckelpunkter

Nyckelpunkter. (Expo Go, 2024):

1. **Omedelbara uppdateringar:** Kod ändringar syns direkt i Expo Go applikationen tack vare “hot reloading”.
2. **Ingen “Native Build” krävs:** Android Studio eller Xcode behövs inte för att förhandsvisa sin applikation på en fysisk telefon. Expo Go sköter allting internt.
3. **Plattformsoberoende testning:** Samma Expo Go applikation fungerar för både Android och iOS.
4. **Tillgång till Expo APIs:** Expo Go kommer med många APIs som sensorer och kameror vilket ger tillgång till de inbyggda funktionerna utan att måsta skriva egen gjord kod för det.

2.5 Varför Expo och React Native över Android Studio

Expo hämtar med sig plattformsoberoende utveckling, vilket specifikt för detta arbete inte är ännu relevant men inom framtidsplanerna blir det. Detta innebär endast lite extra arbete för

att kunna distribuera till iOS och Android. Expo tar bort kraven för komplexa inställningar och förhands steg som krävs innan man kan börja utveckla applikationen. Android Studio ger en mera tillgång till diverse inbyggd funktionalitet och applikationens prestanda men kräver mera upprätthållande samt den mera komplexa uppställningen. (Stackshare, 2020).

På grund av dessa skillnader gjordes valet att använda den mera strömlinjeformade utvecklingsmiljön Expo. Applikationen kräver inte den extra komplexiteten eller kontrollen av prestandan som Android Studio skulle hämta.

2.6 UX/UI teori

På grund av vad applikationen skall försöka ersätta undviks onödig komplexitet, en onödigt svår applikation att använda kommer endast skrämma bort användare. Det kända uttrycket "less is more" passar perfekt för UX/UI design. Det skall inte vara ett krav att måsta spendera några timmar på att försöka klura ut hur man skall använda en applikation. (UX Design Institute, 2024)

Konsistens är väldigt viktigt inom design. Kan vara oroväckande för en användare om det kommer femton olika designade knappar flygande mot dem där alla har olika fonter och färger på texten. Istället skall vi försöka använda liknande design som möjligt i alla delar av applikationen, samma bakgrundsfärg palett för alla sidor, liknande stil på diverse knappar och så vidare. (UX Design Institute, 2024)

Feedback att något har gjorts i applikationen är något man inte nödvändigtvis tänker på som utvecklaren eftersom man vet vad allting gör, eller vet vad de förväntas göra. En alldeles ny användare har dock ingen aning hur det skall se ut om något lyckades, alltså vid viktiga tillfällen eller om inte en stor visuell ändring sker kan en "push notification" vara ett bra tillägg. Man skall dock vara försiktig att inte använda dem för mycket som endast irriterar användaren. (UX Design Institute, 2024)

3 Projektets uppsättning

Expo är som sagt ett ramverk och en plattform för att bygga React Native applikationer. Expo hämtar en av de smidigaste realtids testande för projekt och ger möjligheten att inte behöva installera något som “Android Studio”.

3.1 Steg 1: Förberedande av datorn

Första steget är att kontrollera att maskinen är färdig för utvecklande med Expo. Expo kräver Node.js och npm. Efter det installeras Expo CLI (command-line interface) för att skapa och hantera Expo projekt. (Expo Documentation, 2024)

1. **Installera Node.js och npm**
   `sudo apt install nodejs npm`
2. **Installera Expo CLI**
   `npm install --global expo-cli`
3. **Verifiera installationerna**
   `node -v`
   `npm -v`
   `expo --version`

3.2 Steg 2: Skapa Expo projektet

Expo erbjuder på färdiga mallar för att snabbt och lätt sätta igång. Behövs endast köra ett kommando för att skapa projektet. (Expo Documentation, 2024)

1. **Skapa Expo projekt**
   `npx create-expo-app ExampleProject -template blank-typescript`
   Detta kommando har “-typescript” i slutet vilket betyder att projektet kommer konfigureras att använda Typescript som sitt språk
2. **Navigera till det nyligen skapade projektet**
   `cd ExampleProject`
   Kommandot för att gå in i det skapade projektet och se strukturen
3. **Projekt strukturen**
   - `App.tsx` : Startpunkten för applikationen.
   - `package.json` : Listan för “dependencies” och skript för projektet.

- **`node_modules/`** : Förvarar alla installerade paket och bibliotek, är en folder.
- **`tsconfig.json`** : Typescript konfiguartions filen

3.3 Steg 3: Starta utvecklings servern

När projektet är skapat kan man börja utveckla applikationen. Först startar man sin utvecklingsserver. “Metro bundler” som det heter är den lokala “web interface” som ger dig möjligheten att hantera projektet och ansluta enheter. (Expo Documentation, 2024)

Kommando för att starta servern:

`npx expo start`

Detta kommer starta servern som kommer se ut enligt följande:

Figur 2. Metro Bundler demonstrering

3.4 Steg 4: Körandet av applikationen

För att köra sin applikation vid utvecklingsfasen har man tre alternativ: Expo Go, Emulator eller på webbläsare. (Expo Documentation, 2024)

1. **Expo Go på en telefon**

   - Ladda ner applikation “Expo Go” på en Android eller iOS telefon

   - Skanna QR koden som visas i Metro bundler med Expo Go applikationen.

   - Applikationen kommer köras direkt på telefonen och alla ändringar i koden syns genast.

2. **Emulator**

   - Installera Android Studio (eller liknande för iOS).

   - Starta emulatorn med: **`emulator -avd <emulator name>`**

   - Tryck “a” inne i Metro bundler terminalen för att köra applikationen på emulatorn

3. **Webbläsaren**

   Det enklaste alternativet.

   - Öppna en webbläsare

   - Skriv “localhost:8081” i URL-fältet.

   Porten “8081” är standardporten som servern körs på. Om porten är någon annan ser man det i Metro bundler vid “Web is waiting on …”

3.5 Steg 5: Expo bibliotek

Som sagt kommer Expo med en hel del inbyggda APIs i formen av sensorer, kamera tillgång och så vidare. Man kan även installera bibliotek från Expo-ekosystemet. Som exempel använder vi **`expo-location`** biblioteket. (Expo Documentation, 2024)

1. **Installera biblioteket**

   `npm install expo-location`

2. **Biblioteket i koden**

```tsx
// Usage of Location Example
const App: React.FC = () => {
  const [location, setLocation] = useState<Location.LocationObject | null>(null);

  useEffect(() => {
    (async () => {
      let { status } = await Location.requestForegroundPermissionsAsync();
      if (status !== 'granted') {
        console.log('Permission Denied');
        return;
      }

      let currentLocation = await Location.getCurrentPositionAsync({});
      setLocation(currentLocation);
    })();
  }, []);

  return (
    <View>
      <Text>Location: {location ? JSON.stringify(location) : 'Loading...'}</Text>
    </View>
  );
};

export default App;
```

Figur 3. expo-location biblioteket exempel

3. **Hur det ser ut i applikationen**

   Positionsdatan i en väldigt rå form ser ut som exemplet nedanför om man inte filtrerar det.

Location: {"coords":{"latitude":60.1915392,"longitude":24.9790464,"altitude":null,"accuracy":1222.4665578038723,"altitudeAccuracy":null,"heading":null,"speed":null},"timestamp":1729494563127}

Figur 4. Exemplet i applikationen

3.6 Steg 6: Bygg applikationen

Efter man utvecklat färdigt sin applikation så blir nästa steg att bygga den för produktion. (Expo Documentation, 2024)

1. **Kör kommandot för att bygga applikationen**

   `npx expo prebuild`

2. **För Android Application Package (APK) eller iOS behövs Expos Enterprise Application Software (EAS)**

   Först behöver man installera Expos EAS. Sedan körs kommando för att bygga applikationen. Beroende på om man behöver Android eller iOS väljer man kommandot enligt följande:

```
npm install -g eas-cli
eas build --platform android
eas build --platform ios
```

4 Metodik

4.1 Filstruktur

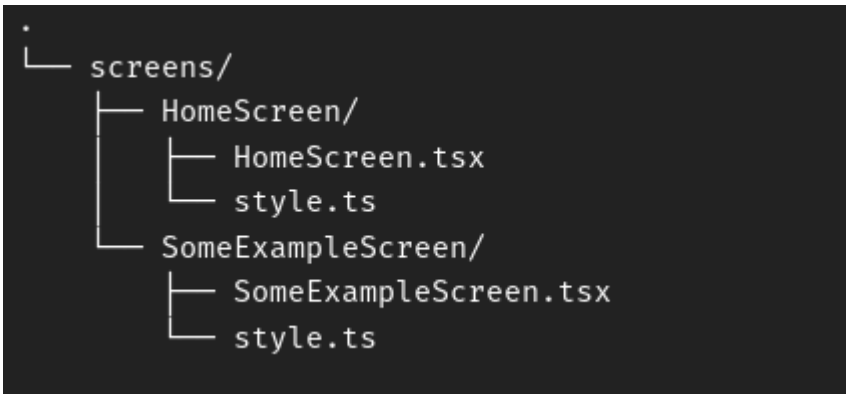
I teorin har filstrukturen inte en skillnad så länge man har all kommunikation mellan filerna rätt, dock kommer Expo med en grundmall när man startar sitt projekt och det rekommenderas inte att börja hur som helst ändra på den grunden. En organiserad filstruktur gör livet lättare för alla och sparar en hel del huvudvärk i framtiden om man inte arbetat med koden på ett tag. Hjälper också att skriva kommentarer och/eller dokumentera koden.

4.1.1 Bra praxis

Som en generell regel sägs det att endast en komponent skall exporteras ur en fil och filnamnet skall vara samma som komponentens namn. Det är bättre att ha en lång rad med **`.tsx`** filer i foldern **`components`** istället för en enda **`components.tsx`** var man sedan måste gå genom en massa kod för att hitta vad man söker. Inget stoppar en från att därpå ha en egen folder för till exempel olika rullgardinsmenyer. (Kraman, 2024)

Samma kan göras för olika funktioner, skapar alltid en funktion om det används flera gånger. Istället för att skapa funktionen och ta upp utrymme i en komponentfil så kan man förvara den i en egen fil som är i **`functions`** foldern. Det finns alltid undantag och ett bra exempel brukar vara funktioner som hanterar förändringar i komponenten, en **`handleInputChange()`** som hanterar förändringar i ett skrivfält är mer praktisk att ha på samma ställe som skrivfältet. Hanterandet av lagringssystemet är dock bra att ha på ett eget ställe. I projektet är det gjort i en **`storage.ts`** fil där alla funktioner exporteras, till exempel funktioner som **`setItem(parameters)`** och **`getItem(parameters)`**. Sedan importeras dem med **`import * as Storage from "../functions/storage"`**. För att använda dem använder man sedan **`Storage.setItem(parameters)`**.

Det finns olika sätt att göra de olika sidorna för en applikation i Expo, i detta projekt är det gjort med helt enkelt screens. Varje skärm så att säga skall ha en egen folder där det sedan finns själva skärmen i **`.tsx`** format, är också populärt och smart att placera **`style.ts`** filen med designen för den specifika skärmen med i skärmens folder.



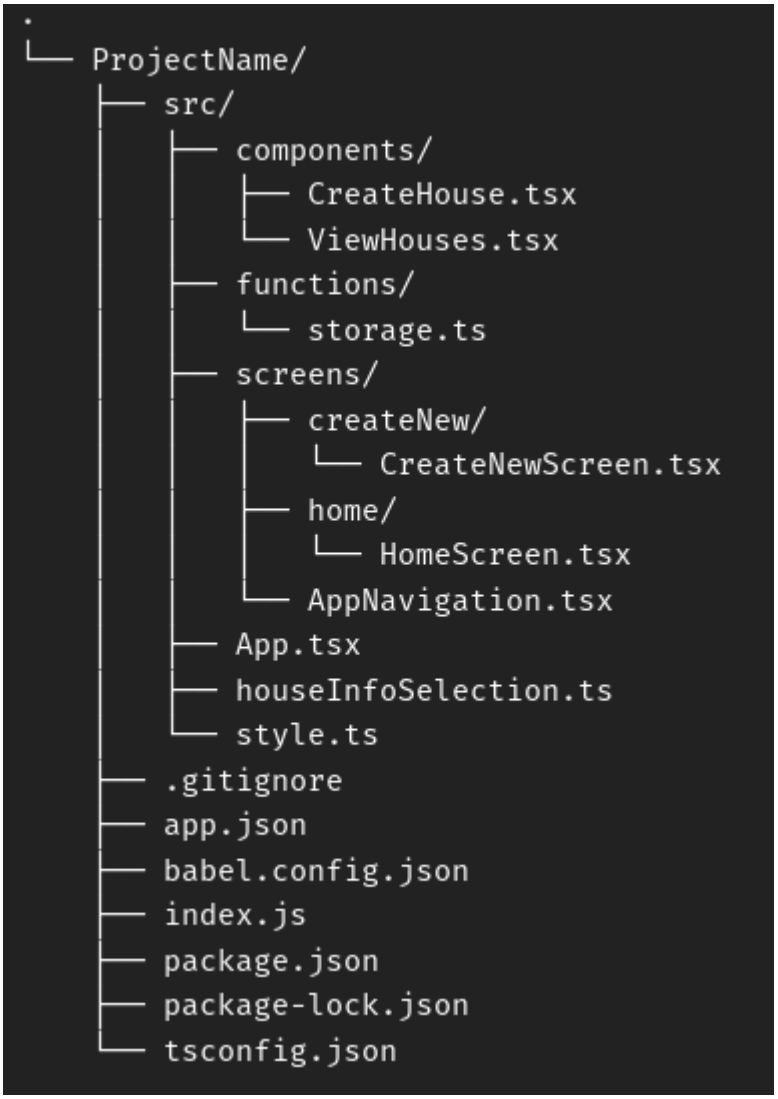
Figur 5. Visualisering av strukturen för skärmar

I detta projekt så är all stil konfigurering i en och samma fil men separerad och organiserad i filen. Det är dock som allt annat och är inte ristat i sten, aldrig omöjligt att det kommer att skapas flera **`style.ts`** filer.

4.1.2 Strukturen som används

Allra först kommer det vara projektets folder med sitt namn. Den välbekanta **`src`** foldern innehåller det mesta i projektet, i princip kommer alltid diverse filer relaterade till den allmänna konfigurationen av applikationen att befinna sig alldeles längst ut i hierarkin alltså utanför **`src`**, till exempel sådant som **`package.json`** och **`tsconfig.json`**. Vi hittar även **`.gitignore`** på samma ställe. Utanför src är också där det alltid finns en **`README.md`**.

Projektet har inte samma filstruktur som kommandot **`npx create-expo-app@latest`** ger som grund. Det finns likheter men projektets struktur är milt baserat på strukturen från andra projekt jag tidigare jobbat på. Sist och slutligen är det alltid personlig preferens hur man vill att det exakt skall se ut, inget stoppar en heller från att senare ändra på strukturen om man upptäcker ett mer logiskt sätt att göra det för sitt projekt.



Figur 6. Filstrukturen för projektet

4.2 React navigation

Både Expo och React kommer båda med möjligheter för navigationen i en applikation, i detta fall används Reacts bibliotek. Specifikt **`bottom-tabs`** navigation vilket skapar ett fält nere på skärmen var man sedan kan lägga till klickbara länkar som tar en till den sidan man vill. Denna stil av navigation gör det också snabbare att komma till en annan sida eftersom det endast kräver ett steg, en rullgardinsmeny kunde vara en möjlighet men vore inte logisk att använda om det inte finns väldigt många olika sidor. (React Navigation, 2024)

4.2.1 Navigation komponenten

Som man ser i struktur förklaringen finns det en **`AppNavigation.tsx`**. Den filen innehåller det synliga navigationsfältet som finns i applikationen. Komponenten byggs upp med **`<Tab.Screen>`** inne i en **`<Tab.Navigator>`**.Kräver att installera Reacts navigations bibliotek

`npm install @react-navigation/bottom-tabs`

```
// AppNavigation.tsx
import React from "react"
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs"
import HomeScreen from "./home/HomeScreen"
const Tab = createBottomTabNavigator()
const AppNavigation = () => (
    <Tab.Navigator
        screenOptions={() => ({
            // Inställningar för navigationsfältet
            // Som exempel så dessa inställningar ändrar på färgen i knappen beroende på
            // om det är den aktiva sidan.
            tabBarActiveTintColor: 'red',
            tabBarInactiveTintColor: 'white',
        })}
    >
        <Tab.Screen
            // Inställningar för specifika klickbara "knappen", t.ex. till vilken sida
            // den skall navigera till
            name="Home"// Vad det står på knappen
            component={HomeScreen} // Denna knapp tar en till hem skärmen
        />
        <Tab.Screen
        // Samma som ovan
            name="Home2"
            component={HomeScreen} // Samma så att den inte skall visa errors för exemplet
        />
    </Tab.Navigator>
)
export default AppNavigation
```

Figur 7. Exempelkod för navigationsfältet

`<Tab.Navigator>` är fältet som tar upp nedre delen av skärmen medan **`<Tab.Screen>`** är de enskilda knapparna man kan trycka på vilket tar en till de olika sidorna. Detta är dock endast en komponent och som sådan måste den användas någonstans. **`App.tsx`** är standardingångspunkten för projektet, i **`index.js`** kan man konfigurera det till något annat om man vill, detta är alltså startpunkten och det första som renderas när applikationen startar. Vi vill alltså importera komponenten till vår startpunkt. (React Navigation, 2024)

```tsx
// App.tsx
import { NavigationContainer } from "@react-navigation/native"
import AppNavigation from "./screens/AppNavigation"


export default function App() {
  return (
    <NavigationContainer>
      <AppNavigation/>
    </NavigationContainer>
  )
}
```

Figur 8. Exempelkod för importerande och användande av navigationsfältet

Containern hanterar applikationens navigerings tillstånd, den hanterar all navigation logik och utan den skulle inte applikationen veta hur den ska röra sig mellan de olika sidorna. (React Navigation, 2024)

4.3 React Native Async Storage

Async Storage är ett lagringssystem i React Native. Den förvarar data lokalt på användarens enhet även när applikationen stängs. Async storage fungerar på ett liknande sätt som "local storage" i webbapplikationer. Detta lagringssystem kommer att användas för den första iterationen av applikationen och blir utbytt i framtiden till molnlagring. (Async Storage, 2024)

4.3.1 Async Storage i applikationen

I applikationen har jag all logik i en **`storage.ts`** fil och sedan exporteras funktionerna från den. Viktigt att komma ihåg är att Async Storage sparar allting som en "string", vilket är också en av orsakerna varför framtida iterationer kommer kräva utvecklingen av en backend. När sparandet av information sker behövs en nyckel. Nyckeln används för att sedan kunna hämta den specifika data. För att kunna spara datan behövs den serialiseras till JavaScript Object Notation (JSON). **`JSON.stringify()`** används för att spara datan och **`JSON.parse()`** när man vill hämta datan. (Async Storage, 2024)

Funktionerna för att spara och hämta data ser ut på detta vis:

```
import AsyncStorage from "@react-native-async-storage/async-storage"
import { Alert } from "react-native"

export const setItem = async (key: string, value: string) => {
    try {
        await AsyncStorage.setItem(key, value)
        Alert.alert("Success", "Saved successfully")
        console.log("Item Set")
    } catch (error) {
        console.log("Set item error", error)
    }
}

export const getItem = async (key: any) => {
    try {
        const result = await AsyncStorage.getItem(key)
        return result != null ? JSON.parse(result) : null //Parse it Object
    } catch (error) {
        console.log("Get item error", error)
    }
}
```

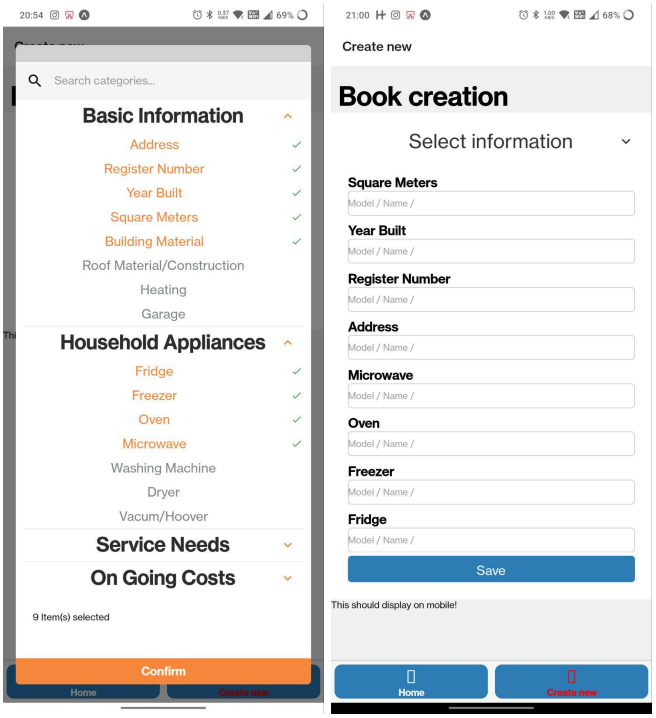
Figur 9. Källkod från applikation för lagringshantering

4.4 Applikationens funktion

4.4.1 Skapandet av en bok

Användaren kan skapa en bok med hjälp av en rullgardinsmeny för att välja vilken information som skall komma med. Efter det får användaren fylla i skrivfält för den valda informationen, detta ger användaren möjligheten att skriva vad de själva vill.

Specifikt är det en importerad React Native **`<sectionedMultiSelect>`** rullgardinsmeny som används. Denna meny innehåller underrubriker för att bättre organisera alternativen.

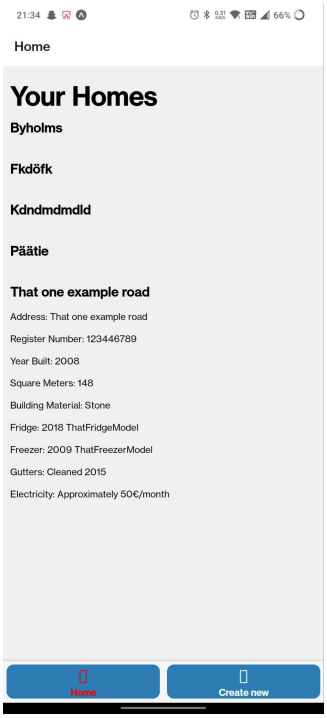


Figur 10 (vänster). *Rullgardinsmenyn från applikationen. PS: Färdiga stilen och innehållet är inte närvarande.*
Figur 11 (höger). *Skrivandet av informationen. PS: Färdiga stilen och innehållet är inte närvarande.*

Efter att man klickat “Confirm” visas sidan där man sedan får skriva exakt vad man vill. Det är mycket sannolikt att stilen av applikationen kommer till någon form variera mellan iterationerna.

4.4.2 Användarens böcker

Poängen är att användaren sedan kan se sitt eller sina hem så snabbt som möjligt efter att applikationen öppnas. Lätt redigering av information är en av de viktigaste delarna med applikationen, det fungerar med att helt enkelt klicka på den information man vill ändra på. Användarens alla hem syns på framsidan, med att klicka på en av dem kommer hemmets information bli synligt.



Figur 12. Användarens skapade böcker. PS: Färdiga stilen och innehållet är inte närvarande

4.5 Användningen av stil i applikationen

Istället för massor av extra rader i antingen komponent eller skärm filen så sätter vi stilen in i en egen eller egna filer och importerar dem. Vi använder oss av **`StyleSheet`**, en React Native API för definierande och organiserande av stilar till komponenter.

Man kan göra det med att ha allting inne i en och samma **`StyleSheet`** var man sedan importerar den som sådan. Jag har istället gjort det med flera stycken som är organiserade i kategorier. Då importerar man endast den eller de man behöver. Gör det också lite lättare att organisera det i själva filen. Som exempel:

```
// style.ts
import { StyleSheet } from "react-native";

export const textStyles = StyleSheet.create({
    titleText: {
        fontFamily: "sans-serif",
        fontSize: 40,
        fontWeight: "bold",
    },
    smallTitleText: {
        fontFamily: "sans-serfi",
        fontSize: 20,
        fontWeight: "bold",
    },
})
```

Figur 13. Exempelkod för skapandet av en "style sheet"

När man sedan vill använda stilen i en komponent börjar man med att importera **`textStyles`**. Sedan är det enkelt att lägga in stilen var man vill ha den.

`<Text style={textStyles.titleText}> Min rubrik här </Text>`

På detta vis använder den sig av den definierade stilen. Nu är det endast en rad och inte flera.

5 Applikationens design

Målet med applikationen är att ersätta pappersformatet. I samband med detta mål kommer kravet att göra applikationen så okomplicerad som möjligt in. Därför använder applikationen ett navigationsfält nere på skärmen vilket låter navigeringen endast kräva ett klick för att byta sida.

Konfigurerbarhet var också ett krav vilket betyder att skapandet av en "bok" innehåller tyvärr några steg, men dessa steg är väljande vilken information användaren vill att boken innehåller. Det är löst med en rullgardinsmeny där man klickar på vad man vill ha med. Efter det är det endast ett simpelt skrivfält som behövs fyllas i och detta tillåter användaren skriva helt vad han eller hon själv vill, till exempel om man har information om skorstenens sotande "10.4.2010 , Sota skorstenen innan julgubben fastnar". Om användaren har till exempel en stuga är det väldigt lätt att bara skapa en till bok för den.

Editeringen av en av sina böcker är gjort så att man endast behöver klicka på namnet i listan, efter att all information syns är det bara att klicka på den information man vill ändra på. Om man köper en ny tvättmaskin är det enkelt att bara gå in i applikationen och ändra informationen för tvättmaskinen.

På grund av vad applikationen försöker ersätta finns det ingen poäng med onödig komplexitet, en onödigt svåranvänd applikation kommer endast skrämma bort användare. Det kan inte vara ett krav att måsta spendera några timmar på att klura ut hur man skall använda en applikation.

6 Resultat

Resultatet av arbetet är en mobilapplikation som effektiviserar hanteringen av hemförvaltningsinformation genom att flytta användarens underhållsbok till deras telefoner. Applikationen är användarvänlig, konfigurerbar enligt behov och tillåter användaren att lagra och redigera information relaterat till deras hem. Expo och React Native har använts för att utveckla en mobilapplikation vars fokus är enkelhet och snabb åtkomst.

Påståendet att applikationen ger snabb tillgång till användarens information baserar sig på designprinciper inom användarvänlighet, med fokus på tydlig navigering och en logisk struktur för informationen (UX Design Institute, 2023). Ett navigationsfält i kombination med konfigurerbart innehåll strävar till att minimera komplexiteten och tiden som krävs för att hitta och redigera sin information.

Antagandet att uppmuntra användare fylla i informationen grundar sig på ovannämnda designprinciper och den massiva mängden smarta telefoner som används (Press, 2024). Den minimalistiska designen och möjligheten att konfigurera vilken information som inkluderas förväntas sänka tröskeln för användaren.

6.1 Följande steg för framtiden

Största steget för en applikation kommer nästan alltid vara vägen till en App store men det kommer högst antagligen vara ett av de senare stegen. Före det finns det på den nuvarande agendan att göra det möjligt för användaren att spara all data i molnet, möjligheten att få informationen ut i PDF-format, överföra sin sparade "bok" till en annan användare i någon form dock detta kommer med stor sannolikhet vara i samband med online uppdateringen.

En testgrupp för applikationen kommer inte att använda en version som finns på Google Play Store utan den fasen kommer att hållas i Expo Go miljön, vid detta skede sker också överföringen till online versionen.

Med hundra procent säkerhet kommer de största delarna vara backends utvecklingen och hela processen att få en applikation godkänd i Google Play Store. I den större bilden är hela potentiella företagsverksamheten för applikationen ett massivt projekt men det är inte inom ramen för detta specifika projekt.

7 Slutsatser

I arbetet tas det fram hur man använder sig av Expo och React Native för att utveckla en mobilapplikation. Praktiska exempel finns som både bilder ur den utvecklade applikationen och specifikt skapade kodsnuttar.

I princip borde alla hemägare äga en underhållsbok för sitt hus. Problemet är om ägaren inte har en eller helt enkelt inte använder en. Därför tar vi istället och skapar en lättanvänd version som befinner sig på ägarens telefon. Den lätt använda applikationen har som mål att göra inmatningen så enkel som möjlig utan många onödiga steg.

Expo i kombination med React Native gör utvecklingen av en mobilapplikation till en njutbar och bekväm process. React Native och alla dess diverse bibliotek underlättar utvecklingen, därpå tillåter ramverket community-paket, paket som är alltså gjorda av andra människor. I arbetet finns det ett perfekt exempel på ett sådant paket med Async Storage, importen av paketet liknar mycket det gamla som inte används mera och man kan lätt tro att det är React Natives egna. När vi sedan ännu använder oss av Typescript istället för JavaScript tar vi därpå

bort onödiga misstag. På grund av att Typescript tillåter inte fel typer, props eller states vid fel tillfällen. Detta tar bort den extra tid som skulle användas för att hitta felet.

Expo hämtar extra bekvämligheter på de redan existerande bekvämligheterna från React Native. Man får tillgång till de inbyggda funktionerna i enheten utan att måsta konfigurera en plattform specifik miljö som Android Studio. Expo låter en utvecklare fokusera huvudsakligen på att skriva fronted-koden för applikationen. Expo låter utvecklaren också utnyttja mobilapplikationen "Expo Go". En applikation till telefonen för att lätt kunna hoppa in i utvecklarens Expo applikation och förhandsgranska den med hjälp av "hot reloading".

Slutresultatet av arbetet är en mobilapplikation som tillåter en användare välja själv vilken information om deras hem de vill spara och kan senare editera dem vid behov. Applikationen har målet att ersätta behovet för en underhållsbok med sin simplicitet och bekvämlighet.

Källor

1800 INFORMATION (24 september 2008). *What is declarative programming?* Stackoverflow comment. https://stackoverflow.com/questions/129628/what-is-declarative-programming

Async Storage. (2024). *Async Storage. Data storage system from React Native.* *https://react-native-async-storage.github.io/async-storage/*

Energiatehokas koti (2008). *Pientalon huoltokirja.* PDF. https://www.energiatehokaskoti.fi/files/503/Pientalon_huoltokirja.pdf

Expo. (2024). *Expo and EAS are an ecosystem of tools that help you develop, review & deploy*. https://expo.dev/

Expo Documentation. (2024). *Create amazing apps that run everywhere.* https://docs.expo.dev/

Expo Go. (2024). *Expo Go is a sandbox that enables you to quickly experiment with building native Android and iOS apps.* https://expo.dev/go

Farid. A (13 april 2024). *Building apps with Expo and React Native: pros and cons.* Uppstack studio blog. https://upstackstudio.com/blog/expo-react-native/

GilPress. (31 januari 2024). *How many people own Smartphones? (2024-2029).* What's the big data. https://whatsthebigdata.com/smartphone-stats/

InApp (31 oktober 2024) *React Native vs. Flutter vs. Ionic vs. Xamarin vs. Nativescript (A Detailed Comparison).* Medium. https://inapp-inc.medium.com/react-native-vs-flutter-vs-ionic-vs-xamarin-vs-nativescript-a-detailed-comparison-bfa99d72c4bb

Kraman, K. (14 mars 2024). *12 tips for setting up your next Expo project.* Expo Blog. https://expo.dev/blog/12-tips-for-setting-up-your-next-expo-project

Norbert. K (10 oktober 2024). *React Native pros and cons.* Pagepro blog. https://pagepro.co/blog/react-native-pros-and-cons/

React Native. (2024). *React Native. Learn once, write everywhere* https://reactnative.dev/

React Native Documentation. (2024). *Welcome to the very start of your React Native journey!* https://reactnative.dev/docs/getting-started

React Navigation. (2024). *Bottom Tabs Navigator.* *https://reactnavigation.org/docs/bottom-tab-navigator/*

Stackshare. (2020) *Android SDK vs Expo* https://stackshare.io/stackups/android-vs-expo#:~:text=Cross%2Dplatform%20Development%3A%20While%20Android,both%20iOS%20and%20Android%20platforms.

Tree. (2024). *An online tree-like utility for generating ASCII folder structure diagrams.* https://tree.nathanfriend.com/

TypeScript. (2024). *TypeScript is JavaScript with syntax for types*. https://www.typescriptlang.org/

UX Design Institute. (2023). *The ultimate guide to mobile app design.* https://www.uxdesigninstitute.com/blog/ultimate-guide-to-mobile-app-design/