YRKESHÖGSKOLAN
NOVIA

Delning av kod mellan React och React Native

Oscar Härtull

Examensarbete för ingenjörsexamen (YH)-examen

Utbildningsprogrammet för el- och automationsteknik

Vasa 2021

EXAMENSARBETE

Författare:	Oscar Härtull
Utbildning och ort:	El- och automationsteknik, Vasa
Inriktningsalternativ:	Informationsteknik
Handledare:	Jan Berglund, Karl Herler

Titel: Delning av kod mellan React och React Native

Datum: 27.04.2021 Sidantal: 28

Abstrakt

Detta examensarbete har utförts på begäran av Comsel System Ab och omfattar delning av kod mellan webb- och mobilapplikationer, som är byggda med teknologierna React respektive React Native.

Ändamålet med detta examensarbete var att undersöka vilket tillvägagångssätt som kunde användas för implementering av koddelning till företagets existerande applikationer. Syftet med denna delning av kod mellan applikationer var att underlätta underhållet av programmeringskoden, eftersom en del av koden var snarlik och innehöll liknande funktioner inom både mobil- och webbdelen.

Teoridelen består av en genomgång av grundläggande termer som är relevanta för arbetet och därefter följer en jämförelse av olika metoder för att möjliggöra uppgiften. Efter det ges en planering av hur den slutgiltiga valda metoden ska implementeras. Den praktiska delen innefattar dokumentering över hur tillämpningen av delningsmetoden gick till genom programmering.

Resultatet blev ett funktionellt tillvägagångsätt för delning av kod, genom användandet av npm-paket, som fungerar med företagets nuvarande applikationer.

Språk: svenska Nyckelord: React, React Native, TypeScript, npm, koddelning

---

OPINNÄYTETYÖ

Tekijä:	Oscar Härtull
Koulutus ja paikkakunta:	Sähkö- ja automaatiotekniikka, Vaasa
Suuntautumisvaihtoehto:	Tietotekniikka
Ohjaajat:	Jan Berglund, Karl Herler

Nimike: Koodin jakaminen Reactin ja React Nativen välillä

Päivämäärä: 27.04.2021 Sivumäärä: 28

Tiivistelmä

Tämä opinnäytetyö on tehty Comsel System Oy:n pyynnöstä, ja se käsittelee koodin jakamista verkko- ja mobiilisovellusten välillä, jotka on rakennettu React- sekä React Native-teknologialla.

Opinnäytetyön tarkoituksena oli selvittää, mitä lähestymistapaa voitaisiin käyttää koodin jakamisen toteuttamiseen yrityksen olemassa oleviin sovelluksiin. Sovellusten välisellä koodin jakamisella tarkoituksena on helpottaa ohjelmointikoodin ylläpitoa, koska osa koodista oli samanlainen mobiili- ja verkko-osassa ja sisälsi näiden osalta samanlaisia toimintoja.

Teoreettinen osa koostuu työhön liittyvien perustermien tarkastelusta, jonka jälkeen eri menetelmiä vertaillaan tehtävän mahdollistamiseksi. Sen jälkeen seuraa suunnitelma siitä, miten lopullinen valittu menetelmä toteutetaan. Käytännön osa koostuu dokumentaatiosta siitä, miten jakomenetelmää sovellettiin ohjelmoinnin avulla.

Tuloksena on toimiva koodinjakomenetelmä, jossa käytettiin npm-paketteja, jotka toimivat yrityksen nykyisten sovellusten kanssa.

Kieli: ruotsi Avainsanat: React, React Native, TypeScript, npm, koodin jakaminen

---

BACHELOR'S THESIS

Author:	Oscar Härtull
Degree Programme:	Electrical Engineering
Specialization:	Information Technology
Supervisors:	Jan Berglund, Karl Herler

Title: Sharing of Code Between React and React Native

Date: 27.04.2021 Number of pages: 28

Abstract

This bachelor's thesis has been carried out at the request of Comsel System Ltd and comprises code sharing between web and mobile applications, which are built on the technologies React and React Native, respectively.

The purpose of this bachelor's thesis was to investigate which approach could be used to implement code sharing to the company's existing applications. The aim of this code sharing between applications was to facilitate the maintenance of the programming code, as part of the code was similar for both the mobile and web part and contained similar functions for them.

The theoretical part consists of a review of basic terms that are relevant for the work, whereafter different methods to carry out the task are compared. After that, a plan is given for how the final chosen method will be implemented. The practical part consists of documentation of how the sharing method was applied through programming.

The result was a functional approach to code sharing, using npm packages, which work with the company's current applications.

Language: Swedish Key words: React, React Native, TypeScript, npm, code sharing

---

Innehållsförteckning

Förkortningar

API	Application programming interface, applikationsprogrammeringsgränssnitt
CLI	Command-line interface, kommandotolk
DOM	Document Object Model, dokumentobjektmodell
IDE	Integrated Development Environment, integrerad utvecklingsmiljö

Terminologi

branch	En version som divergerar från huvudversionen och inte påverkar den med egna förändringar
constructor	Metod som skapar en instans av ett objekt från en klass
dependency	Beroende, paket som beror på ett annat paket för att fungera
directory	Katalog för filer
module	Modul, i detta fall fil eller directory som kan inkluderas i ett projekt från node_modules
repository	En datastruktur som lagrar metadata för arkivets struktur
stack	En mängd mjukvara som tillsammans bygger upp en större helhet
subdirectory	Underliggande katalog till ett directory
submodule	Underliggande module till en module
transpiler	Översättning av kod från ett programmeringsspråk till ett annat språk, med ekvivalent funktionalitet

1 Inledning

Detta examensarbete är utfört på uppdrag av och i samarbete med Comsel System Ab. Arbetet handlar om hur delad programmeringskod mellan applikationer utvecklade i React och React Native ska realiseras effektivast. Med delningen avses gemensam användning av kod mellan olika applikationer.

I den teoretiska delen i arbetet behandlas först fundamentalt JavaScript, TypeScript, React och React Native. Efter detta följer en analys av vilka metoder som kan användas för delning av kod. Den praktiska delen av examensarbetet utgörs först av planeringen av implementation för en vald metod, utgående från den nämnda teoridelen. Därefter följer själva implementeringen.

1.1 Uppdragsgivare

Comsel System Ab är ett finländskt produktdesign- och utvecklingsföretag som grundades 1989, förutom i Finland verkar företaget även på den skandinaviska marknaden. Till en början utgjordes verksamheten av planläggning och utförande av elinstallationer, datanätverkstjänster och utvecklandet av automationsutrustning. Dessa är även väsentliga delar idag för Comsel Systems produkter.

I dagsläget fokuserar företaget på att utveckla och industrialisera system och lösningar, såsom IoT-produkter (Internet of Things), för smart energimätning inom energisektorn. Datamätningarna görs för bland annat el-, fjärrvärme-, gas- och vattenförbrukning. Fram till år 2020 hade Comsel System skickat över 1 biljon mätvärden från el-, fjärrvärme-, vatten- och gasinstallationer. [1].

1.2 Syfte och bakgrund

Comsel System Ab använder sig av React och React Native för sina webb- och mobilapplikationer. Dessa applikationers kod delas delvis mellan varandra, men en del av kodens funktionalitet skulle kunna optimeras för smidigare delbarhet och underhåll. Genom delning av kod skulle man exempelvis i praktiken kunna underlätta arbetet, i och med att man då enbart behöver koda en funktion en gång för att få samma funktionalitet i mobilapplikationen som på den motsvarande webbapplikationen. Allt det här skulle då spara tid och pengar på utveckling och underhåll. Syftet med detta examensarbete var därmed att

utreda vilken metod som är mest lämplig att använda sig av för delning av kod mellan mobil- och webbapplikationer, samt att förverkliga den valda metoden.

2 Programmeringsspråk

Det här kapitlet behandlar de programmeringsspråk som är relaterade till examensarbetet. I nuläget blir allt fler applikationer webbaserade och körs i webbläsare. Därför är det naturligt att programmeringsspråket JavaScript och härledda versioner av det får allt större användning och publicitet.

2.1 JavaScript

JavaScript, förkortat JS, är ett programmeringsspråk som offentliggjordes 1995. Den officiella standarden för JavaScript är ECMAScript, som specificeras av ECMA, tidigare en akronym för European Computer Manufacturers Association. JavaScript är den främsta implementationen av denna standard och stöder all funktionalitet som anges av ECMA. Exempelvis språkets syntax, olika typer och inbyggda objekt samt funktioner omfattas av ECMAScript. Alla moderna webbläsare sedan 2012 har stöd för ECMAScript 5.1, medan äldre webbläsare stöder åtminstone versioner upp till ECMAScript 3.

Språkets funktionaliteter är bland annat att det är dynamiskt samt objekt-orienterat baserat på prototyper. I motsats till ett objekt-orienterat klassbaserat programmeringsspråk kan det prototypbaserade JavaScripts objekt ärva från andra objekt och modifiera ett objekts egenskaper under körtiden. Med ECMAScript 2015 introducerades klasser, men dessa är bara ett tillägg till språkets syntax för användarvänlighetens skull och lägger inte till en objekt-orienterad arvsmodell.

Nu för tiden är programmeringsspråket ytterst utbrett och dess användningsområden utökas hela tiden. Synligast är JavaScript på klientsidan i webbsystem och används för att programmera samt designa händelser på webbsidor, där till exempel musklick och sidnavigering är ett av dessa. Utöver detta så finner man JavaScript även på serversidan med Node.js.

Node.js bygger på öppen källkod och exekverar JavaScriptkod utanför webbläsaren med hjälp av JavaScriptmotorn V8. Standarden ECMAScript är fullt kompatibel med Node.js. Pakethanteraren i Node.js, kallad npm, inklusive dess egenskaper, redogörs för i ett senare kapitel. [2].

För detta examensarbete är JavaScripts användningsområde främst inom programbiblioteket React och ramverket React Native, varav båda två använder sig av språket med hjälp av tre följande motorer:

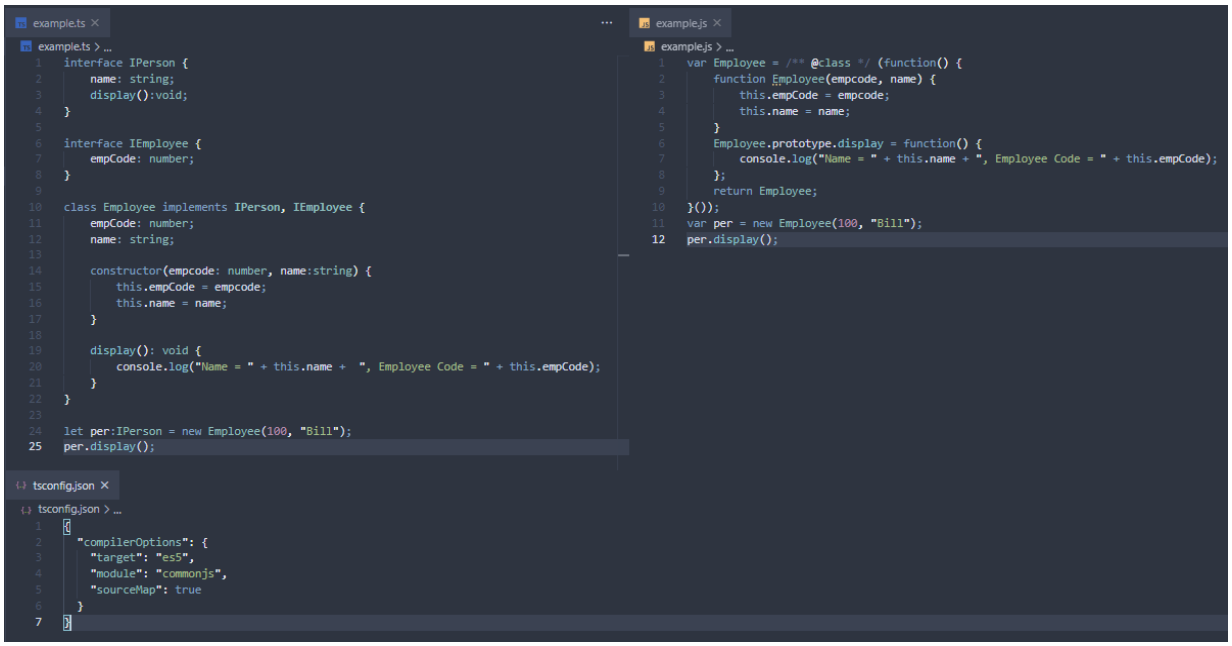
- Hermes, som är en ny motor och används av React Native när applikationer ska köras på Googles Android.
- V8, används vid felsökning av React Native applikationer i Chrome.
- JavaScriptCore, som används både för Apples iOS och Android.

[3].

2.2 TypeScript

TypeScript, som är utvecklat av Microsoft, har framställts för att ge möjligheten att validera JavaScriptkod som funktionell före exekvering av ett program, vilket underlättar i programmeringen av större programvara.

TypeScript omfattar JavaScript och kompletterar det med statiska typdefinitioner, vilket medför att all syntax i JavaScript är tillåten i TypeScript, även om syntaxen innehåller felaktigheter. Varningar utfärdas bara för hur olika värden hanteras och TypeScript hindrar inte i sin tur koden från att köras. Kompilationen till JavaScript sker genom TypeScripts egna kompilator tsc. Se figur 1 för en jämförelse av de båda. [4].



Figur 1. TypeScript kompilerat till JavaScript.

3 Programbibliotek

Ett programbibliotek består av färdigskrivna klasser och funktioner som underlättar vid utvecklingen av mjukvara. Programbibliotekets kod är skapad så att funktioner kan importeras och användas oberoende av hur strukturen för den existerande programkoden ser ut, för den applikationen som använder sig av biblioteket. De tillgängliga funktionerna i koden kan kallas på eller användas utan att de måste definieras inuti den kod som skrivs.

3.1 JavaScriptbibliotek

Tack vare JavaScripts tillväxt har ett stort urval av programbibliotek dykt upp från utvecklare. De inriktar sig på olika områden, såsom visualisering, webbapplikationer och så vidare. Exempelvis kan komplexa uträkningar lösas med ett programbibliotek som fokuserar på matematikfunktioner.

3.2 React

React är ett JavaScriptbibliotek med öppen källkod ämnad för utvecklandet av användargränssnitt. Även om det ibland kallas ett ramverk är det enligt Reacts egna dokumentation ett bibliotek, eftersom dess stack vid utvecklingen kan väljas fritt. React är komponentbaserat och individuella komponenter bygger tillsammans upp användargränssnittet. Om man går enligt Model-View-Controller (MVC) arkitekturmönstret så är React en del av presentationen, view.

React kan verkställas som en single-page application (SPA). En single-page application laddar in endast en HTML sida som sedan uppdateras dynamiskt med innehåll. Därtill är kompletterandet av ett existerande projekt med biblioteket enkelt eftersom React är tillgängligt över ett Content Delivery Network (CDN), innehållsleveransnätverk.

Att skapa en SPA som använder sig av React görs med pakethanteraren npm (Node Package Manager) eller paketexekveraren npx (Node Package Execute) i Node.js. Detta skapar bara en front-end och vad som ska användas till back-end kan väljas fritt på basis av egna krav. Exempelvis kan det nämnas att Facebook, utvecklarna av biblioteket, använder React tillsammans med PHP. [5].

3.2.1 JSX

Element i React skapas med syntaxextensionen JavaScript XML (JSX). JSX gör det möjligt att kombinera HTML/XML-liknande element i JavaScriptkod. Avsikten med att använda JSX är att underlätta skrivningen och läsningen av kod för användargränssnitt inuti JavaScript. Från JSX till standard JavaScript ändras alla JSX-uttryck till normala JavaScriptfunktioner. Konverteringen av koden görs med Babel som är en transpiler, samtidigt som detta sker optimeras kodens prestanda. [6].

3.2.2 Rendering

Dessa element renderas sedan med hjälp av en funktion kallad render från ReactDOM (Figur 2). ReactDOM är ett paket som importeras och har tillgång till modifikation av DOM-element på webbsidan. För varje DOM-objekt finns ett Virtual DOM-objekt som håller en kopia av DOM i minnet. När en uppdatering för ett element sker så uppdaterar ReactDOM först Virtual DOM, varefter en jämförelse med denna mot DOM sker. Endast det som skiljer dem åt behöver då uppdateras. [5].

```
const element = <p>This is made with JSX</p>;
ReactDOM.render(element, document.getElementById('root'));
```

Figur 2. Rendering av ett element.

3.2.3 Komponenter och Hooks

Ett viktigt särdrag hos React är dess uppdelning av användargränssnittet i komponenter. Dessa inkapslade komponenter kan återanvändas på olika ställen i en applikation enligt behov och en komponent kan även delas in i mindre komponenter. I princip kan komponenter ses som JavaScriptfunktioner som renderar HTML-element. En komponent går igenom tre skeden; montering, uppdatering och nermontering, dessa metoder benämns som lifecycle methods.

Komponenterna kan skapas antingen som funktionsbaserade eller klassbaserade. Skillnaden mellan de två alternativen är att den sistnämnda behövs om en constructor är nödvändig, dock har den ej stöd för Hooks. Funktionsbaserade komponenter kan med tillägget Hooks använda state och lifecycle methods, vilket förr var exklusivt för den klassbaserade varianten när man ville använda sig av dynamiska komponenter.

3.2.4 Props och state

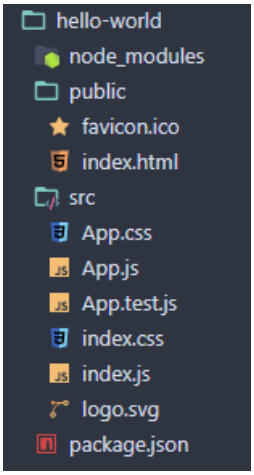
Properties, förenklat props, används för att leverera data mellan komponenter. Dataöverföringen går enbart i riktning från parent-komponent till child-komponent, eller till komponenter på samma nivå. Props är inte modifierbara. Komponenter har även ett state-objekt som lagrar props. När state-objektet ändras så renderas komponenten på nytt. Till skillnad från props så kan komponenter inte leverera data med state, utan ändamålet med state är att hantera data. Figur 3 ger ett exempel för användning av state.

```
class Car extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        brand: "Alfa Romeo"
      };
    }
    render() {
      return (
        <div>
          <h1>This is the car: {this.state.brand}</h1>
        </div>
      );
    }
  }
```

Figur 3. Användning av state.

3.2.5 Struktur

Projektets struktur (Figur 4) måste innehålla en index.js-fil i mappen src, respektive en index.html-fil i mappen public för att man ska kunna göra en kompilation av koden. Alla filer med extensionerna css och js ska även de placeras under src för att webpack, den statiska module bundlern som används i React, ska kunna nå dem när den genererar en eller flera bundler. Med bundling sammanfogas flera js-filer för att minska på antalet serverförfrågningar efter dem. [5].



Figur 4. Typexempel på projektets grundstruktur.

4 Ramverk

Ett ramverk ger strukturen och direktiv för en applikation som ska skapas för ett specifikt användningsmål. Att använda sig av ett ramverk kan främja utvecklingen, då utvecklaren kan fokusera på högnivåfunktioner medan ramverket hanterar lågnivåfunktionalitet. Ramverken kan innehålla stödprogram, kodbibliotek, kompilatorer, och en API för att sammanknyta de olika delarna till en fungerande helhet.

Jämfört med ett programbibliotek har ramverk IoC, expansionsmöjlighet och icke-modifierbar ramverkskod. IoC, Inversion of Control, är antingen en utvecklingsmetod eller ett designmönster för hantering av beroenden och deras konfigurering för ett visst objekt. Expansionsmöjligheten medför att nya kodtillägg påverkar den tidigaravarande minimalt.

4.1 Webbramverk

Webbramverk är ramverk avsedda för utvecklingen av webbapplikationer och tillhandahåller en teknik för att skapa, strukturera och distribuera dessa applikationer till internet. De kan delas upp i två kategorier; server- och klientbaserade.

Serverbaserade webbramverk fyller flera användningsområden. De kan ta hand om databashantering- och manipulering, URL-mapping och HTTP-förfrågningar, samt HTML, XML och JSON utmatningsformat med en template engine.

Ett klientbaserat webbramverk verkar inuti webbläsaren. Till React och React Native finns det en mängd konkurrerande klientbaserade webbramverk såsom Vue.js, Ember.js och Angular.

4.2 React Native

React Native är ett ramverk ämnat för utveckling av native mobilapplikationer till Android och iOS. Stora delar av koden kan återanvändas till båda operativsystemen, men en del funktionalitet och komponenter är dock plattformsspecifika. React-biblioteket implementeras av React Native i sitt ramverk.

Enkelt sammanfattat är React Native en uppsättning av React komponenter, där varje komponent motsvarar en jämförlig native view och native komponent. En native view är ett element som kan visa text, bilder eller ta emot användarinmatning. [7].

4.2.1 Rendering

I motsats till en del andra tillvägagångssätt för plattformsoberoende applikationsutveckling använder sig inte React Native av den traditionella kollektionen. Den traditionella kollektionen består av HTML som strukturerar innehållet, CSS som sköter stylingen och JavaScript som hanterar events i UIWebView samt WebView. I stället använder sig React Native av iOS och Androids egen native API. JavaScript körs på värdens JavaScriptmotor vid exekveringen av applikationen. Med hjälp av JavaScript nås denna native API för iOS eller Android, för att rendera begärda komponenter. Renderingen av komponenter kan göras med upp till 60 bildrutor per sekund, vilket WebView-baserade verktyg inte klarar av. Se figur 5 för ett exempel av renderingen.

```
import React, { Component } from 'react';
import { Text } from 'react-native';
import { View } from 'react-native';

class Dog extends Component {
  render() {
    return (
        <View>
            <Text>Hello, I am your dog!</Text>
        </View>
    );
  }
}
export default Dog;
```

Figur 5. Rendering av komponent i React Native.

4.2.2 Syntax

Syntaxen i React Native är väldigt lik den som används i React, som ersättning för , <p> och <img> används <View>, <Text> och <Image>. Stylingen av applikationen görs med hjälp av JavaScript. Även i det fallet är syntaxen lik vanliga CSS förutom att namn skrivs med kamelnotation, så att exempelvis font-weight och font-size blir fontWeight och fontSize.

5 Metoder för delning av kod

Delningen och struktureringen av koden mellan en webbapplikation och en mobilapplikation kan göras på ett flertal olika sätt, varav alternativen jämförs i detta kapitel.

5.1 ReactXP

Microsoft lanserade 2017 biblioteket ReactXP som är implementerat med TypeScript och verkar som ett lager ovanpå React och React Native. Ändelsen XP står för X-Platform och det är designat för att förenkla utvecklingen av plattformsoberoende applikationer med mottot ”write once, deploy anywhere”. Plattformarna som stöds är Android, iOS, webbaserade och Universal Windows Platform.

ReactXP ger användaren tillgång till API:s och komponenter som fungerar på alla plattformar, eftersom de med plattformsspecifik funktionalitet lämnas bort. Dessa API:s och komponenter utgår ifrån React Native. Applikationerna som skapas med ReactXP kommer att köras likadant oberoende av vilken plattform de kodas och felsöks på.

Vid tidpunkten för detta examensarbete var den senaste versionen, 2.0.0, av ReactXP daterad till den 30 november 2019. Eftersom inga uppdateringar har släppts på över ett år sedan det så kan man anta att Microsoft har lagt utvecklingen för projektet på is. Att använda sig av ReactXP i detta examensarbete skulle därmed inte vara ett bra alternativ på lång sikt, eftersom dess säkerhet för framtiden verkar vara oviss. Saknaden av någon form av långvarigt underhåll skulle i så fall troligtvis leda till att man måste byta till en annan teknik om ReactXP inte hänger med i utvecklingen av React och React Native. [8].

5.2 React Native for Web

Ett annat alternativ för att bara använda en kodbas för flera olika plattformar är React Native for Web. Den här varianten ger möjligheten till att använda en stor del av React Natives API:s och komponenter i webbapplikationer med hjälp av användandet av ReactDOM. Eftersom koden ska köras på flera plattformar är det bättre att hålla sig till det restriktivare React Native som grund. Om React Native for Web skulle användas borde en befintlig React Native applikation fungera som en webbapplikation direkt utan större problem.

Hastigheten för renderingen av användargränssnittet i mobila applikationer är som bäst när den görs på native sidan. Då det här uteblir i React Native for Web för kompatibilitet med

webbapplikationer kommer prestandan att bli sämre. Ytterligare en nackdel med att använda React Native for Web är att plattformarnas varierande skärmstorlekar gör att användargränssnittet måste anpassas med olika villkor, för att få ett korrekt utseende som tar nytta av hela skärmens storlek. [9].

5.3 Hybridapplikation

En hybridapplikation, bestående av HTML, CSS och JavaScript konverteras till en native applikation med verktyg som exempelvis Cordova, PhoneGap eller via Android Studio. Denna typ av applikation använder de mobila plattformarnas inbyggda browsers för att köra och visa applikationen, UIWebView i iOS och WebView i Android.

Alla typer av bibliotek och ramverk för webbapplikationer fungerar med denna metod, så en webbapplikation i React kan smidigt konverteras till en mobilapplikation. Om några särskilda hårdvaru- eller plattformsfunktionaliteter i mobiltelefonen behövs kan dessa exempelvis med Cordova importeras som plugins. Plugins består av JavaScriptgränssnitt som motsvarar inbyggda kodbibliotek för varje plattform. På detta sätt behövs endast en kodbas för programmet vilket gör det lättare att underhålla applikationerna för framtiden.

Dock är prestandan för en hybridapplikation märkbart sämre jämfört med en native mobilapplikation, eftersom dess hastighet beror på plattformarnas inbyggda browsers optimering. Ett annat problem är buggar och att stylingen av en applikation kan variera på olika sätt mellan plattformarnas specifika browsers. Därtill är även supporten kring dessa verktyg för hybridapplikationer nedgående, eftersom de flesta större aktörer och intressegrupper har alltmer övergått till att använda React Native.

5.4 Koddelning med npm-paket

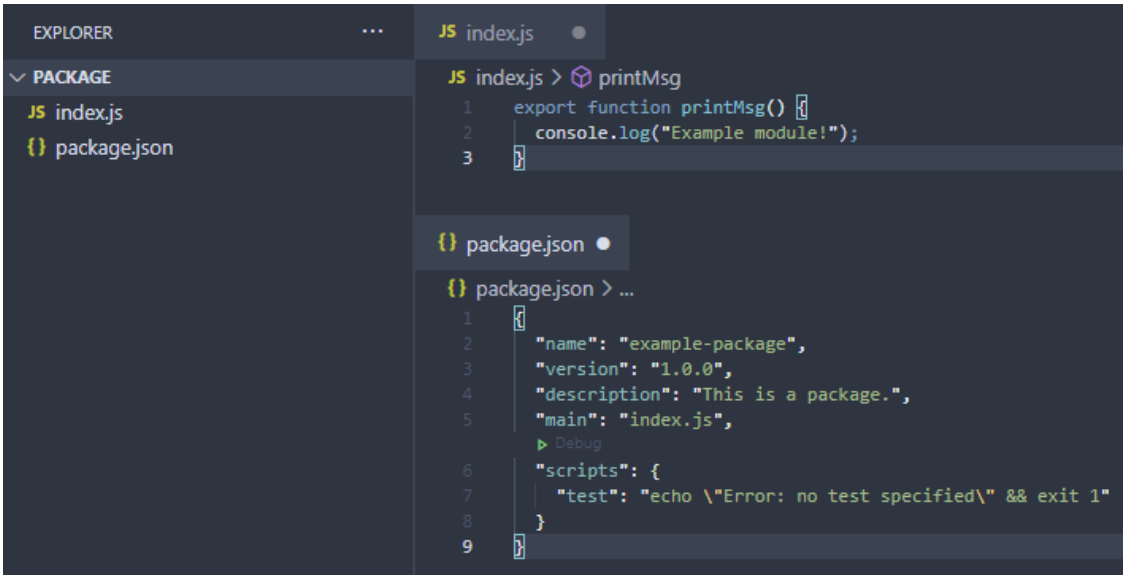
Node Package Manager, nuförtiden npm, är en pakethanteraren för JavaScript och följer med vid installation av Node.js. Den består av en CLI för nerladdning och installation av mjukvara, främst från det officiella npm-registret, som är en databas bestående av JavaScriptpaket. Utöver det officiella npm-registret finns även privata alternativ.

Ett npm-paket (Figur 6) är en eller flera modules grupperade tillsammans. Paketets definitioner måste vara gjorda i filen package.json, där åtminstone namn och version måste anges. En npm module är en antingen en JavaScriptfil eller ett directory tillsammans med en

package.json-fil. som kan laddas med Node.js require() eller import-funktion. Alla modules är paket, men alla paket är inte ämnade att användas som modules.

Lagring av kod i en npm module bestående av endast ren JavaScript skulle vara ett tillvägagångsätt för att dela funktionell kod mellan React och React Native. Om en module exempelvis använder sig av HTML så skulle den inte vara kompatibel med React Native, så all kod för rendering går inte att delas. Dessa modules ska kunna stå som grundstenar till applikationerna och laddas in enligt behov.

Då koden har blivit separerad till modules blir underhållet till en början mycket större, eftersom varje module måste underhållas enskilt och en förändring som berör alla modules därmed tar längre tid att verkställa. Därtill krävs ett betalt individuellt konto och ett organisationskonto för att dela privata npm-paket via det officiella npm-registret, men det är även möjligt att publicera dessa genom andra tjänster, till exempel GitLab. [10].

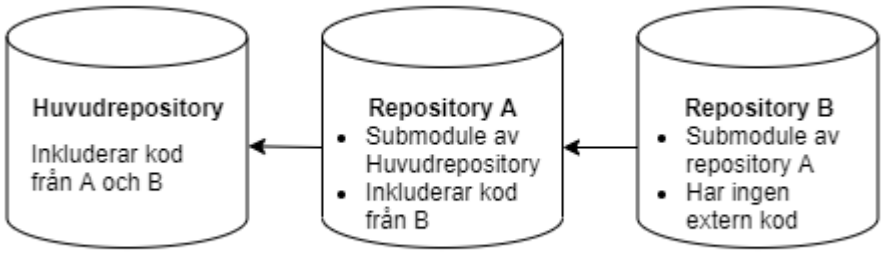


Figur 6. Exempel för ett npm-paket.

5.5 Git Submodules

Git Submodules låter ett git repository användas som ett subdirectory för ett annat git repository. På det viset kan ett repository klonas in till flera olika projekt och hålla förändringar separata från den version som används vid kloningen (Figur 7). Uppdelandet av kod gör på så sätt en komponent till en submodule, med en mera organiserad git logg, eftersom denna logg blir knuten till per komponent. Exempelvis när ett bibliotek som används måste anpassas till det projekt som implementerar det, vore det lättare att ta i bruk Git Submodules tack vare dessa egenskaper.

Olägenheter med Git Submodules är bland annat att inlärningsnivån är hög och att Git inte är skapat för att hålla reda på relationer mellan komponenter eller för att hantera dependencies. Därtill kan problem uppstå vid byte av brancher som innehåller submodules, eftersom vid ett byte från en branch med en submodule till en branch utan submodule, lämnar ett directory kvar som ett ospårat directory i den submodule som användes. [11].



Figur 7. Överblick av Git Submodules.

5.6 Monorepo

Ett monorepo är en utvecklingsstrategi där kod för flera projekt lagras i ett och samma directory. Då en komponent ska delas placeras den på samma nivå som projekten, därefter importeras den enligt behov (Figur 8). Flera individuella team kan arbeta på sina egna projekt och behändigt återanvända delade komponenter.

En nackdel med ett monorepo är att det uppstår säkerhetsproblem och problem med versionshanteringen. Säkerheten blir försvagad då varje projekt har läsrättigheter till allting inom sitt directory. Monorepo lämpar sig inte heller för många och stora projekt, därför att versionshanteringen blir tidskrävande när många förändringar kan göras på en dag.

Det finns flera verktyg för att implementera ett monorepo, varav de flesta har liknande funktioner sinsemellan.

- Yarn Workspaces är ett alternativ som kan sammanlänka flera npm dependencies till ett workspace för att dela det mellan flera projekt. Yarn Workspaces verkar med lågnivåfunktioner och kan kombineras med andra verktyg som erbjuder högnivåfunktioner. [12].

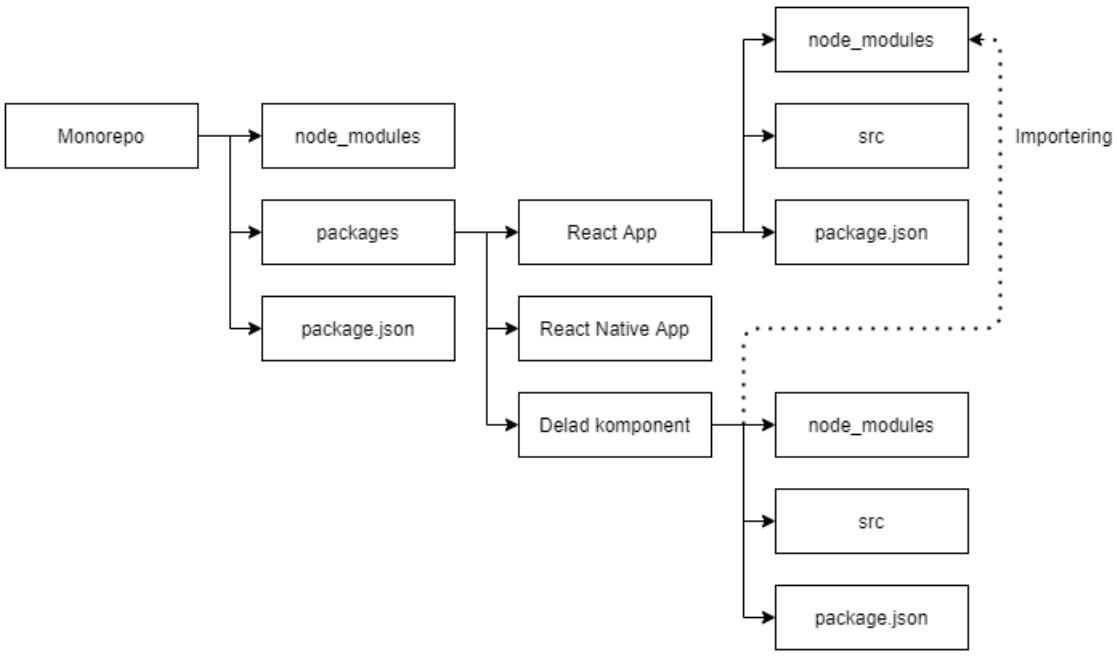
- Lerna använder sig av Git och npm. Strukturen är lös, vilket gör det möjligt att använda en blandning av de flesta ramverk och programmeringsspråk när deras paket

definieras enskilt i varje egen package.json. Bland annat versionshantering, val av npm-skript för testning och publicering av paket är några av Lernas funktioner. Tillsammans med Yarn Workspaces som grund för ett monorepo och dessa funktioner kan det skapas en bra synergi. [13].

- Bit baserar sig på import av isolerade komponenter till projekt. Med Bit läggs ett semantiskt lager till över det repository som hänvisar till komponenterna. En Bit komponent kan vara antingen en komponent gjord i React, en delad stilmall (CSS/SCSS) eller en funktion. En komponent förpackad med Bit är ett giltigt npm-paket och Bit fungerar även tillsammans med Git, npm och Yarn. [14].

- Nx är en uppsättning av verktyg som förenklar utvecklingsflödet och stöder React samt React Native. Kommandon som exempelvis build och test har möjlighet att endast köras på kod som påverkas an en aktuell ändring, detta ger en kortare exekveringstid. [15].

- Rush, utvecklat av Microsoft, är en skalbar hanterare för ett monorepo. Alla dependencies installeras i en gemensam mapp. Inuti ett repository sammankopplas alla projekt automatiskt med varandra med symboliska länkar. Likt NX så kan build kommandot exekveras endast på kod med en aktuell ändring. [16].



Figur 8. Exempel på struktur med monorepo.

6 Planering av implementation

Delningsmetoden som valdes blev användandet av npm-paket med modules. Den främsta anledningen var att i ett redan fungerande projektsystem som redan använder sig av npm modules så faller det naturligt att dela upp och plocka ut befintlig kod till olika paket, som därefter importeras när de behövs. Eftersom Comsel System ibland behöver anpassa applikationerna till sina kunder så är det enklare att bara importera komponenter med den version som behövs än att börja anpassa dessa utifrån ett monorepo. Npm har också ett stort stöd från utvecklare och uppdateras kontinuerligt, vilket kan garantera att livslängden på projekten hålls långa. Comsel System har därtill egna servrar, så kostnaden för publicering av privata npm-paket blir inte en negativ faktor. Metoden är dessutom prövad i projekt såsom React Router [17].

Att använda sig av mindre populära tekniker som ReactXP och React Native for Web skulle leda till att man förlitar sig på en tredje part för underhåll och fortsatt utveckling. I värsta fall måste delningsmetoden bytas till ett annat alternativ om React eller React Native vid en uppdatering förlorar kompatibilitet med någondera av de nämnda teknikerna.

Hybridapplikationer bygger på en kodbas och är en acceptabel lösning om applikationernas egenskaper är spartanska. Har man native-baserade applikationer för flera plattformar blir underhållet mycket mera tidskrävande, vilket hybridapplikationer skulle vinna på. Till nackdelarna hör att hybridapplikationer lider av sämre prestanda jämfört med native-baserade lösningar. Dessutom tillkommer begränsad funktionalitet eftersom de inte kan ta del av native-funktioner som introduceras med tiden efter, en API för exempelvis kameran kan saknas.

Med Git Submodules vore det möjligt att separera komponenterna till flera submodules, som sedan kan läggas till i de repositories som behöver dem. Att Git Submodules lägger till extra komplexitet till den versionshantering som används vid utvecklingen vore dock en nackdel.

Av de övriga tillvägagångssätten för att dela kod kan det därmed konstateras att det alternativ som låg närmare till hands än de andra var monorepo. Att konvertera nuvarande projekt till ett monorepo skulle kräva mera tid och versionshanteringen blir tidsmässigt utdragen på grund av dess storlek.

6.1 Utgångsläge

För att kunna utföra examensarbetets uppgift krävs följande: Android Studio, Node.js, React, React Native, TypeScript och en valfri IDE.

Android Studio behövs för att möjliggöra testning av mobilapplikationer gjorda i React Native, som dessutom kräver Java SE Development Kit (JDK). I Android Studio måste komponenterna Android SDK, Android SDK Platform och Android Virtual Device installeras och aktiveras.

Node.js är nödvändigt för att använda npm och npx, som i sin tur behövs för att skapa en React Native applikation och använda Create React App. Create React App är ett verktyg för att smidigt skapa grunden för en React applikation. Tanken med deras användning är att först göra lokala applikationer som kan testas med ett lokalt skapat npm-paket, som importeras till applikationerna och delar kod. Koden som delas tas från existerande projekt från Comsel System. Efter att ha säkerställt att npm-paketet fungerar som den är avsedd att göra, kommer den att publiceras till Comsel Systems egna privata npm-register.

Kodexempel 1. Kommando för nytt projekt i React Native med TypeScript

```
npx react-native init mobileapp --template react-native-template-typescript
```

Kodexempel 2. Ny React applikation med TypeScript

```
npm init

npm install create-react-app

npx create-react-app webapp --template typescript
```

TypeScript används av Comsel System i företagets applikationsutveckling och kommer även därmed användas i kodningen av npm-paket och modules. TypeScript kan antingen installeras globalt eller lokalt för ett projekt.

Kodexempel 3. Global installation av TypeScript

```
npm install -g typescript
```

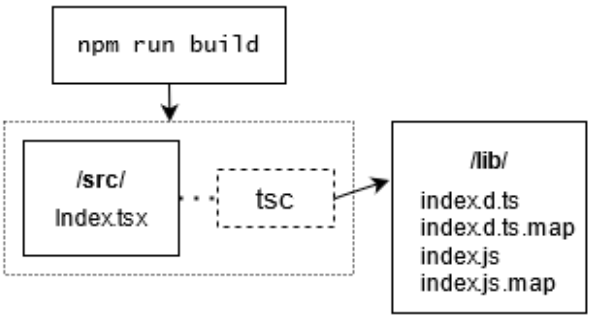
Kodexempel 4. Lokal installation av TypeScript

```
npm install typescript --save-dev
```

6.2 Struktur för ett npm-paket

Den struktur som byggs upp för ett npm-paket gjort i TypeScript är väldigt likt ett normalt npm-paket. Eftersom TypeScript används måste man utöver filen package.json även inkludera tsconfig.json. Filändelsen för .jsx-filer ska även bytas till .tsx.

För det här examensarbetet blir innehållet i ett npm-paket en package.json, tsconfig.json, en mapp vid namn src med index.tsx (som är ett valfritt namn) och en mapp kallad lib, var den kompilerade koden hamnar tillsammans med .map-filer, se figur 9. Dessa .map-filer innehåller information och länkar varje del av den kompilerade JavaScriptkoden tillbaka till rätt rad i motsvarande TypeScriptfil. Felsökningen är då möjlig i TypeScript i stället för i JavaScript. Index.tsx innehåller den kod som kommer att delas eller så exporterar den vidare .tsx-filer från samma katalogsnivå.



Figur 9. Index.tsx kompilerad till index.js med .map-filer.

I package.json är följande alternativ valda: name, version, main, types, files, scripts (build). Main specificerar vilken fil som är entry fil till programmet. För types ska den peka mot var den bundlade deklarationsfilens plats är. Till files väljs den folder, i det här fallet lib, som ska inkluderas i projektet. I build under scripts läggs kommandot *tsc -p* . . Vid exekvering av kommandot *npm run build* så kommer projektet att kompileras.

I tsconfig.json är följande för compilerOptions givet: target, module, jsx, declaration, declarationMap, sourceMap, outDir, rootDir, strict, esModuleInterop, skipLibCheck och forceConsistentCasingInFileNames. Även include och exclude med respektive src och lib samt node_modules anges (Figur 10).

Kommandot *npm pack* kan köras efter en lyckad kompilering av koden för att lokalt skapa en arkiverad fil med formatet .tgz. Denna kan behändigt installeras på applikation för testning med *npm install .//..//test-package-1.0.0.tgz*. När man kör *npm publish* skapas samma arkiverade fil med *npm pack* som ett underkommando, men skillnaden är att filen efteråt publiceras till det register som man har valt.

Figur 10. Överblick av tsconfig.json.

6.3 Olika typer av dependencies

När ett npm-paket installeras till ett projekt installeras det som en dependency. En dependency kan ses som ett paket som programmet är beroende av. Dessa kan ses i package.json-filen under kategorin dependencies. Förutom dependencies finns det devDependencies och peerDependencies. Den förstnämnda samlar upp paket som installeras med taggningen *-D* eller *--save-dev*. Avsikten med sådana paket är att bara använda de vid utvecklingen och de uteblir från lanseringen, hit hör exempelvis testpaket. Till peerDependencies kan det anges vilka paket och med vilken version som host-applikationen måste ha installerade för att ett paket ska fungera felfritt vid installation. [2].

7 Implementering

Implementering kräver först en utredning av vad som kan delas från existerande kod som används av en applikation utvecklad med React och React Native. Det som kommer att delas är till största delen idel TypeScriptkod, såsom klasser och funktioner.

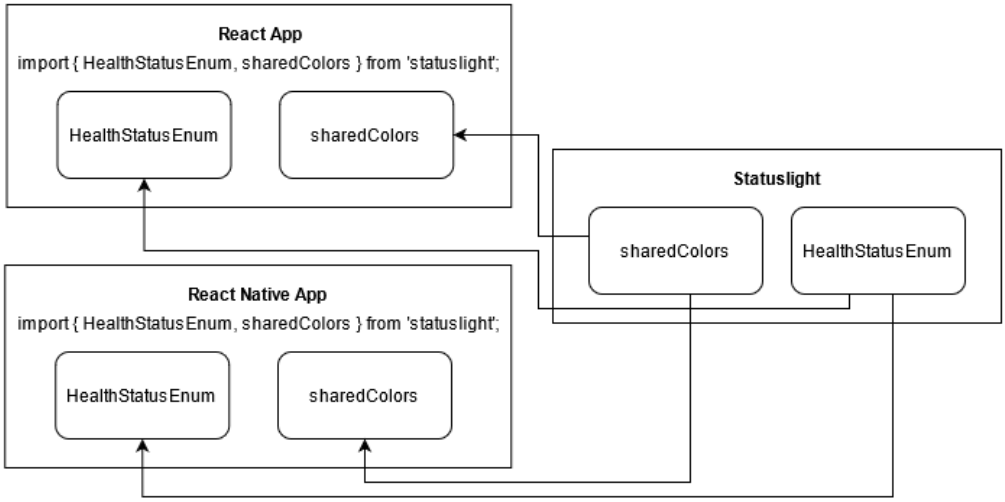
Alla renderingsfunktioner kommer att hållas separata enligt applikationstyp. Stylingen för element med exempelvis typsnittsstorlekar, marginaler och positionering kommer därmed inte delas, eftersom användargränssnittet kommer att anpassas enligt enhetens skärmstorlek. En webbsida och mobilapplikation skiljer sig i hög grad var element placeras ut på grund av olika navigeringssätt, och användarupplevelsen påverkas av hur bra utformningen är gjord. Det kan tilläggas att fastän syntaxen är lik mellan stylingen i React och React Native så är skillnaden så pass påverkande att det medför flera problem att dela stylingen än att behålla dem separerade. Däremot skulle färgkoder kunna delas för att hålla en konsekvent stil och för att behändigt kunna modifiera dem efter behov, exempelvis byts temafärger då genom att modifiera några variabler i en module.

De npm-paket som skapas kommer att kräva en del dependencies, exempelvis kan nämnas Material-UI och react-native-svg. Versionerna för dessa måste hållas under uppsyn för att garantera kompatibilitet mellan alla applikationer och npm-paket.

Underrubrikerna 7.1 och 7.2 ger insyn av hur koddelningen går till i praktiken. När det som ska delas har valts ut placeras den kompatibla delen av funktionaliteten i en ny module. Efter testning att det gjorda npm-paketet fungerar och är kompatibelt, publiceras det till ett privat npm-register som Comsel System innehar. Därefter upprepas denna procedur för varje delbar kod mellan företagets applikationer. De efterföljande underrubrikerna ger därför en god bild utav hur den praktiska delen går till, eftersom det som sker upprepas på nästan identiskt tillvägagångsätt för alla andra komponenter i kodbasen.

7.1 Oberoende paket

Härefter följer en tydlig presentation av hur en del av koden, som hanterar status och anger standardfärger för en specifik ikon, plockas ihop in i en module därefter sedan till ett paket. Figur 11 visar delningen från detta Statuslight-paket till mobil- och webbapplikationen.



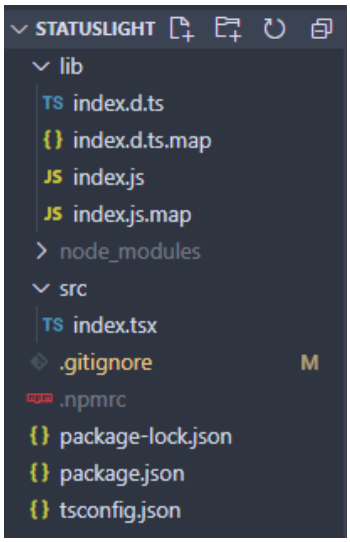
Figur 11. Den delade koden i en module.

Färgerna importeras från Material-UI och definieras, statuskonstanterna anges i uppräkningstypen HealthStatusEnum. Export används för att ett utomstående program ska kunna importera och använda dessa, vilket visas i figur 12.

```
src > TS index.tsx > HealthStatusEnum
1   import grey from '@material-ui/core/colors/grey';
2   import red from '@material-ui/core/colors/red';
3   import yellow from '@material-ui/core/colors/yellow';
4   import green from '@material-ui/core/colors/green';
5
6
7   export const sharedColors = {
8       unknown: grey[500],
9       error: red[500],
10      warning: yellow[600],
11      ok: green[600],
12      statusIcon: '#fff'
13  }
14
15  export enum HealthStatusEnum {
16      UNKNOWN,
17      ERROR,
18      WARNING,
19      OK
20  }
```

Figur 12. Den delade koden i en module.

Exempelvis vid ett byte av färgvärden skulle denna förändring bara behöva göras på ett ställe för att påverka både webb- och mobilapplikationen. Det slutgiltiga utseendet på filstrukturen för ett npm-paket i TypeScript kan ses i figur 13.



Figur 13. Filstruktur för paketet.

I webb- och mobilapplikationen importeras därefter behövligt innehåll för användning från den module som innehöll sharedColors och HealthStatusEnum, se figur 14 och 15.

```
import { HealthStatusEnum, sharedColors } from 'statuslight';

const styles = (theme: Theme) => createStyles({
  statusLight: {···
  },
  statusIcon: {···
  },
  unknown: {
    background: sharedColors.unknown
  },
  ok: {
    background: sharedColors.ok
  },
  warning: {
    background: sharedColors.warning
  },
  error: {
    background: sharedColors.error
  },
});

type Props = WithStyles<typeof styles> & {
  status: HealthStatusEnum,
  tooltip?: string,
};
```

Figur 14. Användning av koden i webbapplikationen.

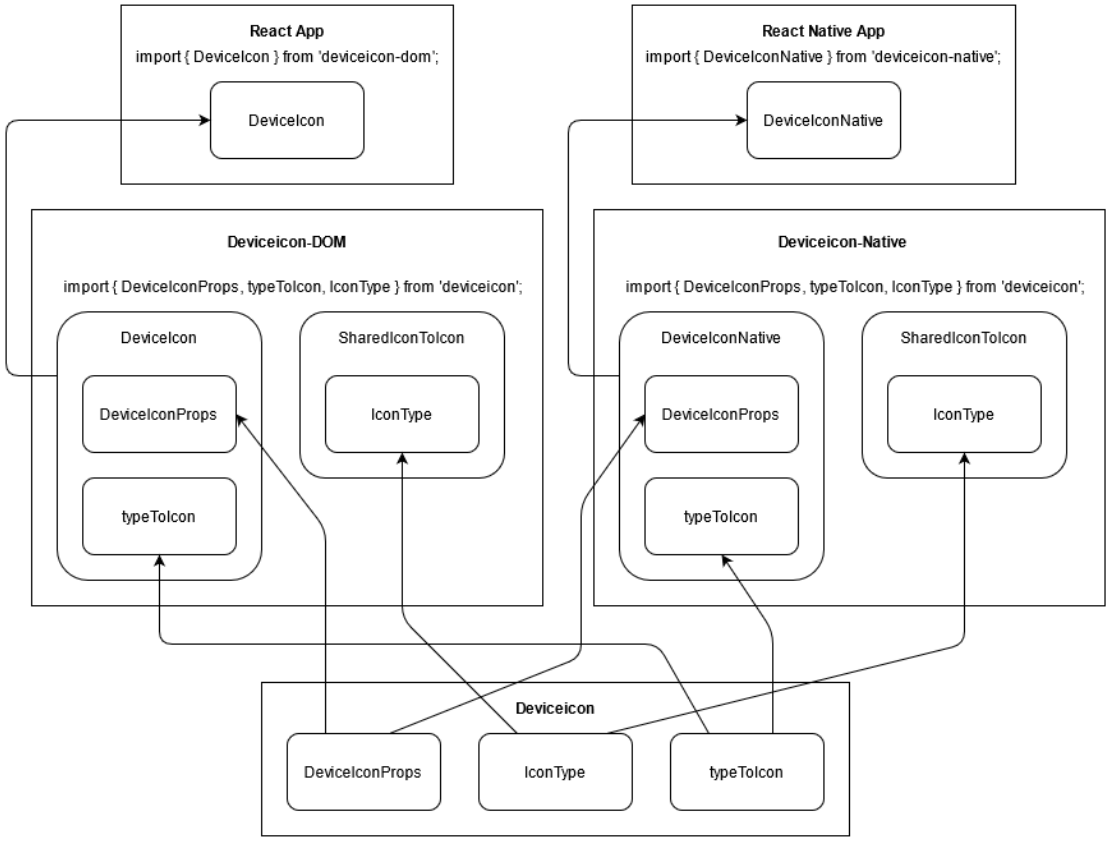
TS StatuslightNative.tsx > ...

```tsx
import { HealthStatusEnum, sharedColors} from 'statuslight'

interface Props {
  status: HealthStatusEnum;
}

export const StatusLight = (props: Props) => {
  const theme = useContext(ThemeContext);
  let background = {backgroundColor: themes.disabled.background};
  if (props.status===HealthStatusEnum.OK) {
    background.backgroundColor = sharedColors.ok
  }
  if (props.status===HealthStatusEnum.WARNING) {
    background.backgroundColor = sharedColors.warning
  }
  if (props.status===HealthStatusEnum.ERROR) {
    background.backgroundColor = sharedColors.error
  }
  if (props.status===HealthStatusEnum.UNKNOWN) {
    background.backgroundColor = sharedColors.unknown
  }

  return <View style={[styles.circle, background]}>
    {props.status===HealthStatusEnum.UNKNOWN? <Text style={styles.unknownStatusIcon}>?</Text> : <></>}
    {props.status===HealthStatusEnum.OK? <Icon name="done" color="#fff" /> : <></>}
    {props.status===HealthStatusEnum.WARNING? <Text style={styles.unknownStatusIcon}>!</Text> : <></>}
    {props.status===HealthStatusEnum.ERROR? <Icon name="clear" color="#fff" /> : <></>}
  </View>;
};
```

Figur 15. Användning av koden i mobilapplikationen.

7.2 Paket beroende av andra paket

I det här andra fallet skapas ett paket som innehåller ikoner som används till gränssnittet för mobil- och webbapplikationer. Eftersom renderingen fungerar på olika sätt mellan webb- och mobilapplikationer placeras renderingsfunktionen i två skilda, men tämligen lika paket (Figur 16).



Figur 16. Översikt av delningen.

Paketet Deviceicon innehåller props, alla ikontyper och en funktion som ger en vald typ dess rätta ikon. Deviceicon-DOM och Deviceicon-Native paketen importerar DeviceIconProps (Figur 17), typeToIcon, och IconType (Figur 18) från Deviceicon-paketet. De två förstnämnda används i deras egna DeviceIcon och DeviceIconNative, medan den sistnämnda, IconType, behövs för SharedIconToIcon. Figur 19 visar allt inuti DeviceIcon.

src > TS DeviceIconProps.tsx > ...

```tsx
import { GraphNode, Unit} from "corona";

export interface DeviceIconProps {
    type: GraphNode|Unit,
    color?: any,
    size?: number
}
```

Figur 17. DeviceIconProps.

src > TS IconHandling.tsx > ...

```tsx
import { DeviceType, Model, Medium } from "corona";

export enum IconType {
    UNKNOWN,
    Globe,
    Home,
    DeviceHub,
    PhoneInTalk,
    Oculus,
    Zodiac,
    Gemini,
    EVStation,
    Memory,
    FlashOn,
    WhatsHot,
    Opacity,
    Speed,
    AcUnit,
    DirectionsBus,
    SmokingRooms,
    Thermometer,
    Iso,
    PowerSettingsNew,
    Delete,
    PVArray,
    C
}

export function typeToIcon(type: DeviceType, model: Model): IconType {···
}
```

Figur 18. IconType och typeToIcon.

src > TS DeviceIcon.tsx > ...

```tsx
import { DeviceIconProps, typeToIcon, IconType } from 'deviceicon';

interface Props {
  icon: IconType,
  color: any,
  size: number
}

function SharedIconToIcon(props: Props): JSX.Element {···
}

export function DeviceIcon(props: DeviceIconProps) {
  return <SharedIconToIcon icon={typeToIcon(props.type.deviceType, props.type.model)} color={props.color} size={props.size!} />
}
```

Figur 19. Innehållet för DeviceIcon.

De flesta ikoner finns färdigskapade i Material-UI, medan en handfull är skräddarsydda och läggs till manuellt i Deviceicon-DOM och Deviceicon-Native, se figur 20 och 21. Ikonerna är i SVG-format och behöver tredjepartspaketet react-native-svg för att fungera i React Native. Här märks det att vissa små förändringar måste göras, även om koden är väldigt lik.

src > icons > devices > TS ThermometerIcon.tsx > ...

```tsx
import React from 'react';
import SvgIcon, { SvgIconProps } from '@material-ui/core/SvgIcon';

function ThermometerIcon(props: SvgIconProps) {
  return (
    <SvgIcon {...props}>
      <path d="M15 13V5A3 3 0 0 0 9 5V13A5 5 0 1 0 15 13M12 4A1 1 0 0 1 13 5V8H11V5A1 1 0 0 1 12 4Z" />
    </SvgIcon>
  );
}

export default ThermometerIcon;
```

Figur 20. SVG-ikon i React.

icons > devices > TS ThermometerIcon.tsx > ...

```tsx
import React from 'react';
import { View } from 'react-native';
import Svg, { Path } from 'react-native-svg';

type Props = {
  color: any,
  size: number
}

function ThermometerIcon(props: Props) {
  return (
    <View style={{width: props.size, height: props.size}}>
      <Svg width="100%" height="100%" viewBox="0 0 23 23">
        <Path fill={props.color} d="M15 13V5A3 3 0 0 0 9 5V13A5 5 0 1 0 15 13M12 4A1 1 0 0 1 13 5V8H11V5A1 1 0 0 1 12 4Z" />
      </Svg>
    </View>
  );
}

export default ThermometerIcon;
```

Figur 21. SVG-ikon i React Native.

Till sist kan ikonerna renderas i applikationerna med DeviceIcon från antingen Deviceicon-DOM eller Deviceicon-Native, beroende på den plattform som används. Figur 22 och 23 visar hur en modell genereras som ger korresponderande ikoner. I figur 24 importeras komponenterna till respektive applikationen och resultatet av de två renderade ikonerna, elektricitets- och G3-ikon syns för var sin plattform i figur 25.

```tsx
src > TS iconTesting.tsx > ...
import { Unit, Model, EmptyUUID, Medium, DeviceType } from 'corona';
import { DeviceIcon } from 'deviceicon-dom';

export const IconTesting = () => {
    const modelGenerator = (medium: Medium, deviceType=DeviceType.Meter, manufacturer="Comsel", model=""): Unit => {···
    }
    const testunitG3 = modelGenerator(Medium.Unknown, DeviceType.Logging, "Comsel", "G3");
    const testunitElectricity = modelGenerator(Medium.Electricity);
    const size = 14

    return <div style={{display: "flex", flexDirection: "column", textAlign: 'left', marginLeft: '40%'}}>

        <div>G3 Icon: <DeviceIcon type={testunitG3} color={"black"} size={size} /></div>
        <div>Electricity Icon: {testunitElectricity.model.medium} - <DeviceIcon type={testunitElectricity} color="black" size={size} /></div>
    </div>;
}
```

Figur 22. Generering av elektricitets- och G3-ikon i React.

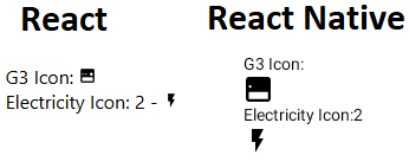
```tsx
TS iconTesting.tsx > ...
import React from 'react';
import { Unit, Model, EmptyUUID, Medium, DeviceType } from 'corona';
import { DeviceIcon } from 'deviceicon-native';
import { View, Text, ScrollView } from 'react-native';

export const IconTesting = () => {
    const modelGenerator = (medium: Medium, deviceType = DeviceType.Meter, manufacturer = "Comsel", model = ""): Unit => {···
    }
    const testunitG3 = modelGenerator(Medium.Unknown, DeviceType.Logging, "Comsel", "G3");
    const testunitElectricity = modelGenerator(Medium.Electricity);
    const size = 24
    const color = 'rgba(0, 0, 0, 1)';

    return (
    <ScrollView  style={{ display: "flex", flexDirection: "column" }}>
        <View><Text>G3 Icon:</Text><DeviceIcon type={testunitG3} color={color} size={size} /></View>
        <View><Text>Electricity Icon:{testunitElectricity.model.medium}</Text><DeviceIcon type={testunitElectricity} color={color} size={size} /></View>
    </ScrollView>
    )
}
```

Figur 23. Generering av elektricitets- och G3-ikon i React Native.

```tsx
src > TS App.tsx > ...
import { IconTesting } from './iconTesting';

function App() {

  return (
    <div className="App">
      <h1>Test</h1>
        <IconTesting />
    </div>
  );
}

export default App;
```

```tsx
TS App.tsx > ...
import { IconTesting } from './iconTesting'

declare const global: { HermesInternal: null | {} };
/* ···
const App = () => {
  return (
    <>
      <SafeAreaView>
        <View style={styles.body}>
            < IconTesting />
        </View>
      </SafeAreaView>
    </>
```

Figur 24. Användning av testikonerna i applikationerna.



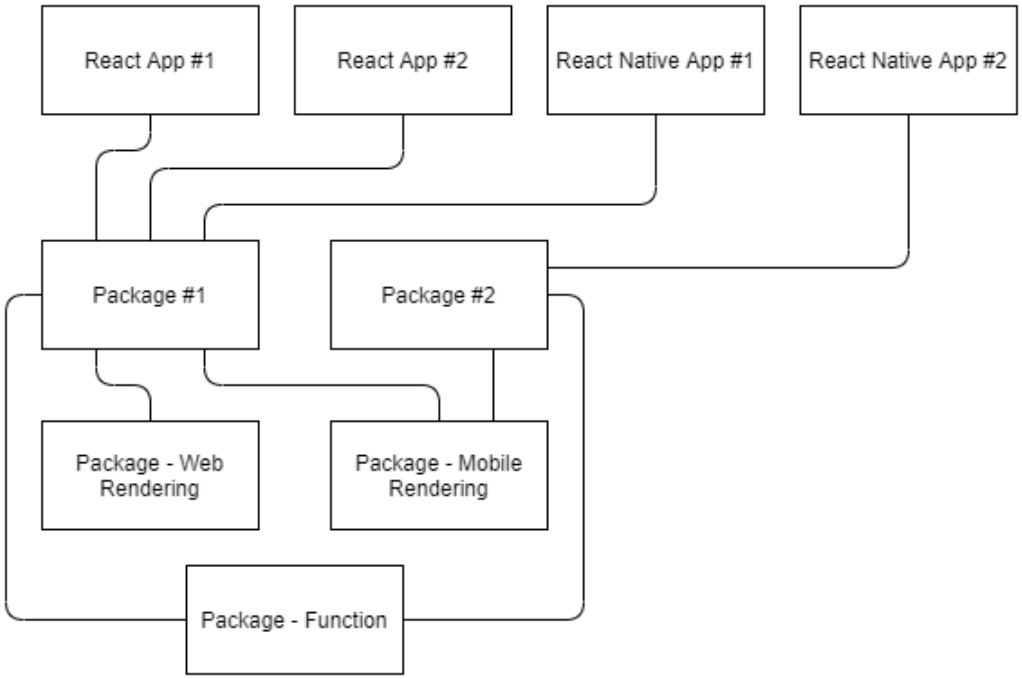
Figur 25. Ikonerna renderade.

8 Resultat

I en tid som formas alltmer av digitalisering och där både webb- och mobilapplikationer spelar en väsentlig roll, är det viktigt att utvecklingen och underhållet av dessa kan utföras så smidigt och effektivt som möjligt. Examensarbetets ändamål var att komma fram till ett fungerande system för koddelningen som skulle stödja dessa syften.

Enligt överenskommelse med Comsel System efter en diskussion blev koddelningen gjord med npm-paket. På basis av de erfarenheter projektet har gett hittills kan det konstateras att metoden som används ger ett funktionellt resultat. Därmed finns det goda förutsättningar för en fortsatt användning av metoden i framtiden. Med andra ord kommer koddelningen av Comsel Systems existerande kodbas förhoppningsvis gå igenom den fungerande modellen för delningsprocessen med npm-paket.

I framtiden kan man förvänta sig att paketen växer till antalet. För att främja framtida utveckling av projekt är ett förslag till fortsatt utveckling ett skapande av någon form av dokumentation eller övergripande relationsschema, se exempel i figur 26. Dessa skulle ge en helhetsbild av de olika paketens funktioner och av deras inbördes relation till varandra. Annars kan det bli alltför bekymmersamt att kontrollera kompatibilitet för paketen sinsemellan.



Figur 26. Enkelt relationsschema.

9 Sammanfattning

Examensarbetets syfte var att i samarbete med Comsel System komma fram till ett lösningsförslag för delning av kod mellan företagets nuvarande applikationer byggda på React och React Native, samt genomföra förslaget. Tillvägagångssättet som valdes blev användandet av npm-paket och resultatet var en välfungerande metod för delning av kod.

Detta projekt har till stor del varit rakt på sak och redan från början formades en god bild utav hur examensarbetet skulle genomföras, både teoretiskt och praktiskt. Trots det vill jag även lyfta fram de hinder och de lärdomar jag har tagit åt mig under hela processen.

Mängden fysiska källor som behandlar React och React Native är begränsad, eftersom dessa är relativt nya teknologier som kontinuerligt uppdateras och i bokform skulle deras information snabbt bli utdaterad. Att finna digitala källor utanför de använda verktygens egna dokumentationer var väldigt mödosamt och blev en utmaning. Majoriteten av utanförståendeinformation fanns i form av webbjournaler, vars innehåll inte alltid överensstämde med den officiella dokumentationen. Det var även svårt att förklara de engelska termer och begrepp som används inom ramen för arbetet, eftersom en direkt översättning till svenska inte alltid skulle få samma innebörd. För många enskilda ord existerar inte heller någon svensk motsvarighet.

Att utarbeta kapitlet för den praktiska delen, själva implementeringen, som bestod av programmering, blev även en aning komplicerat. Dels för att innehållet enligt det exempel som nämns i texten motsvarade vad majoriteten av hur koddelningen kom att såg ut, dels för att koden var konfidentiell.

Det som dock tog upp mest tid kring den praktiska delen av arbetet var att sätta sig in i hur ett npm-paket skulle byggas upp med TypeScript, samt hur React och React Native fungerar, vilka två jag inte hade några tidigare erfarenheter av. En väldigt givande och en mycket positiv aspekt av examensarbetet har varit den kunskap som jag har erhållit under processens gång. Detta gäller både användningen av React, React Native och TypeScript i en verklig arbetsmiljö, samt hur alla delar av utvecklingsprocessen sammanfogas för att skapa en fungerande helhet som är kompatibel med företagets nuvarande applikationer.

Slutligen vill jag ge ett tack till Comsel System för att ha gett mig möjligheten att genomföra detta examensarbete, Karl Herler som handledare från företaget och Jan Berglund som verkade som handledare från Novia.

10 Källförteckning

[1] Comsel System, ”About us,” 2021. [Online]. Available: https://comselsystem.com/about.html. [Använd 3 1 2021].

[2] OpenJS Foundation, ”Node.js,” 2021. [Online]. Available: https://nodejs.org/. [Använd 3 1 2021].

[3] Mozilla, ”MDN Web Docs,” 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Använd 3 1 2021].

[4] Microsoft, ”TypeScript,” 2021. [Online]. Available: https://www.typescriptlang.org/docs/. [Använd 3 1 2021].

[5] Facebook Inc., ”React,” 2021. [Online]. Available: https://reactjs.org/. [Använd 3 1 2021].

[6] Facebook Inc., ”Draft: JSX Specification,” 2021. [Online]. Available: https://facebook.github.io/jsx/. [Använd 3 1 2021].

[7] Facebook Inc., ”React Native,” 2021. [Online]. Available: https://reactnative.dev/docs. [Använd 3 1 2021].

[8] Microsoft, ”ReactXP,” 2021. [Online]. Available: https://microsoft.github.io/reactxp/. [Använd 5 1 2021].

[9] N. Gallagher, ”GitHub: React Native for Web,” 2021. [Online]. Available: https://github.com/necolas/react-native-web. [Använd 5 1 2021].

[10] npm, Inc., ”npm Docs,” 2021. [Online]. Available: https://docs.npmjs.com/. [Använd 5 1 2021].

[11] Git, ”Git Tools - Submodules,” 2021. [Online]. Available: https://git-scm.com/book/en/v2/Git-Tools-Submodules. [Använd 20 2 2021].

[12] Yarn, ”Workspaces,” 2021. [Online]. Available: https://classic.yarnpkg.com/en/docs/workspaces/. [Använd 20 2 2021].

[13] Lerna, ”Lerna,” 2021. [Online]. Available: https://lerna.js.org/. [Använd 20 2 2021].

[14] Cocycles, ”Bit,” 2021. [Online]. Available: https://bit.dev/. [Använd 20 2 2021].

[15] Nrwl, ”Nx,” 2021. [Online]. Available: https://nx.dev/react. [Använd 20 2 2021].

[16] Microsoft, ”Rush,” 2021. [Online]. Available: https://rushjs.io/pages/intro/welcome/. [Använd 20 2 2021].

[17] React Training, ”GitHub: React Router,” 2021. [Online]. Available: https://github.com/ReactTraining/react-router. [Använd 10 3 2021].