ARCADA

Jämförelse av Capacitor och React Native för mobilapplikationsutveckling

Joel Kackur

Lärdomsprov

Informationsteknik

2024

Lärdomsprov

Joel Kackur

Jämförelse av Capacitor och React Native för mobilapplikationsutveckling.

Yrkeshögskolan Arcada: Informationsteknik, 2024.

Uppdragsgivare:

Codebite Oy

Sammandrag:

I denna avhandling undersöks övergången från webbapplikationer byggda med React till mobilapplikationer genom användning av två olika teknologier: Capacitor och React Native. Målet var att evaluera den ekonomiska och tekniska hållbarheten i att utveckla korsplattformslösningar för företaget Codebite. Studien involverade utvecklingen av två testapplikationer: en enkel lunchmeny för en pizzarestaurang och en mer komplex väderapplikation. Dessa applikationer utvecklades först som webbapplikationer med React och konverterades sedan till mobilapplikationer med hjälp av Capacitor respektive React Native. Metodologin inkluderade praktisk kodning, tidtagning av utvecklingsprocessen, analys av applikationernas prestanda, och identifiering av problem under konverteringen. Det huvudsakliga resultatet visar att medan Capacitor erbjuder en snabb och kostnadseffektiv metod för att omvandla befintlig React-kod till mobilapplikationer, medför det vissa begränsningar i navigering och designflexibilitet. Å andra sidan, medan React Native presenterar en mer tidskrävande process, ger det mer nativa och anpassningsbara applikationer. Avhandlingen konstaterar att valet mellan Capacitor och React Native bör baseras på projektets krav och komplexitet. För enklare projekt där kostnadseffektivitet och snabb utveckling prioriteras kan Capacitor vara det föredragna alternativet. För mer avancerade applikationer som kräver en högre grad av anpassning och en mer nativ användarupplevelse, är React Native att föredra.

Nyckelord:

React, React Native, Capacitor, Codebite, Android Studio, Expo, Konvertering

Degree Thesis

Joel Kackur

Comparison of Capactior and React Native for mobile application development.

Arcada University of Applied Sciences: Information Technology, 2024.

Commissioned by:

Codebite Oy

Abstract:

In this thesis, the transition from web applications built with React to mobile applications is investigated using two different technologies: Capacitor and React Native. The goal was to evaluate the economic and technical sustainability of developing cross-platform solutions for the company Codebite. The study involved the development of two test applications: a simple lunch menu for a pizza restaurant and a more complex weather application. These applications were initially developed as web applications using React and were then converted into mobile applications using Capacitor and React Native, respectively. The methodology included practical coding, timing the development process, analyzing the applications' performance, and identifying issues during the conversion. The main result shows that while Capacitor offers a fast and cost-effective method for transforming existing React code into mobile applications, it comes with certain limitations in terms of navigation and design flexibility. On the other hand, React Native, although more time-consuming, provides more native and customizable applications. The thesis concludes that the choice between Capacitor and React Native should be based on the project's requirements and complexity. For simpler projects prioritizing cost-effectiveness and rapid development, Capacitor may be the preferred option. For more advanced applications requiring a higher degree of customization and a more native user experience, React Native is preferable.

Keywords:

React, React Native, Capacitor, Codebite, Android Studio, Expo, Conversion

INNEHÅLL

1 Inledning 7
1.1 Bakgrund 7
1.2 Syfte och mål 8
1.3 Metoder och avgränsningar 9
2 Verktyg 10
2.1 React 10
2.2 React Native 12
2.3 Capacitor 13
3 Applikationernas uppbyggnad 13
3.1 Pizza-Maestro lunchmenyn 14
3.2 Tempestic 17
4 Konverteringarna 19
4.1 Hårdvaran 19
4.2 Mjukvaran 20
4.3 Konvertering till Capacitor 21
4.3.1 Konverteringsprocessen 21
4.3.2 Tidtagningen 23
4.4 Konvertering till React Native 23
4.4.1 Skapandet av applikationerna 24
4.4.2 Tidtagningen 26
4.5 Problem som uppstod 26
4.5.1 Capacitor 26
4.5.2 React Native 27
5 Testning av applikationerna 28
5.1 Buggar som uppkom 28
5.2 Hastighet på de olika plattformarna 29
6 Resultat 30
7 Slutsatser 33
Källor 35

FIGURER

Figur 1. Popularitet av frågor frågade om olika ramverk (Stack Overflow 2023). 11
Figur 2 En bild på navigationsfältet på Pizza Maestro-hemsidan. 15
Figur 3 En bild på Pizza Maestro hemsidan. 16
Figur 4 En bild på Pizza Maestro lunchmenyn på hemsidan. 16
Figur 5 En bild på Pizza Maestro om oss på hemsidan. 17
Figur 6 Bild på Tempestic hemsidan. 19
Figur 7 Bilder på Capacitor-applikationerna 22
Figur 8 Tidtagning av utveckling på Capacitor. 23
Figur 9 Bild på React Natvie applikationerna 25
Figur 10 Tidtagning av utveckling på React Native. 26
Figur 11 Prestanda av Tempestic applikationen på två olika plattformar. 30
Figur 12 Pizza Maestro applikationen, Capacitor till vänster och React Native till höger. 31
Figur 13 Tempestic applikationen, Capacitor till vänster och React Native till höger. . 32

Definitioner

API (Application Programming Interface) – Ett gränssnitt som definierar interaktioner mellan olika mjukvarukomponenter, ofta använd för att tillåta tredjepartsprogram att interagera med ett system.
APK (Android Package) – Ett paketfilformat använd för distribution och installation av mjukvara på Android-operativsystemet.
CLI (Command Line Interface) – Kommandoradsgränssnitt, ett textbaserat användargränssnitt för att interagera med en datorprogram.
JSON (JavaScript Object Notation) - är ett lättläst datautbytesformat som är enkelt för människor att skriva och läsa samt för maskiner att tolka och generera. Detta är standarden för att skicka och ta emot data via internet.
NPM (Node Package Manager) – Pakethanterare för JavaScript, använd för att installera, dela och hantera programkodsmoduler i JavaScript-projekt.
Open-source – Programvara vars källkod är öppen och tillgänglig för användning eller ändring från dess användare.
WWW (World Wide Web) - vilket är ett system av sammanlänkade dokument som nås via internet.
XML (Extensible Markup Language) – Språk för att strukturera data som kan skickas via internet.

1 INLEDNING

I dagens värld underlättar en webbsida kommunikationen mellan företag och kunder. Många företag övergår också från att spara information som behövs inom företaget på papper till att digitalisera den processen. För att hantera den informationen behövs anpassade digitala lösningar efter företagens specifika behov. De kan vara dyra att utveckla, särskilt om man strävar efter att erbjuda dessa tjänster både som mobilapplikationer och webbsidor.

Några exempel på varför ett företag vill ha dessa tjänster på olika plattformar kan vara att företagets arbetare som är ute på fältet behöver kunna få notifikationer av kontorspersonalen, använda kameran för att ladda upp foton till tjänsten eller bara att erbjuda kunderna olika interaktionsalternativ med företagets tjänster. För små företag med en begränsad användarbas kan investeringar i separata utvecklingsstråk för webb- och mobilbaserade tjänster, som utför likvärdiga funktioner, vara ekonomiskt ohållbart.

Eftersom begäran på applikationer ständigt ökar och de blir alltmer komplexa, uppstår en situation där det kan finnas brist på tillgängliga utvecklare för att skapa alla nödvändiga applikationer. Därför skapas hela tiden lösningar som är snabbare och mer effektiva för utvecklare att använda sig av. För företag som specialiserar sig på programutveckling är det därför en utmaning att konstant välja den bästa plattformen för sina applikationer.

1.1 Bakgrund

Enligt Buck (2023) har användningen av mobila applikationer har kraftigt övertagit mobila webbsidor i användning på mobila enheter. Vuxna användare i USA spenderar ungefär fyra timmar om dagen på nätbaserade aktiviteter och 88 procent av den tiden är på applikationer. Det kan bero på att applikationer anpassade för mobilen är mer användarvänliga jämfört med webbsidor. Ytterligare fördelar för ett företag att ha en applikation är för att när en användare stänger en webbsida så är den borta från användarens mobil men om användaren stänger en applikation så finns den ännu kvar. Detta betyder att användaren blir konstant påmind om ens varumärke.

Detta forskningsarbete är gjort för företaget Codebite, ett relativt nytt företag som grundades 2020 i Helsingfors. För tillfället har Codebite färre än fem anställda. Företaget erbjuder olika skräddarsydda applikationer, både i form av webbsidor och mobilapplikationer. Codebite skapades för att stödja sjöfartsindustrin i Finland genom att automatisera deras agenters dagliga pappersarbete och ge dem den informationen som är nödvändig för deras verksamhet.

Codebite använder sig av React för webbsidor respektive React Native för mobilapplikationer. De här ramverken berättas om i verktygskapitlet. De har funderat om det finns ett mer kostnadseffektivt alternativ som är lika bra eller till och med en bättre lösning för att minska kostnaderna för kunder som vill ha både en webbsida och en mobilapplikation.

För ett litet företag är de bättre att begränsa antalet programmeringsspråk för att ha en så effektiv utvecklingsprocess som möjligt. Därför finns verktyg som Capacitor för att kunna skriva samma React kod för olika plattformar. Capacitor kan omvandla ramverksbaserade webbsidor såsom de byggda med React till nativa mobila applikationer. Detta underlättar utvecklingsarbetet genom att minska antalet kodbaser för samma applikation och öka återanvändbarheten av koden.

1.2 Syfte och mål

Detta arbete kommer att fokusera på två webbsidor som är byggda med React. Anledningen till detta är att Codebite använder React och det är enkelt att jämföra med React Native, som företaget också använt för tidigare applikationer.

Den första applikationen utgör en enkel lunchmeny för en fiktiv pizzarestaurang, medan den andra applikationen är en väderapplikation. Codebite för närvarande inte är involverade i några klientprojekt. Av den anledningen utvecklas de två applikationerna primärt för projektets syfte och inte för någon specifik kund.

Målet med arbetet är att jämföra konverteringsprocessen mellan de två olika ramverken. Det som kommer att mätas är hur länge det tar att konvertera för mig som arbetar dagligen med React, hur länge det tar för applikationerna att starta upp och utföra applikationens funktionalitet och om de ser ut som nativa applikationer.

Syftet med arbetet är att undersöka om det är lönsamt för Codebite att investera i att erbjuda lösningar med Capacitor eller om de ska fortsätta med React Native. Det inkluderar även att ta reda på vilka problem som uppstår inom konverteringsprocesserna och om båda alternativen kan erbjudas som en lösning för kunderna, förstås till olika priser utgående från deras behov.

I den teoretiska delen, som beskriver om vad verktygen är, fokuseras det på framtida stöd för de två lösningarna och vilka risker som är förknippade med att investera tid i att lära sig en ny lösning för ett företag. Denna del presenterar också plattformarnas företagsbakgrund och identifierar eventuella röda flaggor som måste beaktas.

1.3 Metoder och avgränsningar

Det största fokuset i det här arbetet kommer att vara på konverteringsprocessen från React till Capacitor och React Native, för att i slutändan ha två versioner av två mobila applikationer som jämförs med varandra. För att skapa de här används tidigare erfarenhet inom branschen, guider, dokumentation från ramverkens hemsidor och diskussionsforum.

Orsaken varför jag har två applikationer är för att ha en applikation som är nästan enbart skriven utan utomstående NPM paket. Den andra är en mer komplex applikation som har liknande funktioner som en kund skulle kunna fråga efter. Den första applikationen existerar som en baslinje för att jämföra den andra applikationen med, det gör det lättare att identifiera orsakerna till eventuella problem som kan uppstå.

För att ta reda på hur länge det tar att konvertera så tar jag tid på mig själv. Detta är rättvist eftersom jag måste lära mig båda lösningarna. Tiden tas från att jag börjar lära mig tills det att jag har nått slutprodukten. Jag skriver också ner mina egna tankar om hur lätt det var att hitta lösningar på de problem som uppstod med under konverteringsprocessen.

Projektet kommer att innehålla några egengjorda grafer om hastigheten mellan de två olika lösningarna men de är bara gjorda för att se vilken som är snabbare, inte för att visa exakt hur snabba de är. Jag fokuserar också bara på hur applikationerna lämpar sig för ett litet företag och tar inte i beaktande vad stora företag gör som har råd att anställa olika utvecklare för olika ramverk.

2 VERKTYG

För att skapa en simpel Android-applikation krävs inte många verktyg, men det bli mera komplicerat om man ska ha stöd för Apples produkter och om man vill lägga ut applikationerna på marknaden. För att skapa koden så använder jag mig av Visual Studio Code som är ett populärt verktyg skapad av Microsoft och går att använda på Linux, Mac och Windows. Orsaken till detta är för att det finns bra stöd för tillägg och för att det finns mycket stöd om jag får problem med något, till exempel om jag vill ha kortkommandon för att skapa till exempel en React komponent.

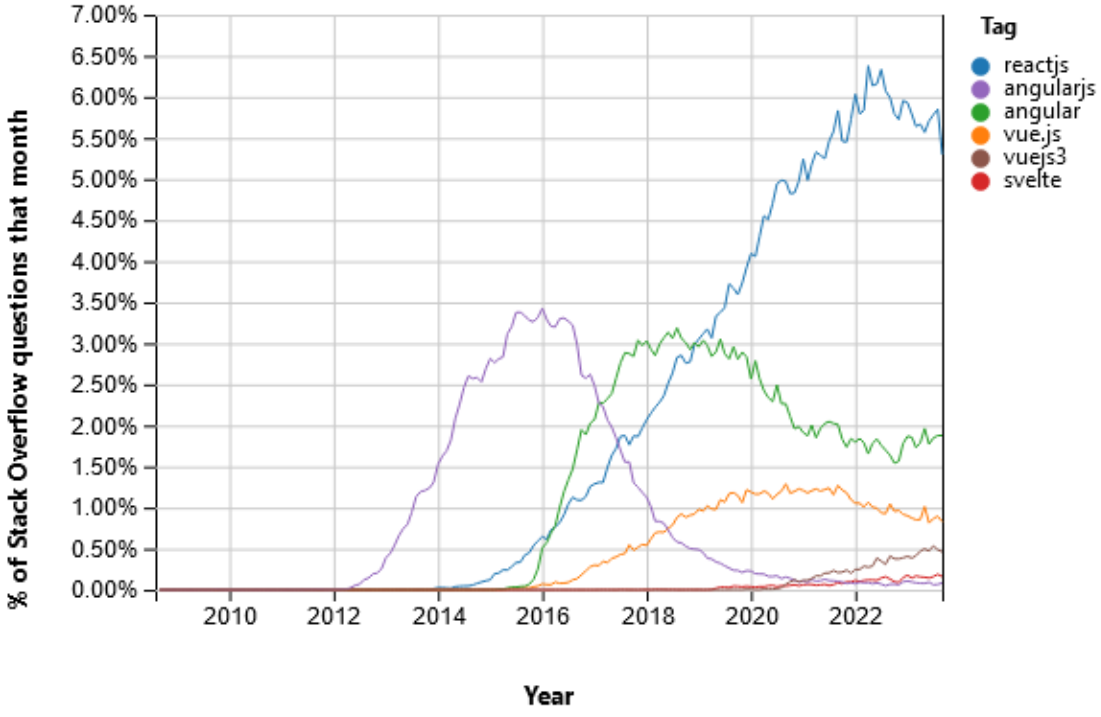
Det jag använder för att installera React och komponenterna för applikationerna är NPM. Det är en pakethanterare för att via en terminal snabbt kunna ladda ner och installera paket i ens applikation som stöder det. På NPMs hemsida (NPM, 2024) hittar man de flesta paket som man kan behöva för ett projekt och om man har något problem med något paket så kan man gå in och ändra på det paketet.

För att utveckla applikationerna så används en Windows 10 dator. Den hårdvaran som jag testar mobilapplikationerna på och tar resultaten från är en Samsung Galaxy S9. Den har en 5.8 tums display och slutresultatet på applikationernas utseende kan variera på vilken storlek, längd och bredd man har på sin mobila enhet. Därför kommer jag också att testa applikationerna på en Samsung Galaxy S21 Ultra.

För att föra över Capacitor-applikationerna till mobilen används Android Studio. Det kopplas upp till mobilen och gör att jag kan installera applikationerna som en APK på mobilen direkt från datorn. För React Native använder jag Expo applikationen (Expo, 2024) som fungerar trådlöst via nätet.

2.1 React

React är ett ramverk som är utvecklat av Meta för att bygga webbsidor. Det är det mest populära ramverket och används i stor utsträckning av över 40 procent av professionella utvecklare (Gathoni, 2022). React tillåter utvecklare att bygga komponentbaserade och interaktiva användargränssnitt på webben. Det är en ”open-source” teknik vilket gör det lätt för nya användare att hitta information om det och lära sig av det. Webbsidor som använder sig av React är bland annat BBC, Airbnb och Dropbox.



Figur 1. Popularitet av frågor frågade om olika ramverk (Stack Overflow 2023).

React började användas av CodeBite på grund av dess popularitet, för att det är lätt att komma i gång med och det krävs inte mycket arbete för att starta upp en applikation. React används i samband med Node för att skapa både framsidan och baksidan av deras webbsidor.

React kan köras med både JavaScript och TypeScript. För det här arbetet används TypeScript. Det har också stöd för många NPM paket som är skapade av andra än Meta. De kan man använda för till exempel en kalender i ett bokningssystem eller om man vill ha färdigt designade komponenter i sin applikation.

Med beaktande av dess utveckling och stöd från Meta, är det sannolikt att React kommer att förbli relevant om tio år. För närvarande är React i en ledande position, men denna situation kan snabbt förändras, vilket har observerats nyligen. Ett exempel på detta är hur kommandot "Create React App", tidigare en standard för att initialisera React-applikationer, upphörde att fungera under 2023 till följd av framväxten av överlägsna alternativ såsom Vite, Rollup och Webpack (Belovarich, 2023). Med detta i åtanke är det

fördelaktigt att inneha färdigheter inom de grundläggande språken inom webbutveckling - HTML, CSS och JavaScript - för att vara beredd att övergå till nya teknologier vid behov.

2.2 React Native

React Native låter utvecklare använda samma struktur som React ramverket tillsammans med nativa mobila egenskaper för att skapa applikationer för Androids och Apples ekosystem. React Native är också skapat av Meta och stöder JavaScript och TypeScript.

En orsak som gör att React Native kan vara bättre att använda än nativ kod är att man kan använda sig av vanliga React komponenter med bara lite omskrivning som sparar tid och pengar för ett företag. Om man utvecklar applikationer för både Android och iOS så kan man använda nästan samma kod för båda plattformarna. Med React Native kan man också använda live omladdning av applikationen så att man ser ändringarna man gör i realtid.

Ramverket skapades år 2015 och har blivit väldigt populärt för att skapa applikationer. React Native har öppen källkod och kan därför förbättras inte bara av Meta men också av folk runt om i hela världen. React Native är känd för att vara lätt att använda och utvecklare som är vana vid JavaScript och React kan lätt förstå hur allt fungerar. Många stora företag använder sig av React Native, såsom Instagram, Tesla, Walmart, och Aribnb (Intellectsoft 2023).

Eftersom det är samma grundare för både React och React Native har de liknande bekymmer när det kommer till om det är säkert att investera i för ett företag. React Native är långsammare än om man skriver nativ kod för mobila applikationer såsom Java. Det är på grund av att Android inte direkt kan läsa JavaScript kod så det måste vara en översättare som översätter koden för mobilen. Med det sagt så fortsätter ändå React Native att växa och enligt Intellectsofts (2023) rapport är 14% av de 500 mest nedladdade applikationerna i USA skapat med React Native.

2.3 Capacitor

Capacitor, och även Cordova, är ett krossplattformverktyg för att lätt skapa mobilapplikationer. Cordova skapades 2009 och användes av Ionic tills de skapade sitt eget verktyg år 2018 för att de inte tyckte om vissa saker om hur Cordova fungerade, de kommer dock fortsätta stöda Cordova i deras Ionic bibliotek.

Enligt Lynch (2020) så fungerar Capacitor och Cordova ungefär på samma sätt. De slår in en inbyggd webbvy och förbättrar möjligheten för JavaScript i Webbvyn att kommunicera med inbyggda funktioner i mobilen.

Capacitor använder sig av en CAPBridgeViewController (CBVC) som tar hand om en webbvy vilket är en brygga som översätter JavaScript till nativ kommunikation. Då CBVCn startar laddar den en URL som är specificerad i applikationens konfiguration, därifrån öppnar webbvyn lika som i en webbläsare URLn i applikationen, med det undantaget att det har tillgång till många fler API begäran via Capacitor.

Capacitor kan installeras i vilket modernt JavaScript ramverk som helst, bara det finns en package.json fil, en index.html fil och en separat mapp för webbtillgångar, såsom dist eller www. Därefter kan man ladda ner Capacitor med NPM pakethanteraren, välja om man vill skapa ett Android projekt eller ett iOS projekt och installera respektive plattform.

Capacitor har också öppen källkod och enligt Lynch (2020) är deras filosofi att inte släppa stora uppdateringar utan att i stället hålla det stabilt och välunderhållet. Det här kan betyda att i framtiden faller det bakom andra lösningar men det är det ingen som vet. Med detta i åtanke och med tanke på hur lätt det är att integrera med webbsidor så kan man använda det här i framtiden. Man får dock vara redo att byta lösning som är sanningen med all programmerings ramverk.

3 APPLIKATIONERNAS UPPBYGGNAD

Det finns några likheter mellan de två applikationerna, den största är att de båda två skapades med Vite (Vite, 2024). En orsak till att jag har övergått till Vite för att skapa React applikationer är för det första att Meta har slutat stöda ”Create React App” kommandot som förr har varit det enklaste sättet att skapa applikationer.

Den andra orsaken varför jag övergick till Vite är för att, som Vite betyder, det är snabbare att starta upp applikationen. Vite installerar varken onödiga komponenter eller onödig kod, som man oftast inte behöver. Fördelen med det är att man inte behöver vänta lika lång tid för att starta om applikationen när man ändrar på koden.

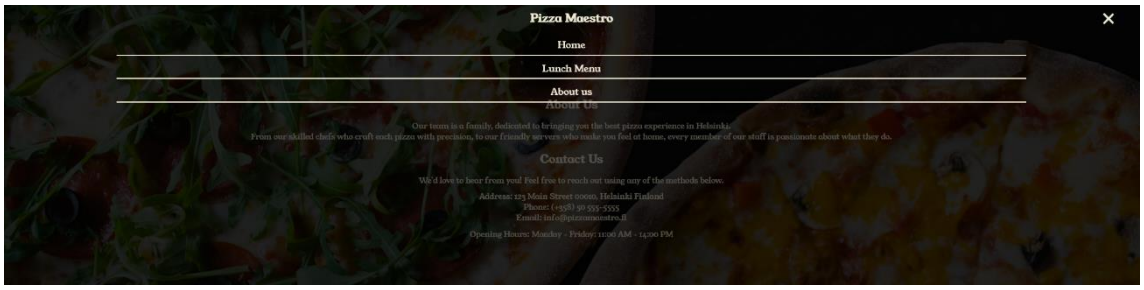
Båda applikationerna använder Typescript för att det är standard inom de flesta företag men också för att det gör att alltmer strukturerat. Om man har en bug i sin kod och har skrivit koden för länge sedan så är det lättare att fixa när man vet vad alla variabler innehåller eller förväntas innehålla för data.

Båda applikationerna har även installerat React Router DOM (React router, 2024), ett paket som förenklar skapandet av länkar till de olika sidorna så att man kan navigera mellan dem. Med detta paket behöver man inte använda vanlig JavaScript kod för att byta URL åt användaren, utan paketet har en färdig funktion som bara byter ut slutet av URLn så att det går lätt att implementera med React komponenter. Sidorna kan struktureras så att man först skapar sina navigeringsfält och andra liknande statiska komponenter runt om sin router, därefter skapas sidornas innehåll inom de statiska komponenterna.

3.1 Pizza-Maestro lunchmenyn

För den simpla applikationen har jag valt att göra en webbsida för en fiktiv pizzarestaurang. Innehållet på den här sidan är påhittat, men funktionellt fungerar den som en riktig webbsida.

Den är uppbyggd med tre sidor, Home, Lunch Menu och About us. Navigations komponenten är lokaliserad på toppen av sidan och är placerad utanför React Routern i koden för att vara med på varenda sida. Den är optimerad för mobil vänlighet med en hamburger meny, var man kan navigera mellan de tre sidorna. Alla komponenter är skapade med helt vanlig HTML och CSS.

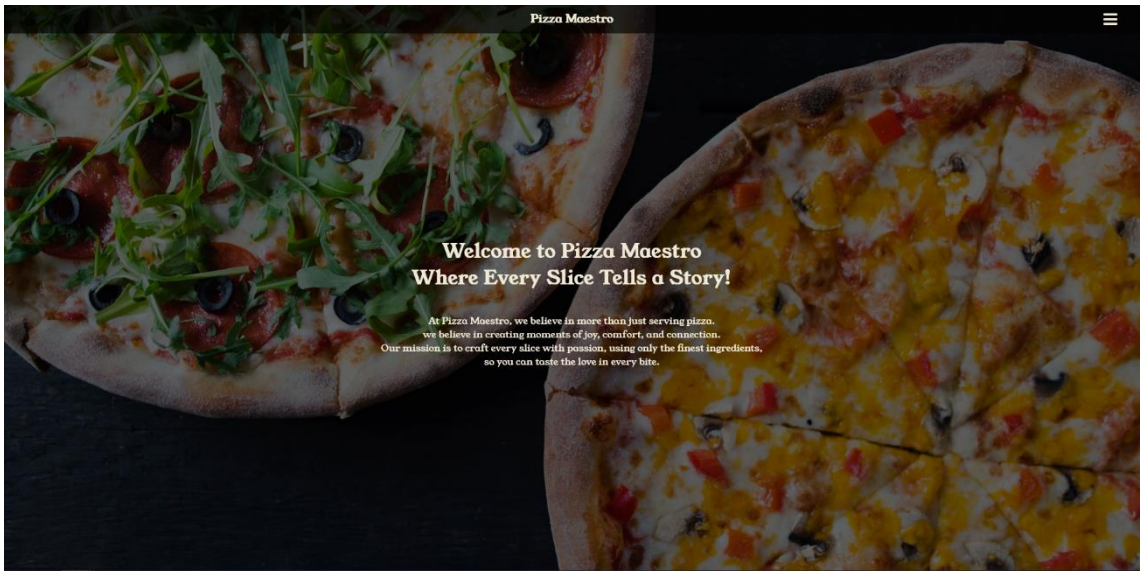


Figur 2 En bild på navigationsfältet på Pizza Maestro-hemsidan.

Bakgrunden på applikationen är en bild på två pizzor sida vid sida för ett mer modernt utseende. När man använder bilder för sina applikationer är det viktigt att ta i beaktande vilka licenser som bilden har. Om man själv, eller någon man har fått rätt av, tagit bilden är detta inget problem. Om man hittat bilden/bilderna på internet måste man kontrollera vilka användningsrättigheter bilderna har.

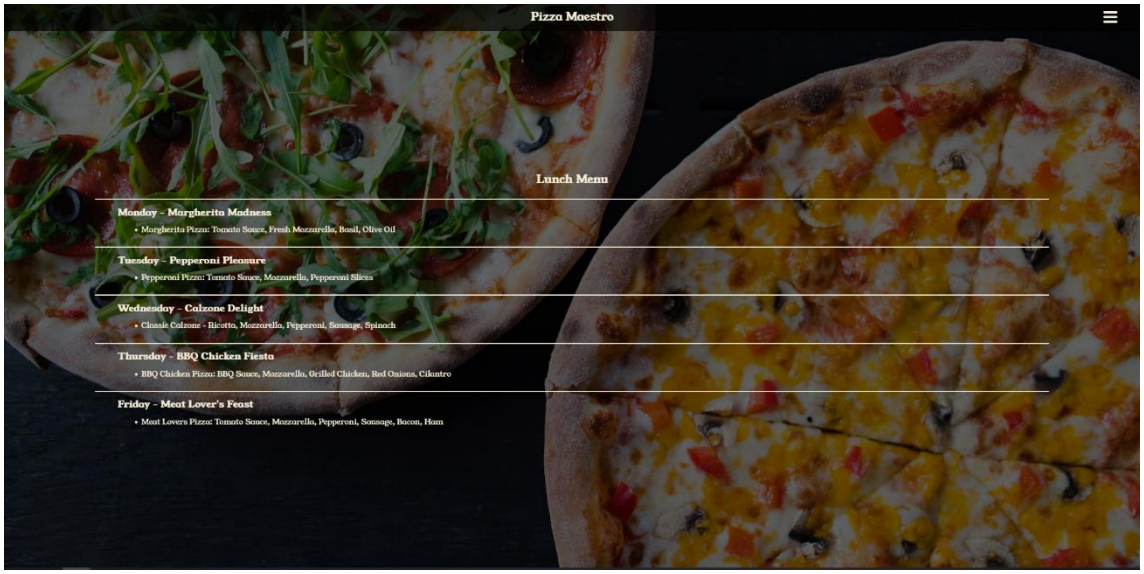
Stylingen för sidan är skapad med React moduler där man importerar klasser. Den enda stylingen som är importerad är fonten på sidan som är OpenSans. För att bakgrundsbilden inte ska vara för distraherande så har ljusstyrkan dragits ner och därför kan texten på hela sidan också vara vit.

På ”Home” sidan som är den första sidan finns titeln som välkomnar användaren till sidan. Under den finns en text som beskriver mera om vad stället är. Texten är stylad så att den är både mobil- och datorvänlig.



Figur 3 En bild på Pizza Maestro hemsidan.

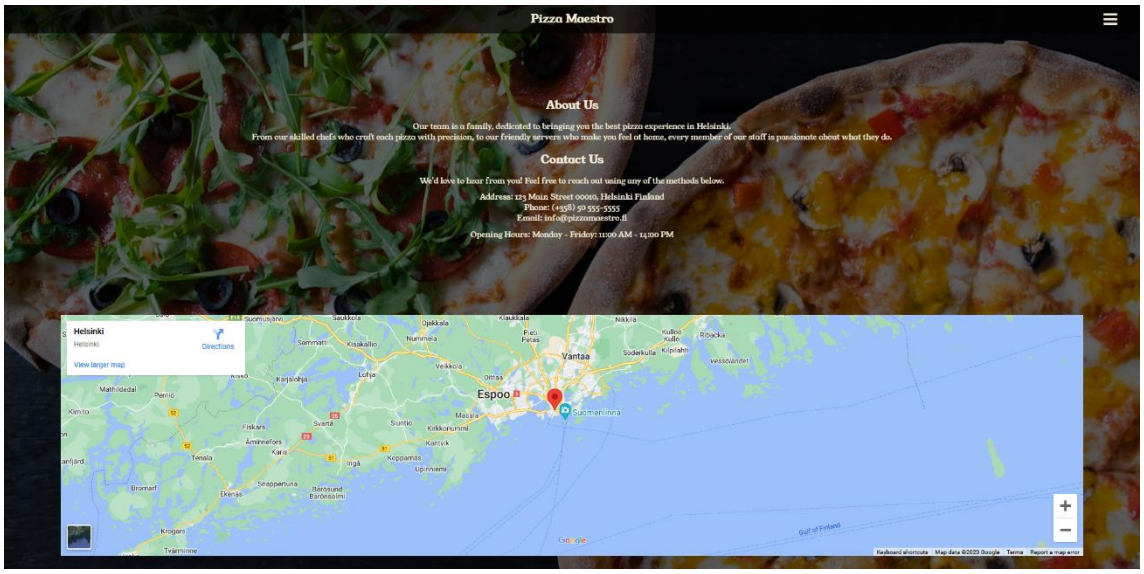
På själva ”Lunch Menu” sidan finns en genererad lunchmeny från måndag till fredag och dagarna är uppdelade med en vit linje. Emellan linjerna står det vilken dag det är, namnet på pizzan och pizzaingredienserna. Texten skapades med en variation på H4 och listtaggar.



Figur 4 En bild på Pizza Maestro lunchmenyn på hemsidan.

En restaurangsida ska också innehålla kontaktuppgifter och inkludera en karta på var restaurangen finns. På den här sidan skapades texten på samma vis som på ”Home” sidan och överst finns det lite information om företaget.

Under det befinner sig kontaktuppgifterna. Därefter har jag inkluderat Google Maps som tar upp nästan halva sidan för att det ska vara en så ändamålsenlig upplevelse som möjligt för slutanvändaren. Kartan är implementerad som en iframe-element som inbäddar en Google Maps-vy i centrum av Helsingfors.



Figur 5 En bild på Pizza Maestro om oss på hemsidan.

Denna applikation är gjord så enkel som möjligt för att skapa den bästa möjliga situationen för konverteringsmetoderna. Då blir slutresultatet det bästa. Den har bara ett importerat paket och det är kanske det mest använda paketet som finns för React så det borde inte skapa några problem.

3.2 Tempestic

För den mer komplicerade applikationen har jag valt att göra en väderapplikation. Den här applikationen är kapabel att visa en fullständig prognos för de flesta platser i världen sju dagar framåt. Den har också prognos för varje timme på dagen.

För att få tag på väderprognosen så har jag använt mig av meteorologiska institutets API (Meteorologiska institutet, 2024). Den stöder all väderdata som behövs för en väderapplikation. En utmaning är att deras dokumentation inte är den lättaste att förstå och det kräver en hel del sökande för att hitta vilken förfrågan till API:n man ska använda. De jobbar dock ständigt på att förbättra den men uppdateringarna sker alltför sällan.

De flesta förfrågningarna returnerar XML format och för nya kodare som bara lärt sig JSON så kan det vara en utmaning att förstå hur man kan använda sig av svaren man får därifrån. De har dock börjat skapa svar i JSON format och jag hoppas att de konverterar om allt till det i framtiden.

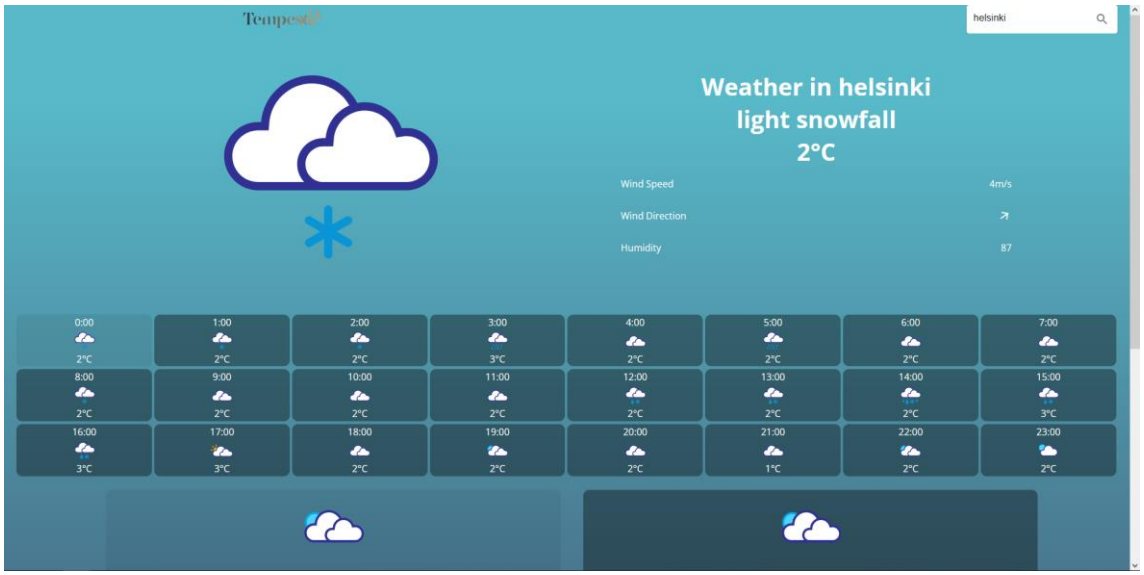
Jag kommer använda mig av Timeseries API:n (Meteorologiska institutet, 2024) på grund av att den returnerar alla de grundliga värden man behöver som till exempel temperatur, nederbörd och så vidare. Med två förfrågningar till den här API:n kan vi skapa en fullt fungerande väderapplikation.

I den första förfrågningen hämtar jag vädret för den dag jag har valt och får vädret för varje timme den dagen. För den andra förfrågningen hämtar jag data för flera dagar för att sedan kunna visa upp det som en lista var man väljer vilken dag det visar på sidan.

För att visa ikonerna så hämtar jag en sträng från Meteorologiska institutet API, den returnera ett nummer och det är deras sätt att välja vilken symbol för vädret man kan använda. De har också sina symboler på Github bara att ladda ned med samma namn som det nummer jag just hämtat. De här ikonerna är i SVG format vilket kan vara ett problem eftersom React Native inte stöder SVG bilder utan ett tilläggspaket. Med den här informationen är det lätt att visa rätt bild för rätt dag.

Codebite använder för det mesta MUI paketet (MUI, 2024) för deras komponenter så det är det jag valt att inkludera i denna applikation. Ionic som har skapat Capacitor har också sina egna bibliotek för komponenter. Att använda mobilorienterade komponenter för att bygga en webbsida kan avvika från projektets ursprungliga mål, vilket är att skapa en standard webbsida med React.

Hela webbsidan är byggd med Grid- och Flex-egenskaper för att den ska vara mobilvänlig. I navigationsfältet skriver jag in vilken stad jag vill ha prognosen för och sedan uppdaterar sidan när jag trycker på sökknappen. Sökfältet och alla ikoner är tagna från MUI. Under navigationsfältet visar det all data som jag har för den timmen och den dagen jag valt.



Figur 6 Bild på Tempestic hemsidan.

Den här applikationen har jag valt att göra för att den har en mer realistisk uppbyggnad för vad som skulle kunna göras inom Codebite. Den är dock inte alls lika komplicerad som de applikationer Codebite gör men den borde visa vilka problem som kan uppstå i ett större projekt.

4 KONVERTERINGARNA

4.1 Hårdvaran

Ingen iOS-produkt användes för detta arbete. Det går att utveckla iOS-applikationer på Windows eller Linux men det är svårare att konfigurera än om man har en Apple dator. Därför så användes en Windows 10 Dator för att utföra konverteringen.

Både React Native och Capacitor prövades på två olika telefoner för att se skillnaderna mellan de två olika skärmarna. Den första var en Samsung Galaxy S9, storleken på telefonens skärm är 5,8 tum och den har ett bildförhållande som är 18,5:9 (GSMARENA, 2023). Den andra telefonen som applikationerna testades på var en Samsung Galaxy S23 Ultra, storleken på den telefonens skärm är 6,8 tum och den har en längre skärm som har ett bildförhållande på 20:9 (GSMARENA, 2023).

Varför bildförhållandena är viktiga att ta i beaktande när man skapar mobila applikationer är för att applikationerna kan fungera på en telefonskärm men när applikationen öppnas på en annan skärm så kan vissa komponenter vara på helt fel plats. Ifall applikationen ska användas på flera olika enheter behöver detta testas innan man släpper applikationen.

En USB-C kabel användes för att föra över Capacitor-applikationen till telefonerna och genom Android Studio kunde man uppdatera applikationen manuellt. Med React Native finns Expo applikationen att laddas ner från Play store och med den kopplas telefonen upp till datorn trådlöst via nätet. Där uppdateras applikationen automatiskt när man sparar ett dokument i Visual Studio Code.

4.2 Mjukvaran

För att testa kompatibilitet så användes S9 telefonen Android 10 och S21 Ultra telefonen Android 14 vilket är den nyaste versionen av Android som fanns tillgänglig när arbetet gjordes. Båda telefonerna har Samsungs One UI versionen av Android.

Här är en lista på vilka versioner som användes av alla program:

- Capacitor 5.5.1
- React Native 0.72.6
- Android Studio Giraffe 2022.3.1 Patch 3
- Expo 2.29.8

Det finns två alternativ att använda för att utveckla React Native och de heter React Native CLI och Expo. Vilken man ska välja beror på projektets krav och utvecklarens preferenser. React Native CLI ger utvecklare mera kontroll över projektet och tillåter direkt tillgång till alla React Natives konfigurationer.

Expo ger utvecklare en snabbare start genom att tillhandahålla en uppsättning färdiga verktyg och tjänster. Expo passar bättre för mindre komplexa projekt och prototyper där snabb prototyputveckling är viktigt (React Native, 2023).

Expo användes på grund av att de mobila applikationer som Codebite har användning för är mest små projekt som till exempel en administrationspanel, där användaren bara ska uppdatera en databas.

4.3 Konvertering till Capacitor

Innan konverteringen påbörjas skulle det bästa sättet vara att analysera och förbereda koden. Detta inkluderar att identifiera och åtgärda eventuella plattformsberoende buggar samt att se över bibliotek och paket för att säkerställa att de är kompatibla med Capacitor. Men med tanke på att vi håller kodbasen den samma som React-applikationen så är tanken att direkt konvertera koden. Det är också fördelen med Capacitor att samma kod kan användas för två olika plattformar, bara koden är skapad med Capacitor i åtanke.

Capacitor är skapad av Ionic teamet och inga av de applikationer som vi konverterat använder Ionic komponenter på grund av att Codebite använder MUI komponenter. Om Codebite skapade applikationer med Ionic komponenter skulle konverteringen vara bättre optimerad för mobila applikationer så det kan vara ett sätt att använda Capacitor i framtiden.

4.3.1 Konverteringsprocessen

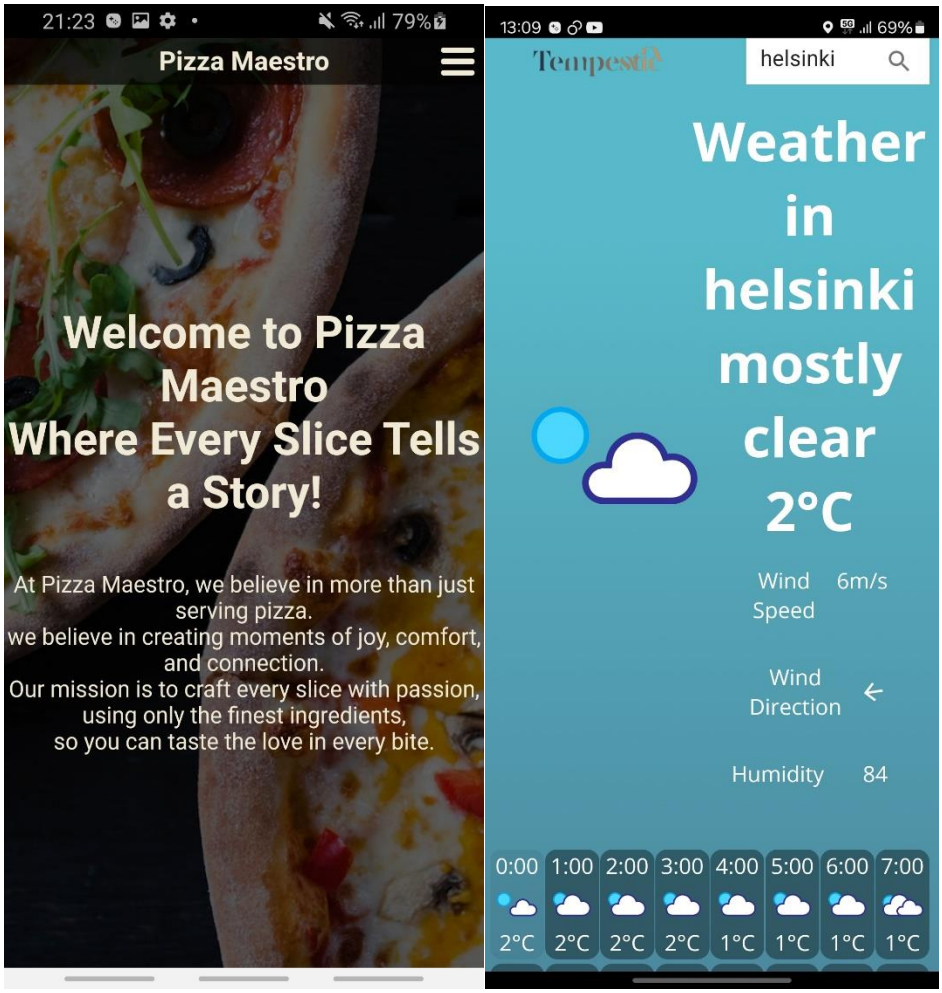
För att börja konverteringsprocessen skapades en ny ”Branch” i Github för att separera koden från webbsidan. Tanken i framtiden är att ha allt i samma ”Branch” om man skapar ett sådant här projekt men för att inte förstöra något delade jag på det. Sedan installerades Capacitor i projektet genom att köra de kommandon som hittas på Capacitors hemsida. Capacitor Core och Capacitor CLI installerades som Node moduler. Sedan initierades de med ”npx cap” kommandot för att skapa konfigurationen för applikationen.

Därefter byggde jag applikationen vilket kopieras in i den nativa plattformen under följande steg. Det steget är att installera Android och/eller iOS paketen för Capacitor som Node moduler och sätta in dem i projektet.

Här kunde man också anpassa projektstrukturen och konfigurationen för att passa mobilplattformen. Detta kan inkludera att lägga till ikoner, startskärmar och andra mobil-

specifika resurser. I konfigurations filen kan namnet på applikationen och dess id också ändras.

Den mobila applikationen kan köras genom att köra ”npx cap run andriod” i konsolen varefter Android Studio startas. I Android Studio väljer man den rätta konfigurationen för sin mobila enhet och kör applikationen endera i en virtuell miljö eller på en riktig telefon.



Figur 7 Bilder på Capacitor-applikationerna

För att skapa en APK fil för Android telefoner i Android Studio så är det bara att öppna bygg fliken och sedan generera en APK som är lätt att flytta över till telefonen dit den installeras.

4.3.2 Tidtagningen

Capacitor	
Infosökning	
Dokumentation	55 min
Android Studio	
Konfiguration	1h 40 min
Arbete för Pizza Maestro applikationen	
Github	10 min
Kodning	30 min
Arbete för Tempestic applikationen	
Github	5 min
Kodning	15 min

Figur 8 Tidtagning av utveckling på Capacitor.

Tidtagningen visar att det som tog mest tid var att konfigurera Android Studio. Orsaken till det var att i början så var de installerade paketen fel för den Android version som jag körde på Samsung telefonerna. Den tiden kan räknas bort för framtida applikationer. Utöver det var det här det snabbaste sättet jag stött på att göra en mobilapplikation.

4.4 Konvertering till React Native

Med React Native så grundades en ny kodbas för två olika applikationer. En ny React Native-applikation skapades från grunden för de båda projekten samt också nya Github Repositories för dem. Fördelen med att skapa en ny applikation för en ny plattform var att jag kan bygga upp utseendet för applikationen just för den plattformen.

Konvertering från React till React Native är inte en helt ny kodbas på grund av att de båda använder samma språk och samma komponentstruktur. Den största skillnaden var hur webbläsare tolkar CSS och hur React Natives motor tolkar den kod man skrivit.

Codebites anställda har väldigt lite erfarenhet av hur man skall skriva stilmallskoden för React Native. Därför skulle användningen av React Native i framtiden kunna utgöra den

mest tidskrävande komponenten. Problemet underlättas dock på grund av hur populärt ramverket är och hur många färdiga komponenter som redan finns att installera.

4.4.1 Skapandet av applikationerna

För att skapa en React Native applikation borde Node och NPM vara installerat på datorn, precis som med React. Det är med NPM som jag kommer installera allt det jag behöver för applikationerna.

Enligt React Natives dokumentation finns två vanliga sätt att skapa en applikation på. Det första alternativet är React Native CLI som behöver Android Studio eller Xcode för att fungera. Det här alternativet är mer komplext och kan användas om en mer anpassad konfiguration behövs för applikationen.

I detta arbete användes Expo Go vilket är ett verktyg som förenklar kodningsprocessen. Expo bygger ovanpå React Native och inkluderar verktyg och tjänster för att göra utvecklingen enklare. Expo är särskilt användbart för att skapa snabba prototyper och utveckla applikationer utan att behöva hantera konfigurationer.

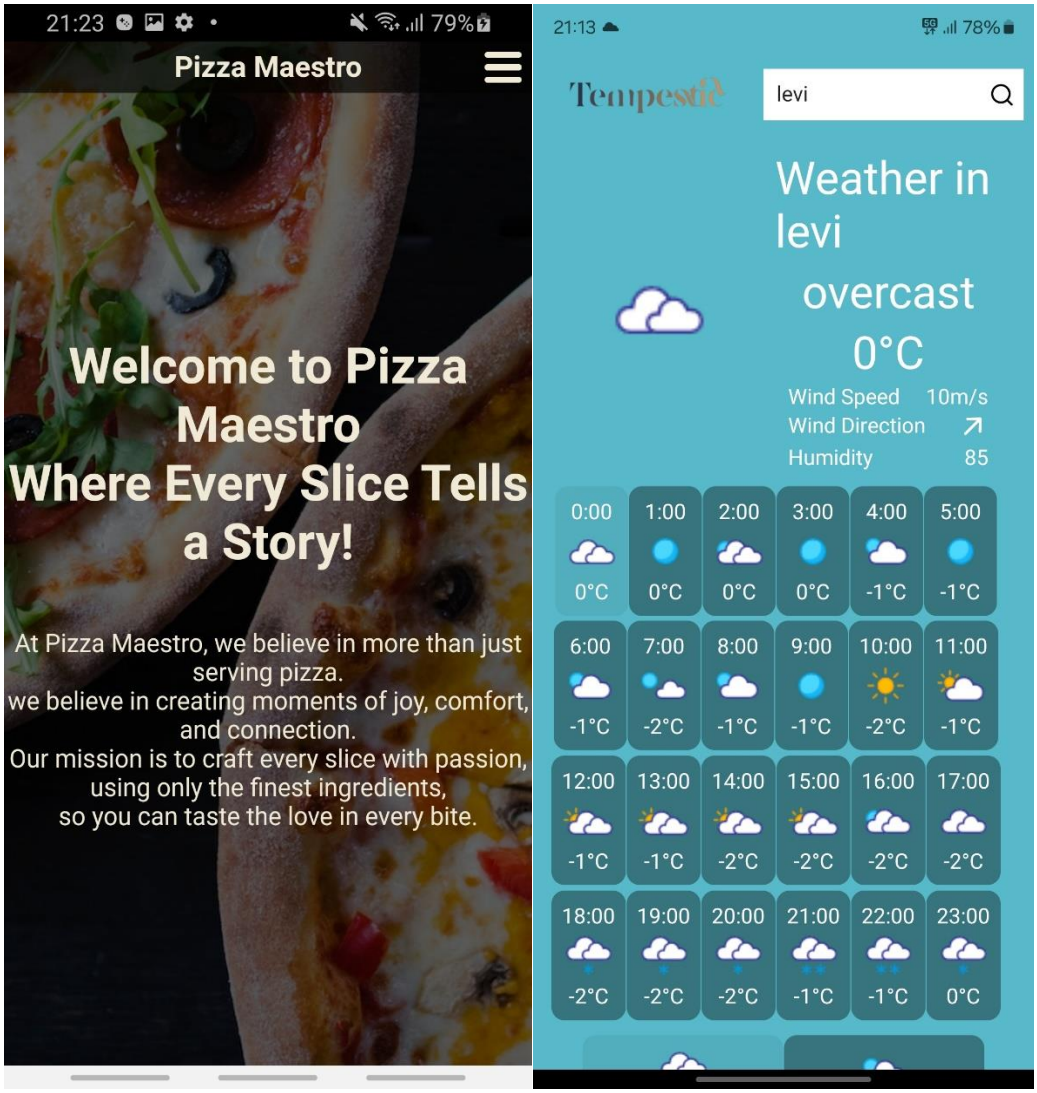
Expo har en applikation i Androids Play butik och i Apples App butik som gör att man direkt kan ansluta trådlöst till applikationen man utvecklar och koden man skriver uppdateras i realtid. Expo använder sig inte av Android Studio och därför kan man snabbt börja utveckla projektet.

För att skapa de nya applikationerna kördes ”npx create-expo-app” kommandot vilket byggde en ny Expo version av React Native. Sedan kunde jag starta applikationen med ”npx expo start” och skanna QR koden med Expo applikationen i telefonen. Nästa steg var att kopiera över koden från React applikationerna till React Native applikationerna.

JavaScript funktionerna fungerade genom att bara kopiera över koden. Men en viktig punkt att tänka på är att React Native använder olika komponenter för det mobila gränssnittet jämfört med webben.

Till exempel ersätts HTML-element som Div och P med React Native-komponenter som View och Text. Dessutom finns det skillnader i hur stilar hanteras och hur layouten struktureras.

När webbsidans HTML-element och stilar ersatts med motsvarande React Native-varianter, behöver också navigationslogiken bytas ut. React Native har sina egna navigationslösningar, så navigationsflödet behöver också anpassas efter det. Det kan innebära att byta ut react-router-dom mot React Natives egna navigator, som till exempel createStackNavigator.



Figur 9 Bild på React Natvie applikationerna

För att generera en APK med Expo så måste ett gratis Expo konto skapas. Det görs i Expos kontrollpanel och där byggs även applikationen. För att skicka en förfrågan för att bygga applikationen, måste jag först installera EAS-CLI.

Därefter konfigurerar jag filen eas.json med de inställningarna för applikationen jag vill ha. Här måste jag specificera att jag vill ha en APK för annars så bygger den en AAB fil

som är menad för Google play store. När den är konfigurerad kan jag köra kommandot ” eas build -p android --profile [profil namnet vi använder]” och då startar en arbetare i Expo kontrollpanelen med att bygga upp applikationen. När den är färdig så laddar man ned APKn från sidan och installerar den på telefonen.

4.4.2 Tidtagningen

React Native	
Infosökning	
Dokumentation	2h 10 min
Expo Go	
Konfiguration	20 min
Arbete för Pizza Maestro applikationen	
Github	7 min
Kodning	3 h 20 min
Arbete för Tempestic applikationen	
Github	4 min
Kodning	6 h 40 min

Figur 10 Tidtagning av utveckling på React Native.

När man jämför tidtagningen av React Native med Capacitor så ser man att React natives konfiguration var lättare att konfigurera på grund av Expo applikationen. Det var dock det enda som gick snabbare med React Native.

4.5 Problem som uppstod

4.5.1 Capacitor

Capacitor fungerade bra med React och det är ett av de snabbaste alternativen att konvertera en webbsida med. Men precis som med alla teknologier kan det uppstå vissa problem.

Det största problemet som uppstod var att navigeringen på Pizza Maestro applikationen slutade att fungera. Det är för att applikationen använder React-router-dom som Capacitor inte stöder. Det är inte ett problem för Tempestic applikationen som bara har en sida. Men om man navigerar till nästa sida så öppnar applikationen webben och går till samma URL som datorn använder för den sidan, i det här fallet http://localhost:5137/about för ”About” sidan.

Problemet med navigationen kan fixas med Ionics egna system men det gör att man endera måste ändra hur webbsidan också fungerar. Om det inte går så tar det bort den största fördelen med Capacitor, vilket är att man kan ha samma kod för både webbsidan och för den mobila applikationen.

Ett till problem som uppstod var kompatibilitet med versionerna på mjukvaran man kör, som till exempel Android på telefonen och på det Android som var installerat i Android Studio. I framtiden kan också problem uppstå om du inte har kompatibla versioner av React och Capacitor. Om Codebite går den här vägen är det viktigt att vara uppdaterad om de senaste versionerna och se till att de är kompatibla med varandra för att undvika konflikter och buggar.

4.5.2 React Native

Att konvertera med React Native kan vara en utmanande process på grund av skillnaderna i hur en webbsida och en mobilapplikation fungerar, därför uppstod det även problem som inte hade en uppenbar lösning som en webbutvecklare kunde att hitta.

Den största utmaningen var att React Native inte använder HTML och CSS. I stället används React Native komponenter med sin egna stilkod. Att översätta vanlig CSS till React Native-stilar var tidskrävande och för att en webbutvecklare ska kunna skapa samma applikationer på båda plattformarna behövs det spenderas mycket mer tid på React Native applikationen.

För att använda SVG i React Native behöver utvecklare använda sig av tredjepartsbibliotek som react-native-svg. Detta bibliotek innehåller komponenter som klarar av att integrera SVG filer i React Native applikationer. Problemet som uppkom i detta fall var att kunna stöda dynamiska uppdateringar av ikoner för Meteorologiska Institutet. Den API:n

innehåller över hundra olika ikoner som dynamiskt uppdateras beroende på vilket väder det är.

För att kunna uppdatera en variabel dynamiskt så använder man sig av ett ”useState” som i React används i funktionella komponenter för att införa tillstånd i komponenten. Problemet var att få den här tillståndsvariabeln och SVG biblioteket att samarbeta. Meteorologiska institutets API returnerar bara namnet på vilken ikon man ska använda och inte hela SVG ikonen, så därför måste ikonen hämtas med hjälp av namnet.

Det finns ingen lättillgänglig dokumentation på hur man kan åstadkomma detta och det resulterade till att ikonerna konverterades till PNG filer med hjälp av Adobes konverterare. Sedan byttes SVG ikonerna ut till PNG ikonerna och de fungerade direkt med React Natives inbyggda ”Image” komponent.

5 TESTNING AV APPLIKATIONERNA

5.1 Buggar som uppkom

En bugg som skiljde sig åt på de båda applikationerna var hur notifikationsfältet visades på telefonerna. På Capacitor-applikationen visades inte applikationen över notifikationsfältet och den kördes bara mellan den och navigationsfältet, vilket ser ut som att den inte är en helt nativ applikation.

React Natvie-applikationen kördes på hela skärmen och om jag inte gav den en marginal i toppen av skärmen så kunde den vara ovanpå telefonens egna notifikationer. Om jag också har en mörk bakgrundsbild så syns inte heller notifikationerna på grund av att de är svarta.

Genom hela applikationernas utveckling testade jag de på min Samsung Galaxy S9. När jag nådde den punkten där jag kände att de var färdiga, behövde jag testa dem på enheter med olika skärmstorlekar. Det var då jag prövade dem på min Samsung Galaxy S21 Ultra för att se hur applikationerna ser ut på den.

Pizza Maestro-applikationen såg ut precis som förväntat men det var bara en applikation med text, ett navigeringsfält och en bakgrundsbild så det var ingen stor överraskning. På

Tempestic-applikationen uppenbarade det sig att ikonerna var i olika format, vissa av dem passade inte in i den stilmall som jag gett dem på den större skärmen.

När jag tittade på den mindre skärmen igen noterade jag att de också var olika på den men det såg jag inte förrän jag prövade den större skärmen. Det är en av orsakerna till att testning av olika skärmstorlekar är viktigt.

Det som också uppkom i Tempestic var att Capacitor-applikationens sökfält betedde sig som den gjorde på datorn. I React Native applikationen så kunde man inte söka direkt från tangentbordet i telefonen utan man måste manuellt trycka på förstoringsglasikonen. Man måste där komma på olika lösningar på samma problem, vilket igen kräver tid i utvecklingsprocessen.

5.2 Hastighet på de olika plattformarna

Jag hittade inget bra verktyg för att testa hastigheterna av applikationerna på telefonerna och den dator jag använde klarade inte av att emulera Android när jag testade det så jag gjorde det med en vanlig tidtagare. På grund av detta så är den här informationen inte exakt utan en jämförelse mellan de två plattformarna. Jag försökte dock göra det så rättvist som möjligt genom att utföra tidtagningen när telefonerna var i samma tillstånd. Testet gjordes tre gånger för att få en så bra bild på hur snabba de är.

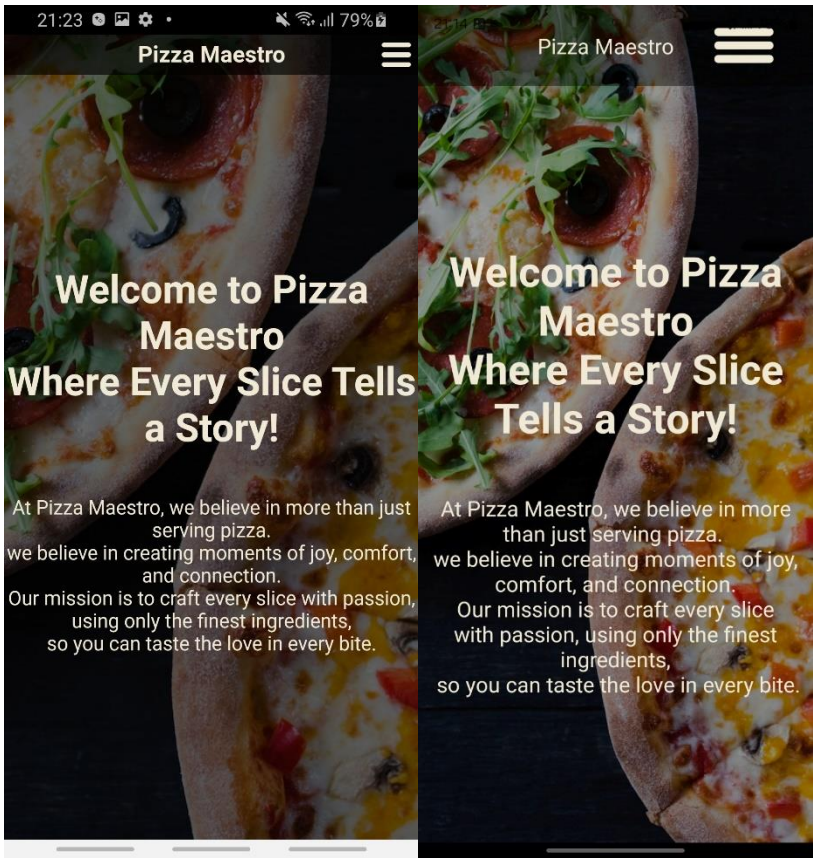
Capacitor	
Start av applikationen	≈150ms
Hämta in ny väderinfo från en ny plats	≈34ms
Byt dag	≈40ms
React Native	
Start av applikationen	≈115ms
Hämta in ny väderinfo från en ny plats	≈73ms
Byt dag	≈60ms

Figur 11 Prestanda av Tempestic applikationen på två olika plattformar.

Ovan ser vi att Capacitor-applikationen tar längre tid att starta upp och det kan bero på att den körs via en webbvy. Vi ser också att när applikationerna har startats är Capacitor betydligt snabbare än React Native applikationen och det märktes när man bytte dag eller sökte efter en ny plats att få vädret för. Det kan vara att det går att optimera React Native applikationen men funktionerna är uppbyggda på samma sätt på båda plattformarna så det borde då också gå att optimera dem på Capacitor-applikationen.

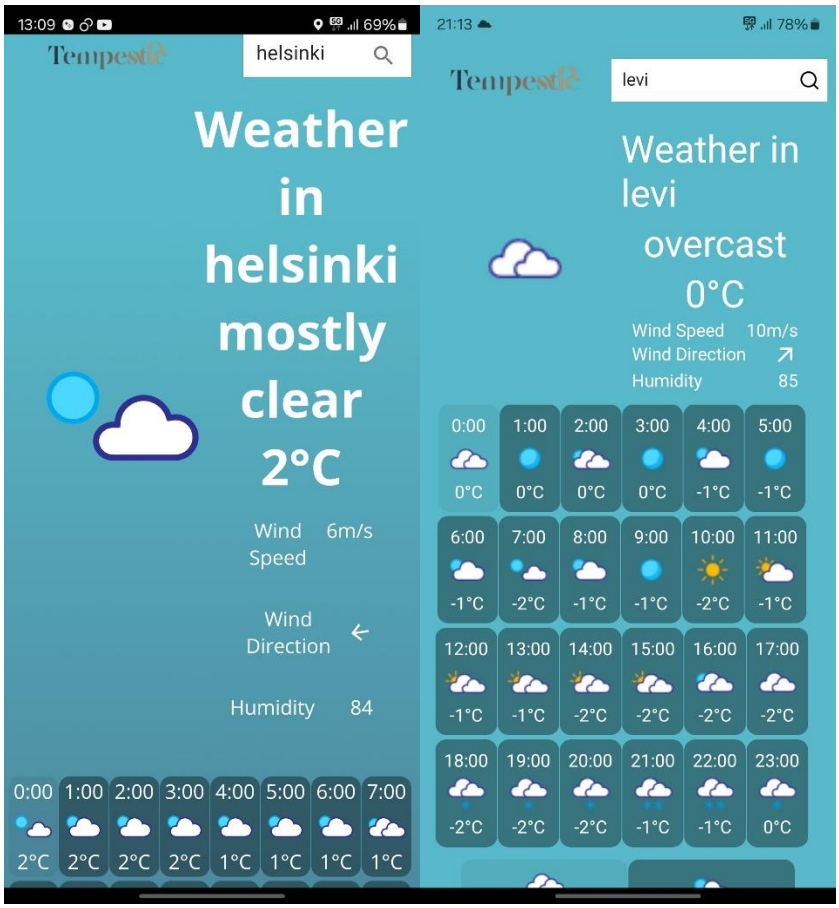
6 RESULTAT

De här två applikationerna har olika funktioner och visar upp två sätt på hur plattformarna fungerar. Tack vare det här ser man hur plattformarna hanterar två olika scenarion.



Figur 12 Pizza Maestro applikationen, Capacitor till vänster och React Native till höger.

Pizza Maestro applikationen är helt programmerad med vanlig React kod utan något annat NPM paket installerat Det enda problemet där var hur Capacitor fungerar med React navigeringen. Så det måsta man tänka på i framtiden om man skapar en applikation med React och Capacitor.



Figur 13 Tempestic applikationen, Capacitor till vänster och React Native till höger.

Tempestic applikationen är uppbyggd med NPM paket som MUI, Axios och Date-fns. De här paketen är de som används främst av Codebite och valdes därför för den här applikationen. MUI fanns dock inte för React Native och därför stylades komponenterna för den applikationen manuellt.

En av de mest framträdande utmaningarna med dessa två plattformar är att underhåll av en gemensam kodbas inte är möjlig, vilket potentiellt kan leda till en fördubbling av arbetsbelastningen relaterad till framtida uppdateringar och underhåll. När det gäller Capacitor, är det genomförbart att utveckla applikationen med React på ett sätt som säkerställer kompatibilitet och funktionalitet utan fel. Detta kan innebära en övergång från att använda MUI till att implementera Ionics egna bibliotek. Emellertid medför detta ett behov av att tillägna sig kunskap om det nya bibliotekets funktioner, vilket kräver en tidsinvestering.

Det man också måste ta i beaktande är hur länge företagen stöder de här biblioteken. I det här fallet är det ganska säkert att Meta stöder React och React Native en lång tid ännu

med tanke på hur populära de är men Ionic som är utvecklaren av Capacitor är ett mindre företag, dock har den har den öppen källkod vilket brukar hjälpa med att underhålla en plattform.

Capacitors stora fördel är att det är lättare att konvertera med och tar mycket mindre tid än React Native vilket gör att nästan vilken React-utvecklare som helst kan arbeta med det. React Native har större stöd bakom sig vilket betyder att det finns mer dokumentation när man stöter på problem. En nackdel är att man måste spendera mer tid och pengar på utvecklingen.

I det här fallet fungerade ändå Capacitor lika bra som React native för båda projekten. Capacitor hade sina problem som man måste tänka på men det kan också vara ett alternativ i större projekt. Jag tror ändå att om kunden vill ha en mer skräddarsydd applikation med mer funktionalitet så är React Native fortfarande ett bättre alternativ.

7 SLUTSATSER

Syftet med arbetet var att jämföra konverteringsprocessen mellan Capacitor och React Native från en React hemsida. Fastän båda ramverken har utmaningar som man måste lösa. Alla alternativ som finns tillgängliga har utmaningar. Det som använts tidigare inom företaget har varit React Native och därför ville jag lära mig mer om det och jämföra det med ett annat alternativ.

De här applikationerna kan jag finslipa och i framtiden visa upp åt kollegorna på Codebite för att kunna ta ett bättre beslut om vad vi ska rekommendera åt kunderna i framtiden. Det svåraste när man väljer vilken plattform man ska bygga en produkt med har varit att välja den rätta plattformen, ofta utifrån att man bara har läst om dem. Nu har vi en konkret möjlighet att prova, och vid behov, implementera funktioner i applikationerna för att undersöka vilka problem som kan uppstå.

Det svåraste med det här arbetet har varit att styla React native applikationen på grund av hur mycket det skiljer sig från vanlig CSS och det är den aspekten av arbetet jag spenderade den största delen av tiden på. Fastän det var svårare att göra det så var det lättare att få det och se mer naturligt ut på telefonen än med Capacitor på grund av att man inte

behövde ha samma stilkod för fler än en enhet. Jag har insett hur enkelt det är att konvertera med Capacitor och vilka problem som kan uppstå.

Målet med det här projektet var att jämföra konverteringsprocessen och att undersöka om det är lönsamt. Lönsamheten är en viktig aspekt inom Codebite. Vi har endast små projekt där skapandet av en applikation inte får kosta för mycket för kunden. Jag har uppnått målet.

KÄLLOR

Bolovarich, S (17 mars 2023). The Problem With React. DEV. https://dev.to/steveblue/the-problem-with-react-46mg

Buck, A (2023). Mobile Apps vs Mobile Websites: Which is Best for 2023?. MobiLoud. https://www.mobiloud.com/blog/mobile-apps-vs-mobile-websites

Capacitor. (2023). Capacitor docs. Capacitor. https://capacitorjs.com/docs

Capacitor. (2023). React & Capacitor. Capacitor. https://capacitorjs.com/solution/react

Expo. (2024), Expo. Meta plattforms. https://expo.dev/

Gathoni, M (6 september 2022). The Most Popular JavaScript Frameworks of 2022. Makeuseof. https://www.makeuseof.com/most-popular-javascript-frameworks/

GSMARENA. (2023), Samsung Galaxy S9. GSMARENA. https://www.gsmarena.com/samsung_galaxy_s9-8966.php

GSMARENA. (2023), Samsung Galaxy S21 Ultra 5G. GSMARENA. https://www.gsmarena.com/samsung_galaxy_s21_ultra_5g-10596.php

Intellectsoft (28 augusti 2023). 8 Best Android Frameworks for App Development in 2023. Intellectsoft. https://www.intellectsoft.net/blog/best-android-frameworks/

Lynch, M. (28 februari 2020). How Capacitor Works. tinyletter. https://tinyletter.com/ionic-max/letters/how-capacitor-works

Meteorologiska institutet. (2024), The Finnish Meteorological Institute's open data. Meteorologiska institutet. https://en.ilmatieteenlaitos.fi/open-data

Meteorologiska institutet. (2024), WFS Time Series Data. Meteorologiska institutet. https://en.ilmatieteenlaitos.fi/open-data-manual-time-series-data

MUI. (2024), MUI. Material UI SAS. https://mui.com

NPM. (2024), NPM. NPM Inc. https://www.npmjs.com/

React Native. (2023). React Native docs. Meta Plattforms. https://reactnative.dev/docs

React Router. (2024), React Roter. Remix Software, Inc. https://reactrouter.com/en/main

Stack Overflow. (8 oktober 2023). Stack Overflow Trends. Stack Overflow. https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3

Vite. (2024), Vite. Evan You & Vite Contributors. https://vitejs.dev