UNIVERSITETET I AGDER

IS-304: 2024

Tittel: Developing an in-house web push solution for an existing client application

Emnekode	IS-304
Emnenavn	Bacheloroppgave i informasjonssystemer
Emneansvarlig:	Hallgeir Nilsen
Veileder	Janis Gailis
Oppdragsgiver:	CuroTech AS

Student:

Etternavn	**Fornavn**
Päivärinta	Ansu

Jeg/vi bekrefter at vi ikke siterer eller på annen måte bruker andres arbeider uten at dette er oppgitt, og at alle referanser er oppgitt i litteraturlisten.	JA X	NEI___
Kan besvarelsen brukes til undervisningsformål?	JA __	NEI X
Vi bekrefter at alle i gruppa har bidratt til besvarelsen	JA X	NEI___

Abstract

Push messaging to notify end users of new, relevant information is an increasingly prevalent functionality in modern software. During my bachelor's thesis this past semester, I have had the opportunity to work alongside a team of developers at CuroTech AS with the goal of developing such functionality for their product, FDVweb.

This report documents the process thus far, as well as the technologies used to develop a working prototype that is currently in beta testing. Although the original task was loosely defined to merely explore whether it was feasible to implement a solution for push notifications into FDVweb, the speed with which a proof of concept was in place prodded us to begin work on an actual implementation.

As a result of getting to work in such a dynamic way, I have had a meaningful learning experience within software development. Thanks to the autonomy I have been afforded during my work on this project, I have also gained great insights into my own capabilities and limitations.

Table of Contents

List of figures

Abbreviations

API: Application Programming Interface

DB: Database

DTO: Data transfer object

FCM: Firebase Cloud Messaging

FDV: Forvaltning, Drift og Vedlikehold (translated: Management, Operation and Maintenance)

MVC: Model-View-Controller

MVP: Minimally Viable Product

POC: Proof of concept

PWA: Progressive Web Application

SW: ServiceWorker

Introduction

During my bachelor project I have been working for CuroTech with the task of implementing functionality for push messaging in their product, FDVweb. FDVweb is a software solution for taking care of their clients' management, operations- and maintenance needs. It comes as a web application with its own dedicated progressive web application instead of a native application for mobile use. The main reason for the lack of push functionality is the fact that Apple first began supporting web push on PWA with iOS version 16.4, which launched in March 2023, and as a large portion of CuroTech's clientele use iPhones, it was not desirable to develop a solution that could only be used by non-iPhone users at the time.

The team

Instead of a group of students, I will be working as a sole student alongside the developer team at CuroTech, thus fulfilling the university's requirement that the bachelor project be done in a group setting.

Background for project selection

While there are many existing platforms that offer push notification services, e.g. Firebase Cloud Messaging and OneSignal, they tend to either get pricy or use data collected through the use of their services for monetization. As for FCM, they require full integration of Firebase within the application, something CuroTech was not interested in taking on, as that brings up uncertainties regarding data retention.

Project definition

To figure out whether or not it is possible to implement a satisfactory solution for push messaging in FDVweb, in which case to start developing an in-house solution for push notifications.

Central decisions

Methodology

During this project, I have followed some of the principles found in the Manifesto for Agile Software Development. This was not so much a conscious decision as it was a natural result of the status quo within the team of developers at the company. The prevalent culture that permeates the team goes hand in hand with the four main values of agile software development, namely:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

(Beck et al., 2001).

Agile methodology made the most sense in that workflow consisted mainly of me focusing on the next vital bit to get closer to a working MVP, with available support for when I got stuck. I opted not to use any conventional version of a Kanban board, but instead make use of sketches whenever it would be necessary to gain an overview of the current state of the project. This would allow me to minimize the amount of meta work as I was the sole developer of the functionality. During the project I will make sure to write clear enough code to avoid the need of thorough documentation for other developers to get relatively quickly up to date and proficient in working with it in the future.

Principle 7 of the agile manifesto is that working software is the primary measure of progress (Beck et al., 2001). Following this principle, I chose to focus most of my time and efforts to ensure progress on the coding front with the cost of neglected documentation of the work that has been done. This also ties in with the 10th principle: simplicity, maximizing the amount of work not done (Beck et al., 2001), as documentation for its own sake provides little to no value. The concepts are already well-documented and

developers that will work on the code in the future are better served by simply digging in and getting to work.

Project management

Since I was to be the sole developer in charge of researching and implementing the functionality for push messaging, as well as the loose project definition, I quickly decided on a dynamic strategy consisting of continuous cycles of research, implementation, testing, feedback, reflection and iteration. I made an early conscious decision not to spend much time devising a more concrete strategy as well as thorough planning, as I figured I would be bound to constantly choose between revising the plans or to abandon them completely. This decision was quickly affirmed by the flexibility of the prevalent dynamic within the team of developers.

As this project is focused solely on added backend logic to existing software, I was spared the highly time-consuming task of designing a user interface and worrying too much about user experience. Instead, it was quickly evident that most of my time would be spent researching possible solutions and getting comfortable with the technologies at hand, while coding whatever amount possible at any given time.

Ensuring quality

To ensure quality in practice, my code would undergo continuous reviews by one of the team members. Any pull requests would also require thorough reviews before merging with the main branch.

Initial tasks

The initial list of tasks was as follows:

- write a minimally working API to handle main logic regarding push notifications
- connect it to a database
- get hands on the subscription object provided by the browser
- store the subscription in a DB table together with information about the user it belongs to
- be able to send push messages to all desired devices/operation systems (Windows, macOS, iOS, Android)
- if time, start looking at how to implement this into FDVweb

Requirements from client

The client did not have any explicit requirements regarding which technologies to use in the project, quite the contrary, they maintained that I should not feel restricted by the technologies they currently use while working on the project. This was made possible by the fact that my project would be an external API, that the client software would make calls to. Of course, integrating the usage of the API within the client software would have to be done in their current language, C# with the .NET framework, so I opted to also write the API itself using the same technologies to minimize the amount tools to be learned.

As for functionality, the company had a clear requirement that notifications will have to be able to be delivered to PWAs on both Android and iOS. Another requirement was that the notification must be able to open the relevant view within the PWA when an end user clicks on the notification.

There were also a few implicit requirements that go along with the nature of notifications, namely being able to determine when they are being sent, who will receive them, and to define the contents of the notification, including an icon, a title and a message body.

Tools used

The tools used in this project are as follows:

C# with .NET 6 for writing the API as well as the necessary integration code within FDVweb.

SQL Server for the DB containing all relevant subscription data.

Entity Framework Core for building the DB based on a model defined in the API as well as handling any necessary future migrations, it also functions as the O/RM for handling the DB queries initiated by the API.

JavaScript for handling events that trigger FDVweb to handle saving and deletion of user subscriptions as well as within the SW for handling the presentation of notifications to end users.

The package web-push-csharp by coryjthompson was used to create notifications that pass encryption requirements.

Azure DevOps for version control and logging of history.

Standards

FDVweb is written using the MVC pattern and makes use of Dependency Injection and services, so I followed the same pattern while integrating the usage of the API within FDVweb.

Running the project

The team consists of me as the sole student, as well as access to the company's three full-time developers for guidance. It was clear from the start that I was the head of this project, with the other team members serving a support role for whenever I needed assistance, be it technical, navigational or the inner workings of their codebase. Seeing as the team is quite small and the communication between members is as open, the dynamic settled on me working independently as long as there were no major obstacles to the progression of the project.

The team makes use of weekly status meetings as a measure to keep track of what each member is working on as well as bringing up any major roadblocks they might have. These meetings also make for a great occasion to re-prioritize pending tasks and bring to light any issues that may have previously been overlooked.

To refine my understanding of the problem domain and gain a better overview of the eventual solution, I sketched out some diagrams and a basic flow-chart over the course of my research of the existing documentation.

I used simple note documents and post-it notes to keep track of active and future tasks, as there always seemed to be a clear next step that had to be solved before continuation of the project was possible. The clarity came from close communication with the developer team, both through frequent conversations as well as the weekly status meetings. A history of what was done, and when, is available through a log of my pull requests to the company's Azure DevOps repositories.

Estimates were made continuously during the project. To minimize the risk of spending too much time creating estimates that may or may not turn out to be accurate, they were given in the format of days/weeks, rather than attempting to calculate time expenditure down to the last hour of estimated workload.

Ensuring quality

To ensure a deeper understanding of the inner workings of the code, I employed the use of repetition. Halfway through the semester, after achieving a working minimal proof of concept, I made sure to solidify what I had learned so far by completely rewriting the API. This in turn helped me to improve upon its models and controller for the next iteration, which would become the base of the final implementation.

Before I submitted any code to the repositories, its functionality was thoroughly tested on all devices and operation systems in use by the end users. This was done by testing sending notifications to the browsers Google Chrome, Mozilla Firefox, Microsoft Edge and Safari on desktop, Windows and macOS respectively, as well as Chrome and Firefox on Android and Safari on iOS.

Quality of code was assured through continuous reviews of my submissions by one of the full-time developers within the company.

The product

Intro to web push

The web push protocol is a spec that defines the format of network requests from a server to a push service, that in turn delivers push messages to end users' devices.

A push notification, or push message is data sent to a web application from an application server [ref]. It is then delivered to the SW associated with the subscription, if the worker is not currently running, it will be started to ensure delivery of the message (Beverloo et al., 2023).

Each subscription has a unique endpoint to which the application server can send messages to. As each subscription belongs to a SW registration, they are limited to one device (Beverloo et al., 2023). However, a user can be subscribed with several devices. To be able to send a notification to all devices belonging to the user, one must make sure to save relevant user data along with the data of all their subscriptions in a database.

Another requirement for the sending of push notifications is that one must make use of the Push API that is accessible through the ServiceWorkerRegistration interface. The Push API enables the subscription of a user to a push service, of which there is one per main web browser (Beverloo et al., 2023). It also takes care of the sending of push messages to these services, that in turn delivers the message to the end users' subscription endpoints.

For a notification to be allowed through the existing push services, messages must be encrypted according to the Web Push spec (Beverloo et al., 2023).

Proof of concept

My main task was to work on a proof of concept to figure out if and how to get web push working on a PWA on iOS and worry about eventual integration with FDVweb only if the POC turns out successful.

To achieve this, we decided that I would be working on a prototype outside of FDVweb to allow me not to worry about introducing bugs and go at my own pace. Which entailed writing a very simplified PWA with its own SW and manifest files, as well as an API and its corresponding DB for storing the subscriptions. I kept the PWA as minimal as functionally possible to focus on the API that would later be integrated with FDVweb in the event of a working POC.

The PWA consisted of two buttons (as seen in Figure 1), one for getting user permission and saving the subscription object provided by the browser and one for sending a very basic notification to, at this stage, all subscription endpoints saved in the DB. Both of which were achieved through making calls to the API.

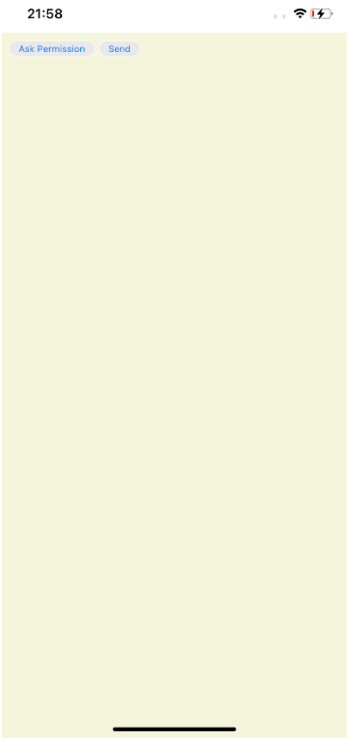

Figure 1 - early POC view

User permission

To ask user for permission, one must use the requestPermission method found in the Notification interface of the Notifications API, which is available through SW (mdn, 2024).

Apple demands user initiation for the above method to work as planned (Apple, n.d.), for example by clicking a button.

PushSubscription

After permission is given, the subscribe method found in the pushManager interface available through the SW registration can be called (mdn, 2024), which prompts the browser's push service to return a subscription object with an endpoint URL as well as a P-256 ECDH key pair for encryption and an authorization secret (Beverloo et al., 2023).

API

The subscription object is then sent to the API in the body of a call, after which the API makes use of a DTO to store the subscription in the DB.

When an event occurs that is predetermined to notify a user, an API call containing a data object is made to an endpoint creates a notification with a payload that is determined by the values of the fields within the data object.

For the sending of notifications I made use of a package called web-push-csharp (coreyjthompson, n.d.) to simplify the inherent complexity that accompanies the encryption requirements for a notification to be allowed through.

Serviceworker

The SW includes code for handling "push" and "notificationclick" events. For it to be possible to send a user to the desired view when they click on a notification, also known as deeplinking, one must include logic in the "notificationclick" event handler to open the PWA to a link sent within the notification payload.

To start testing deeplinking in push notifications, I first added a field to the data payload object and set its value to https://www.google.com. This works both on desktop and Android, but not on iOS. After a quick round of research, I could not find any information as to the reason why. I then decided to add a second page to my PWA to test if iOS would open a link if it leads to a page within the scope of the PWA, which turned out successful. This lead me to the conclusion that Apple probably has some sort of check in place to ensure that notifications cannot be used to send users on external adventures.

Integration with FDVweb

It was not until towards the end of March 2024 that it became evident that sending push notifications to the desired devices was indeed possible, prompting the start of integrating the use of the API.

I kept the API with its DB connection and added a button (to satisfy previously mentioned requirements by Apple) to FDVweb for end users to accept the reception of push notifications, which in turn had FDVweb make an API call to save said subscription in the DB. Then I created a PushHttpClient and a PushService with their respective interfaces. The HTTP client is used for making calls to the API while the service contains the logic for the functions that can be called from relevant places within FDVweb, made possible by injecting the service where needed. For example, a general function for creating a notification. This function is reusable as it takes in any relevant information as parameters and in turn creates the payload that will be included in the API call that in turn creates and sends the final notification.

Current state

As of writing this report, I have implemented the API with endpoints for storing users' subscriptions as well as an endpoint for dealing with sending notifications. So far, I have written logic for the creation of notification data within FDVweb, as well as applied it to one main use case of when to notify users. This particular use case is to notify a user upon the creation of atask they are responsible for completing.

Figures 2 and 3, show the notification on the home screen as well as the view that opens when a user opens said notification, on Android and iOS respectively.

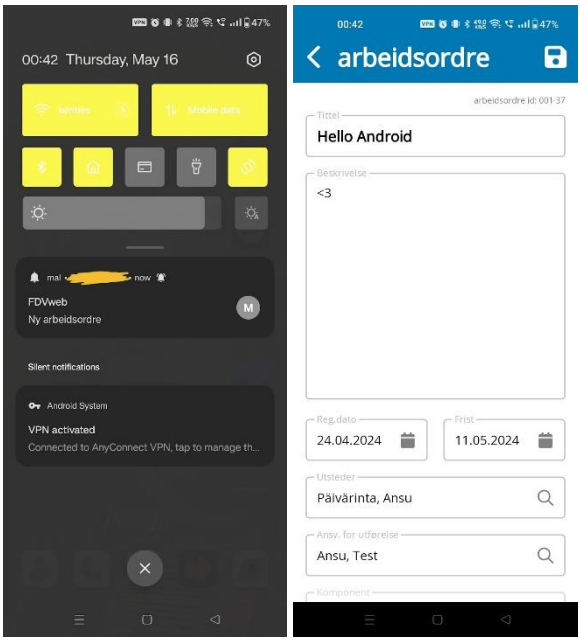

Figure 3 - Android notification.

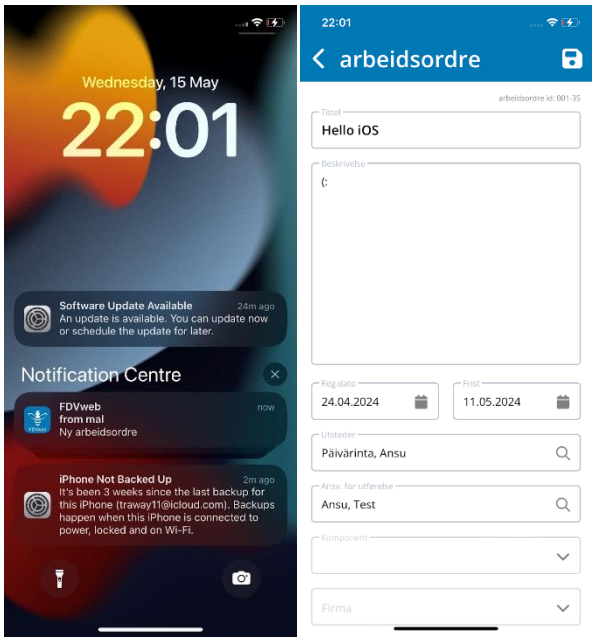

Figure 2 - iOS notification.

TODO: going forward

There are still a few things that need to be improved before the solution can be published as part of FDVweb.

- The API is still in need of an endpoint to deal with the deletion of subscriptions when a user opts out of receiving notifications. Although they will stop receiving notifications the moment they opt out, the now inactive subscription is still retained in the DB, which in time will lead to the DB being cluttered with old, inactive subscriptions.
- Additional triggers for when to send notifications need to be added. Exactly which events are going to trigger notifications is yet to be fully determined.
- Revising the method that creates a notification so that it takes all payload data in as parameters instead of the currently hard coded values for some of the fields.

Reflection

Early on it became clear to me that I had to focus mainly on either the practical or academic side of the project. I quickly opted to go for the former in order to maximize acquisition of real life skills and to the best of my ability integrate myself with the team of developers at CuroTech for an invaluable learning experience. This choice led to me having to somewhat disregard the more academic, university-centric part of the project, with my work capacity being the limiting factor.

Environment

Working at CuroTech has provided me with a stress-free work environment, where the focus is not on absolute deadlines, but rather on solving the problems at hand. This lack of external pressure has allowed some sort of internal drive to step in, and has at times resulted in me working the sort of hours that previously seemed impossible. I am truly grateful for the welcoming culture they have managed to attain and it has single handedly rekindled my joy for learning.

Challenges

There were occasions where seemingly simple issues wasted hours, and even days of production time. Not to mention the challenges arising from Apple's unique demands regarding some parts of the implementation.

For one, there is this thing called cross-origin resource sharing, CORS for short, that I got to become acquainted with during the creation of my proof of concept. In short, it is an HTTP-header based mechanism put in place to allow a server to explicitly indicate which origins, other than its own, should be able to access its resources (mdn, 2024). Working with it turned out to be extremely fiddly and straight up painful. There were times when I finally got things to work how I wanted them to, only for things to break a few days later without having touched any code in charge of CORS permissions. Luckily this stopped being an issue once I moved on from the POC, and things are finally working as expected.

Another issue that springs to mind is that at one point, while working on the proof of concept, I had erroneously written http://* rather than https://* at one spot in the code. This was luckily spotted on the same day, although not before several hours had been spent trying to find the error.

References

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, W., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001, Feb 11-13.). Manifesto for Agile Software Development. Retrieved 16. May 2024 from

https://agilemanifesto.org/

coreyjthompson. (n.d.). web-push-csharp. In *github*. Retrieved 15. May 2024 from

https://github.com/web-push-libs/web-push-csharp

mdn. (2024, Mar 6). *Notifications API.* Retrieved 15. May from

https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API

Gaunt, M. (2016, June 30). The Web Push Protocol. In *web.dev*. Retrieved 16. May 2024 from

https://web.dev/articles/push-notifications-web-push-protocol

mdn. (2024, Apr 22). *PushSubscription.* Retrieved 15. May 2024 from

https://developer.mozilla.org/en-US/docs/Web/API/PushSubscription

Beverloo, P., Thomson, M., Caceres, M. (2023, Dec 11). *Push API.* W3C. Retrieved 15. May 2024 from

https://www.w3.org/TR/push-api/

Sending web push notifications in web apps and browsers. (n.d.). In *Apple Developer.* Retrieved 14. May 2024 from

https://developer.apple.com/documentation/usernotifications/sending-web-push-notifications-in-web-apps-and-browsers

Appendix

Self-evaluation

Concrete learning

During my time working on this project I have become comfortable with two programming languages and several other tools. Having barely touched C# using .NET in a previous semester, I am now at a level where I feel comfortable both programming in C# and navigating its documentation. My previously basic JavaScript skills have also improved. Additionally I have gained experience in the use of Entity Framework Core with SQL Server, both in creating/updating a database structure as well as writing queries to manipulate existing data. I have also been using SQL Server Management Studio to gain an overview of the database and make sure everything worked as intended. Finally, I have gained additional practice in working with git through Azure DevOps as well as a basic introduction to their Pipeline feature.

New concepts I have learned during this project include the web push protocol, serviceworkers, progressive web applications, as well as practice writing APIs, services and interfaces. Not to mention more hands on experience with the use of the MVC, and Dependency Injection patterns.

Office attendance

I went to the office between 1-4 days per week, most weeks, with some weeks spent soaking in home studies for when I felt the need to sort out lack of understanding of some topics at hand. This, coupled with repetition of the parts that I did understand, ended up proving to be a functional tool and resulted in the sought after increase of retained knowledge.

Seeking assistance

In the past, I rarely sought assistance when facing difficulties with any given task, much to the detriment of progress. During this project, however, I have been quick to seek guidance from the team members available at CuroTech on occasions where I started to feel like I was banging my head against the wall. This is in and of itself a big improvement on the personal level, but I could have been even better at seeking out assistance, mainly regarding the university side of the project, as I have not employed the use of my assigned supervisor as of the writing of this assessment.

Missed opportunities

I did end up missing the second of two status presentations of the project. Additionally, I did not utilize the opportunities provided to meet up with both a representative from the client company and my university supervisor. I do regret the missed potential positive impact this could have had on my final report, though I can not say for certain that I would have had the energy or capacity to turn said potential into real value.

Personal insights

In addition to the above discussed improvement regarding seeking assistance, I have gained valuable insights into my own capacity to work both independently and in a team setting. During this project I have been thrown out of my comfort zone weekly, if not daily, which has taught me valuable lessons in both resilience and problem solving.

Notat

Pushvarsling i FDVweb

Ansu ble invitert til å gjennomføre sin bacheloroppgave for oss desember 2023, med avslutning våren 2024. Prosjektets utgangspunkt var å hjelpe oss å kode inn «push-varsling» på vår PWA-app for FDVweb.

FDVweb er et skybasert drift- og vedlikeholdssystem som dekker alt fra ordinære kontorbygg og boligkomplekser til avanserte produksjonsanlegg. Som system er FDVweb utviklet med fokus på et logisk og enkelt brukergrensesnitt, for å minimalisere opplæringsbehovet samt sikre at alle rutiner blir fulgt opp i praksis.

Implementering av pushvarsling på vår applikasjon har vært en del av utviklingsplanen vår og Ansu ble tidlig tatt inn i prosessen med utviklingen. Som del av prosessen har Ansu vært involvert i både kartlegging, planlegging og utvikling av pushvarslingen de siste 5 månedene. Ansu har jobbet både selvstendig og tett med utviklingsteamet for å etablere løsningen som nå er i test-fase. Ansu har nådd milepælene på veien og tidvis overgått våre forventninger til hvor kjapt han har satt seg inn i eksisterende produkt og kode. Løsningen har blitt beta testet og ligger i planlagt utrulling til 5,000 brukere av FDVweb før sommeren.