CERN SUMMER STUDENT PROGRAMME 2017

CERN: EP-AID-DA

ALICE Experiment

Final Report

Web Notification Infrastructure for the ALICE O²

Student
Anirudh Goel

Supervisor
Adam Wegrzynek

12 June - 8 September
Contents

I. Introduction

II. Motivation

III. Implementation
   A. Push Notifications using Web Sockets
   B. Push Notifications using Service Workers
   C. Push Notifications using Apple Push Notification Service (APNS)
   D. Tests and Documentation

IV. Result

V. Future Perspective

VI. Acknowledgement

VII. References
Introduction

The ALICE Experiment is being prepared for major upgrade of its computing system that will take place during Long Shutdown 2. The upgrade foresees software, hardware replacement and integration of following sub-systems: Data Acquisition chain and High Level Trigger. The new, common system will be able to collect 100x more data than during Run 2. It will require advanced and intuitive user interfaces to control and smoothly operate the experiment.

During 13 weeks Summer Student Programme, I worked in the Data Acquisition Team of the ALICE Experiment on developing the Web Notification Infrastructure for the future ALICE O² System.

The infrastructure can push the notification via 3 different means of transport:

- Using Web Sockets [1],
- Using Service Workers [2] (Chrome, Mozilla and other major browsers),
- Using Apple Push Notification Service [3] (APNS) for Safari as it doesn’t support Service Workers as of now (latest version till date - 10.1.2).

Initially, I started my work by understanding the existing code and working on implementing Push Notifications with Web Sockets. During ALICE Work Package 8 meeting I gave a presentation on ‘Web Push Notifications’ showcasing the work that I had done after 4 weeks of my programme. This presentation included a brief introduction of Push Notifications, Service Workers and a live demo of Push Notifications in browsers that support Service Workers. Afterwards, I proceeded with integration of the Push Notifications for Safari using Apple Push Notification Service (APNS) and simultaneously, I also covered my code with Mocha [4] tests, provided with JSDoc [5] documentation and used Travis [6] for continuous integration.
Motivation

Web Push Notifications were selected as the default way of notifying users in the web application of ALICE O² as they are modern, simple, effective, customisable and responsive. They are displayed in the form of subtle pop up messages, so the user doesn’t need to open a different application to receive them. In addition, the users can choose what type of notifications they want to receive.

Implementation

A. Push Notifications using Web Sockets

Web Sockets is a communication protocol that establishes a persistent connection between the web browser client and the server. Both parties can exchange data at any time. The new ALICE O² web applications will use Web Sockets to communicate with the backend.
I implemented WebSocket Push Notifications as jQuery Widget [7] that checks if the message received by any web browser client is of type ‘Notification’. Then, if the user has enabled push notifications in his browser, the widget obtains the content of notification from the message and generates a notification.

To test the infrastructure I developed a dummy agent that keeps on sending notifications over ZeroMQ [8] at fixed intervals. Then the notification are received by web application backend, translated into WebSocket messages and pushed to all subscribers where they are parsed and presented as Push Notifications. This dummy message agent can later be used as the actual server inside the machine to send error or log messages to the clients.

A limitation of the Web Socket approach is that it requires the web application to opened in the browser otherwise the messages are not received.

**B. Push Notifications using Service Workers**

**Service Workers** are background processes that run in a browser. They are lightweight and they can be used to receive messages from a Server and display them as notifications. Service Workers are currently supported in the latest versions of Google Chrome, Mozilla Firefox and Opera. Safari doesn’t support Service Workers but Push Notifications can be sent using Apple Push Notifications Service (APNS), which is discussed in the next section.

The major advantage of sending Push Notifications using Service Workers is that the user doesn’t need to open the web application or even the browser to receive them. Once the user has subscribed to the notifications, a service worker is registered in the browser and it receives the notifications regardless of the fact that the browser is running or not.

Integrating this feature required development of two components-

1. Registering a service worker and receiving the notifications on client side.
2. Generating the notification on server side and sending it to the clients.

Registering a service worker and receiving notifications
I followed the tutorial by Matt Gaunt [9], that guided me through:
• registering a service worker
• generating application server VAPID Keys
• creating the UI and initialising it
• subscribing and unsubscribing the user
• handling it if the user denies permission for push notifications
• handling an incoming push request using service worker

Based on it, I’ve developed the client part of this infrastructure, which registers a service worker on client system and handles the part of receiving and displaying notifications from the server.

Generating the notification on server side and sending it to the clients
While implementing this feature, I discovered different browsers have different guidelines for its implementation. Hence, to avoid the hassle of managing different keys (GCM [10], FCM [11] and VAPID [12]) and also to handle legacy support, I decided to use a Node.js package called ‘web-push’ [13].

For developing this part, I followed another tutorial by Matt Gaunt [14].
This tutorial walked me through:
• saving a subscription object on the server after the user has subscribed
• fetching these subscription object and sending push messages to them

On the top of that I added features to differentiate types of notifications and a preference panel where people could subscribe to what kind of notifications they want to receive. As notification types can change in time, I implemented them in a generic way so they can be modified easily.

The major challenges involved in developing these features were:
• finding reliable resources for understanding it as it’s relatively new technology
• choosing correct cloud platform; as Google recently announced that GCM (Google Cloud Messaging) will be replaced by FCM (Firebase Cloud Messaging), but some older versions of browsers work with GCM, so it was after a lot of searching around that I decided to use the ‘web-push’ package.

• making the preferences section generic, I had to write the code in such a way that the number of modifications needed to be performed while changing the preferences were minimal. After brainstorming through a lot of different approaches, I finally implemented it in such a way that the whole preferences section is generated dynamically from a Javascript array, so the developer only needs to change the options in that array and it is ready to go.

C. Push Notifications using Apple Push Notification Service

Safari currently doesn’t support Service Workers, so the ‘web-push’ package used for other browsers cannot work with it. Apple provides another way to send push notifications on macOS - **Apple Push Notification Service (APNS)**. Push Notifications sent using APNS work just like push notifications for apps. They display website icon and custom notification text, which users can click to go right to our website and they don’t even need Safari to run in the background.

Implementing this feature had a lot of challenges, as it needed multiple steps and the Official Apple Documentation [15] lacks a lot of information. I found the tutorial by Samuli Hakoniemi [16], which states the information missing in the official docs very clearly. So, I followed both of these tutorials to implement the feature. The main challenges that I faced during this are:

• As compared to the relatively easy first step of registering a Website Push ID, **generating a .p12 certificate** required a lot more efforts as the Official Apple Docs
don’t list the steps to do it and this process was very confusing even with Samuli’s tutorial.

- The next major challenge was creating the push package. This part turned out to be the most difficult as the Companion File [17] provided in the Official Apple Documentation has some ambiguous fields and the code finally worked after I discovered that some fields are redundant and they need to be commented out.
- Also, the structure of the pushpackage directory was different from what is specified in either of the documentations. So, I figured all this out after browsing different resources and finally created a documentation of my own, which you can find on the Github repository [18][19].
- One final challenge was to integrate the APNS and Service Worker Push Notifications into one system. This turned out to be a tradeoff between redundant and manageable code, because both of these technologies had some common functions but to keep the code generic, some amount of redundancy had to be introduced.

**D. Tests and Documentation**

The last task that I performed was writing tests for my code. I wrote extensive Mocha tests using Assert and Chai assertion library. I wrote both, pass and fail tests for every function in my code and as a result the code coverage for my code was as high as 91.3%.

I also wrote very detailed documentation about the features that I had developed. This documentation contains all the steps needed for installation and configuration of the app. It also explains the architecture and working of the system with appropriate illustrations, workflows and tutorial links.
Result

After skimming through several tutorials, comparing multiple frameworks, libraries, and packages and writing about 3000 lines of code, the results I finally obtained were:

- A fully functional Notification Architecture, which supports notifications by 3 different methods (web sockets, service workers and APNS).
- A notification trigger for Web Sockets.
- A unified notification trigger which acts as a common trigger for sending notifications to both, browsers supporting service workers and to APNS.
- An easily customisable preference panel where the users can choose the type of notifications that they want to receive.

All of my code and corresponding documentation is available on Github [18][19].

Future Perspective

Apple recently announced that Safari will start supporting service workers, so the future developments that can be performed in this architecture is that once the service workers are supported by Safari, the APNS part of this architecture can be modified to make it simpler or can be removed altogether if it becomes redundant.
Acknowledgement

I am highly grateful to Adam Wegrynek, for being a polymath supervisor. He always gave me challenging work which helped me to learn a lot of new things. I worked on technologies that were completely new for me including Node.js, Web Sockets, Push Notifications, APNS, Gulp, Travis and lot more.

Also, I express my thanks to all the people at CERN who made this summer the greatest experience of my life.

References

[2]: https://developers.google.com/web/fundamentals/getting-started/primers/service-workers
[3]: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW1
[4]: https://mochajs.org
[5]: http://usejsdoc.org
[6]: https://travis-ci.org/
[7]: https://jqueryui.com/widget/
[8]: http://zeromq.org
[9]: https://developers.google.com/web/fundamentals/getting-started/codelabs/push-notifications/
[10]: https://developers.google.com/cloud-messaging/
[13]: https://www.npmjs.com/package/web-push
[16]: http://samuli.hakoniemi.net/how-to-implement-safari-push-notifications-on-your-website/#registering-a-website-push-id
[18]: https://github.com/AliceO2Group/ControlGui
[19]: https://github.com/AliceO2Group/Gui