ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THPV011

NOTIFICATIONS WITH NATIVE MOBILE APPLICATIONS

B. Bertrand*, J. Forsberg, MAX IV, Lund, Sweden
E. Laface, G.Weiss, European Spallation Source ERIC, Lund, Sweden

Abstract

Notifications are an essential part of any control system. Many people want to be notified of specific events. There are several ways to send notifications: SMS, e-mails or messaging applications like Slack and Telegram are some common ones. Those solutions frequently require some central configuration to record who will receive messages, which is difficult to maintain. ESS developed a native mobile application, both for iOS and Android, to manage notifications. The application allows the users to subscribe to the topics they are interested in, removing the need for a central configuration. A web server is used as gateway to send all notifications following Apple and Google protocols. This server exposes a REST API that is used both by clients to send messages and mobile applications to retrieve and manage those messages. This paper will detail the technical implementation as well as the lessons learnt from this approach.

INTRODUCTION

The European Spallation Source (ESS) is under rapid development in Lund, Sweden. More and more parts of the control system are put into place, which means a growing number of messages and alarms are triggered. People want to be notified to keep track of what is happening and to know if any action is required. There were several ways to send notifications depending on the application. IT had a service to send SMS, some applications were relying on e-mails. We also developed Telegram and Slack bots. The Telegram bot was quite popular as users could check messages on their phone via the native mobile application. The issue was that the configuration was centralized: each new user had to be added manually with the list of topics he was interested in. That was demanding to maintain and didn't scale well. The other problem was the inclusion in existing applications which wasn't trivial. We wanted a general purpose solution that would be easy to use from any application and would unify the user experience.

CONCEPTS

We wanted users to be able to subscribe themselves to the notifications they are interested in. This would remove the need for a central configuration that is laborious to manage. To achieve this, notifications have to be grouped in categories named *services*. The number of *services* doesn't have any limit in theory. In practice, too many services would make it difficult to decide which to subscribe to. With too few services, there is a risk that the user will receive many unwanted notifications. Some services used at ESS are called Logbook On-Call, Logbook TS2, OpenXAL and Prometheus CSI. Those examples show that a service can be linked to an application (like OpenXAL), but that one application can also send messages to different services (like the LogBook).

Smartphones are part of an infrastructure that makes it easy to dispatch notifications. We chose to develop a specific mobile application that we could customize to our need. This application would be used to subscribe to the *services*, receive notifications and read messages. Two clients are available, one for Apple iOS devices [1] and one for Android users [2].

Sending notifications to a mobile phone can be done relying on Apple and Google infrastructure. We decided to design a REST API that is used both to communicate with the mobile clients and to forward the notifications received from the system. Sending a notification only requires a POST to this central server, named Notify server [3], making the integration in existing application simple.

NOTIFY SERVER

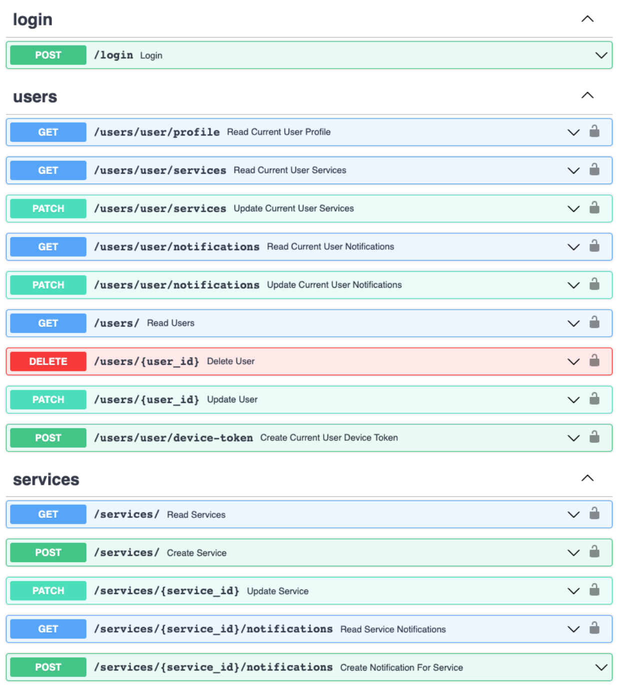
The Notify server was developed with FastAPI [4], an async Python web framework, that quickly became very popular in the past years. It is based on Starlette [5], a lightweight ASGI framework, and Pydantic [6], a data validation library using python type annotations. PostgreSQL [7] is used as database. FastAPI was designed to make writing API easy and is based on OpenAPI standard [8]. It automatically generates an interactive API documentation with exploration via Swagger UI [9]. This interface, showed in Fig. 1, is used by admin users to perform basic operation, like creating new services.

Message Sending

As described earlier, a notification has to be linked to a *service*. Only admin users can register a new service. When creating a new service, an UUID is generated and used to identify the service. This id has to be used by clients to send a notification associated to that service. This is done by sending a POST to */services/{service_id}/notifications* with the fields *title*, *subtitle* and *url* in the body. Only the title is required. Other fields are optional. The subtitle usually contains a longer description. The URL can be used to redirect to a specific website, like a LogBook entry, to have more details. This link will make the messages clickable in the mobile app. Figure 2 and Figure 3 show how to send a notification using curl or Python.

As there is no authentication, a filtering based on IP address is performed to avoid receiving messages from an untrusted source. Once received, the message is forwarded to all users who subscribed to this service using Apple and Google push infrastructure.

* benjamin.bertrand@maxiv.lu.se



login

- POST /login Login

users

- GET /users/user/profile Read Current User Profile
- GET /users/user/services Read Current User Services
- PATCH /users/user/services Update Current User Services
- GET /users/user/notifications Read Current User Notifications
- PATCH /users/user/notifications Update Current User Notifications
- GET /users/ Read Users
- DELETE /users/{user_id} Delete User
- PATCH /users/{user_id} Update User
- POST /users/user/device-token Create Current User Device Token

services

- GET /services/ Read Services
- POST /services/ Create Service
- PATCH /services/{service_id} Update Service
- GET /services/{service_id}/notifications Read Service Notifications
- POST /services/{service_id}/notifications Create Notification For Service

Figure 1: Notify server API

```
curl -X 'POST' \
  'https://notify.maxiv.lu.se/api/v2/services/f848b451-e3c8-497b-a6da-2cb571d9d95e/notifications' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "title": "My message",
  "subtitle": "This is a test",
  "url": ""
}'
```

Figure 2: Sending a notification with curl

```
data = {"title": title, "subtitle": subtitle, "url": url}
response = httpx.post(
    f"{server}/api/v1/services/{service_id}/notifications", json=data
)
```

Figure 3: Sending a notification from Python

Communication with Mobile Clients

Mobile clients need to authenticate to access the REST API. To login, username and password are sent to the server and checked against an Active Directory service. If the credentials are correct, the server sends back a JSON Web Token (JWT). This token shall be included in the header of any future request. It is valid for a limited number of days and the user will only be forced to authenticate again when it has expired. This solution allows to not store any credentials in the phone or server.

Once logged in, the mobile app can use the API endpoints to get the list of services, subscribe or unsubscribe and read notifications linked to the current user. For notifications to be sent to that phone, the app must send a token to identify the device. This is done via the */users/user/device-token* endpoint. This device token is associated to the user and saved on the server. This is the token used to send notifications via Apple or Google services.

The workflow to send a notification is depicted in Fig. 4:

1. An application sends a message to the Notify server (the message is linked to a service).
2. The server sends a notification to both Apple Push Notification service [10] and Firebase Cloud Messaging [11] (depending on the device token type) for all users who subscribed to that service.
3. The notification is sent by Apple and Google cloud services to the user device.
4. When the user opens the notification, the full message is read from the Notify server API.

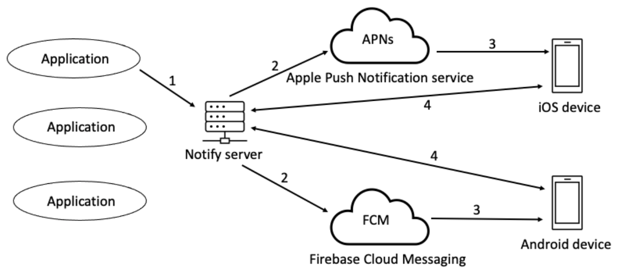


Figure 4: Notification workflow

The Notify server is deployed using docker in a Virtual Machine at ESS and on Kubernetes at MAX IV. All site specific parameters can be defined in a configuration file.

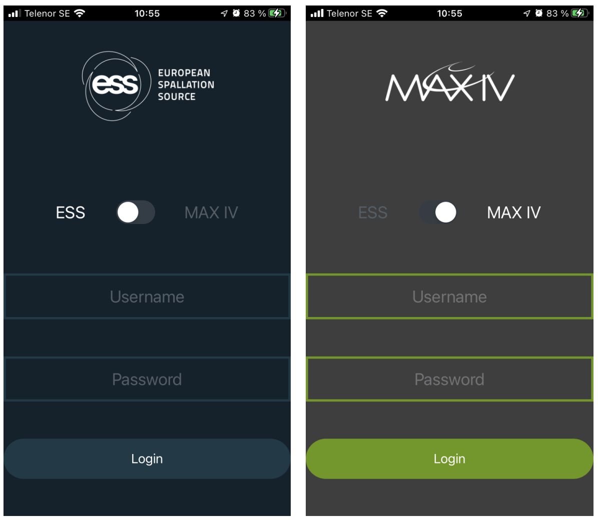
APPLE IOS CLIENT

The iOS client is a GUI for the REST API described previously. Once installed it will contact the server through a link that is hard coded in the application. In our current version we support two possible links and customizations, one for ESS server and one for MAX IV server as in Fig. 5. As described previously, upon successful login a JSON Web Token is stored in the app and will be used for future authentications.

The app will also ask the user to accept notifications and in case of an affirmative answer it will request to Apple a push token that is sent to the Notify server. This token is stored in the profile of the user and used to push any new notification to the phone through the Apple Push notification API.

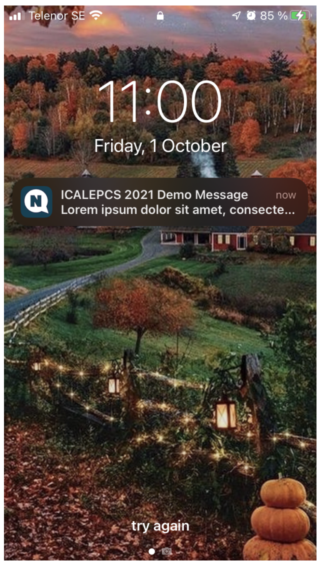
Once the app is properly registered on the server the available services can be selected for subscription (Fig. 6). The app will communicate to the server which service the user wants to subscribe to and the server will associate the push token of the user to the service. In this way every time a new message is delivered to a service, all the subscribers will be notified about the new message (Fig. 7).

When the app is opened, the main screen will display the list of available notifications (Fig. 8a), these are downloaded from the server at every startup of the app or if a new notification arrives and the app is running on the phone. The notifications are marked with a red dot if unread, and a badge



(a) ESS (b) MAX IV

Figure 5: Login Screen



ICALEPCS 2021 Demo Message
Lorem ipsum dolor sit amet, consecte...

Figure 7: Push notification alert

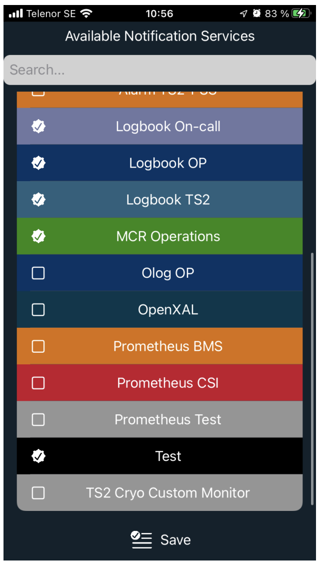
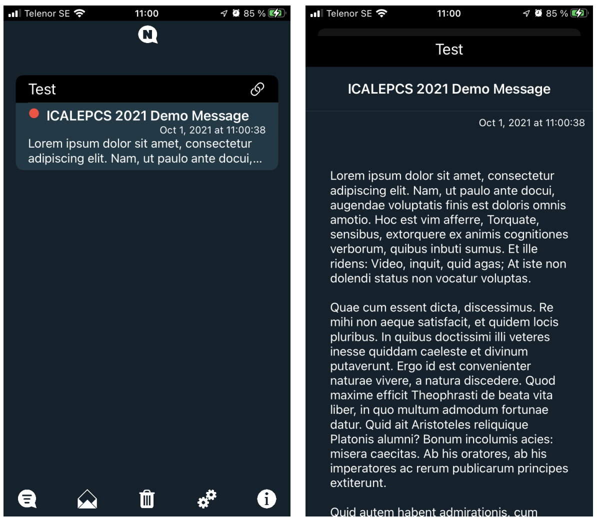


Figure 6: Services



ICALEPCS 2021 Demo Message
Oct 1, 2021 at 11:00:38
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam, ut paulo ante docui,...

ICALEPCS 2021 Demo Message
Oct 1, 2021 at 11:00:38

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam, ut paulo ante docui, augendae voluptatis finis est doloris omnis amotio. Hoc est vim afferre, Torquate, sensibus, extorquere ex animis cognitiones verborum, quibus inbuti sumus. Et ille ridens: Video, inquit, quid agas; At iste non dolendi status non vocatur voluptas.

Quae cum essent dicta, discessimus. Re mihi non aeque satisfacit, et quidem locis pluribus. In quibus doctissimi illi veteres inesse quiddam caeleste et divinum putaverunt. Ergo id est convenienter naturae vivere, a natura discedere. Quod maxime efficit Theophrasti de beata vita liber, in quo multum admodum fortunae datur. Quid ait Aristoteles reliquique Platonis alumni? Bonum incolumis acies: misera caecitas. Ab his oratores, ab his imperatores ac rerum publicarum principes extiterunt.

Quid autem habent admirationis, cum

(a) Messages list (b) Single message view

Figure 8: Main Screen

with the number of unread notifications is shown on the icon of the app. The top bar of the notification contains the name and the color of the service that is notifying, and on the right corner there is a button with a link to the URL sent together with the message, if present.

Tapping on a notification is possible to expand the text and read the full content (Fig. 8b).

Deployment

To be able to send push notifications to an Apple device the developer needs to request a push key that has to be used in order to send the notifications to the app. This is the reason why we developed a unique app with two possible logins (ESS and MAX IV), because in this way we can use the push key registered under the ESS license to push messages also from MAX IV. The alternative is that MAX IV register its own developer on the developer program of Apple and requests a key for push notifications that is set in the Notify server settings. This is under consideration for the future.

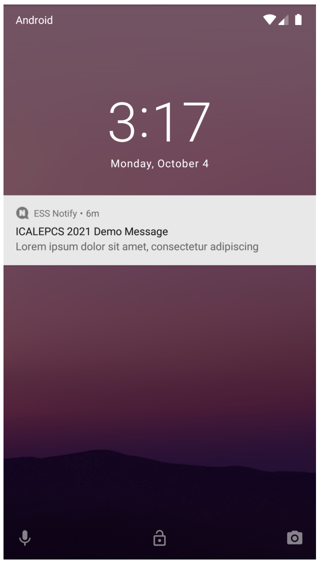
Apple offers multiple way to deploy applications. The most common way is to use the App Store, but our application has an access restriction to the users of ESS and MAX IV and Apple forbids the distribution of restricted applications on the App Store. The solution that we adopted to distribute the app is then through the Mobile Device Management (MDM) system. The whole process is exactly like to distribute the app through the App Store, it means that the app must be signed with the distribution signature generated on the Apple Developer portal and sent to the App Store Connect, that is the website that Apple uses to manage the distribution of apps. Once the app is on the portal and all the relevant information for distribution are filled, the app is sent to Apple for review as Private. A normal app in general is sent as public, but because we do not go to the App Store but we will stay within our MDM, the app has to be marked as private.

Apple will then verify the application running it, this means that together with the app they will require a demo account to test the functionalities, so we created in our LDAP server a test account that we enable every time we submit a new revision of the app for approval. After the checks of Apple the application is marked as Ready for Sale and appears automatically to the registered Business web page connected to our license for distribution. For ESS we use the Self Service application to distribute internal apps.

This process, except the push key that is always the same, has to be repeated for each version of the app submitted to Apple.

ANDROID CLIENT

The Android client was initially developed after the iOS counterpart, consequently the functionality and user interface (UI) were copied to be basically identical. Some minor difference in the UI layout exist, though. Figure 9 shows the lock screen showing a test notification message.



ICALEPCS 2021 Demo Message
Lorem ipsum dolor sit amet, consectetur adipiscing

Figure 9: Push notification alert

The Android client does not support selection between ESS and MAX IV, it currently only supports the ESS use case. To be used at MAX IV, the application was recompiled after changing the Notify server url. This was enough for testing purpose. More customization is planned to make the theme MAX IV specific.

The current code base of the Android client requires a minimum version of Android 7.0, which was released August 2016.

Deployment

Development of Android apps is free of charge, as are development tools. Installing an Android app by downloading an apk (Android application PacKage) file to any compatible target device - or using USB tethering - is possible and does not incur any limitations to the app usage. ESS has however opted for distribution over Google Play due to its simplicity. The Google Play approval process is slightly simpler compared to the process used by Apple, e.g. there is no test account needed in order for Google to be able to login to the application. This distribution will be considered by MAX IV when the app is customized.

In order to publish apps on Google Play one needs to register a developer account, currently priced at $25 (one-time charge). The developer account must be linked to a Gmail account.

APPLICATION INTEGRATION

As we saw earlier, sending a notification only requires a POST request to the Notify server. ESS integrated this system in several applications: LogBook, OpenXAL, EPICS alarms and Prometheus are some examples. Integration can be done in different ways depending on the application. For Prometheus, a bridge [12] was created to forward and curate messages from the Alert manager to the Notify server. EPICS alarms are sent via Kafka, which makes it quite flexible. A client is used to listen to the desired Kafka topics and forward alarms to the server. At MAX IV, preliminary tests were performed successfully with Achtung, a new alarm management system for Tango.

CONCLUSION

The solution put in place at ESS relies on modern tools and native mobile applications for both iOS and Android. The management of who will receive notifications is delegated to the users themselves, which removes the need for a central configuration that can be fastidious to keep up to date. This is really beneficial to the users who can subscribe and unsubscribe as they want, from their phone. It gives an unified way to receive notifications from different systems. The implementation is generic and can be re-used. The Notify server was straightforward to deploy at MAX IV and the concept was tested successfully. The customization of the mobile clients requires a bit more work but nothing major.

REFERENCES

[1] E. Laface. (2021) Notify iOS Client. https://gitlab.esss.lu.se/ics-software/ess-notify

[2] G. Weiss. (2021) Notify Android Client. https://gitlab.esss.lu.se/ics-software/ess-notify-android

[3] B. Bertrand and E. Laface. (2021) Notify Server. https://github.com/EuropeanSpallationSource/notify-server

[4] S. Ramírez. FastAPI. https://fastapi.tiangolo.com

[5] T. Christie. Starlette. https://www.starlette.io

[6] S. Colvin. pydantic. https://pydantic-docs.helpmanual.io

[7] PostgreSQL. https://www.postgresql.org

[8] OpenAPI standard. https://github.com/OAI/OpenAPI-Specification

[9] Swagger UI. https://github.com/swagger-api/swagger-ui

[10] Apple Push Notification service. https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/sending_notification_requests_to_apns/

[11] Firebase Cloud Messaging. https://firebase.google.com/docs/cloud-messaging

[12] A. Curri. (2021) Alertmanager To Ess Notify. https://gitlab.esss.lu.se/ics-infrastructure/alertmanager-to-ess-notify