Firebase Push Notifications

Namazu Elements Core Features · https://namazustudios.com/docs/namazu-elements-core/firebase-push-notifications/

Namazu Elements includes a full server-side push notification system built on the Firebase Admin SDK. It has three parts: a per-Application Firebase configuration, a device registration token stored per Profile, and a general-purpose notification-sending API that your own server-side code calls to actually deliver a push. There is no REST endpoint for triggering a send; sending is meant to be driven by your own business logic, typically from a Custom Element. See Custom Elements if you haven't built one yet.

Configuring Firebase for an Application

Firebase credentials are configured per Application, as one of that Application's Application Configurations, alongside things like IAP platform and matchmaking configuration. There is no server-wide Firebase configuration; each Application that wants to send push notifications needs its own.

Field	Description
`projectId`	The Firebase project ID.
`serviceAccountCredentials`	The full JSON contents of a Firebase service account key file (not a file path). This is the same file you'd generate from Firebase Console under Project Settings > Service Accounts > Generate New Private Key.

Method	Path	Description
`POST`	`/application/{applicationNameOrId}/configuration/firebase`	Creates the Application's Firebase configuration.

Method	Path	Description
`GET`	`/application/{applicationNameOrId}/configuration/firebase/{applicationConfigurationNameOrId}`	Gets a Firebase configuration.
`PUT`	`/application/{applicationNameOrId}/configuration/firebase/{applicationConfigurationNameOrId}`	Updates a Firebase configuration.
`DELETE`	`/application/{applicationNameOrId}/configuration/firebase/{applicationConfigurationNameOrId}`	Deletes a Firebase configuration.

Managing an Application's Firebase configuration is Superuser-only; there is no User or Anonymous access to this endpoint. If an Application has no Firebase configuration attached, attempting to send a notification through it fails with a configuration error rather than silently doing nothing.

Registering a Device Token

Once a client obtains a Firebase registration token (via the Firebase client SDK on the device), it registers that token with Elements against the current Profile:

Method	Path	Description
`POST`	`/notification/fcm`	Registers a Firebase token for a Profile.
`PUT`	`/notification/fcm/{fcmRegistrationId}`	Updates an existing registration (typically to refresh a rotated token).
`DELETE`	`/notification/fcm/{fcmRegistrationId}`	Removes a registration (e.g. on sign-out or uninstall).

A registration is just an id, a `registrationToken`, and the owning Profile. Each Profile can hold only one stored registration at a time; registering a new token for a Profile that already has one replaces it rather than adding a second device.

As a normal User, you can only manage the registration for your own current Profile: if you omit `profile` in the request body it defaults to your current Profile, and if you supply one that isn't yours,

the request is rejected. A Superuser can create, update, or delete a registration for any Profile. Anonymous callers cannot use this endpoint at all.

Sending a Notification

There is no REST endpoint to send a push notification. Sending is done programmatically, by injecting `NotificationService` into your own Custom Element and building a notification with its fluent builder:

```
@Inject
private NotificationService notificationService;

public void notifyPlayer(Profile recipient) {
    notificationService
        .getBuilder()
        .recipient(recipient)
        .title("Your turn!")
        .message("It's your move in Match #482.")
        .sound()
        .add("matchId", "482")
        .build()
        .send();
}
```

The builder exposes:

- `recipient(Profile)` - the Profile to notify. Elements looks up that Profile's registered token(s) to determine where to actually deliver the push.
- `sender(Profile)` / `application(Application)` - which Application's Firebase configuration to send under. If you set a sender Profile, its owning Application is used automatically; you rarely need to call `application(...)` directly.
- `title(String)` / `message(String)` - the notification's title and body text.
- `sound()` / `sound(String)` - enables a notification sound, defaulting to `"default"`, or a specific named sound.
- `add(key, value)` / `addAll(Map)` - a custom string key/value data payload delivered alongside the notification, for your client code to act on (e.g. deep-linking to a specific screen).

`build()` throws a configuration exception if the target Application has no Firebase configuration attached. `send()` returns the number of destinations notified, and has an overload that takes success/failure callbacks per destination:

```
int notified = notificationService
    .getBuilder()
    .recipient(recipient)
    .title("Your turn!")
    .message("It's your move in Match #482.")
    .build()
    .send(
        event -> logger.info("Delivered to token {}", event.getTokenId()),
        error -> logger.warn("Delivery failed", error)
    );
```

Under the hood, each call sends a single, direct-to-device Firebase message per registered token for the recipient (there's no topic broadcast or multicast batching). Android and iOS-specific notification fields are populated automatically from `title` / `message` / `sound`, so you don't need to build platform-specific payloads yourself.

Error Handling

If Firebase reports a token as unregistered (the device uninstalled the app, or the token otherwise expired), Elements automatically deletes that Profile's stored registration so you don't keep sending to a dead token. Any other delivery failure (network error, malformed payload, and so on) is only reported to the failure callback you supply; there is no built-in retry or backoff, so retry logic is your Element's responsibility if you need it.

Events

Registering, updating, and deleting an FCM registration each publish an event your Element can subscribe to (registration created, updated, and deleted), the same way most other data types in Elements do. See Events. Sending a notification itself does not publish an event; use the success/failure callbacks on `send(...)` if you need to react to delivery outcomes.