Back End Development 1

Sesi 21

Best Practice : Integration Firebase Cloud Messaging with Spring-boot and Secure Spring-boot App

Introductions

Firebase Cloud Messaging adalah suatu service yang disediakan oleh Firebase untuk mendapatkan *push notification* pada perangkat Android dan iOS atau bahkan front-end.

Mengapa *push notification*? Jika sebuah perangkat melakukan *pull* setiap sekian detik untuk mengecek notifikasi, aplikasi akan menjadi boros daya, *server* yang melayani akan down jika dilakukan oleh banyak perangkat dalam waktu yang bersamaan, dan akan ada *delay* untuk mendapatkan notifikasi sehingga tidak *realtime*.

Firebase Cloud Messaging sangat bermanfaat untuk sebuah aplikasi yang membutuhkan *push notification* seperti aplikasi kami.

Nah pada sesi ini kita akan belajar bagaimana integrasi Firebase Cloud Messaging pada app Back End dalam hal ini. springboot .

Sebelum memulai alangkah baiknya kita perhatikan Arsitektur Project yang akan kita buat.

Understanding the High-Level Architecture

Untuk memulai, penting untuk memahami : bagaimana project ini akan bekerja.

Saat pengguna pertama kali membuka aplikasi, kita akan request permissions untuk mengirim notifikasi. Jika hasilnya memberikan akses (grant permission), maka Firebase akan mengirimkan token untuk mengidentifikasi perangkat mereka.

Kemudian, klien mengirimkan token ke aplikasi Spring Boot kita sehingga kita dapat menggunakannya untuk mengirim notifications ke pengguna tersebut.

Setiap kali backend kita ingin mengirim notifikasi, maka akan memberikan detail tentang notifikasi yang diinginkan ke Firebase.

Dari proses ini lah, backend Firebase akan mengirimkan notifikasi ke perangkat yang benar.

What will Happen : Server-Side

Di sisi server, aplikasi Spring Boot kita akan menggunakan SDK dari Firebase yang disebut Admin SDK. SDK ini memungkinkan aplikasi kita berinteraksi dengan Firebase untuk mengirimkan notifikasi kepada kita.

Saat inisialisasi Firebase di konsol Firebase, akan memberikan kita file JSON khusus yang akan digunakan untuk authorize aplikasi Spring Boot kita dalam mengirim notifikasi nantinya

Kapanpun kita ingin mengirim notifikasi, kita harus membuat objek Message, yang berisi semua info tentang notifikasi yang ingin dikirim. Termasuk mencakup judul, deskripsi, URL ikon, dan informasi khusus platform apa pun.

Kita memiliki dua cara untuk mengirim notifikasi - topik dan direct.

Lalu Apa itu Notifikasi Topic dan Direct ?

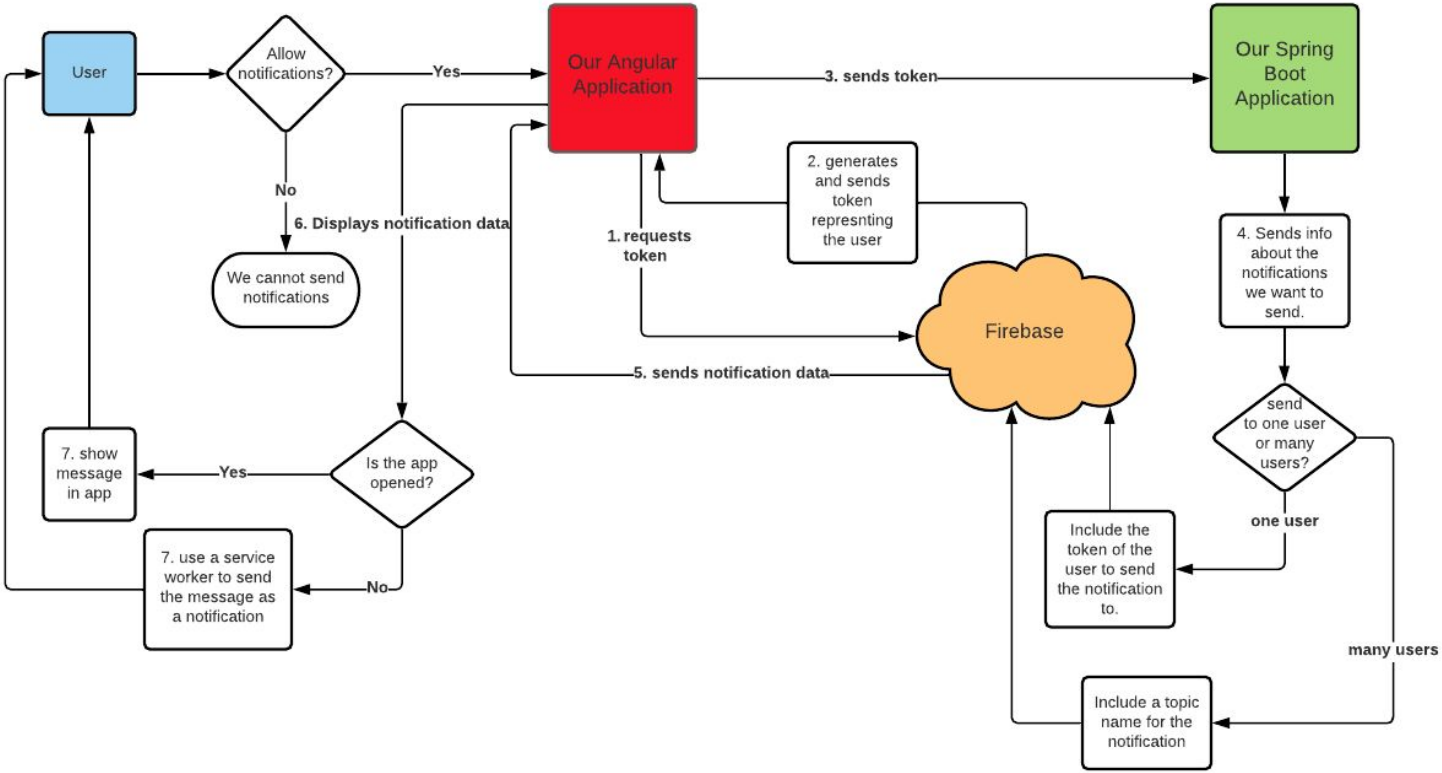
Topic Notifications vs Direct Notifications

Notifikasi topik adalah notifikasi dengan tag/spesifikasi tertentu yang dikenal sebagai topik. Pengguna akan berlangganan untuk diberi tahu tentang pesan apapun dengan topik pilihan mereka.

Setiap kali pengguna berlangganan suatu topik, sistem akan mengirimkan token mereka dan nama topik untuk berlangganan, ke aplikasi Springboot kita. Nah Dengan menggunakan alur ini, kita dapat memberi tahu Firebase untuk mengirimi mereka notifikasi tentang topik itu.

Selain itu, kita juga memiliki opsi untuk mengirim direct notifications untuk menentukan di objek Pesan, token pengguna untuk memberi notif. Kemudian, Firebase akan mengirimkan notifikasi ke pengguna tersebut.

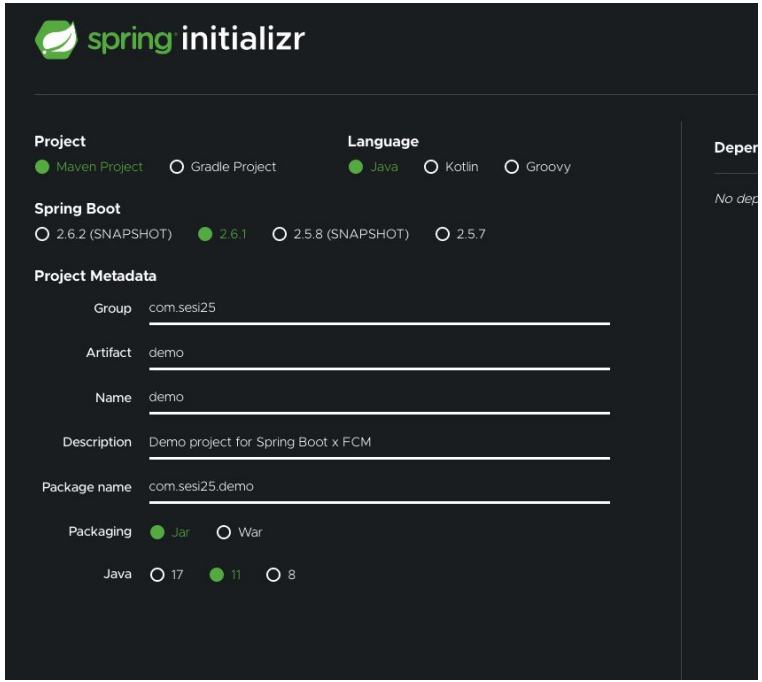
Gambaran FULL ARSITEKTUR



Setting Up Your Back-End

Pada sesi ini kita hanya akan fokus pada Back End Springboot.

Seperti biasa silahkan generate springboot project menggunakan Spring Initializer via https://start.spring.io/



Setting Up Firebase

1. Buka Firebase website dan lakukan login dengan akun Google.
2. Klik Go to console pada pojok atas kanan.
3. Pilih Add project untuk create a new Firebase project. Lanjutkan saja hingga bisa create the project.
4. Tekan icon gear di kiri-atas/sebelah Project Overview Button, lalu pilih projects settings.
5. Klik Service accounts di area atas tepatnya dibawah Project settings.
6. Generate Private Key Baru, dimana private key adalah file JSON yang sudah disebutkan diawal untuk authorize back end kita.
7. Tambahkan property ke application.properties file dengan file path ke private key:

```
app.firebase-config-file=firebase-config/[your-file-name-goes-here].json
```

Setelah kita punya private key, kita sudah bisa mulai integrasi-kan firebase dengan app Springboot

Sebelum memulai, letakkan file JSON pada resources folder dengan nama firebase-config.

Selanjutnya kita tambahkan Firebase Admin SDK ke project kita menggunakan MAVEN.

Sisipkan dependency dibawah ini diantara tag dependencies pada file pom.xml :

```
<dependency>
    <groupId>com.google.Firebase</groupId>
    <artifactId>Firebase-admin</artifactId>
    <version>7.2.0</version>
</dependency>
```

Selanjutnya, kita harus create a new service bean dimana yang akan kita gunakan untuk add Firebase ke BackEnd.

Gunakan anotasi @Value, Pertama-tama silahkan inject file path dari private key pada field :

```
@Service
class FirebaseInitializer {
    @Value("\${app.Firebase-config-file}")
    lateinit var FirebaseConfigPath: String
}
```

Springboot akan membaca nama property diantara ( ) dan melakukan inject dari value property pada field yang dituju.

Diantara class yang ada, kita juga perlu create a function annotated dengan @PostConstruct untuk mendapatkan akses ke firebase.
@PostConstruct memberi info ke Spring untuk menjalankan function setelah bean's properties di inisialisasi:

```
@Value("\${app.Firebase-config-file}")
lateinit var FirebaseConfigPath: String

// creates a logger we can use to log messages to the console. This is just to format our console messages nicely.
var logger: Logger = LoggerFactory.getLogger(FirebaseInitializer::class.java)

@PostConstruct
fun initialize(){
   // Get our credentials to authorize this Spring Boot application.
   try {
       val options = FirebaseOptions.builder()
               .setCredentials(GoogleCredentials.fromStream(ClassPathResource(FirebaseConfigPath).inputStream)).build()
       // If our app Firebase application was not initialized, do so.
       if (FirebaseApp.getApps().isEmpty()) {
           FirebaseApp.initializeApp(options)
           logger.info("Firebase application has been initialized")
       }
   } catch (e: IOException) {
       logger.error(e.message)
   }

}
```

Create FCM Service

Silahkan buat service untuk mengirimkan notifikasi dan subscribe users ke topik notifications. Sebelum itu kita akan buat beberapa model class untuk represent notifikasi nya :

```
abstract class AppNotification(open val title: String, open val message: String)

data class TopicNotification(val topic: String, override val title: String,
                             override val message: String): AppNotification(title, message)

data class DirectNotification(val target: String, override val title: String,
                              override val message: String): AppNotification(title, message)
```

Using these classes, we can create a function in our service to send direct notifications:

```
@Service
class FCMService {
    fun sendNotificationToTarget(notification: DirectNotification){
        val message = Message.builder()
                // Set the configuration for our web notification
                .setWebpushConfig(
                        // Create and pass a WebpushConfig object setting the notification
                        WebpushConfig.builder()
                                .setNotification(
                                        // Create and pass a web notification object with the specified title, body, and icon URL
                                        WebpushNotification.builder()
                                                .setTitle(notification.title)
                                                .setBody(notification.message)
                                                .setIcon("https://assets.mapquestapi.com/icon/v2/circle@2x.png")
                                                .build()
                                ).build()
                )
                // Specify the user to send it to in the form of their token
                .setToken(notification.target)
                .build()
        FirebaseMessaging.getInstance().sendAsync(message)
    }
}
```

Seperti yang Kita lihat, membuat notifikasi cukup mudah jika kita mengetahui pola desain pattern nya. Karena platform kita adalah web, kita ***pass*** objek `WebpushConfig` ke builder.

Terakhir, setelah mengatur data notifikasi, kita juga harus mengatur token untuk menentukan kepada siapa pesan akan disampaikan. Untuk itu kita memanggil method sendAsync untuk mengirim pesan.

Demikian juga, untuk mengirim notifikasi topik, kita membuat similar function di services.

Satu-satunya perbedaan adalah kita menentukan topik instead token:

Untuk menyelesaikan service ini, mari kita buat function untuk membuat pengguna bisa subscribe topik tertentu.

Buat objek model untuk represent subscription request :

```
// The subscriber field specifies the token of the subscribing user
data class SubscriptionRequest(val subscriber: String, val topic: String)
```

Dengan class yang sudah dibuat, berikut adalah fungsi untuk membuat pengguna subscribe suatu topik:

```
fun subscribeToTopic(subscription: SubscriptionRequest){

   FirebaseMessaging.getInstance().subscribeToTopic(listOf(subscription.subscriber), subscription.topic)

}
```

REST Controller

Sekarang setelah kita membuat service Firebase Cloud Messaging, Selanjutnya yang perlu kita lakukan hanyalah membuat REST Controller.

Misalnya, kita akan meminta klien mengirim request ke controller ini untuk mengirim notifikasi ke dirinya sendiri. A Real Production app tidak akan dibuild seperti itu.

```
@RestController
class NotificationController(private val fcm: FCMService) {
    @PostMapping("/notification")
    fun sendTargetedNotification(@RequestBody notification: DirectNotification){
        fcm.sendNotificationToTarget(notification)
    }

    @PostMapping("/topic/notification")
    fun sendNotificationToTopic(@RequestBody notification: TopicNotification){
        fcm.sendNotificationToTopic(notification)
    }

    @PostMapping("/topic/subscription")
    fun subscribeToTopic(@RequestBody subscription: SubscriptionRequest){
        fcm.subscribeToTopic(subscription)
    }
}
```

Configuring CORS

Hal terakhir yang perlu kita lakukan agar ini berfungsi adalah konfigurasi CORS.
Dengan cara ini, klien akan diizinkan untuk mengirim permintaan apa pun yang diinginkannya ke backend yang kita buat

Untuk melakukan ini, tambahkan bean berikut:

```
@Bean
fun cors(): WebMvcConfigurer {
    return object : WebMvcConfigurer{
        override fun addCorsMappings(registry: CorsRegistry) {
            // Allow our client (on localhost:4200) to send requests anywhere in our backend
            registry.addMapping("/**").allowedOrigins("http://localhost:4200")
        }
    }
}
```

Nah itulah cara melakukan integrasi app springboot dengan Firebase Cloud Messaging,
Selanjutnya nanti akan kita tampilkan notifikasi yang sudah kita buat pada saat belajar Front End (Angular).