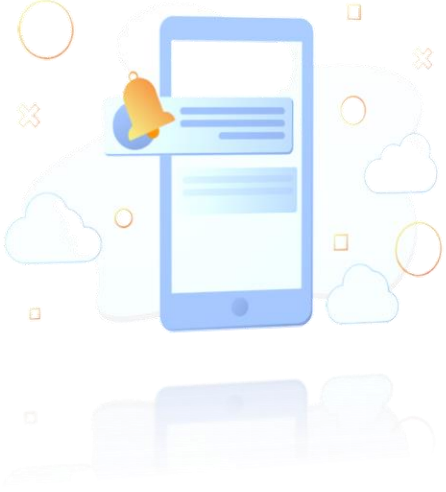
PUSH NOTIFICATION

What Is Push Notifications?

- Push notifications are clickable pop-up messages that appear on your users' browsers irrespective of the device they're using or the browser they're on.
- Push notifications are sent from an app can only be received by people who have the app installed on their phone.
- User must have push enabled to receive notifications.

What do Push Notifications look like?

- Like operating system version.
- Additional message content like images and action buttons.



Why do We use push notification?

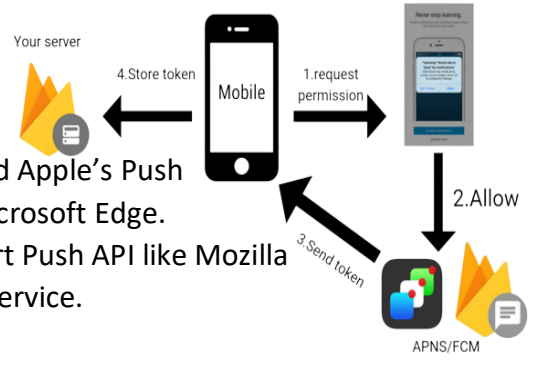
- Push notification allows you to engage user outside your app.
- They key is not over-message and only deliver relevant contact.

Different types of push notifications

- Web Push Notification.
- Desktop Push Notifications.
- Mobile App Push Notifications.
- Push notifications on wearables.

How do push notifications work?

- At the core of push notifications services like Google's Firebase Cloud Messaging (FCM) and Apple's Push Notification Service enable notifications on Safari and Windows Notification Service for Microsoft Edge.
- Although FCM falls under Google's umbrella, it also works with other browsers that support Push API like Mozilla Firefox and Opera. This enables the sending of push notifications to a web app via a push service.



What is FCM?

- Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably send messages at no cost.
- Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention.

What protocol does FCM use?

- FCM supports server protocols HTTP and XMPP which are identical to GCM protocols.

What is the use of APNs?

- Apple Push Notification service (APNs) is a cloud service that allows approved third-party apps installed on Apple
- devices to send push notifications from a remote server to users over a secure connection.

What is APNs certificate in iOS?

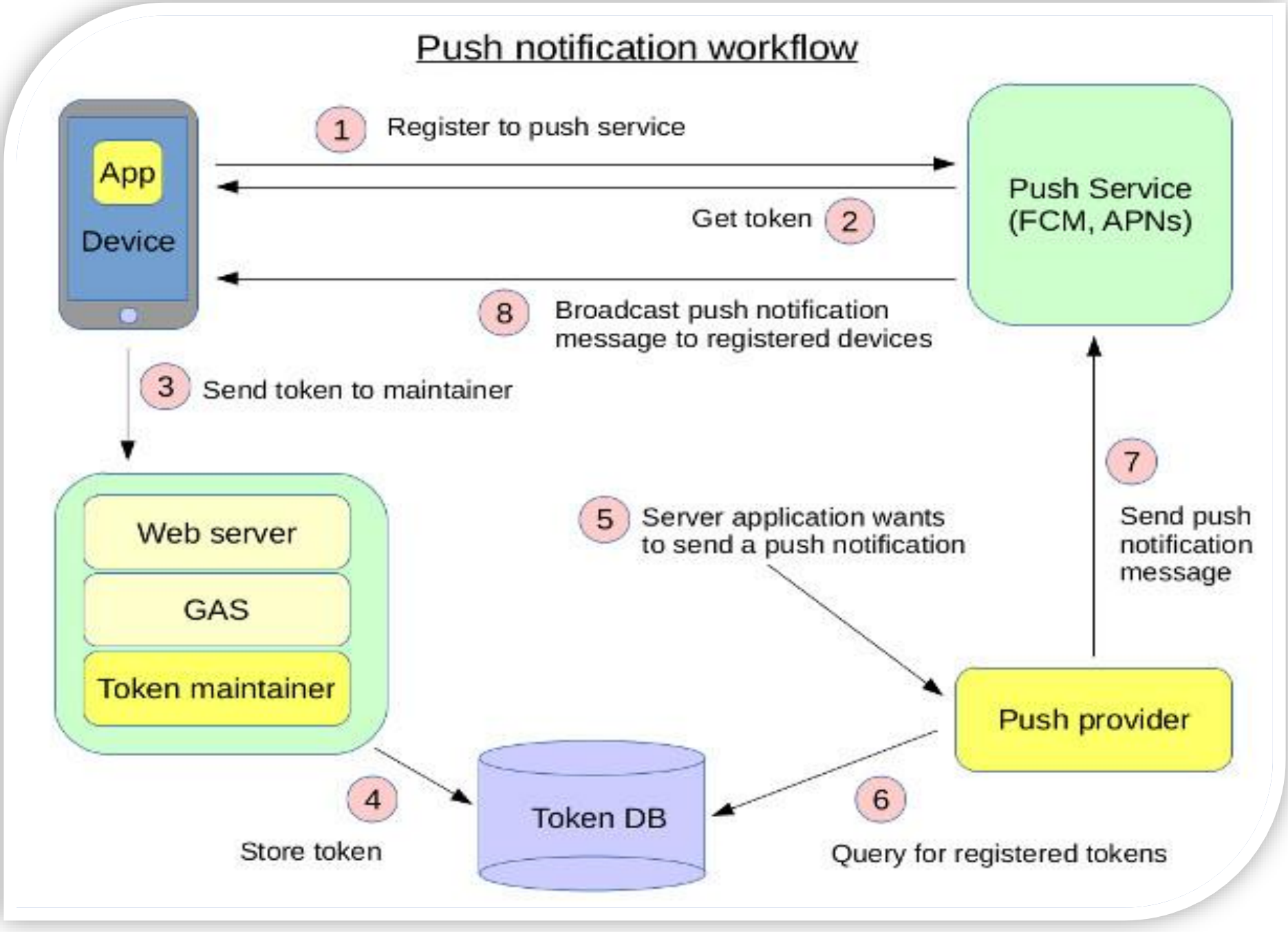
- Apple Push Notification service (APNs) is a platform notification service that enables third-party application developers to send notification data to applications installed on Apple devices.

What is APNs protocol?

- The provider API is based on the HTTP/2 network protocol. Each interaction starts with a POST request, from your provider, which contains a JSON payload and a device token.

Which browsers support push notifications?

Browser	Windows PC	MacOS	ChromeOS	Ubuntu/Linux	Android	iPhone (iOS)
Chrome	YES	YES	YES	YES	YES	NO
Firefox	YES	YES	YES	YES	YES	NO
Safari	N/A	YES	N/A	N/A	N/A	NO
Opera	YES	YES	YES	YES	YES	NO
Microsoft Edge	YES	N/A	N/A	N/A	YES	N/A
Yandex	YES	YES	YES	N/A	YES	N/A
UC Browser	NO	N/A	YES	N/A	YES	NO
Samsung Internet Browser	N/A	N/A	YES	N/A	YES	N/A
Internet Explorer	NO	N/A	N/A	N/A	N/A	N/A


Push notification workflow
App
Device
1
Register to push service
Get token
2
Push Service
(FCM, APNs)
8
Broadcast push notification
message to registered devices
3
Send token to maintainer
Web server
GAS
Token maintainer
5
Server application wants
to send a push notification
7
Send push
notification
message
Push provider
4
Store token
Token DB
6
Query for registered tokens

How to work Push Notification for Android Device using the FCM Server

- First, we need **URL, DEVICE TOKEN, FCM AUTHENTICATION KEY** for the implementation which we will get from client side.
  - **URL: -** When client side will register the application with the FCM server that time we will get the URL.
  - **DEVICE TOKEN: -** When Client-side registers that application to the device it will generate one device token.
  - **FCM AUTHENTICATION KEY: -** When application will be configured with FCM server that time we will get the authentication key.
- Now these three things we must configure with our backend implementation.

- So, first we created a class called **SpringBootPushNotificationsApplication** then declared **URL, DEVICE TOKEN, FCM AUTHENTICATION** as string. Here we used public final (we cannot change the value of a variable if once declared) **static** (value is the same for every instance of the class).

```
public class SpringBootPushNotificationsApplication {
public final static String AUTH_KEY_FCM = “YOUR AUTH_KEY_FCM”
public final static String API_URL_FCM = “YOUR API_URL_FCM”
public final static String DEVICE_TOKEN = “YOUR DEVICE_TOKEN”
```

Common mistakes made while using push notifications! **Push notifications are an exceptionally handy tool to engage your audience in real-time, and in a very personalized manner. A robust push notification campaign helps increase repeat visits and direct traffic to best-performing content. Sounds exciting, right? It surely is!**

❖ Then we created a method called **sendPushNotification ()** and pass the result as string and we throw **IOException** and **JsonSyntaxException.**

- **IO Exception: -** IO Exception is an exception which programmers use in the code to throw a failure in Input & Output operations. It is a checked exception.
- **JsonSyntaxException: -** This exception is raised when Gson attempts to read (or write) a malformed JSON element.

```
public static void sendPushNotification() throws IOException,
JsonSyntaxException {
String result = "";
```

❖ Now we started configuring **URL** part and established a new **HttpsURLConnection** for **FCM URL**. We have used **url.openConnection** here because whenever we hit the URL, **url.openConnection** will manage the protocol of the **URL**.

- **PROTOCOL: -** a set of rules or procedures for transmitting data between electronic devices, such as computers.

```
URL url = new URL(API_URL_FCM);
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
```

- Now we have used **setRequestMethod** for sending http post request using **HttpURLConnection** and **setRequestProperty** to set value of the specified request header field **(Like content – type, Authorization)**. The value will only be used by the current **URLConnection** instance.

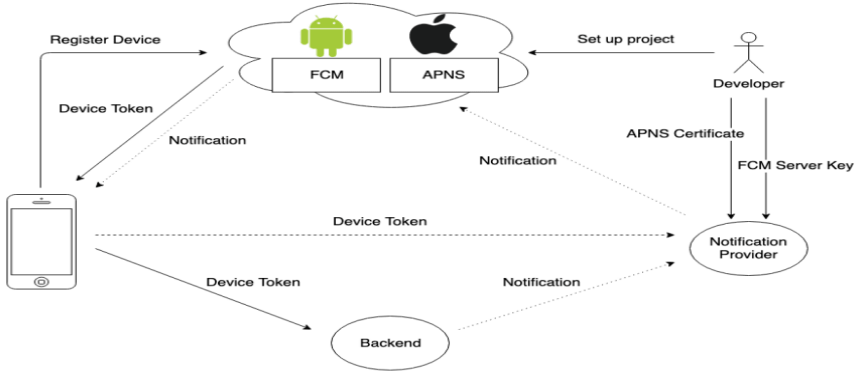
- Now we use **setCaches** for storing the cache data from memory, **setDoOutput** is for used with **POST** to allow sending a body via the connection and **setDoInput** is for used to fetch the response and is true by default.

```
conn.setRequestMethod("POST");
conn.setRequestProperty("Authorization", "key=AUTH_KEY_FCM ");
conn.setRequestProperty("Content-Type", "application/json");
conn.setUseCaches(false);
conn.setDoInput(true);
conn.setDoOutput(true);
```

- Now here I created String body, but we cannot pass the data as a String to the JSON object, so we converted String to JSON format and using **JsonObject json_body = (JsonObject) JsonParser.parseString(body)** command we created JSON String to Parse JSON.
  - **Parse JSON: -** Provides forward, read-only access to JSON data in a streaming way. This is the most efficient way for reading JSON data.

- Now we created a try catch block. Inside try block we used outputStream for passing or write some data to the destination or connection.
- Using this piece of code, we created an object called wr and we passed getOutputStream because it returns an output stream for the given socket.
  - **SOCKET: -** Socket is nothing but two-way communication. It may be over network, program, or machine.
    - Now we used wr.write for writing string. as we have written string before, so we just passed the parameter like**(json_body).**
    - Then we used flush() method for if element is there inside the stream, it will clear the stream.
    - After completing this 10th step, we are ready to write the data to the connection.

```
try {
OutputStreamWriter wr = new
OutputStreamWriter(conn.getOutputStream());
wr. write(json_body.toString());
wr.flush():
```



- Now we have created a classed called commonconstants and inside that create two field for SUCCESS and FALIURE.

```
package com.example.demo;
public class CommonConstants {
public static final String SUCCESS = "200";
public static final String FAILURE = "400";
}
```

- Then we passed the commonconstants class as result.

```
result = CommonConstants.SUCCESS;
```

- And we printed the result using **System.out.println()**. Here our try block closed.

```
System.out.println("Notification is sent successfully. STATUS CODE: " + result);
```

- We have written Catch block for define the error occurs in the try block.

```
catch (Exception e) {

e.printStackTrace();

System.out.println("Notification is sent failed. STATUS CODE: " + result);

result = CommonConstants.FAILURE;
```

- **After completing try** - Catch block we printed **getResponseCode** just to check the status code from HTTP response message.

```
System.out.println(conn.getResponseCode());
```

- Then we have created a main method for Run and Debugging.

```
public static void main (String [] args) {
SpringApplication.run (SpringBootPushNotificationsApplication.class, args);
```

- Then inside the main method we have created a **try catch block** and called **sendpushNotification()** method.

```
try {
sendPushNotification();
} catch (IOException e) {
e.printStackTrace();
}
```

----------END----------0----------END----------