Push Notifications Research

Push notification use two different APIs: **push** is invoked when a server supplies information to a service worker; a **notification** is the action of a service worker or web page script showing information to a user.

Push API is based on **service workers** because of ability working in background even when browser is closed, which service workers provides.

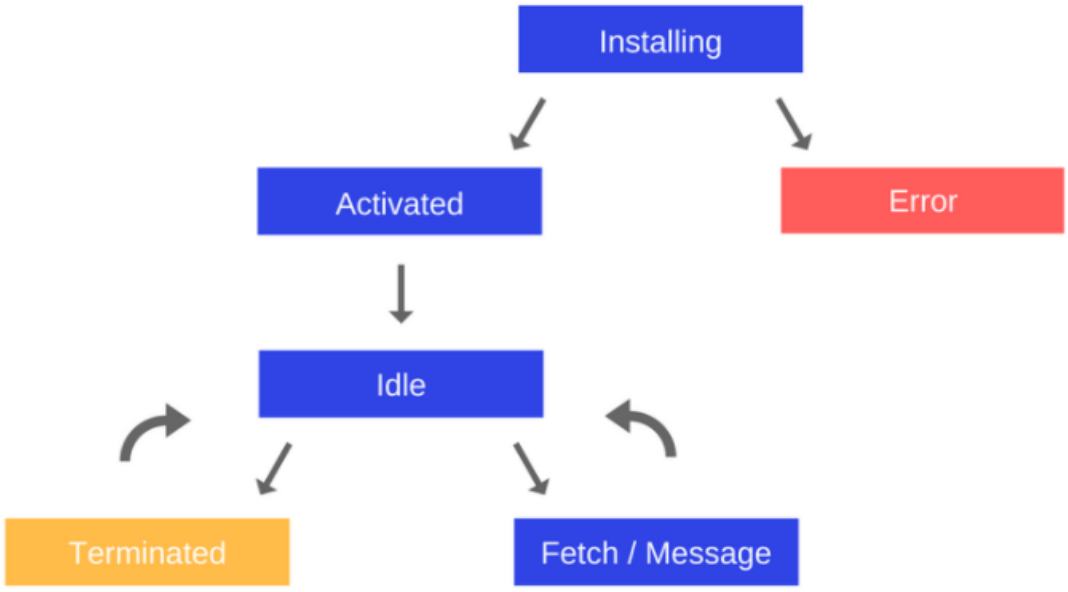
	Notification API	Push API	Service Workers
Chrome Desktop	+	+	+
Chrome Android	-	+	+
Firefox Desktop	?	+	+
Firefox Android	?	+	+
IE10	-	-	-
IE11	-	-	-
Edge	+	+	+
Safari Desktop	?	-	+
Safari IOS	-	-	+

+ **- full support of API**
? **- partial support of API**
- **- doesn't support API**

From data in the table, all required API will work only in Chrome and Edge (chromium-based from 15 Jan 2020). In Firefox - with some minor restrictions.

Service Workers

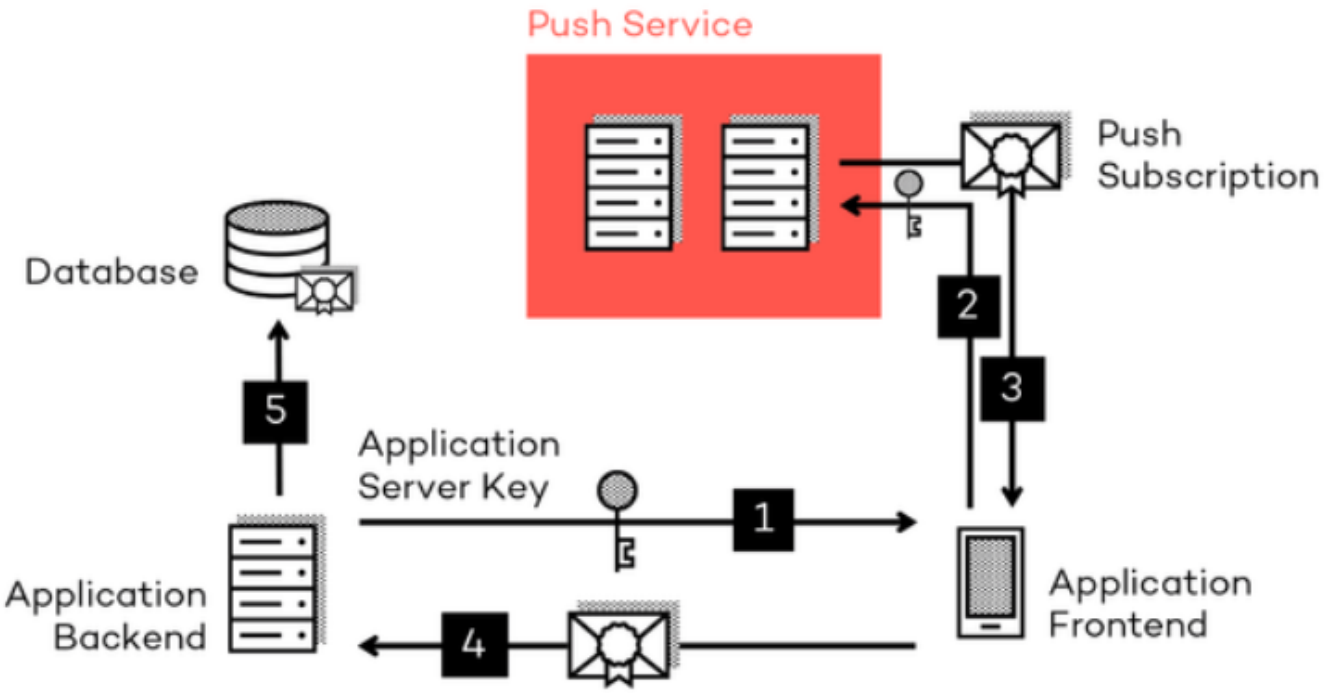
Service worker lifecycle:



1. The `install` event is the first event a service worker gets, and it only happens once.
2. A promise passed to `installEvent.waitUntil()` signals the duration and success or failure of your install.

3. A service worker won't receive events like `fetch` and `push` until it successfully finishes installing and becomes "active".
4. By default, a page's fetches won't go through a service worker unless the page request itself went through a service worker. So you'll need to refresh the page to see the effects of the service worker.
5. `clients.claim()` can override this default, and take control of non-controlled pages.

Push API



1. Application Server Key - *Uint8Array* with public VAPID key. It makes sure that only the application server will be able to send messages to the device. Typically provided by the application server.
2. *PushManager* on the client side using `subcribe()` to send Application Server Key to the browser push service. Chrome and Firefox had there own services for such feature, its impossible to use custom service. No any API credentials required, the browser vendors provide the push service infrastructure completely for free.
3. If the browser was able to subscribe for push messages, it will return the Push Subscription. This subscription consists of a push service endpoint which the application server can use to talk to the right push service and a public key.
4. Push Subscription is being sent to the application server
5. Application server stores endpoint url for sending push notifications

Push Manager Subscribe Parameter Interface

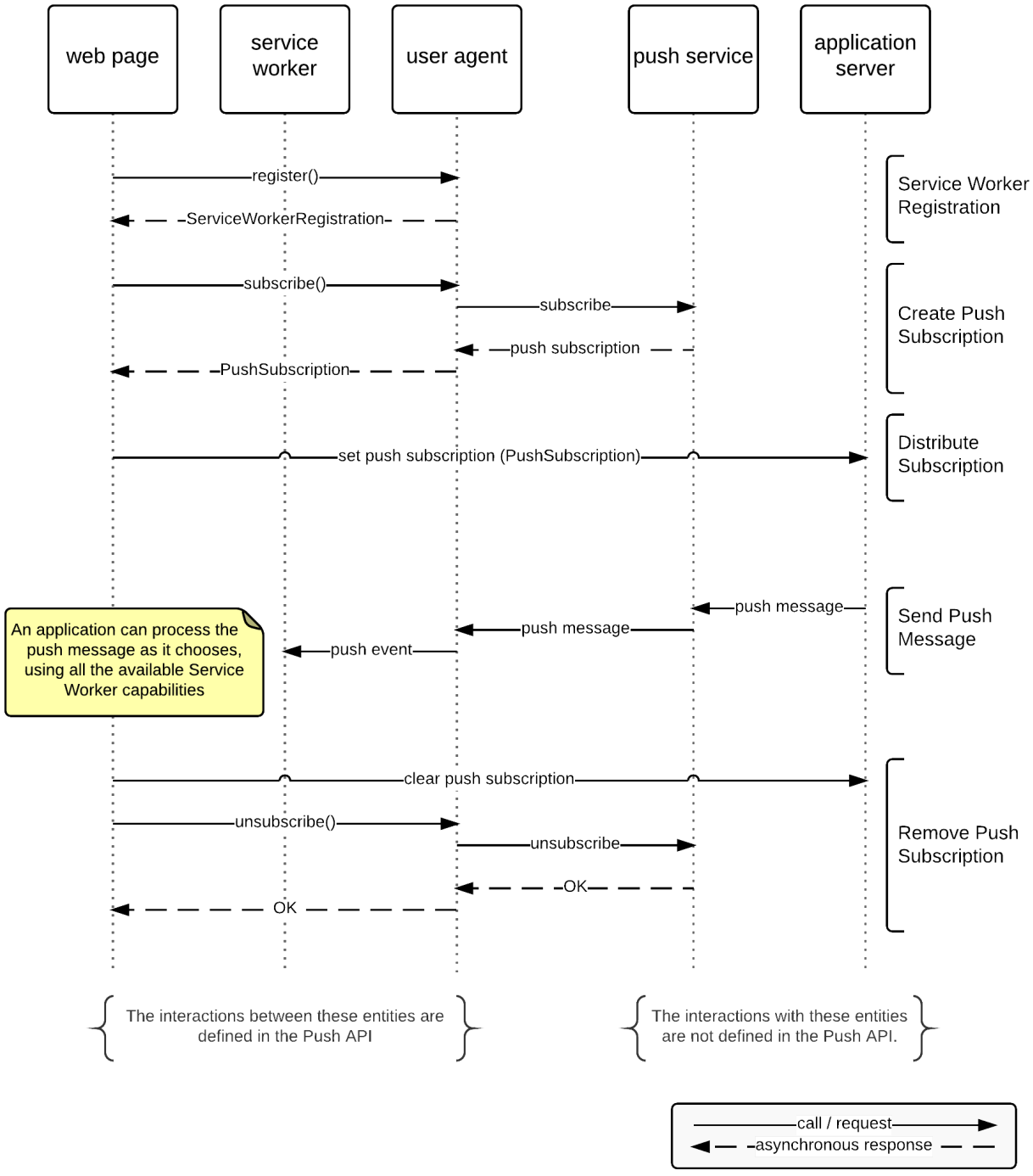
```
 {
        applicationServerKey: BufferSource | string | null;
    userVisibleOnly: boolean;
}
```

`userVisibleOnly` should always be *true. False* value for silent push messaging (user won't get notification), which for now doesn't work in Chrome and maybe will never be allowed

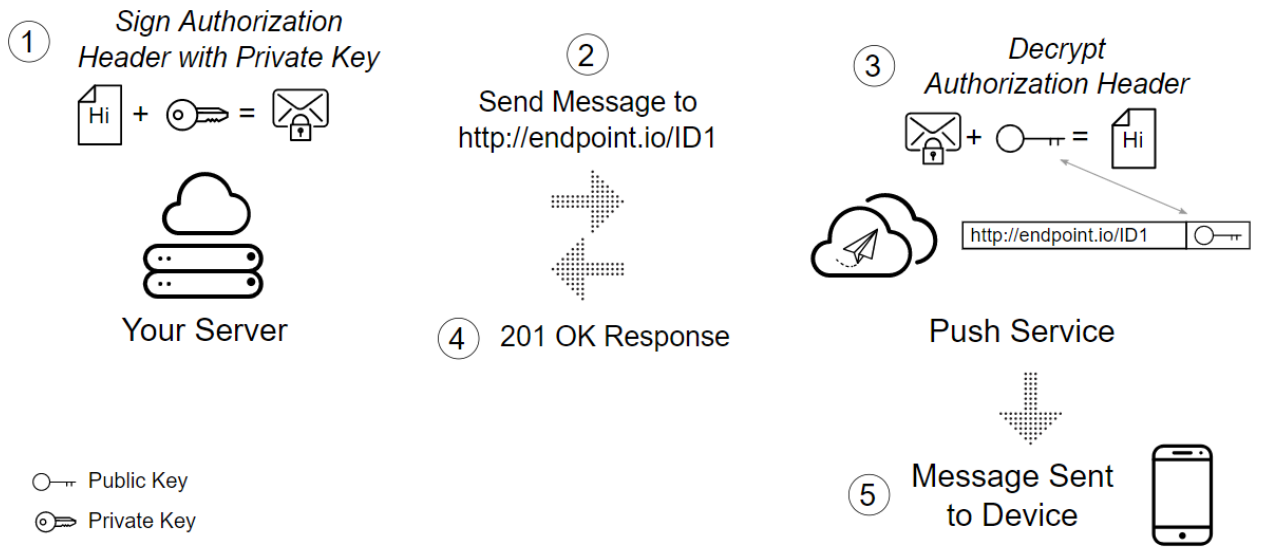
Push Subscription Interface

```
{
  "endpoint": <String Url>,
  "expirationTime": number | null,
  "options": {
    "applicationServerKey": string,
    "userVisibleOnly": boolean
  }
}
```

Push Notification lifecycle:



Sending push message by application server:



1. The application server signs some *JSON* information with it's **private application key**.
2. This signed information is sent to the push service as a header in a *POST* request.
3. The push service uses the stored public key it received from `pushManager.subscribe()` to check the received information is signed by the private key relating to the public key. The public key is the `applicationServerKey` passed into the subscribe call.
4. If the signed information is valid the push service sends the push message to the user.

More about Web Push Protocol

Notification API

Notification interface:

Notification Interface

`{`
`  title: string,`
`  options?: NotificationOptions`
`}`

NotificationOptions Interface

```
{
    actions?: NotificationAction[];
    badge?: string;
    body?: string;
    data?: any;
    dir?: "auto" | "ltr" | "rtl";
    icon?: string;
    image?: string;
    lang?: string;
    renotify?: boolean;
    requireInteraction?: boolean;
    silent?: boolean;
    tag?: string;
    timestamp?: number;
    vibrate?: number | number[];
}
```

`tag` - works like ID that groups notifications together, so that any old notifications that are currently displayed will be closed if they have the same tag as a new notification. New notification will appear without a sound or vibration.

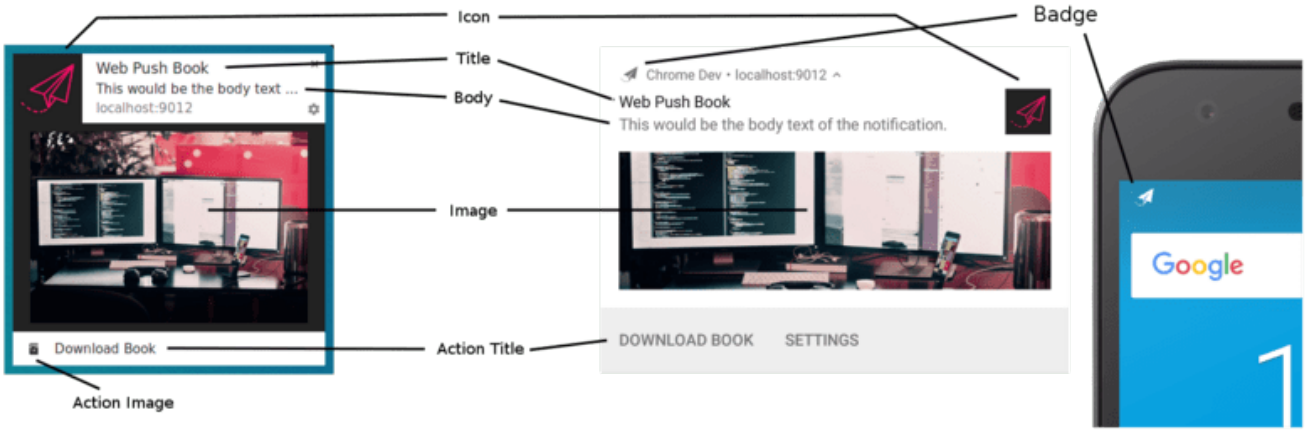
`renotify` - determine if new notification which was grouped by `tag` will work with sound or vibration.

NotificationAction Interface

```
{
    action: string;
    icon?: string;
    title: string;
}
```

`action`- works like ID for current action, allows client app to determine which action was triggered.

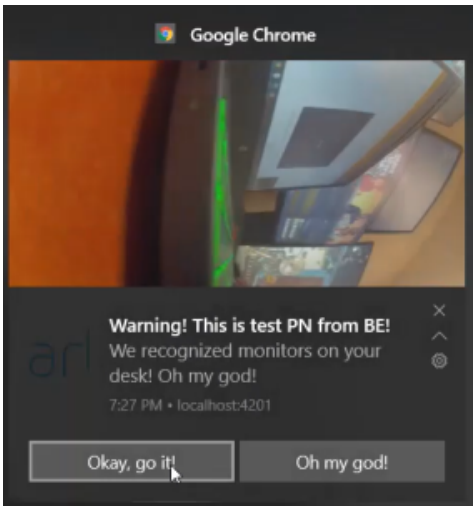
Example of the notification:



Demo of working concept

Video: PN Demo.mp4

Screenshot of notifications from Chrome on Windows



UX Note: Design of notification can't be changed. Only images, text, number of buttons and text in them can be changed

Test Push Notification with local BE

Allow Push Notification logic by enabling feature flags `pushApiMock` and `pushNotification`

add web-push library:

yarn command

```
yarn add web-push
```

Open command line in project app root and using command line go to server file location:

Server file location

```
cd src/push-server
```

Start PN server mock:

PN Server

```
node pn_server
```

In `push-notification.service.ts` in `_temp` variable set `isBackEndReady` and `enableLocalTest` as true

push-notification.service.ts

```
_temp = {
        isBackEndReady: true,
    enableLocalTest: true
}
```

isBackEndReady	true	true	false
enableLocalTest	true	false	doesn't metter
result	PN from local BE	PN from BE	PN from Dev Tools