BUSINESS ACADEMY AARHUS
SCHOOL OF APPLIED SCIENCES

PWA Push Notifications

Push Notifications in PWAs

- Link to Github repo will be available at the end of this talk.
- Small self-contained example built on the MERN stack
  - Express.js / node.js for the backend
  - React for the front-end

Push Notifications in PWAs

- The **Push API** makes it possible for PWAs to receive messages pushed from a server. <u>The PWA does not have to be in the foreground, or even currently loaded.</u>
- For an app to receive push messages, it must have an active **service worker**.
- And if you want to engage the user, you can use the **Notification API** to display a system notification on the user's device when a message is received.

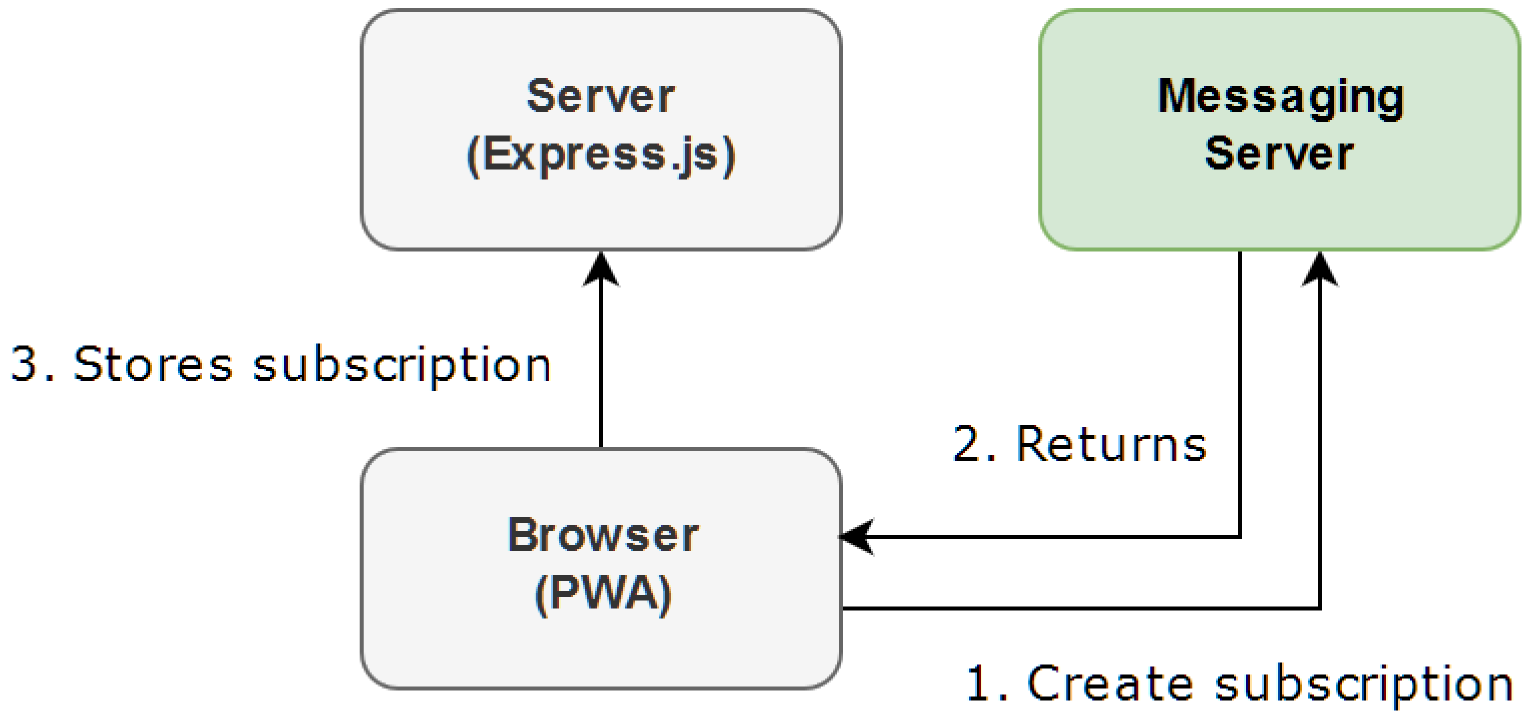
Push Notifications use cases

- Some examples:
  - Updates to your order (transactional)
  - “It’s your birthday.”
  - “It’s going to rain”.
  - Someone sent you are message
  - Someone beat your high score in a game
  - Remember your dentist appointment
  - Something one your wish list is on sale
  - …

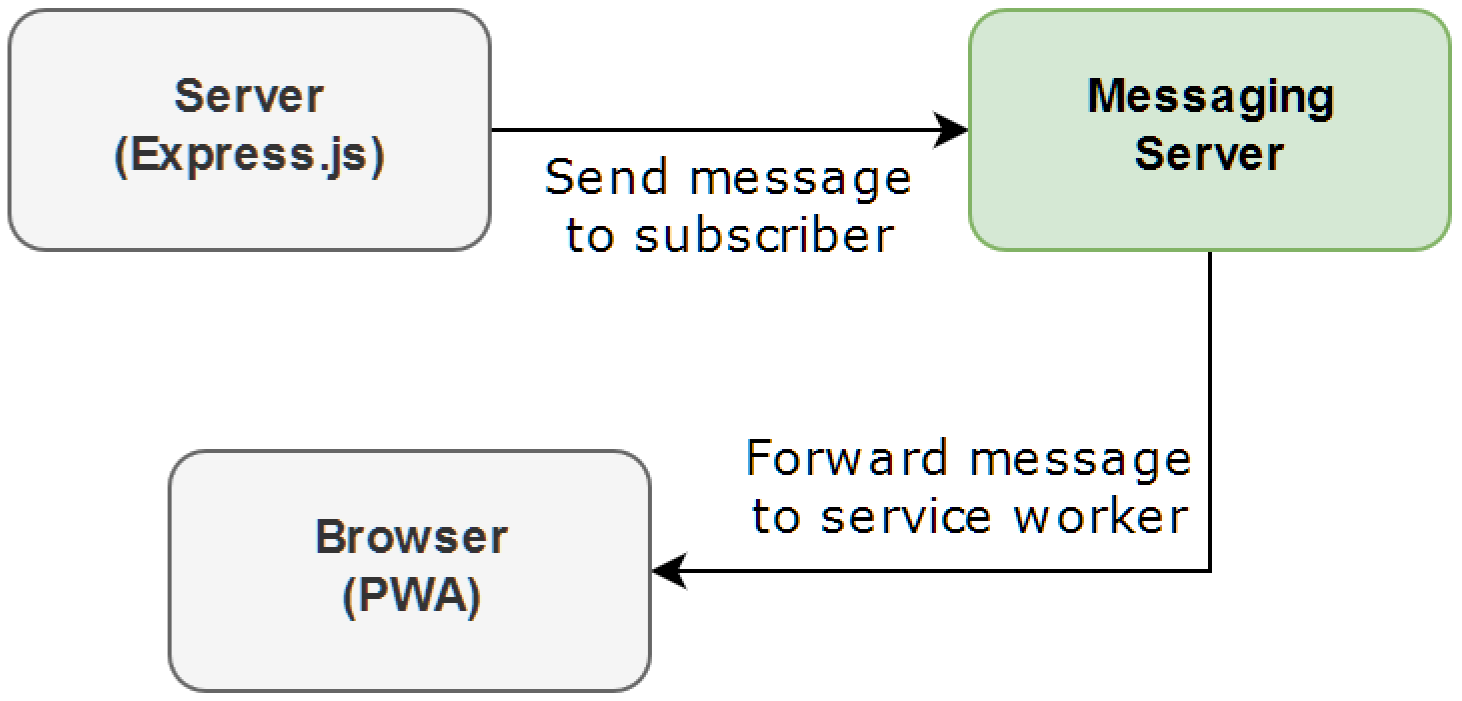
Technology and Architecture

- The **Notification API**
- The **Push API**
- A **Service Worker** (to receive push messages)
- A **server** (to send messages)
- A **messaging server** (provided by the browser vendor)

Creating a subscription



Sending a push message



The server

- One way of sending messages using a server is to implement the server yourself and use a **web push** library.
  - **https://www.npmjs.com/package/web-push**
- I’ll show you an example using **node.js** and **express.js**

Server to **messaging server** communication

- **The server** uses VAPID to identify itself against the **messaging server** using a public/private key pair.
- This is a one-time setup:
  1. Create a public/private key pair (RSA) for the server.
  2. The public key is given to the web app.
  3. The private key is hidden on the server.
- When the server sends a push message, it is signed with the private key.
- Only holders of the private key can send push messages.

Creating the public/private keys

- Create the keys by using the **web-push** library from the command line:
  - `npm install web-push -g`
  - `web-push generate-vapid-keys`
- Then keep the keys somewhere safe!

DEMO

Client-side implementation (1/3)

- When registering the Service Worker, use the **registration object** to subscribe to push

```
navigator.serviceWorker.ready.then(
    function (serviceWorkerRegistration) {
        // Register to push events here
```

Client-side implementation (2/3)

- Add a listener on **push** events in the Service Worker implementation.

```
self.addEventListener('push', function (event) {
    const data = event.data.json();
    // TODO: Do stuff with push data here
});
```

Client-side implementation (3/3)

- Use `self.registration.showNotification(...)`
  to show notification to the user from the **push** event listener in the Service Worker.

```
event.waitUntil(
    self.registration.showNotification(data.title, {
        body: data.msg,
        vibrate: [500, 100, 500]
    })
);
```

Server-side implementation (1/2)

Send subscriptions from the client to the server using a HTTP POST request:

```
app.post('/api/subscribe', (req, res) => {
    const subscription = req.body;
    // TODO: Store subscription in database
```

Server-side implementation (2/2)

Send push messages from the Server using the **web-push** library:

```
subscriptions.forEach(sub => {
    const payload = JSON.stringify({
        msg: text,
        title: title
    });
    webpush.sendNotification(sub, payload).catch(
        error => {
            console.error(error.stack);
        });
});
```

Example on Github.

- Check it out here:
- https://github.com/kdorland/web-push
- Questions?