Notification Systems Handover Document

Purui Xu
Siyu Tao

Table of Contents

1. Introduction

This document provides a detailed overview of the notification project, including system architecture, source code structure, key functionalities, and more. It's intended to enable a seamless transition of this project to the next responsible party.

The notification system is at the heart of this application. It is designed to send push notifications to users based on their preferences and schedules. Users can subscribe and unsubscribe from notifications, allowing them to have control over the content they receive. The database structure is simple but effective, containing the essential information for handling notifications. It includes user IDs, notification messages, time of schedule, and a Boolean to control the subscription status.

2. Project Overview

This project enables sending notifications to subscribed users. Users can subscribe to receive notifications, manage their subscriptions, and receive daily notifications at scheduled times.

2.1 Objectives

- **User Engagement:** To maintain a consistent and personalized engagement with users by sending relevant notifications based on preferences and schedules.

- **Cross-Platform Compatibility:** To deliver a seamless experience across different browsers and devices, ensuring the widest possible reach.

2.2 Key Features

- **Subscription Management:** Users can subscribe or unsubscribe to various topics and set preferences for notifications.
- **Scheduled Notifications:** Timed and automated notifications allow users to receive reminders or alerts at specific times.
- **Push Notifications:** Utilizes web push technology to send real-time notifications to subscribed users.
- **Adaptive UI:** An intuitive and responsive design that automatically adapts to different screen sizes and orientations.

3. System Architecture

The system architecture of our PWA is designed to be scalable, maintainable, and robust, consisting of several interconnected components:

3.1 Front-end

HTML, CSS, and JavaScript are used to build the user interface and register the service worker for push notifications.

- **HTML/CSS/JavaScript:** The building blocks of the web, used to create a responsive and interactive user interface.
- **Materialize CSS Framework:** Ensures consistency and visual appeal across the application.
- **Service Workers:** Handles background tasks like caching assets, managing offline behavior, and handling push notifications.

3.2 Back-end Server

Built with Express, handles the subscription and sending of push notifications, as well as interactions with an SQLite database.

- **Express.js/server.js:** A fast, unopinionated, and minimal web framework for Node.js that handles server-side logic and routing.
- **Web Push:** Enables subscription and sending of push notifications.
- **SQLite3 Database:** Stores data related to notifications, user preferences, and subscriptions.

3.3 Integration

- **Node Schedule:** Handles scheduling tasks like sending daily notifications on scheduled times.
- **Moment.js:** Assists in managing date and time operations, vital for scheduling.

4. Source Code Structure

4.1 Source Code Structure from Purui Xu's GitHub folder

In our PWA application, we have 3 main directories: pages, CSS and JS folder.

Folder Name	Files Contained	Description
page	about.html contact.html fallback. html	HTML files designed as branch websites of the home page index.html
css	materialize.min.css style.css	materialize.min.css: a popular front-end framework that provides a set of pre-designed CSS components for building responsive and visually appealing web applications.
		style.css: defines the main layout of the PWA website.
js	materialize.min.js app.js ui.js	materialize.min.js: uses the pre-designed JS components for the website building.
		app.js: is responsible for registering a service worker in a web application
		ui.js: contains JavaScript code that enhances the user interface (UI) of a web application by initializing and configuring certain components using the Materialize CSS framework.

Despite these files. In the root directory, there are still some important single files:

File Name	Description
index.html	defines the website of the home page of the PWA application.
manifest.json	is a configuration file used for PWA.
package-lock.json	is a JSON file that is automatically generated when using the Node.js package manager (npm) to install or update packages for the project. It serves as a record of the exact versions of all the dependencies that were installed for your project, along with their sub-dependencies (dependencies of dependencies) and their specific versions.
sw.js	is a JavaScript file that defines a service worker for a web application. A service worker is a script that runs in the background of a web page and can intercept and handle network requests, cache resources, enable push notifications, and perform various other tasks.

web-push.js	is a JavaScript script that sets up a server-side application using the Express.js framework to handle scheduling and sending of push notifications to subscribers. It also includes functionality to manage subscription status for users. This script is designed to work alongside a web application and a service worker that handles push notifications on the client side.

4.2 Source Code Structure from Siyu Tao's GitHub

Our system includes frontend user interaction, backend management of notifications, and database setup for user subscriptions and messages. It leverages service workers, VAPID key generation, and push management to provide a comprehensive solution.

- Frontend

File Name	Description
index.html	contains the structure of the page, including notification buttons and welcoming information.
index.js	contains the main JavaScript code for the client-side, handling push notifications, subscriptions, and service worker registration
style.css	contains the styling for the HTML elements on the front end.

- Backend

File Name	Description
server.js	includes all routes, push notification handling, scheduling, and interactions with the notifications database. (core Express backend code)
userDatabase.js	handles user database connections, specifically to SQLite3 \`users.db\`, and the creation of the user table.
database.js	manages the connection and initialization for the notifications database (\`notifications.db\`). Includes the creation of messages table and insertion of default messages.
generate-valid-keys.js	contains code to generate VAPID keys, essential for push notifications.

- Service Worker

File Name	Description
worker.js	the service worker file, responsible for listening to and handling push events, and displaying notifications to the user.

5. Key Functionalities

5.1 Key Functionalities from Purui Xu's GitHub folder

The functionalities are mainly defined in sw.js and web-push.js files.

- sw.js:

Functionality	Detailed Description
Caching Assets	• The install event is triggered when the service worker is first registered or updated. • During installation, it caches specified assets (HTML files, JavaScript files, CSS files, images, fonts, etc.) using the caches API. The assets are stored in the staticCacheName. • This ensures that these assets are available offline and can be quickly retrieved from the cache.
Managing Cache Size	• This function helps manage the size of the cache by deleting older entries when the cache size exceeds a specified limit.
Activating Service Worker	• The activate event is triggered when the service worker is activated, usually after installation. • It cleans up old caches by deleting caches with different names than the current staticCacheName and dynamicCacheName.
Fetch Activities	• The fetch event is triggered whenever a network request is made by the web application. • The service worker attempts to respond to the request from the cache (caches.match) before fetching the resource from the network. • If the requested resource is not in the cache, it is fetched from the network, stored in the dynamicCacheName cache, and then the cache size is checked and managed using the limitCacheSize function.
Push Notifications	• The service worker handles push notifications by listening for the push event. • When a push notification is received, the service worker displays a notification using the data provided in the push message. • When the user clicks on the notification, the notificationclick event is triggered, and the service worker opens a specific URL (in this case, https://app.askanjlee.com).
Subscription Management	• The service worker includes logic to request subscription permission for push notifications using the askForSubscriptionPermission function. • It handles the user's response to the subscription permission request and opens a window when the user clicks the "Allow" action. • It also handles the event when the subscription is canceled, allowing the user to unsubscribe from push notifications.

- web-push.js:

Functionality	**Detailed Description**
Import and Setup	• The script imports necessary packages, including web-push, express, body-parser, and a custom database module. • It sets up an Express application and configures middleware.
Vapid Keys	• VAPID (Voluntary Application Server Identification) keys are used to authenticate the server when sending push notifications. • The script sets up the public and private VAPID keys using the webpush.setVapidDetails() function.
API Endpoint to Schedule Notifications	• The script defines an API endpoint (/schedule) that receives requests to schedule push notifications. • The request includes the user's ID, payload (notification content), and the desired schedule time. • The script checks if the user is subscribed and schedules the notification accordingly.
Notifications Scheduling and Sending	• The script includes functions to schedule push notifications for specific times of the day. • A background task (setInterval) runs periodically to check the scheduled notifications and sends them to subscribers if the scheduled time has been reached.
API Endpoints to Manage Subscriptions	• API endpoints (/unsubscribe, /user/:id/subscription, /user/:id/subscription) are defined to handle subscription management. • Users can subscribe or unsubscribe from push notifications using these endpoints.

6. Database Design

SQLite3 Database with the following schema:

Attribute	**Type**	**Description**
id	INTEGER	PRIMARY KEY
message	TEXT	stores the message content. It should be NOT NULL.
time	TIME	stores the scheduled time of the message. It should be NOT NULL.

7. Environment Setup

Steps	Detailed Description
Node.js	• Visit the official Node.js website: https://nodejs.org/ • Run the downloaded installer and follow the installation prompts. • After the installation is complete, open a terminal window and type node -v to verify that Node.js is installed. You should also be able to run the Node.js REPL by typing node.
Database	• In the integrated terminal, use the following command to install the sqlite3 package globally by using: npm install -g sqlite3
Server	• Run server with `node server.js`
Get VAPID keys	• Install the `web-push` package in the terminal by using: npm install -g web-push.js • Generate the VAPID keys by using: web-push generate-vapid-keys

8. Known Issues and Solutions

- VAPID Key Mismatch: If VAPID keys are inconsistent, regenerate them.
- Endpoint Mismatch: Ensure the latest endpoint is used for notifications.
- Connection Errors: Verify server and client are on the correct port.

9. Contacts

Purui Xu: puruix@andrew.cmu.edu
Siyu Tao: siyutao@andrew.cmu.edu