Real-time synchronizace pro offline-first klienty

Originální zadání

Máme backend v cloudu (AWS) a několik frontendových aplikací (web, mobile).

Data z těchto aplikací je potřeba synchronizovat v reálném čase, tzn. když něco změníme na webu, musí se to do nějakého definovaného času (2s) zobrazit na mobilní aplikaci (a naopak). Zároveň je potřeba, aby mobilní aplikace uměly pracovat neomezeně dlouhou dobu offline.

Jaké bys použil přístupy? Jaké technologie? Jakou použiješ databázi a jak bude vypadat formát výměny dat? Jak bys zajistil real-time synchronizaci dat v offline-first aplikaci? Co se týče přístupu a architektury, kterou popisuješ, vidíš nějaké úskalí? Nebo naopak, je něco, co bys vyzdvihl jako unikátní nebo skvělé řešení?

Zestručněné pracovní zadání/poznámky

- Existující backend infrastruktura.
- Množství klientských aplikací napříč různými platformami.
- Real-time synchronizace mezi serverem a klienty, real-time timeframe 2s.
- Offline-first architektura klientských aplikací.
- Specifikovat technologie, DB, komunikační protokol, atd. viz originální zadání.

Zaměření dokumentu

Zadání specifikuje především požadavek na real-time synchronizaci a offline-first přístup. Obě vlastnosti lze analyzovat odděleně a závislost mezi nimi vzniká při propojení obou v klientské aplikaci. Úvodní verze řešení se zaměřuje pouze na tyto dvě klíčové vlastnosti.

Komentář před vypracováním řešení

Zbyněk Růžička: Oba přístupy real-time synchronizace a offline-first klientské přístupy jsem dosud na předešlých pracovních projektech neřešil, ale po úvodním seznámení s touto tématickou doménou považuji obě zmíněné vlastnosti aplikací a velmi přínosné a rád se budu touto doménou dále zabývat.

Real-time synchronizace

Řešíme-li komunikaci mezi oddělenými aplikacemi, standardem je používat mezi serverem a klientem definované a popsané API. Při real-time synchronizaci mezi klientskými aplikacemi se lze zaměřit na tzv. Real-time API.

Je vhodné mít možnost obousměrné komunikace mezi serverem a klientskými aplikacemi, aby klient nemusel vykonávat množství requestů vůči serveru ve snaze odhalit novou aktualizaci dat. Pro tyto účely slouží WebSockets.

WebSockets (RFC 6455) umožňují po navázání spojení obousměrnou výměnu zpráv mezi serverem a klientem. Po navázání úvodního spojení tento protokol již dále nepoužívá standardní HTTP hlavičky a návratové kódy a je třeba si definovat a implementovat vlastní model významu zpráv a jejich výměny nebo použít existující řešení.

Pro možnost notifikovat klienty ohledně nových dat bude třeba definovat kategorie, pro které se mohou klienti registrovat. Každá kategorie tedy bude mít své listeners.

Definované kategorie budou záviset na doméně a zaměření aplikace - u messengerů může být kategorií každá komunikační skupina, u obchodní aplikace mohou být kategorie jako zboží, objednávky, registrovaní uživatelé.

Push notifikace

Tyto notifikace umožňují ze strany serveru informovat klienta o změně stavu nebo dat ve chvíli, kdy k takové změně dojde.

U push notifikací je důležité vyjasnit/upřesnit, co vše chceme při push notifikaci posílat. Oba dále uvedené přístupy mají níže uvedené výhody (+) a nevýhody (-).

A. Kompletní aktualizace dat.
   (+) Šetří množství requestů vůči serveru.
   (-) Klient stahuje kompletní objem aktualizací, přestože data aktuálně nemusí potřebovat.
B. Pouze informace, že uvedená kategorie je aktualizována a umožníme/očekáváme, že klient stáhne kompletní aktualizaci pouze na jeho explicitní request.
   (+) Klient stahuje jen minimum nezbytných nebo požadovaných dat.
   (-) Dle výše počtu klientů lze očekávat i vyšší množství requestů na server, kdy klient potvrzuje zájem o zaslání kompletního objemu aktualizace.

Volba technologie pro synchronizaci

Na úvodní prototypování preferuji použít existující řešení pro synchronizaci a nejdříve odhalit jeho silné a slabé stránky, než se pouštět do čistě vlastní implementace.

Pusher (https://pusher.com/) nabízí Real-time API postavené na WebSockets a současně umožňuje definovat kategorie/kanály pro zasílání a odběr aktualizací. Lze tedy použít pro real-time synchronizaci.

//TODO vyjasnit klady a zápory pro Pusher a porovnat je s dalšími existujícími IaaS řešeními (PubNub, Ably, SocketIO)...

//TODO zjistit, zda Pusher nabízí real-time timeframe a jak se konfiguruje.

Offline-first klientské aplikace

Při implementaci offline-first aplikací je současným standardem používání technologie ServiceWorker a tento přístup lze použít i pro naše řešení. ServiceWorker umožňuje definovat skript, který je spuštěn prohlížečem jako prostředník mezi webovou aplikací a síťovými požadavky.

V offline-first přístupu je dále klíčové použití lokální cache, která umožňuje poskytovat dříve registrovaná a stažená data i v případě offline režimu nebo ve stavu nedostatečně pomalého připojení. Servírování dříve uložených dat je možné definovat v rámci "fetch" události, která je vykonána při požadavku na načtení stránky.

Dalším krokem je zde rozhodnout se pro použití konkrétní klientské lightweight databáze, která registrovaná data udrží i po vypnutí/restartu prohlížeče.

Po úvodním hledání se nabízí možnost použít PouchDB řešení. PouchDB nabízí propracovanější dokumentaci, širší uživatelskou základnu a je to free open source, oproti konkurenční SlashDB. Zejména finální volbě vhodné DB bych ale doporučil věnovat více času a rozhodnout se až po úvodním prototypování a ověření, že DB splňuje očekávání a je snadno použitelná jako cache pro offline resources. Do prototypování bych rád zahrnul i defaultní ServiceWorker cache.

Pro možnost snáze později změnit výběr DB doporučuji definovat a používat v implementaci vlastní cache interface, který bude zahrnovat všechny základní operace s registrovanými daty a který bude vytvářet abstrakci od konkrétního výběru DB.

Q&A

1. Jaké bys použil přístupy?
    a. Pro synchronizaci komunikace přes WebSockets a použití principu Real-time API a push notifikací.
    b. Pro offline-first se používají skripty registrované na eventech pro ServiceWorker a lokální cache.
2. Jaké technologie?
    a. Pro synchronizaci se nabízí existující řešení Pusher.
    b. Pro offline-first bude použit ServiceWorker a skrze prototypování je třeba potvrdit použití PouchDB jako cache.
3. Jakou použiješ databázi a jak bude vypadat formát výměny dat?
    a. Jako úvodní DB bude použita PouchDB.
    b. Výměna synchronizovaných dat bude probíhat skrze definované kategorie/kanály.
4. Jak bys zajistil real-time synchronizaci dat v offline-first aplikaci?
    a. Klientské aplikace budou registrovány jako listeners/subscribers na jednotlivé tématické kanály do real-time API a server jako publisher bude zasílat push notifikace pro jednotlivé aktualizované kanály.
5. Co se týče přístupu a architektury, kterou popisuješ, vidíš nějaké úskalí?
    a. V současnosti vnímám jako citlivé použití lokální cache a vyladění jejího správného použití pro fetch requesty.

    b. Bylo by vhodné napsat registraci offline dat tak, aby se všechny existující kategorie/kanály ukládaly do cache na základě iterace přes existující kanály, nikoliv tak, že ukládání každého kanálu by řešil samostatný manuálně definovaný a explicitně volaný request.
6. Nebo naopak, je něco, co bys vyzdvihl jako unikátní nebo skvělé řešení?
    a. Existující synchronizační nástroje považuji za vyspělé a jejich nasazení by mělo být méně komplikované.
    b. Určitou unikátnost spatřuji v možnosti, že klient bude stahovat jen minimum potřebných dat, jak je více popsáno v kapitole Push notifikace, ale tento přístup musí být nejdříve schválen i ostatními, zda je to vlastnost, kterou klienti ocení. Tato konkrétní feature se hodí více pro aplikace připojené přes mobilní data v terénu, než pro POS systémy, které budou většinou připojeny do sítě přes WiFi nebo kabel.