УДК 004.67

Моісейкін О.С.,
Артамонов Є.Б., к.т.н.,

МЕТОДИ ОРГАНІЗАЦІЇ ВИКОРИСТАННЯ ТЕХНОЛОГІЙ РОЗРОБКИ МЕРЕЖЕВИХ МУЛЬТИПЛАТФОРМНИХ СИСТЕМ

Національний авіаційний університет

master@cifr.us
eart@ukr.net

Описані методи організації використання технологій, що застосовуються при розробці мультиплатформних мережевих систем. Особлива увага приділяється проблемі зменшення часу очікування завантаження динамічних веб-додатків. Описуються підходи проектування програмного коду, пов'язані з кешуванням компонентів інтерфейсу, організації взаємодій з клієнтськими базами даних і передачі накопиченої інформації від клієнта до сервера. Пропонований матеріал широко використовується при створенні інформаційних систем. На практиці застосовується в системах адаптивного навчання і системах аналізу маркетингових досліджень

Ключові слова: інтернет, прогресивні веб-додатки, клієнт-серверна архітектура, база даних, методологія, передача даних, кешування, динамічне формування вмісту

Вступ

Сучасне програмне забезпечення вийшло далеко за межі стаціонарних персональних комп'ютерів і велика частина його можливостей, реалізована, в так званих «веб-додатках», здатних завантажуватися через браузер без необхідності додаткової установки.

Оскільки велика частина програмного забезпечення реалізовує набір функцій для роботи з базою даних, то підходи пропоновані веб-додатками, вирішують проблеми сумісності програмного коду, проблеми підтримки програмного продукту, здешевлюють процес розробки готових продуктів і все це завдяки клієнт-серверному підходу використовуваному в мережі Інтернет.

З розвитком комп'ютерних пристроїв, запуск веб-додатків став можливий на смартфоні, планшетному комп'ютері або будь-якому іншому мобільному пристрої з браузером відповідним специфікаціям *W3C*.

Завантажується пропрограмний код здатний візуально адаптуватися під розміри екранів, що робить зручним використання практичним на будь-якому сучасному пристрої без необхідності перекомпілювання і повторної розсилки програмного коду.

Браузер виступає не просто як візуалізатор гіпертекстових документів, а й надає завантажуваному скриптовому коду *API* для роботи з локальними ресурсами. До них можна віднести локальні бази даних *webSQL*, модулі глобального позиціонування *GPS* і апаратні мережеві інтерфейси *Bluetooth*.

Специфіка роботи веб-додатків на потребує наявність сервера для надання функціональних можливостей, але в даному випадку весь необхідний програмний код може бути завантажений в пам'ять клієнтського пристрою і бути використаний без підключення до мережі Інтернет.

Постановка проблеми

У класичному алгоритмі взаємодії клієнта і сервера, тільки сервер є джерелом інформації, а клієнт - споживачем і частково здатний впливати на сервер маніпулювання надісланими клієнтськими скриптами. У такій взаємодії є чітко виражений архітектурний ланцюжок «клієнт відправляє запит», «сервер обробляє запит і обробляє дані», «клієнт отримує результат обробки» [1].

Штучно створена фрагментованість, дозволяє використовувати веб-додатки в двосторонньому порядку, не тільки одержуючи інформацію на мобільний пристрій, а й використовуючи саме веб-додаток, створювати інформацію і обмінюватися нею з початковим джерелом, породжуючи при цьому архітектурно неправильно організовані системи. Це тягне за собою підвищення навантаження на мережу і збільшення часу очікування завантаження вмісту.

Розвиток галузі веб-орієнтованого програмного забезпечення змушує створювати більш просунуті і прогресивні програмні продукти, в яких клієнт як споживач інформації, так само може бути її джерелом і змінювати стан, взаємодіючі з сервером, при цьому не змінюючи початкову роль. Їх використання передбачає переорганізацію комп'ютерних мереж, що можливо за рахунок емуляції поведінки запитів шляхом використання методології розробки «Прогресивних веб-додатків» (англ. «*Progressive Web Apps*», *PWA*).

Переорганізація обміну даних посилює необхідність вирішення проблеми узгодженості, множинного дублювання і перезапису, оскільки пропонована методологія тільки реалізує програмне забезпечення і ще не вирішує проблеми обміну даними. При цьому основною невирішеною проблемою залишається організація методології розробки і доказ користі від практичного застосування такого програмного забезпечення [2].

Аналіз досліджень

Вирішенням проблемних питань, пов'язаних з мережею Інтернет, займається організація *W3C - World Wide Web Consortium*. Яка розробляє єдині принципи і стандарти, звані «рекомендаціями», які впроваджуються виробниками програмних продуктів і апаратного обладнання. Відносно пропонованої методології, *W3C* допускає розвиток *PWA* за умови використання висунутого маніфесту і реєстрації коду додатків через *Service Workers*, тим самим закривши питання безпеки використання такого програмного коду.

Одним з основних прихильників висунутого підходу, є компанія *Google*, яка з середини 2015 року дотримується таких принципів, за якими додаток повинен бути адаптивним, незалежним від з'єднання, самооновлюваним, безпечним, легким у використанні [3].

Дослідження, що узагальнюють проблеми обміну даними в прогресивних веб-додатках, на даний момент не проводилися. Ймовірною, причиною цьому послужила відносна новизна даної напрямку розробки.

Етапи вирішення задачі

За основний показник швидкодії, приймається час очікування повного завантаження поточної сторінки веб-додаткf.

Для забезпечення якісно побудованої архітектурної бази, пропонується логічно розділити розробку веб-додатка на такі підзадачі:

1) Поділ графічного інтерфейсу;
2) Зберігання виконуваного коду і статичних ресурсів;
3) Організація обміну даними з сервером.

Обов'язковими умовами є ідентична функціональність, як при наявності підключення до мережі, так і без нього, а також відсутність повторюваних частин виконуваного коду.

Оскільки, самі по собі, проблеми рішення обміну даними і їх синхронізації не є новим, частина виникаючих проблем, може бути вирішена в програмних продуктах, що використовують *CouchDB* і *CouchDB Replication Protocol*. *CouchDB* є документно-орієнтованою системою управління базами даних і не вимагає опису схеми даних, тобто реалізована в рамках підходу *NoSQL*. Дозволяє створити точну синхронізацію на базі *P2P*-з'єднання з будь-якою кількістю користувачів, не зберігає дані і зв'язки в таблицях[4].

Для управління розподіленим доступом використовується механізм *MVC*-

C, що дозволяє уникнути необхідності блокування файлу бази даних під час багатопоточного запису, але логіка вирішення конфліктів повинна відноситися до області відповідальності логіки самого веб-додатка.

Виклад основного матеріалу

Оскільки мова йде про динамічні веб-додатки, реалізація яких може істотно відрізнятися від області застосування і виконуваних завдань, як найбільш зрозумілий приклад, вибрано веб-додаток для метрично-адаптивного читання.

Основне завдання такого додатка полягає в динамічній зміні подальшого тексту на основі аналізу метрик поведінки користувача під час читання.

На першому етапі визначається, які елементи розмітки сторінки підлягають кешуванню в пам'яті клієнта. Як зазначено на рисунку 1, в даному прикладі статичними елементами позначені розмітка призначена для меню користувача і блок виведення тексту, так як ці елементи описані в фалі *index.html*, для нього прописується відповідна директива маніфесту.

Приклад вмісту файлу:

```
<!DOCTYPE html>
<html manifest="cache.manifest">
<body>
  <div class="menu">
    ...
  </div>
 <div class="content">
    ...
  </div>
  <script>
    ...
  </script>
 </body>
</html>
```

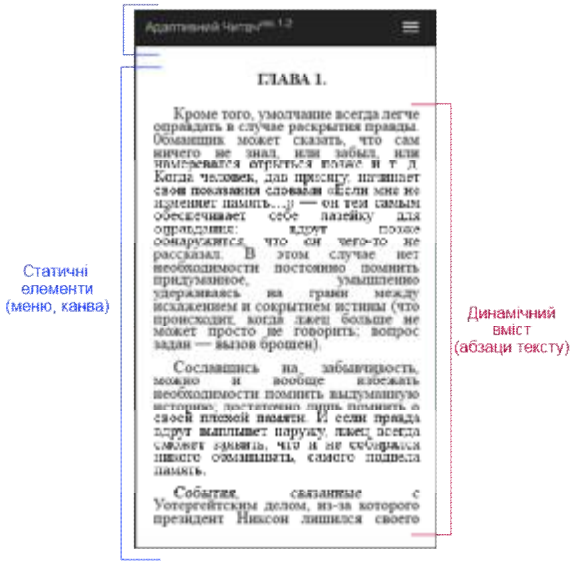

Рис. 1. Схематичне зображення розділення графічного інтерфейсу веб-додатка, на статичну та динамічну складову.

Відповідно до вказівок всередині файлу *cache.manifest*, весь вміст *index.html* буде поміщено в сховище браузера. При цьому виникає конфліктна ситуація. У разі зміни на сервері, кешовані частини коду не будуть оновлені при наступному зверненні.

Рішенням даної проблеми може бути постійний перезапис вмісту кеша при наявності доступу до мережі, але в цілях економії ресурсів необхідно використовувати алгоритм перевірки фактичної зміни. Власна реалізація даного алгоритму полягає в використанні подій *JavaScript - load* і *updateready*. Після завантаження браузера перевіряється наявність доступу до мережі, запитується хеш-сума маніфесту на стороні сервера і порівнюється з хеш-сумою кешованого вмісту. У разі розбіжності значень, відбувається автоматичне спустошення кешу командою *appCache.swapCache*().

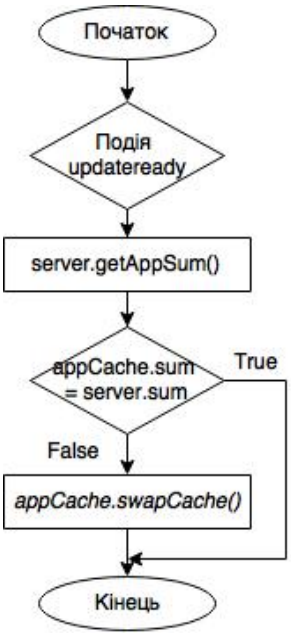

Рис. 2. Схема алгоритму перевірки змін статичних частин коду.

Підміна динамічного вмісту повинна відбуватися ґрунтуючись на показниках поведінки користувача за певний часовий період. У той же час, необхідно мати повну версію структурованого тексту в локальній пам'яті. Обидва випадки неможливі без використання вбудованої бази даних.

Реалізація даної підзадачі забезпечена використанням технології *WebSQL*. Позитивною стороною є наявність майже повноцінного *SQL*, але великим мінусом є виконання запитів через замикання.

У процесі дослідження даної технології, був розроблений підхід організації роботи з внутрішньою базою даних через функції користувальницького коду.

При першому завантаженні сторінки, база даних проходить ініціалізацію функцією *dbInit*(), при цьому глобальний об'єкт *db* повинен отримати всі її властивості і стати точкою підключення.

```
function dbInit() {
   db = openDatabase({
     'name': 'localbase',  'version': '0,1',
     'title' : 'AdaptiveRead',
     size': 5*1024*1024
   });
}
```

Далі слідує виклик функції *dbConnect*(), в якій виконується створення тестової таблиці. При невдалому завершенні, об'єкт *db* отримує ознаку помилки, що унеможливлює його подальше використання.

```
function dbConnect() {
  db.transaction(function(tx) {
     tx.executeSql("SELECT * FROM test",
       [], [], function (tx, error) {
       tx.executeSql("CREATE TABLE test" +
                 "text TEXT, user TEXT, ")");
           });
       });
         db.errorHandler =
         function (transaction, error) {
            alert("Error processing SQL);
            return true;
          }
    }
}
```

Як видно з представленого прикладу користувальницкього коду, виконання запиту наповнено додатковим описом замикаючих конструкцій, що робить код менш читабельним і збільшує обсяг переданих даних. Тому при всіх подальших зверненнях строго використовується обгорткова функція *dbQuery*().

```
function dbQuery(db, query, params) {
   db.transaction(function (tx) {
      tx.executeSql(query, params ,
         function (tx, results) {
           return  fetchObject(results.rows);
        }, null);
   }); }
```

Огортання конструкції запиту в окрему функцію зменшує кількість повторень коду в 4 рази, а результат виконання буде повернений у вигляді колекції однотипних об'єктів. В кінцевому результаті, використання бази даних зводиться до наступного порядку виклику функцій.

```
var db;
dbInit();
dbConnect();
```

```
res = dbQuery('SELECT * FROM table');
```

Пересилання характеристик оцінки користувача та адаптивне завантаження текстових блоків, може бути організоване з використанням асинхронних запитів. Слід звернути увагу на те, що більшість серверних реалізацій визначає такі запити як крос-доменну атаку і відразу ж відхиляють їх [5].

Найбільш прийнятним варіантом організацій обміну даними в описуваному стеку технологій, буде здійснення пересилання з використанням вбудованого в браузер *JavaScript*-класу *webSocket*. Дана технологія є стандартизованої і має досить простий синтаксис, взаємодія з нею зводиться до трьох операцій: створення підключення; передача даних методом *send*; отримання даних через подію *onmessage*.

```
var ws = new WebSocket("ws://cifr.us/ws");
ws.send(user.sense['timeRead'].value);
ws.onmessage = function(event) {
    console.log(event.data);
};
```

У порівнянні з *Ajax*, пересилання даних виконується з більшою швидкістю і має більшу криптостійкість за рахунок внутрішньої організованості *ws*-протоколу [6].

Ключовим показником було прийнято час очікування повного завантаження поточної сторінки веб-додатка, залежний від таких факторів як швидкість з'єднання і обсяг даних при передачі. Описані методи, що включають прийоми кешування, використання інформації в базі даних і організації обміну даними між сервером і клієнтом, виявилися найбільш ефективними.

Для підтвердження цього висновку було проведено практичне дослідження і вимірювання часу передачі при різних умовах. В якості експериментальної бази використовувалися два типи веб-додатків: класичний (К) підхід з серверною *MVC*-архітектурою і прогресивний (П) з використанням описаних технологій. В обох варіантах призначення і функціональні можливості повністю ідентичні описаного прикладу метрико-адаптивного читання. Швидкість з'єднання між клієнтом і сервером - 100 Мбіт/сек, середній час відгуку сервера - 37.461 мс.

Результати замірів наведені в таблиці 1. Представлені значення є усередненими після 10 ітерацій. В якості інструментарію використаний режим розробника в браузері *Safari Mobile*.

Таблиця 1. Результати вимірювань очікування передачі даних

Параметри	К	П
Перший запит	700 мс	800 мс
Повторний запит	500 мс	230 мс
Вхідні даних	260 Кбайт	320 Кбайт
Запит наступних абзаців	8 мс	2 мс
Затримка з'єднання	1000 мс	20 мс
Вихідні дані	400 байт	

При першому запиті виконується повне завантаження сторінки веб-додатки. Час очікування завершення виконання у додатку П на 14% вище, ніж у К, в наслідок завантаження більшого обсягу вхідних даних, що пояснюється необхідністю використання коду роботу з локальною базою даних. При цьому, повторне звернення у П займає на 54% менше часу за рахунок перманентного кешування вмісту сторінки через маніфест додатка.

Відправка даних на сервер. Додаток К встановлює нове з'єднання через *Ajax*-запит, для П використовується *webSocket*-клієнт. В обох випадках, обсяг вихідних даних складає 400 байт. У додатку П відправка відбувається в середньому на 80% швидше, за рахунок підтримування соке-

том низьковитратного з'єднання з сервером.

Висновок

Дослідження розвитку програмного забезпечення виявило необхідність розробки складних програмних продуктів орієнтованих на роботу у веб-середовищі. Найбільш актуальним напрямком визначено розробку «прогресивних вебдодатків», я не мають стандартизованого підходу і чітко вираженої архітектури.

Було проведено аналіз пропонованих варіантів вирішення задач вебдодатків, надані приклади їх використання. Отримані результати тестування демонструють значне скорочення часу передачі.

Можна сказати, що прогресивні додатки є логічним розвитком односторінкових додатків. До відмінних рис відноситься відсутність необхідності в інсталяції і можливість взаємодії без наявності доступу до мережі Інтернет. Це здешевлює вартість кінцевого продукту, а також спрощує поширення і подальшу підтримку. У порівнянні з нативними додатками, мають ідентичні функціональні можливості. У довгостроковій перспективі здатні витіснити останні.

Актуальність і затребуваність підтверджується практичним застосуванням всесвітньо відомими розробниками програмного забезпечення [7].

У майбутньому заплановано: більш глибоке вивчення можливостей технологій застосування прогресивних вебдодатків; розробка моделей систем для перевірки можливостей сучасних підходів проектування програмного забезпечення; аналіз створюваних потоків даних з метою подальшої оптимізації алгоритмів обміну.

Отримані результати і наведений матеріал, можуть бути використані при проектування складних мультиплатформних клієнт-серверних систем.

Список літератури

1. Кристиан Дари. *AJAX* и *PHP*. Разработка динамических веб-приложений. – М.: Символ-Плюс, 2009. – 336 с.

2. Майкл Миковски., Джош К. Пауэлл, Разработка одностраничных веб-приложений. – М.: ДМК Пресс, 2014. – 512 с.

3. *Progressive Web Apps* [Інтернет-ресурс] / *Web*-сайт: *developers.google.com*; Режим доступу: *https://developers.google.com/web/progressive-web-apps/*, вільний.

4. *Chris Anderson. CouchDB: The Definitive Guide // O`Reilly Media – Sebastopol, USA* , 2014. – P. 272.

5. *Andrew Lombardi. WebSocket. Lightweight Client-Server Communications // O`Reilly Media – Sebastopol, USA,* 2015. – P. 144.

6. Моісейкін О.С. Технологія розробки веб-додатків реального часу. // МОДС 2016. – Жукин, 2016. – С. 473-475.

7. *Air Berlin:* реализация *Progressive Web App* (09/2016) [Інтернет-ресурс] / *Web*-сайт: www.*habrahabr.ru*; Режим доступу: *https://habrahabr.ru/company/google/blog/308498/*, вільний.

Статтю подано до редакції 20.02.2017