МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

**Факультет программной инженерии и компьютерной техники**

**Направление (специальность) ⎯ 09.04.02 Информационные системы и технологии**

**Специализация ⎯ Веб-технологии**

**Дисциплина — Проектирование и анализ языков веб-решений**

**Курсовой проект (работа)**

**ТЕМА: Исследование особенностей создания PWA**

ВЫПОЛНИЛ

Студент группы P41622 \_\_\_\_\_\_\_\_\_\_\_\_ Паршин Д.С.

№  группы подпись, дата ФИО

ПРОВЕРИЛ     \_\_\_\_\_\_\_\_\_\_\_\_ Государев И.Б.

ученая степень, должность подпись, дата ФИО

**САНКТ-ПЕТЕРБУРГ  
2019 г.**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc7831841)

[ГЛАВА 1. PROGRESSIVE WEB APP 7](#_Toc7831842)

[ГЛАВА 2. ВИЗУАЛЬНЫЕ ОСОБЕННОСТИ PWA 9](#_Toc7831843)

[2.1. Прогрессивное улучшение 9](#_Toc7831844)

[2.2. Методы вёрстки 11](#_Toc7831845)

[2.3. Application shell 14](#_Toc7831846)

[ГЛАВА 3. ВНУТРЕННИЕ ОСОБЕННОСТИ PWA 15](#_Toc7831847)

[3.1. Web App manifest 15](#_Toc7831848)

[3.2. HTTPS 15](#_Toc7831849)

[3.3. Service Worker 17](#_Toc7831850)

[3.4. Push-уведомления 22](#_Toc7831851)

[ЗАКЛЮЧЕНИЕ 26](#_Toc7831852)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 27](#_Toc7831853)

# **ВВЕДЕНИЕ**

**Актуальность темы**. В 2007 г. был представлен первый смартфон iPhone, который не только перевернул представление о носимых гаджетах, но и начал историю активного использования интернета с носимых устройств. С тех пор количество пользователей мобильного интернета только растёт. Так в 2009-ом году количество мобильных пользователей составляло около 1% (99% desktop), в 2012-ом году 11%, в 2015-ом 35%, а в 2018-ом 52%, что иллюстрирует рисунок 1 [1]. Мы можем наблюдать стремительный рост захода в интернет, а значит и веб-сайты с гаджетов[2].Справедливо предположить, что рост преобладания мобильных пользователей будет только ускоряться. Как следствие, будет наблюдаться рост и числа покупок в интернете, актуальность рекламы[3].

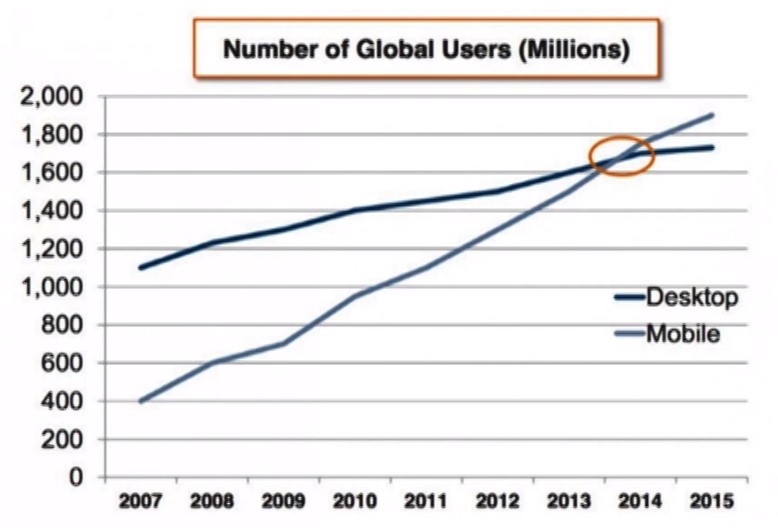
****

Рисунок 1. График роста числа мобильных пользователей интернета [1]

Данная тенденция существует и в России, о чём свидетельствует исследование GFK [24], которое проиллюстрировано на рисунке 2.



Рисунок 2. Рост пользователей мобильного интернета в России [2]

Разработка веб-сайтов под мобильные платформы имеет отличия от ранее сложившейся практики создания веб-сайтов для более мощных и больших устройств. Так, среди особенностей создания сайтов под мобильные платформы можно выделить разнообразие размеров и разрешений дисплеев, медленный интернет и большое количество пользователей с «слабыми» устройствами низкой ценовой категории [4].

Параллельно с ростом числа пользователей, просматривающих веб-сайты, растёт число просмотров мобильных приложений. Согласно comScore [5] пользователи отдают предпочтение мобильным приложениям, что иллюстрирует рисунок 3. Согласно отчёту State of Mobile 2019 от App Annie [6] по данным на 2018-ый год пользователи в среднем проводят по 3 часа в день в мобильных приложениях, рост на 50%. Рост трат на приложения (не игры) составил 120% по сравнению с 2016. Во многом это происходит благодаря подписке. В большинстве стран мобильная экономика растет гораздо быстрее ВВП. В 2019-ом году App Annie прогнозируют, что на 60% больше приложений будет монетизироваться через рекламу.

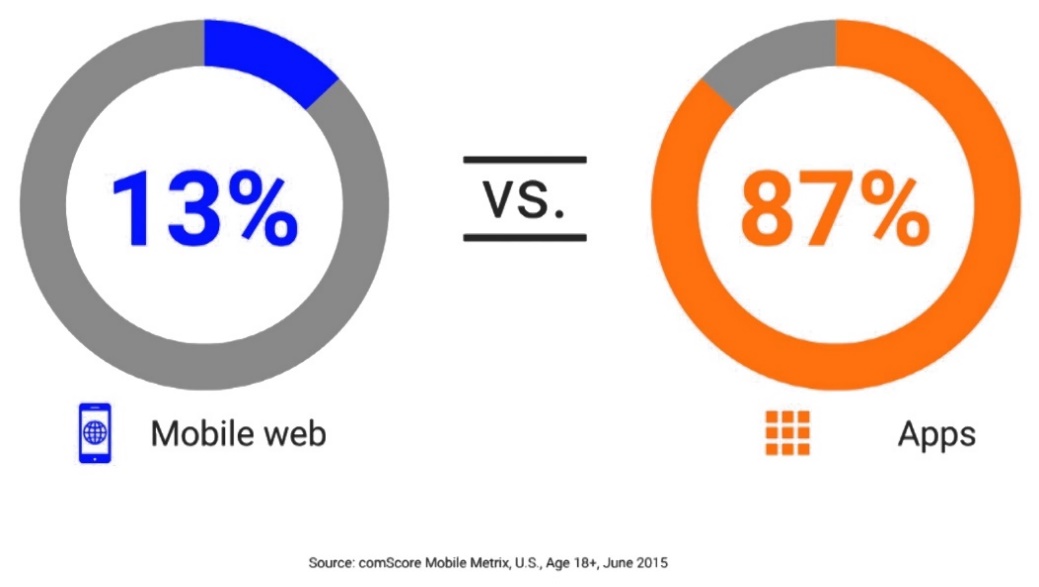
****

Рисунок 3. Соотношение затрат времени пользователей на просмотр контента на сайтах и приложениях

В 2015-ом году Google анонсировали Progressive Web Apps (PWA). PWA – это сайт, построенный с использованием веб-технологий, который взаимодействует с пользователем как приложение, позволяет получать уведомления, может быть установлен на любой гаджет, работать без интернет-соединения, имеет возможность быть установленным на домашний экран.

PWA, соединяя в себе преимущества веб-сайтов и приложений, улучшает пользовательский опыт и повышает вовлечённость людей в использование продукта. Согласно статистике опубликованной Google:

1. AliExpress увеличил коэффициент конверсии для новых пользователей на 104%.
2. United eXtra Electronics показал 4-кратное увеличение возвратов посетителей и на 100% повысил уровень продаж от пользователей, которые приходят в результате взаимодействия со всплывающими уведомлениями.
3. 5miles снизил показатель отказов на 50% и увеличил конверсии на 30%.
4. Konga используют на 92% меньше данных для первоначальной загрузки по сравнению с загрузкой их родного приложения.
5. The Washington Post создали Progressive Web App, который установлен в фоновом режиме, когда люди читают их AMP страницы в результатах поиска Google. В результате количество посещений статей выросло на 12%. От загрузки статей по 8 секунд в 2013 они пришли к 80 миллисекундам в Progressive Web App.

Востребовано исследование, направленное на поиск характерных черт PWA, позволяющих ориентироваться на них при проектировании и разработке веб-приложений.

**Цель исследования:** систематизация отличительных характеристик и особенностей развития progressive web app.

**Задачи:**

1. Анализ литературы и веб-ресурсов по теме PWA.
2. Конкретизация особенностей PWA.

# **ГЛАВА 1. PROGRESSIVE WEB APP**

PWA – это название группы приложений, которые используют стек Web технологий (JS + HTML + CSS) и позволяют соединить простоту использования веб-сайта со специфичными для нативных приложений операционной системы UX и техническими возможностями [7].

Основное предназначение PWA – увеличить конверсию, количество пользователей и удобство использования веб-приложений на мобильных устройствах.

Критерии, которым должно соответствовать PWA:

1. Прогрессивность: способность работать с каждым пользователем вне зависимости от окружения, используя метод постепенного улучшения как основной принцип работы;
2. Адаптивность: способность подстраиваться под любое устройство: desktop, смартфоны, планшеты или что-либо другое;
3. Способность выглядеть как native приложения: соответствовать привычным для пользователя способам взаимодействия и навигации;
4. Независимость от соединения: используя Service Worker приложение должно работать в оффлайн режиме при прерывании или отсутствии соединения;
5. Самообновляемость: приложение должно контролировать процесс автоматического обновления посредством Service Worker API;
6. Безопасность – способность посредством использования HTTPS предотвращать перехват и подмену данных;
7. Определяемость (опознаваемость): посредством W3C манифеста и регистрации через Service Worker приложение идентифицируется как «приложение» поисковыми системами;
8. Способность удерживать: используя технические возможности мотивировать пользователя повторно использовать приложение, например посредством push-уведомлений;
9. Легкость в установке: способность «сохранить» приложение на устройстве пользователя посредством добавления PWA приложения в список установленных приложений без использования магазина приложений;
10. Легкость в использовании: для начала использования приложения достаточно открыть URL. Установка приложения не обязательна.



Рисунок 4. Отличительные черты (критерии) PWA

Следует отметить, что не все описанные критерии являются обязательными. Так, push-уведомления и самообновляемость могут отсутствовать. Однако существуют важные характеристики PWA, которые следуют из соответствия таким критериям, как адаптивность. Рассмотрим их во второй и третьей главах.

# **ГЛАВА 2. ВИЗУАЛЬНЫЕ ОСОБЕННОСТИ PWA**

## 2.1. Прогрессивное улучшение

PWA строятся на технологии прогрессивного улучшения. Прогрессивное улучшение (progressive enhancement) – это подход к разработке, при котором все браузеры получают одинаковый набор функциональных решений, но в последних версиях браузеров этот набор расширяется за счёт поддерживаемых новых свойств.

Прогрессивное улучшение предполагает, что веб-интерфейсы должны создаваться поэтапно, циклически, от простого к сложному. На каждом из этапов должен получаться законченный веб-интерфейс, который будет лучше, красивее и удобнее предыдущего. Можно выделить четыре основных этапа создания сайта с прогрессивным улучшением [17]:

1. HTML
2. CSS
3. CSS3
4. JavaScript

На первом этапе исходное содержание страницы размечается с помощью HTML. На этом этапе необходимо сделать логичную и семантически правильную разметку. Никакого дополнительного оформления не осуществляется. В результате получается корректно размеченный HTML-документ, который браузер отображает так, как считает нужным. Такой документ можно назвать «наименьшим общим знаменателем», так как он будет корректно отображаться в любом, даже самом простом, браузере. Прогрессивное улучшение подразумевает, что первый этап является самым важным, так как в вебе делается акцент на содержании [8].

На втором этапе документ оформляется с помощью CSS и обретает более аккуратный вид: появляется сетка страницы, задаются параметры шрифта элементов, фоновые изображения и т.п.

На третьем этапе применяются новые возможности из семейства спецификаций CSS3, и документ доводится до идеала: полупрозрачные плашки, скругление углов, тени. Также с помощью CSS3 можно добавить различные анимационно-декоративные эффекты: постепенное затухание или смещение элементов, подсвечивание полей форм и т.п.

На четвёртом этапе доводится процесс взаимодействия с интерфейсом: различные AJAX решения, динамические элементы. Тут используется JavaScript. Этот этап отвечает за удобство.

Подводя итог описанным этапам, отметим аспекты PWA:

1. Смысл документа, логическая разметка. Технология: HTML.
2. Внешний вид. Технология: CSS.
3. Снова внешний вид. Технология: CSS3.
4. Взаимодействие, интерактивность, удобство. Технология: JavaScript.

Следует также отметить обратную прогрессивному улучшению технологию, постепенной деградации (graceful degradation). Постепенная деградация выражается в возможности работы при отключённом JavaScript, в достаточно аккуратном отображении интерфейса в браузере, не поддерживающем новые свойства CSS3, в адекватном отображении сайта при отключенных изображениях. В каждом из этих случаев работа пользователя с интерфейсом будет в принципе возможна, хотя и не так удобна [9].

Ключевые различия этих двух технологий:

1. Постепенная деградация — более широкое понятие, чем прогрессивное улучшение, который ограничен только веб-интерфейсами.
2. Прогрессивное улучшение задаёт вектор движения (начинать с простого), а для прогрессивной деградации это не так важно.
3. Прогрессивное улучшение настаивает на важности содержания, а для прогрессивной деградации оно на втором плане.

Прогрессивное улучшение — это более строгая и последовательная идеология разработки веб-интерфейсов.

## 2.2. Методы вёрстки

Прежде чем описать адаптивную и отзывчивую вёрстку, необходимо разобраться, какие методы вёрстки существуют и какие между ними существуют отличия.

Когда «мобильный» интернет не был популярен и большинство пользователей использовали персональные компьютеры с разрешением 1024х768, доминировала фиксированная вёрстка веб-сайтов. На сегодняшний день существует несколько методов реализации вёрстки: резиновая, адаптивная, responsive (отзывчивая). Зачастую можно встретиться с непониманием отличий между ними; несмотря на общую цель эти направления отличаются по своей сущности. Данная работа ставит перед собою цель внести ясность в этот вопрос [10].

**Фиксированная вёрстка** представляет собой самый первый метод вёрстки, удовлетворяющий нужды пользователей интернета с начала его существования; создаётся путём задания фиксированной ширины блоков в пикселях, которые не меняют своих размеров. Выбор ширины зависит от ориентации на определённые разрешение мониторов. Для всех пользователей сайт, построенный таким образом будет выглядеть одинаково [11].

Но при больших разрешениях окажется слишком много пустого пространства, что выглядит дисгармонично. При маленьких разрешениях экрана возможно появление горизонтальной полосы прокрутки, крупные блоки с текстом не будут помещаться на экран. Главными преимуществами фиксированной вёрстки являются её просто, низкая себестоимость, быстрая скорость в её исполнении, гарантия одинакового отображения на различных экранах. Недостатком является невозможность создать универсальный сайт, который будет подстраиваться под разрешение экрана пользователей.

**«Резиновая» вёрстка** стала первым шагом на встречу к пользователям устройств с не стандартным разрешением экрана. Основная особенность данного метода заключается в том, что ширина колонок задаётся в процентах, а не в пикселях, поэтому «резиновый» дизайн при любом разрешении монитора растягиваются в заданных пропорциях. Создать такой дизайн, особенно при использовании блочной вёрстки несколько сложнее, чем фиксированный, зато пользователь не будет видеть пустых полей на странице. Преимущества: отображение элементов сайта в заданных пропорциях, вне зависимости от разрешения монитора. Недостатки: при большом разбросе размеров экранов сложно реализовать дизайн так, чтобы при любом разрешении все элементы выглядели одинаково хорошо, чтобы блоки не «разъезжались», а картинки и видео не вылезали за границы блоков [12].

**Адаптивная вёрстка** – метод, который предложил Арон Густафсон в работе «Adaptive web-design» [13]. Адаптивный метод вёрстки позволяет одинаково хорошо отображать сайт на мобильных телефонах, планшетах и устройствах с большим разрешением экрана. Разработчики, по сути, создают несколько независимых вариантов дизайна для разных устройств. Когда пользователь заходит на сайт, сервер определяет тип устройства и разрешение экрана и «отдаёт» ту или иную версию сайта. При этом каждая версия сайта заранее оптимизирована под то или иное разрешение экрана. Для версий под экраны с небольшими разрешениями используются уменьшенные изображения, а некоторые элементы совсем убираются. Компоновка блоков в дизайне под мобильные версии отличается от «большой» версии. Стили переключаются от одного breakpoint (точки изменения разрешения экрана) к другому, то есть мы имеем фиксированные макеты, например, для iPhonе и iPad, не затрагивая устройства с иными разрешениями экрана [14]**.** Для реализации адаптивной вёрстки используются медиа-выражение, доступные в CSS3 (@Media). Преимущества: адаптация под популярные разрешения устройств, возможность создать «мобильную» версию сайта. Недостатки: ориентация на определённые разрешения экранов.

**«Отзывчивая» (responsive) вёрстка –** в настоящее время самый современный метод, предложенный Итаном Маркоттом и объединяющий в себе «резиновую», адаптивную вёрстки и гибкие изображения [15]. «Отзывчивый» дизайн автоматически подстраивается под любую ширину экрана, как и резиновый, при этом, в процессе подстройки может менять расположение блоков, размер картинок, могут пропадать или изменяться некоторые элементы (например, обычное горизонтальное меню может поменяться на выпадающее) [16].

Все отступы и размеры Итан предлагал указывать в процентах с дробями для большей точности. Особенностью такого подхода стало плавное изменение сайта, с ориентацией не на конкретные устройства, а на содержимое. Подобное объединение резиновой и адаптивной вёрстки в реализации самое сложное. Но результат получается наиболее приемлемый.

**Сравнительный анализ технологий вёрстки:**

Таблица 1. Сравнительный анализ технологий вёрстки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Фиксированная | «Резиновая» | Адаптивная | Responsive |
| Изменение размера блока при изменение размера viewport | Нет | Да | Да | Да |
| Изменение размера пропорционально размеру вью порта | Нет | Да | Нет | Да |
| Предлагает отличающиеся версии композиции сайта в зависимости от viewport | Нет | Нет | Да | Да |
| Плавность изменения размеров блоков | Нет | Да | Нет | Да |

В результате сравнительного анализа выявлено значительное преимущество подхода responsive вёрстки.

## 2.3. Application shell

Appication shell (app shell) – это шаблон графического интерфейса, похожий на оболочку native-приложения. App shell представляет собой минимальный набор HTML, CSS и JS, необходимый для обеспечения нормальной работы пользовательского интерфейса, который может обеспечить мгновенную и надёжную производительность для пользователей при повторном посещении веб-сайта. Он хранится на клиентском устройстве и загружается при запуске приложения, а затем уже в него загружается динамическая информация [18].

Преимущества использования оболочки app shell:

1. Надёжная и быстрая работа.
2. Взаимодействие схожее на работу с native-приложением.
3. Экономия загружаемых данных.

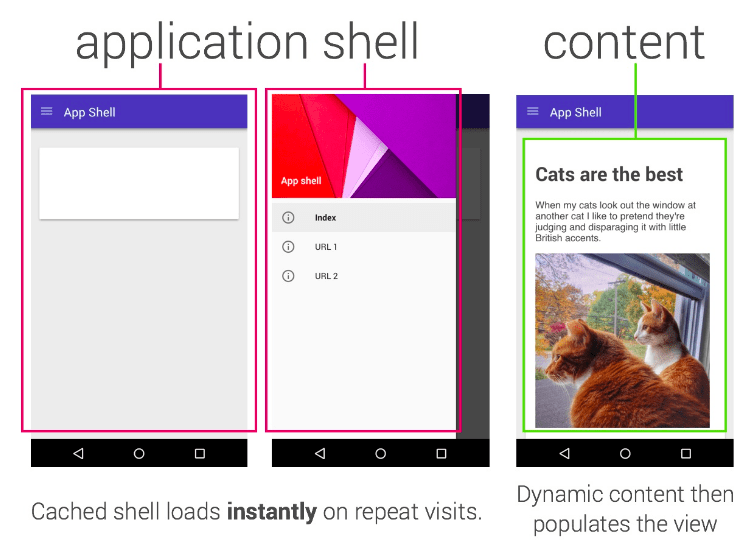


Рисунок 5. Демонстрация App shell с загруженным контентом и без

# **ГЛАВА 3. ВНУТРЕННИЕ ОСОБЕННОСТИ PWA**

* 1. **Web App manifest**

JSON файл, декларативно определяющий для браузера название приложения, иконку, как будет выглядеть PWA (fullscreen, standalone и др.) и некоторые иные параметры. Он позволяет «установить» PWA как отдельное приложение на домашний экран мобильного устройства.

Манифест веб-приложения внедряется с помощью тега ссылки в заголовке документа. Например: <link rel="manifest" href="/manifest.webmanifest"> .

Пример содержания веб-манифеста:

{

"name": "HackerWeb",

"short\_name": "HackerWeb",

"start\_url": ".",

"display": "standalone",

"background\_color": "#fff",

"description": "A simply readable Hacker News app.",

"icons": {

"src": "images/touch/homescreen48.png",

"sizes": "48x48",

"type": "image/png"

},

"related\_applications": {

"platform": "web"

} }

* 1. **HTTPS**

В процессе создания веб-приложения SW будут работать нормально на localhost, но после выпуска приложения необходимо будет использовать HTTPS.

Service worker, который не защищён HTTPS, подвержен атакам со стороны, так как злоумышленник сможет перехватывать соединения и создавать фальшивые ответы на запросы приложения. Именно поэтому, для того чтобы сделать систему безопаснее, программист должен регистрировать SW на страницах, которые обслуживаются по HTTPS соединению. В частности, это даст уверенность в том, что SW, загружаемый в браузер, не был модифицирован во время передачи его кода по сети.

HTTPS не является отдельным протоколом передачи данных, а представляет собой расширение протокола HTTP с надстройкой шифрования. Передаваемые по протоколу HTTP данные не защищены, HTTPS обеспечивает конфиденциальность информации путём её шифрования.

Защиту данных в HTTPS обеспечивает криптографический протокол SSL/TLS, который шифрует передаваемую информацию. По сути, этот протокол является обёрткой для HTTP. Он обеспечивает шифрование данных и делает их недоступными для просмотра посторонними. Протокол SSL/TLS хорош тем, что позволяет двум незнакомым между собой участникам сети установить защищённое соединение через незащищённый канал [19].

Для реализации передачи данных посредством HTTPS на веб-сервере, обрабатывающем запросы от клиентов, должен быть установлен специальный SSL-сертификат.

* 1. **Service Worker**

Веб-воркеры (Web Workers) – это механизм, который позволяет скрипту выполняться в фоновом потоке, который отделен от основного потока веб-приложения. Преимущество заключается в том, что ресурсоёмкие вычисления могут выполняться в отдельном потоке, позволяя запустить основной (обычно пользовательский) поток без блокировки и замедления.

Веб-воркеры реализованы с использованием .js-файлов, которые включаются в страницу с применением асинхронного HTTP-запроса. Эти запросы полностью скрыты от разработчика благодаря Web Worker API.

Воркеры используют механизмы передачи сообщений, характерные для технологий, которые применяются для организации взаимодействия потоков, что позволяет организовать их параллельное выполнение. Они отлично подходят для того, чтобы выполнять тяжёлые вычислительные операции, не замедляя работу пользовательского интерфейса [20].

Веб-воркеры выполняются в изолированных потоках в браузере. Как результат, код, который они выполняют, должен быть включён в отдельный файл.

Существует 3 типа веб-воркеров:

1. Выделенные воркеры (Dedicated Workers) – экземпляры выделенных веб-воркеров создаются главным процессом. Обмениваться данными с ними может только он.
2. Разделяемые воркеры (Shared Workers) – дают доступ к разделяемому воркеру может получить любой процесс, имеющий тот же источник, что и воркер (например — разные вкладки браузера, iframe, и другие разделяемый воркеры).
3. Сервис-воркеры (Service Workers) — это воркеры, управляемые событиями, зарегистрированные с использованием источника их происхождения и пути. Они могут контролировать веб-страницу, с которой связаны, перехватывая и модифицируя команды навигации и запросы ресурсов, и выполняя кэширование данных, которым можно очень точно управлять. Всё это предоставляет средства управления поведением приложения в определённой ситуации (например, когда сеть недоступна).

Service Worker – это проксирующий слой между серверной и клиентской частью, находящийся в браузере. Все запросы браузера идут через него. Данное разделение на два независимых слоя позволяет достичь перехода обычного веб-сайта в PWA максимально простым. Из хранилищ у Service Worker'a есть доступ к Cache Storage для веб-ресурсов, и IndexDB для данных.

Service Worker позволяет принять запрос от браузера, проверить состояние сети, взять данные из хранилища, произвести с ними операции и вернуть некий результат обратно в браузер – который будет считать, что ответ ему пришел от сервера, или не будет – на усмотрение разработчика. Два браузерных слоя (клиентский фронтэнд и Service Worker) позволяют писать полноценные приложения. Для большинства же сайтов будет достаточно кэширующей функциональности Service Worker'a, чтобы стать PWA.

Service Worker представляет собой javascript файл, подключаемый в html коде страницы. В нём программист определяет логику работы с приходящими от клиента запросами и иную функциональность.

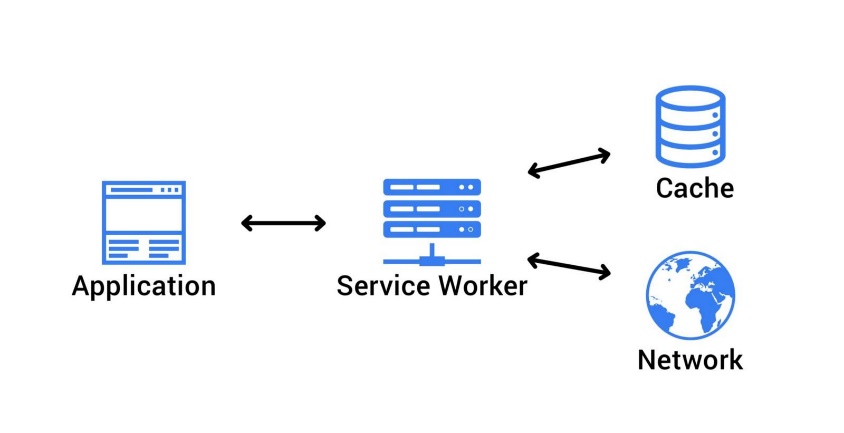


Рисунок 6. Иллюстрация взаимодействия SW с клиентской и серверной частями

Жизненный цикл Service Worker состоит из трёх этапов:

1. Загрузка
2. Установка (регистрация)
3. Активация

На первом этапе веб-браузер загружает js-файл, содержащий код service worker (SW). Прежде чем установить SW, его нужно зарегистрировать. Это делается в JavaScript коде. Когда SW успешно зарегистрирован, браузер запускает его установку в фоновом режиме.

Пример кода, отвечающего за регистрацию SW:

if ('serviceWorker' in navigator) {

window.addEventListener('load', function() {

navigator.serviceWorker.register('/sw.js').then(function(registration) {

console.log('ServiceWorker registration successful');

}, function(err) {

console.log('ServiceWorker registration failed: ', err);

});});}

Здесь мы проверяем, поддерживает ли браузер технологию SW, если да, то загружаем SW, сообщая об этом в консоль. Если при регистрации SW произошла ошибка, то также сообщаем об этом в консоль. Метод register() можно запускать каждый раз при запуске страницы, если SW уже ранее был зарегистрирован, ошибки не произойдёт. Так же на этом этапе возможно поместить необходимые файлы в кэш.

После установки SW необходимо его активировать. На этом этапе у разработчика появляется возможность с данными, которые были ранее были кэшированы. После активации SW может управлять всеми страницами (обрабатывать события), которые попадают в его область видимости. Следует отметить, что механизмы SW не будут действовать на активировавшую его страницу до тех пор, пока она не будет перезагружена.

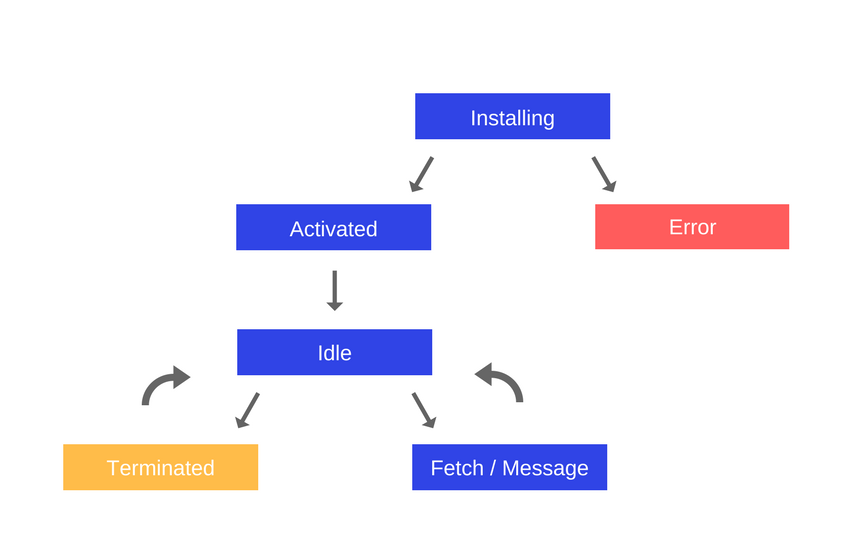


Рисунок 7. Жизненный цикл SW

Обычно в кэш помещают минимальный набор HTML, CSS и JavaScript необходимый для обеспечения работы пользовательского интерфейса (App shell).

Примера кода, отвечающего за помещение в кэш css и js файлов:

let CACHE\_NAME = 'my-web-app-cache';

let urlsToCache =

[ '/styles/main.css',

'/scripts/app.js',

'/scripts/lib.js'];

self.addEventListener('install', function(event) {

event.waitUntil(

caches.open(CACHE\_NAME)

.then(function(cache) {

console.log('Opened cache');

return cache.addAll(urlsToCache);

}) );});

После того как SW будет успешно установлен, и пользователь перейдёт на иную страницу приложения или обновит страницу, SW начнёт получать fetch события.

Пример кода, который анализирует запрос и ищет кэшированные результаты для этого запросы в SW, после чего возвращает ответ:

self.addEventListener('fetch', function(event) {

event.respondWith(

caches.match(event.request)

.then(function(response) {

if (response) {

return response; }

let fetchRequest = event.request.clone();

return fetch(fetchRequest).then(

function(response) {

if(!response || response.status !== 200 || response.type !== 'basic') {

return response;}

var responseToCache = response.clone();

caches.open(CACHE\_NAME)

.then(function(cache) {

cache.put(event.request, responseToCache);

});

return response;

} );}));});

В данном примере event.respondWith() определяет то, как будет проходить реакция на fetch событие. Мы передаём promise из caches.match(), который анализирует запрос и определяет, имеются ли какие-либо кэшированные ответы на подобный запрос, в ранее сохранённом кэше. Если искомое найдено в кэше, то из него извлекается ответ. Если то, что необходимо не найдено, то выполняется fetch операция. Проверяется, что полученный ответ равен «200». По итогам ответ добавляется в кэш.

Когда пользователь заходит на веб-страницу, браузер в фоновом режиме пытается выполнить повторную загрузку JS-файла, содержащего SW. Если имеется хотя бы небольшая разница между старым и новым SW файлами, то браузер запустит новый SW и вызовет install-событие. Следует отметить, что в этот момент пользователь видит работу старой версии SW, новый начнёт полноценно работать после перезагрузки веб-сайта (будет вызвано событие activate). На этом этапе обычно старые версии кэша заменяются новыми.

На данный момент (май 2019-го года) SW поддерживается большинством современных браузеров [21], что иллюстрирует рисунок 8.

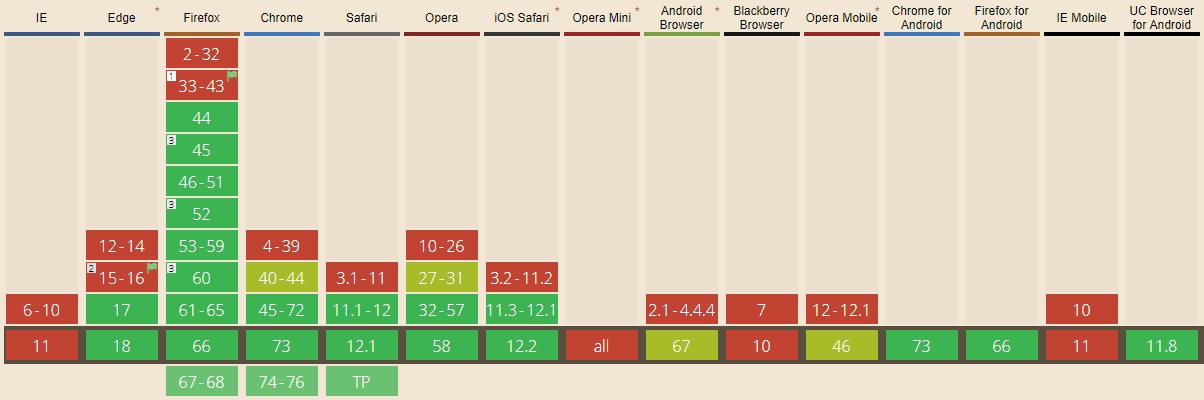


Рисунок 8. Поддержка SW браузерами [21]

* 1. **Push-уведомления**

Технология push-уведомление позволяет пользователям подписываться на периодические уведомления от веб-приложения, которые направлены на привлечение внимания пользователей к приложению.

Одним из механизмов, обеспечивающих работу данной технологии, являются SW. SW обрабатывают push-уведомления, экономно расходуют ресурсы системы, так как их код выполняется только тогда, когда в браузер поступает новое уведомление, за работу с которым ответственен определённый SW.

Под push-уведомлениями обычно подразумевается работа двух технологий. Push API, которая используется, когда сервер посылает сообщение SW, и Notifications API, который применяется, когда SW, или скрипт в веб-приложении намеревается показать пользователю уведомление.

Для реализации механизма push-уведомлений необходимо выполнить три шага [22]:

1. Подготовить пользовательский интерфейс, т.е. подготовить механизм, который позволит пользователю подписаться на уведомления.
2. Отправка push-уведомления. На этом этапе на сервере разработчика происходит обращение к соответствующему API, что приводит к отправке уведомления пользователю.
3. Приём push-сообщения. На данном этапе происходит обработка push-сообщения после того, как оно будет принято браузером.

Перед началом работы следует проверить, поддерживает ли используемый браузер технологии SW и Push-уведомлений (serviceWorker в объекте navigator, PushManager в объекте window). Зарегистрировав SW, необходимо запустить процедуру подписки пользователя на уведомления. Для этого необходимо получить от него разрешение на отправку ему push-уведомлений. Следует обратить внимание, что могут использоваться две версии необходимого API. В более старой версии оно принимало функцию обратного вызова, в новой оно возвращает promise. Хорошим решением будет использовать оба этих подхода.

Пример кода, запускающего регистрацию получения push-уведомлений:

function requestPermission() {

return new Promise(function(resolve, reject) {

const permissionResult = Notification.requestPermission(function(result) {

// Поддержка устаревшей версии с функцией обратного вызова.

resolve(result);

});

if (permissionResult) {

permissionResult.then(resolve, reject);

}})

.then(function(permissionResult) {

if (permissionResult !== 'granted') {

throw new Error('Permission not granted.');

}});}

После того как пользователь выберет один из предложенных браузером вариантов (разрешить или блокировать), или закроет окно, мы получим результат в виде строки, содержащий в зависимости от выбора пользователя «granted», «denied», или «default». Если пользовать выберет вариант «блокировать», то приложение больше не сможет запрашивать у него разрешение на показ уведомлений до тех пор, пока он самостоятельно в настройках не изменит решение.

Получив разрешение на показ уведомлений от пользователя необходимо оформить подписку, вызвав во время регистрации SW метод registration.pushManager.subscribe(). Сервер приложения сгенерирует уникальную пару ключей для приложения (VAPID-ключи). В эту пару входят закрытый и открытые ключи. Открытый передаётся клиенту, а закрытый ключ сервер хранит в тайне. Ключи позволяют сервису push-уведомлений знать о том, какой сервер приложения подписал пользователя, и, что тот же самый сервер отправления уведомления конкретному пользователю.

Далее браузер передаёт applicationServerKey (открытый ключ) push-сервису в ходе оформления подписки. Это означает, что push-сервис сможет связать открытый ключ приложения с подпиской, PushSubscription. Веб-приложение загружается и вызывает subscribe(), передавая серверный ключ. Браузер выполняет сетевой запрос к push-сервису, который генерирует адрес точки входа в собственное API, свяжет этот адрес с ключом и возвращает сведения о нём браузеру. Браузер добавляет эти сведения к объекту PushSubscription, который возвращается через обещание subscribe() [23].

Позже, когда необходимо отправить push-уведомление, нужно создать заголовок Authorization, который содержит информацию, подписанную закрытым серверным ключом приложения. Когда push-сервис получает запрос на отправку уведомления, он проверяет заголовок, используя открытый ключ, который уже связан с точкой входа в API.

Push-сервис – это система, которая получает запрос на отправке push-уведомлений, проверяет их и доставляет пользователю. Push-сервисы – это сторонние службы, неконтролируемые разработчиком (например, Google FCM). Сервер разработчика взаимодействует с push-сервисами через API. Браузер может использовать различные push-сервисы, однако, они имеют одинаковый API.

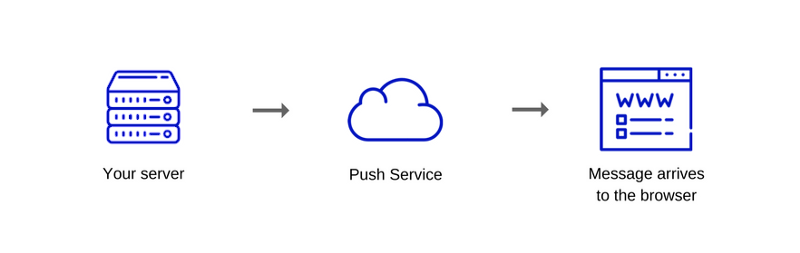


Рисунок 9. Иллюстрация получения push-уведомления пользователем

Для каждого push-уведомления задаются следующие свойства:

1. TTL — определяет срок, который недоставленное push-уведомление может провести в очереди до его удаления.
2. Priority — задаёт приоритет сообщения, что позволяет push-сервису отправлять только высокоприоритетные сообщения в том случае, если нужно экономить заряд батареи устройства пользователя.
3. Topic — назначает push-уведомлению имя темы, что приведёт к замене ожидающих доставки сообщений с той же темой. В результате, как только устройство пользователя окажется активным, пользователь получит актуальное сообщение.

Как только push-сообщение доставлено push-сервису, оно получает состояние ожидания до тех пор, пока устройство не войдёт в сеть или же срок хранения сообщения, определённый при помощи TTL-параметра, не истечёт. Получив сообщение от push-сервиса браузер расшифровывает его и вызывает событие push в зарегистрированном SW. Браузер может вызвать SW даже если соответствующая ему веб-страница закрыта. Это происходит таким образом:

1. Push-сообщение попадает в браузер, который расшифровывает его
2. Браузер активирует SW.
3. Событие push передаётся SW.

# **ЗАКЛЮЧЕНИЕ**

В ходе данной работы были изучены технологии позволяющие создать progressive web app. На основании информационной базы и ресурсов, включающих в себя как отечественные, так и зарубежные источники, были рассмотренные основные техники, которые используются для создания PWA. Были выделенные и проанализированы следующие особенности создания прогрессивного веб-приложения:

1. Прогрессивность – подход к разработке, при котором все браузеры получаются одинаковый набор функциональных решений, но в последних версиях браузеров этот набор расширяется за счёт поддерживаемых новых свойств.
2. Application shell – шаблон графического интерфейса обеспечивающий веб-приложению схожесть с native-приложением, состоит из минимального набора HTML, CSS и JS, необходимого для обеспечения нормальной работы пользовательского интерфейса.
3. Web manifest – JSON файл позволяющий установить веб-приложению на домашний экран гаджета и определяющий его поведение при запуске.
4. HTTPS – безопасный протокол передачи данных основанный на HTTP, поддерживающий шифрование посредством криптографических протоколов SSL и TLS.
5. Service worker – API отвечающий за управление событиями, контролем веб-страницы, перехват запросов, работу вне сети.
6. Push-уведомления в браузере – технология позволяющая пользователям подписываться на уведомления от веб-приложения.

Описанные в работе технологии позволяют увеличить скорость работы с сайтом и улучшить опыт его использования пользователем.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Desktop vs Mobile vs Tablet Market Share Worldwide | Statcounter GlobalStats [Электронный ресурс] / Режим доступа: http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-200901-201811 (дата обращения: 15.03.2019)
2. Коноплев Д.Э. «Закат десктопов: как мобильные платформы меняют медиа и их аудиторию» [Текст] / Д.Э. Коноплев. — Вестник Челябинского университета №5, 2015. — 142-146 с.
3. Жабин В.В. Технология адаптации web-ресурсов к мобильным устройствам [Текст] / В.В. Жабин. — Гаудемаус №2, 2012. — 173 с.
4. Леонтьев К.Б. Web-дизайн. Руководство пользователя [Текст] / К.Б. Леонтьев. — М.: Познавательная книга плюс, 2001. — 321 с.
5. American media measurement and analytics company providing marketing data and analytics to enterprises [Электронный ресурс] / Режим доступа: <https://www.comscore.com/> (дата обращения: 20.03.2019)
6. The State of Mobile in 2019 – The Most Important Trends to Know [Электронный ресурс] / Режим доступа: [www.appannie.com/ru/insights/market-data/the-state-of-mobile-2019](http://www.appannie.com/ru/insights/market-data/the-state-of-mobile-2019) (дата обращения: 20.03.2019)
7. Hume D. A. Progressive web apps. [Текст] / D.A. Hume Manning Publications Co., 2017.
8. Gustafson A. Adaptive web design: crafting rich experiences with progressive enhancement [Текст] /A. Gustafson New Riders, 2015.
9. Desruelle H., Gielen F. Context-driven progressive enhancement of mobile web applications: A multicriteria decision-making approach [Текст] / The Computer Journal. – 2015. – Т. 58. – №. 8. – С. 1732-1746.
10. Макнейл П. Веб-дизайн. Книга для разработчика. [Текст] / П. Макнейл. – СПб.:Питер, 2015. — 289 с.
11. Плохенко В. Г. Применение современных стандартной HTML5 и CSS при проектировании сайта [Текст] / В.Г. Плохенко, Е.И. Симоневич. — Международный журнал экспериментального образования. – 2016. – №. 9-2. – С. 321.
12. Лещев. Д. Создание интерактивного web-сайта: учебный курс [Текст] / Д.Лещев. – СПб.: Питер, 2003. – 540 с.
13. Gustafson A. Adaptive web design: crafting rich experiences with progressive enhancement [текст] / A. Gustafson. – New Riders, 2015.
14. De Bra P. Design issues in adaptive web-site development [Текст] /P. De Bra. — Eindhoven university of technology, 2018. — 36-37 с.
15. Маркотт И. Отзывчивый веб-дизайн [Текст] / И. Маркотт. — Манн, Иванов и Фербер, 2012
16. Kim B. Responsive web design, discoverability, and mobile challenge [Текст] / B. Kib. — Library technology reports, 2013. —29-31 с.
17. Progressive Enhancement [Электронный ресурс] / Режим доступа: <https://htmlacademy.ru/blog/7-progressive-enhancement> (дата обращения: 23.03.2019)
18. The App Shell Model [Электронный ресурс] / Режим доступа: <https://developers.google.com/web/fundamentals/architecture/app-shell> (дата обращения: 23.03.2019)
19. Durumeric Z. et al. Analysis of the HTTPS certificate ecosystem [Текст] / Z. Durumeric — Proceedings of the 2013 conference on Internet measurement conference. – ACM, 2013. – С. 291-304.
20. Namiot D., Sukhomlin V.JavaScript concurrency models [Текст] /D. Namiot, V.Sukhomlin — International Journal of Open Information Technologies. 2015. №6.
21. Поддержка технологии Service Worker [Электронный ресурс] / Режим доступа: <https://caniuse.com/#feat=serviceworkers> (дата обращения: 23.03.2019)
22. Mulligan M., Nykänen P., Toijala J. Web services push gateway [Текст] – M.Milligan, P. Nykänen, J. Toijala 2007.
23. Lynes M. et al. Mobile push notification [Текст] – M. Lynes 2016.
24. Проникновение интернета в России: итоги 2018 года [Электронный ресурс] / Режим доступа: https://www.gfk.com/fileadmin/user\_upload/dyna\_content/RU/Documents/Press\_Releases/2019/GfK\_Rus\_Internet\_Audience\_in\_Russia\_2018.pdf (дата обращения: 26.03.2019)