**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

РАЗРАБОТКА ПРОГРЕССИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ

Курсовая работа

Петрова Андрея Александровича

студента 3 курса,

специальность «прикладная информатика»

Научный руководитель:

старший преподаватель

Карпович Н.А.

Минск

2022

**РЕФЕРАТ**

Курсовая работа, 26 c., 10 рис., 1 таблица.

**Ключевые слова**: PWA, прогрессивное веб-приложение, веб-приложение, нативное приложение.

**Объект исследования** — прогрессивные веб-приложения и использование их возможностей для создания приложений, не зависящих от конкретной платформы. В качестве предмета исследования выбираем разработку прогрессивного веб-приложения.

**Цели работы** — дать определению прогрессивному веб-приложению, рассмотреть его возможности и отличия от иных приложений, принципы его создания, а также разработать веб-приложение, которое будет являться прогрессивным.

**Методы исследования** — а) теоретические: изучение литературы, посвященной проблеме создания нативных, PWA и веб-приложений, их возможностям; б) практические: обобщение опыта разработки и проектирования прогрессивного веб-приложения «schedule.manager».

Результатами являются — прогрессивное веб-приложение «schedule.manager».

**Область применения** — проектирование и разработка прогрессивных веб-приложений в современной индустрии разработки нативных и веб-приложений.

**РЕФЕРАТ**

Курсавая праца, 26 C., 10 мал., 1 табліца.

**Ключавыя словы**: PWA, прагрэсіўнае вэб-прыкладанне, вэб- прыкладанне, натыўнае прыкладанне.

**Аб'ект даследавання** — прагрэсіўныя вэб-прыкладанні і выкарыстанне іх магчымасцяў для стварэння прыкладанняў, якія не залежаць ад канкрэтнай платформы. У якасці прадмета даследавання выбіраем распрацоўку прагрэсіўнага вэб-прыкладання.

**Мэты працы** — даць вызначэнню прагрэсіўнаму вэб-прыкладанню, разгледзець яго магчымасці і адрозненні ад іншых прыкладанняў, прынцыпы яго стварэння, а таксама распрацаваць вэб-прыкладанне, якое будзе з'яўляцца прагрэсіўным.

**Метады даследавання** — а) тэарэтычныя: вывучэнне літаратуры, прысвечанай праблеме стварэння натыўных, PWA і вэб-прыкладанняў, іх магчымасцям; б) практычныя: абагульненне вопыту распрацоўкі і праектавання прагрэсіўнага вэб-прыкладання «schedule.manager».

Вынікамі з'яўляюцца — прагрэсіўнае вэб-прыкладанне «schedule.manager».

**Вобласць прымянення** — праектаванне і распрацоўка прагрэсіўных вэб-прыкладанняў у сучаснай індустрыі распрацоўкі натыўных і вэб-прыкладанняў.

**ESSAY**

Course work, 26 p., 10 illustrations, 1 table.

**Keywords**: PWA, progressive web application, web application, native application.

**Object of research** — progressive web applications and the use of their capabilities to create applications that do not depend on a specific platform. As a subject of research, we choose the development of a progressive web application.

**Purpose** — to define a progressive web application, to consider its capabilities and differences from other applications, the principles of its creation, as well as to develop a web application that will be progressive.

**Methods of research** — a) theoretical: study of literature devoted to the problem of creating native, PWA and web applications, their capabilities; b) practical: generalization of experience in the development and design of a progressive web application "schedule.manager".

The results are — a progressive web application "schedule.manager".

**Scope** — design and development of progressive web applications in the modern industry of native and web application development.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 7](#_Toc103861641)

[ГЛАВА 1. ВВЕДЕНИЕ В ПРОГРЕССИВНЫЕ ВЕБ-ПРИЛОЖЕНИЯМИ 9](#_Toc103861642)

[ГЛАВА 2. СРАВНЕНИТЕЛЬНЫЙ АНАЛИЗ ПРОГРЕССИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ И НАТИВНЫХ 12](#_Toc103861643)

[Раздел 2.1. Преимущества и недостатки PWAs в сравнении с нативными приложениями. 13](#_Toc103861644)

[Раздел 2.2. Использование прогрессивных веб-приложений в коммерческой разработке 18](#_Toc103861645)

[ГЛАВА 3. СОЗДАНИЕ ПРОГРЕССИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ 21](#_Toc103861646)

[Раздел 3.1. Возможности manifest.json 21](#_Toc103861647)

[Раздел 3.2. Использование Service workers 23](#_Toc103861648)

[Раздел 3.3. Возможность установки прогрессивных веб-приложений на устройства пользователей 27](#_Toc103861649)

[Раздел 3.4. Обновление прогрессивных веб-приложений на устройствах пользователей 29](#_Toc103861650)

[ЗАКЛЮЧЕНИЕ 32](#_Toc103861651)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 34](#_Toc103861652)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

PWA прогрессивное веб-приложение (Progressive web application)

HTA HTML application

# **ВВЕДЕНИЕ**

ИТ-индустрия прямо сейчас претерпевает существенные изменения. Благодаря новым инструментам создания программного обеспечения невозможно себе представить бизнес, работающий исключительно в офлайн. Лет десять назад веб рассматривался лишь как инструмент для обмена информацией. Никто и не думал, что первоначальные элементы сети станут базисом для приложений, которые будут работать на всех типах устройств и экранах.

Сегодня веб – это важнейшее средство предоставления простых, понятных и удобных интерфейсов для потребительских и корпоративных приложений. Он настолько распространен, что подавляющее большинство приложений, изначально создававшихся для мобильных и настольных платформ, сейчас разрабатываются с помощью веб-технологий, что требует от разработчиков постоянного наблюдения за развитием веб-технологий, иначе сделать свои проекты успешными просто не получится. Так неотъемлемой частью современной веб-разработки являются прогрессивные веб-приложения (далее – PWA).

Представьте, что ваш сайт взаимодействует с пользователем как приложение. Пользователь может установить его на любой гаджет, получать уведомления и работать с ним даже без интернет-соединения. Это и есть технология PWA, которую активно продвигают крупнейшие компании такие как Google, Mozilla, Microsoft и Apple.

Объектом исследования данной работы является технология PWA, разработка приложений с её использованием, а также влияние этой технологии на современную индустрию веб-разработки.

В данной работе будет рассмотрена тема создания PWA, его особенности и отличия от нативных приложений. Для демонстрации возможноей PWA Будет разработано веб-приложение.

Методы исследования, которые применялись в работе, включают в себя изучение литературы, посвященной особенностям технологии PWA, проблемам проектирования, реализации, установки и обновления современных веб-приложений, разработанных с помощью данной технологии; практическое применение полученных знаний на примере разработка простого PWA, а также изучение статистики разработки и публикации PWAs.

В главе 1 дано определение прогрессивного веб-приложения, рассмотрены его ключевые принципы и краткие исторические сведения.

В главе 2 дано определение нативного приложения, рассмотрены отличия PWA от нативных и веб-приложений, рассмотрены преимущества PWA для бизнеса.

В главе 3 рассмотрены основные этапы создания и обновления прогрессивных веб-приложений, а также предоставлена практическая реализация внедрения технологии PWA в веб-приложение.

# **ГЛАВА 1. ВВЕДЕНИЕ В ПРОГРЕССИВНЫЕ ВЕБ-ПРИЛОЖЕНИЯМИ**

Progressive web applications (PWAs) – это веб-приложения, разработанные с использованием ряда конкретных технологий, таких как web API, service workers, манифесты веб-приложений и стандартных шаблонов, позволяющих им использовать преимущества как веб-функций, поддерживаемых современными браузерами, так и функций нативных приложений (иногда их называют как платформо-зависимые). Такие приложения запускаются на любом устройство, имеющем браузер или приложение со встроенным web view. Их не обязательно устанавливать, чтобы воспользоваться функционалом. При этом PWAs обладают рядом характеристик, обеспечивающих пользователей новым опытом взаимодействия с контентом, позволяя им обновлять веб-приложения до прогрессивных веб-приложений независимо от операционной системы их устройства.

Стоит отметить, что PWA не создаются с помощью одной технологии. Они представляют лишь новую философию (концепцию) создания веб-приложений, включающую определенные шаблоны, API и другие функции. На первый взгляд не так очевидно, является ли веб-приложение PWA или нет. Приложение можно рассматривать как PWA, если оно соответствует определенным требованиям или реализует набор заданных функций: работает в автономном режиме, устанавливается, легко синхронизируется, может отправлять push-уведомления и т.д.

Есть несколько ключевых принципов, которые веб-приложение должно стараться соблюдать, чтобы быть идентифицированным как PWA [3]. Так веб-приложение должно быть:

* *Обнаружаемо (discoverable).* Содержимое веб-приложения можно найти с помощью поисковых систем.
* *Доступно посредством ссылок (linkable).* Приложением можно поделиться, отправив URL-адрес.
* *Автономно (network independent).* Приложением может работать в автономном режиме или с плохим сетевым подключением.
* *Прогрессивно (progressively enhanced).* Веб-приложение может быть по-прежнему доступно с базовым функционалом на старых браузерах, но полностью функционально в новейших.
* *Повторно вовлекаемо (re-engageable).* Веб-приложение может отправлять push-уведомления на устройство для вовлечения пользователя в приложение.
* *Безопасно (secure).* Соединения между пользователем, приложением и сервером защищены от любых третьих сторон, пытающихся получить доступ к конфиденциальным данным.
* *Иметь возможность установки (installable*). Веб-приложение может быть доступно на главном экране устройства или в программе запуска приложений.
* *Иметь отзывчивый дизайн (responsively designed).* Веб-приложение можно использовать на любом устройстве с экраном и браузером.

PWA не новая концепция – в прошлом данная идея неоднократно пересматривались с использованием различных подходов. Прогрессивные улучшения и адаптивный дизайн уже позволяют создавать сайты, оптимизированные для мобильных устройств. Однако PWA предоставляют все это и многое другое, не теряя при этом ни одной из существующих функций.

Так прародителем технологии PWA можно назвать технологию HTML Application (HTA), представленную корпорацией Microsoft в 2000 году. HTA представляет собой приложение Microsoft Windows, являющееся документом HTML, отображаемым в отдельном окне без элементов интерфейса обозревателя таких как строка меню, строка адреса, панель инструментов с помощью движка Microsoft Internet Explorer. На HTA не распространяются большинство ограничений безопасности Internet Explorer, в частности на использование небезопасных элементов ActiveX. Например, HTA может создавать, изменять, удалять файлы и записи системного реестра Windows. HTA поддерживают исполнение сценариев, поэтому они могут являться носителем вредоносного кода, что не позволило данной технологии снискать популярность.

Спустя 7 лет Apple представила свой вариант HTML-приложений на конференции для разработчиков WWDC. iOS 1.0 не поддерживала сторонние приложения и приложения для оригинального iPhone разрабатывались именно на платформе web и были доступны в Safari — браузере для macOS и iOS. Apple старалась продвигать веб-приложения, но в 2007 году HTML-приложения не были готовы для загрузки на платформу из-за малого функционала и недоработок. Спустя год во второй версии операционной системы появился магазин приложений App Store, который не входил в первоначальную дорожную карту, что ознаменовало активное развитие нативных приложений, отложившее развитие технологии PWA.

Широкую известность технология PWA приобрела только в 2015 году благодаря расширению возможностей браузера Google Chrome и продвижению Service Worker и Web App Manifest. В это же дизайнер Фрэнсис Берримэн и инженер Google Chrome Алекс Рассел ввели термин PWA. Позже в марте 2018 года произошли существенные изменения в области реализации web-стандартов. Так Apple добавила поддержку Service Worker в мобильную и десктопную версию Safari, а уже через месяц Microsoft добавила поддержку Service Worker в свой браузер Microsoft Edge и возможность распространения PWA через Microsoft Store.

В настоящее время офлайн работу с PWAs могут обеспечить iOS, Android, Windows, Linux, macOS и Chrome OS в браузерах Chrome, Safari, Firefox, Edge и Samsung Internet.

# **ГЛАВА 2. СРАВНЕНИТЕЛЬНЫЙ АНАЛИЗ ПРОГРЕССИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ И НАТИВНЫХ**

В настоящее время разнообразие операционных систем на компьютерах и смартфонах привело к поразительному увеличению количества как мобильных, так и десктопных приложений. Одним из типов приложений, который приобрел огромную популярность среди пользователей, является платформо-зависимое приложени, или как его еще называют нативное приложение (native application)

Так что такое нативное приложение? В своей статье «What is native app? Definition, tips and best example» для интернет-ресурса Magenest автор Le Pham дает определение нативному приложению. Так по ее мнению нативное приложение – это приложение, устрановленное на устройство пользователей, и к которому они могут получить доступ через главный экран. Уточняя, что обычно нативные приложения приобретаются через App Store и Google Play [8].

Вместе с тем автор многих технических статей для портала TechTarget Alexander S. Gillis в статье «native app» указывает, что термин «нативное приложение» в контексте мобильных приложений используется для обозначения любого приложения, написанного для работы на определенной платформе устройства [9].

Можно сделать вывод, что нативное, или платформо-зависимое приложение – это приложение, разработанное для работы на определенной платформе. Из этого вытекает одно из существенных различий между нативным приложением и веб-приложением – языки программирования. Хотя веб-приложения написаны в основном на Javascript, нативные приложения используют разные языки в зависимости от того, на какой платформе они созданы. Например, Java или Kotlin используется для нативных приложений Android, Swift или Objective-C – для нативных приложений iOS, MacOS, а C#, C++ – для большинства приложений Windows.

Как упоминалось ранее, PWAs являются разновидностью веб-приложений. Всякий раз, посещая веб-сайт, его контент, как правило, находится в актуальном состоянии. Опыт работы с этим сайтом может быть как эфемерным, так и постоянным. Он может быть доступен кому угодно и где угодно, на любом устройстве с помощью единой базы кода. Однако нативные приложения известны своей невероятной функциональностью и надежностью. Они всегда присутствуют на домашних экранах, в доках и на панелях задач и работают независимо от сетевого подключения, запускаясь в собственном автономном режиме. Они могут читать и записывать файлы из локальной файловой системы, получать доступ к оборудованию, подключенному через USB, последовательный порт или Bluetooth, а также взаимодействовать с данными, хранящимися на устройстве пользователя, такими как контакты и события календаря. Кроме того, у нативных приложений есть другие возможности, такие как возможность использовать жесты на мобильных устройствах (стандартные жесты операционной системы или собственные жесты, определяемые приложением), использовать систему уведомлений устройств, делать снимки, просматривать воспроизведение музыкальных композиций или управлять ими в другом приложении. В связи с чем нативные приложения ощущаются как часть устройства, на котором они работают.

Если рассматривать нативные приложения и веб-приложениях с точки зрения возможностей и охвата, первые представляют собой лучшие возможности, а веб-приложения – наиболее доступные.

## **Раздел 2.1. Преимущества и недостатки PWAs в сравнении с нативными приложениями.**

PWA используют современные API-интерфейсы, чтобы обеспечить расширенный функционал, сравнимый с нативными приложения, надежность и возможность установки приложения на устройства пользователей, обеспечивая доступ к контенту и возможностям, в любом месте и на любом устройстве с помощью единой кодовой базы. Концепция PWA была разработана опираясь на три стопа: функциональность, надежность и устанавливаемость, чтобы превратить опыт использования PWA в опыт, подобному при использовании нативных приложений.

Так для ресурса “web.dev”, содержащему рекомендации по современной веб-разработке от компании Google, разработчики Sam Richard и Pete LePage, основываяь на статистике и возможностях PWAs делают небольшой вывод, что прогрессивные веб-приложения являются «лучшим из обоих миров» в контексте нативных и веб-приложений, выделяя следующие отличительные черты PWAs [10]:

* Надёжность
* Устанавливаемость
* Использование возможностей обоих платформ.

Рассмотрим каждый из этих пунктов детальнее. Описывая возможности прогрессивных веб-приложений авторы Sam Richard и Pete LePage в той же статье приводят в пример возможности создания приложения для гиперлокального видеочата, используя WebRTC, геолокацию и push-уведомления. Благодаря возможностям PWA это приложение можно сделать устанавливаемым на любое устройство, имеющего веб-браузер, а разговоры с помощью него виртуальными, использую технологии WebGL и WebVR. Отмечается, что с внедрением технологии WebAssembly разработчики могут подключиться к другим экосистемам, таким как C, C++ и Rust, и привнести в веб десятилетия работы и возможностей нативных приложений [10].

Действительно, объединяя в себе основу веб-приложений, новые веб-стандарты, такие как WaSP, W3C и др., и API интерфейсы, PWA способны претендовать на возможности, которые до недавного времени были доступны только для приложений, разработанных для конкретной платформы. Постепенно PWAs будут обрастать все большим и большим функционалом. Однако уже сейчас, с помощью API можно использовать такие функции, как доступ к файловой системе, управлением мультимедиа и полная поддержка буфера обмена.

Как отмечалось ранее, PWA способны запускаться с домашнего экрана, док-станции или панели задач пользователя в отдельном окне вместо вкладки браузера. Их можно искать на устройстве в списке установленных программ и переключаться между ними с помощью многозадачности, заставляя их чувствовать себя частью устройства, на котором они установлены. Все эти возможности, а также возможности, зарезервированные при работе в браузере, например, сочетания клавиш, открываются после установки веб-приложения на устройство.

Сравнивая нативные приложения и PWA, стоит отметить, что первые практически не существуют вне магазина приложений. Это создает как положительные стороны, так и недостатки для обоих платформ. Так, в случае блокировки нативного приложения в магазине приложений, его распространение может затрудниться или вовсе остановиться. PWA же не имеют такой проблемы, так как они, по сути, создаются на основе сайтов и веб-приложений, что позволяет им «жить» вне магазинов и распространяться посредством ссылок, что в свою очередь создает сложности: более сложную процедуру публикации в магазинах приложений.

Рассматривая надёжность с точки зрения пользователя, мы смотрим на то, насколько стабильно и быстро работает веб-приложение и насколько оно зависит от скорости работы или отсутствия сети. Производительность прямо влияет на то, как пользователи воспринимают приложение. Прокрутка и анимация должны казаться плавными, а элементы UI/UX должны быть красивыми и отзывчивыми, однако PWA ограничены в разработке пользовательского интерфейса лишь средствами веб-технологий, когда нативные приложения не имеют такого ограничения.

Приложение должно работать независимо от сетевого соединения, так как пользователи ожидают, что они взаимодействуют с актуальным контентом, а если получение актуального контента не возможно, они ожидают получить об этом сообщение. До недавнего времени работа в автономном режиме была доступа только нативным приложениям, однако с помощью базы данных IndexedDB и service worker, эту возможность можно реализовать для PWAs, позволяя предоставить пользователям отличное взаимодействие с контентом.

Несмотря на все свои преимущества и положительные стороны PWA, как и какая-либо иная технология, имеет свои ограничения. В частности, многие разработчики и исследователи данной темы, отмечают, что главным ограничением PWA приложений является их поддержка в старых версиях популярных браузеров и ограничение возможностей на разных платформах.

Так автор статьи «All You Need to Know About Progressive Web App to Decide if Your Business Needs One» Kateryna Khalimonchuk приводит сравнительную таблицу доступных возможностей PWA для каждой платформы устройств с указанием поддерживаемых браузеров и их долей рынка. Сверив и добавив актуальные сведения о поддержке функциональных возможностей по состоянию на первую половину 2022 года с помощью ресурса caniuse, рассмотрим данную таблицу [Таблица 2.1].

Можно заметить, что android платформа имеет наиболее полную поддержку функциональных возможностей PWAs в сравнении с другими платформами, в связи с чем длительное время существовало ошибочное мнение, что PWA – технология, доступная лишь для android платформы. На что были причины, ведь отсутствие отсутствие push-уведомлений (на iOS платформе), небольшой размер хранения данных для автономных приложений, который не может превышать 50МБ, ограничивают объем функциональности на iOS и Mac платформах и могут наложить определенные ограничения на внедрение прогрессивного веб-приложения для бизнеса.

Так, касаемо PushApi, с помощью которого реализуется возможность отправки push-уведомлений в PWAs приложениях, на момент написания данной работы не имеется никакой поддержки на iOS платформе (Рисунок 2.1.1)

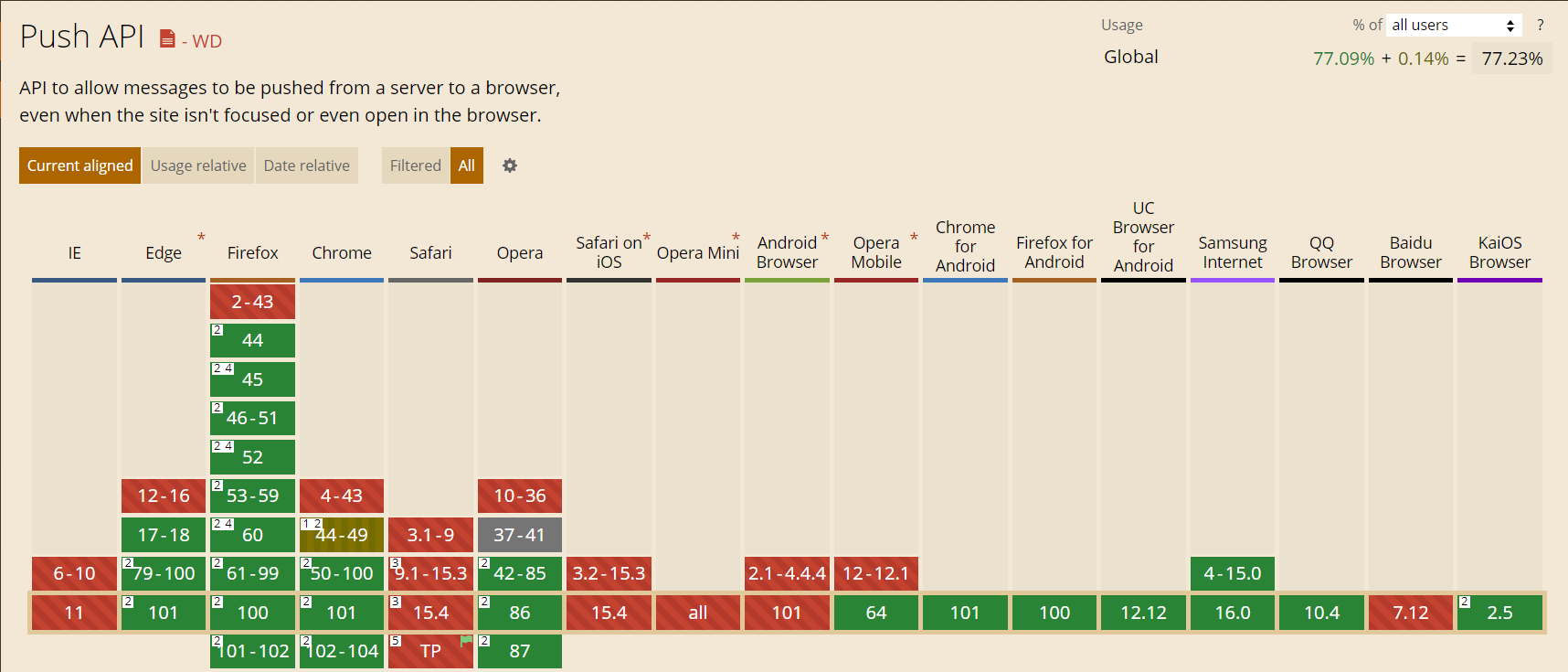


Рисунок 2.1.1 Сведения из ресурса caniuse о поддержке технологии PushApi популярными браузерами

Хоть нативные оболочки (WebView) и позволяют решить большинство проблем, стоит все же отметить, что нативные приложение, имея полный набор SDK, могут использовать доступ ко всем контроллерам и нативным утилитам платформы, когда функциональность PWA зависит от платформы и браузера. Одновременно с этим, стоит отметить, что компания Google активно продвигает и развивает технологию PWA и её поддержку в свои продукты и сервисы. Google Chrome уже сейчас позволяет беспрепятственно установить PWA и поддерживает необходимые возможности для обеспечения хорошего пользовательского опыта с приложением на всех платформах.

Подводя итог вышесказанному, можно определить следующие положительные и отрицательные стороны PWAs. К положительным сторонам можно отнести такие особенности как:

- опыт работы в браузере, похожий на нативное приложение;

- производительность;

- безопасность и надёжность;

- низкая стоимость разработки;

- распространение.

Как и любая технология, PWA имеет свои недостатки. В сравнении с нативными приложениями можно выделить важные и в случае разработки некоторых продуктов, даже критичные, недостатки:

- зависимость функционала приложения от конкретной платформы и браузера;

- возможности пользовательского интерфейса ограничены функциональ-ностью веб-приложений;

- более сложный процесс публикации в магазины приложений.

Несмотря на выявленные недостатки, можно уверенностью сказать, что PWA в ближайшее время не собираются сдавать позиции и способны занять часть доли рынка, принадлежащего некоторым технологиям, возволяющим разрабатывать кросплатформенные приложения, например, Electron.js

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Платформа** | **Android** | **iOS** | **Windows** | **Mac** |
| **Доля рынка браузеров в расчете на OS** | Chrome (87%), Samsung browser (6%), other (7%) | Safari (92%), Chrome (6%), Other (2%) | Chrome (69%), Edge (8%), Firefox (7%), IE (7%), Other (9%) | Chrome(53%), Safari(40%), Firefox(5%), Other(2%) |
| **Общая поддержка** | 99% используемых версий браузеров | 98% используемых версий браузеров | 97% используемых версий браузеров | 99% используемых версий браузеров |
| **Engagement** | Нативные push-уведомления;  In-app уведомления, Email, Messenger bots, SMS | Нет push-уведомлений;  In-app уведомления, Email, Messenger bots, SMS | Push-уведомления (Chrome, Edge) | Push-уведомления (Chrome, Safari) |
| **Иконки приложения** | Есть поддержка | Есть поддержка | Есть поддержка | Есть поддержка |
| **Полноэкранный режим** | Есть поддержка | Есть поддержка | Есть поддержка | Есть поддержка |
| **Установка** | Нативные установочные баннеры и всплывающие окна Google Play Store | Кнопка добавить на главный экран | Microsoft Store;  установочные баннеры и всплывающие окна в Google Chrome, Edge | Баннер автоматической установки в Chrome;  В Safari отсутствуют окна на установку. In-app баннеры/кнопки/ ссылки |
| **Локальное хранилище** | 33% дискового пространства (Chrome, Firefox) | 50MB кэш-памяти (2GB IndexedDB) | +-33% дискового пространства (Chrome, Firefox), максимум 20GB (Edge) | 50MB кэш-памяти (2GB IndexedDB) |
| **Нативные возможности** | GPS, доступ к файлам, Bluetooth, USB, камера, микрофон, ориентация экрана, буфер обмена, расширенные элементы управления камерой, вибрация, магнитомер, акселерометр, гироскоп | GPS, доступ к файлам, камера, микрофон, буфер обмена | Геолокация (ip), доступ к файлам, камера, микрофон, буфер обмена | Геолокация (ip), доступ к файлам, камера, микрофон, буфер обмена |
| **Вход и оплата** | Android login, Android pay, автозаполнение в Chrome, API оплаты, менеджеры паролей | Apple login, Apple pay, автозаполнение в Safari | Автозаполнение в браузере, API оплаты | Автозаполнение в браузере |

Таблица 2.1. Сравнение доступных возможностей PWA для каждой платформы с указанием поддерживаемых браузеров и их долей рынка.

## **Раздел 2.2. Использование прогрессивных веб-приложений в коммерческой разработке**

Хотя прогрессивное веб-приложение довольно новое явление, все большее число компаний уже извлекают выгоду из его обширных возможностей. PWA сочетают в себе возможности веб-сайтов и мобильного программного обеспечения, что позволяет создать иммерсивный пользовательский интерфейс и повысить вовлеченность пользователей и коэффициент конверсии.

Приложения на базе PWA пользователи используют чаще, чем кажется на первый взгляд. Такие бренды как Twitter, Tinder, Uber, Printerest, Telegram, Starbucks, Forbes, AliExpress, Aviasales используют приложения на базе PWA как основное или в дополнение к мобильному приложению.

Исследования показывают, что PWA предоставляют компаниям множество преимуществ, таких как [12]:

* 68% рост мобильного трафика;
* 15-кратное увеличение скорости загрузки и установки;
* Сокращение использования памяти устройства в 25 раз;
* Среднее увеличение конверсии на 52%;
* 78% увеличение среднего сеанса;
* Увеличение вовлеченности на 137%;
* Показатель отказов на 42,86% ниже, чем у мобильных сайтов.
* На 133,67% увеличилось количество просмотров страниц.

Рассматривая конкретные случаи Kateryna Khalimonchuk в статье «All you need to know about Progressive web app to decide if your business needs one», приводит некоторую статистику по приложениям, которые имеют нативные версии и PWA приложения.

Printerest – чрезвычайно популярная платформа для обмена изображениями в социальных сетях. Внедрение PWA версии приложения позволило Printerest расширить свою пользовательскую базу на международном уровне. Результаты очень впечатляющие. Старый мобильный сайт был размером в 650 КБ, приложение для android – 9.6МБ, для iOS – 56МБ. PWA составляет всего 150 КБ! Скорость загрузки резко увеличилась с 28 секунд до 5,6. Рост вовлеченности увеличился на 60%, увеличение времени, затрачиваемого на приложение – на 40%, а доход от рекламы – на 44%.

Twitter был одним из первых PWA. Созданный в 2017 году Twitter Lite PWA был предназначен для повышения вовлеченности пользователей, которые проверяют свой Twitter в среднем 4 раза в день. Официальная статистика Twitter следующая: 65% рост количества страниц за сеанс, 75% рост отправленных твитов, снижение показателя отказов на 20% [11].

Что означают приведенные выше цифры для эффективности бизнеса?

PWA способны максимально упростить использование сервисов или приложений пользователями, следовательно, улучшить критически важные показатели: вовлеченность, конверсию и удержание. Стоит отметить, что это также области, в которых нативные приложения испытывают наибольшие трудности.

В статье «Do You Need A Progressive Web App? PWA Benefits» Patrycja Paterska выделяет некоторые случаи, когда стоит рассматривать PWA вместо нативных приложений:

1. Если вы хотите охватить малоактивных пользователей.
2. Вам нужно снизить входной барьер и сократить воронку продаж.
3. Если ваше приложение является B2C решением и производительность является одним из ключевых показателей эффективности вашего приложения.
4. Если ваше приложение использует аппаратные функции, поддерживаемые PWA [13].

Одна из основных причин, по которой пользователи удаляют приложение (или не хотят устанавливать его вообще), – это ограниченный размер хранилища устройства. Как указывалось ранее, в среднем PWA занимают в 25 раз меньше места на устройстве по сравнению с нативными приложениями. Более того нативные приложения зависят от операционной системы устройства пользователя, а прогрессивные веб-приложения полагаются на Сеть Интернет. PWA может значительно расширить охват и привлечь пользователей с гораздо меньшими затратами, которые не захотят устанавливать мобильное приложение.

Производительность веб-сайтов и приложений способствует удовлетворению запросов клиентов. Это заметно в секторе «Бизнес для клиента» и недавно было задокументировано компанией «Делойт». Их недавнее исследование доказывает, что увеличение скорости даже на 0,1 секунды повышает узнаваемость бренда и приводит к увеличению количества потенциальных клиентов до 10%. Почему это важно? Поскольку PWA являются веб-приложениями, они используют кэш для увеличения скорости работы. PWA Forbes загружается на мобильном устройстве за 2,5 секунды по сравнению с 6,5 секундами в предыдущей версии сайта. В Сети Интернет эти цифры означают гораздо больше, чем где-либо еще: сократив время загрузки, они увеличили количество показов за сеанс на 10 процентов. Если производительность – один из ключевых показателей эффективности приложения, стоит подумать о PWA.

PWA не требуют разных версий для разных устройств; одно прогрессивное приложение отвечает требованиям всех конечных устройств, на которых оно работает. Таким образом, значительно сокращается объем усилий, которые прилагают разработчики, и, как следствие, уменьшается стоимость создания PWA. Если посмотреть стоимость разработки PWA и нативных приложений популярными студиями, можно заметить, что разработка PWA обходится в 2 и более раза дешевле.

Таким образом, компании могут многое выиграть от внедрения прогрессивных веб-приложений: увеличить рост мобильного трафика и скорость загрузки приложения, сократить использование памяти устройства, повысить уровень вовлеченности пользователей. Уже сейчас разработка PWA обходится в несколько раз дешевле разработки нативных приложений, а с учетом того, что крупные игроки, такие как Google и Microsoft, уделяют все большее и большее внимание поддержке PWA, меняя рынок разработки приложений и создавая инструменты для внедрения PWA, стоимость разработки может снизиться еще сильнее.

# **ГЛАВА 3. СОЗДАНИЕ ПРОГРЕССИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ**

В течение последних нескольких лет веб-платформа продолжала развиваться, сокращая разрыв между веб-приложениями и нативными и позволяя разработчикам предоставлять пользователям более широкие возможности и новые способы оставляться пользователей вовлеченными.

Создать прогрессивное веб-приложение довольно просто. Для начала нужно разработать веб-сайт с отзывчивым дизайном, а после понадобятся только добавить manifest.json и service-worker.

## **Раздел 3.1. Возможности manifest.json**

Манифест веб-приложения – это файл формата JSON, который предоставляет разработчикам централизованное место для размещения метаданных, связанных с веб-приложением. Эти метаданные включают, помимо прочего, имя веб-приложения, ссылки на значки, а также предпочтительный URL-адрес, который нужно открывать, когда пользователь запускает веб-приложение. Манифест также позволяет разработчикам объявлять ориентацию экрана по умолчанию для своего веб-приложения, а также предоставляет возможность устанавливать режим отображения для приложения (например, в полноэкранном режиме).

Так w3.org предоставляет следующую типовую структуру manifest.json (Рисукнок 3.2)[14]



Рисунок 3.2. Типовая структура manifest.json

Рассмотрим основные свойства манифеста:

* name – имя, используемое в запросе на установку приложения.
* short\_name – короткое имя, используемое на домашнем экране пользователя, в программе запуска или в других местах с ограничением места.
* start\_url – сообщает браузеру стартовый URL-адрес приложения.
* display – предоставляет возможность настройки отображения UI браузера при запуске приложения. Наиболее часто используемое значение – «standalone». Оно открывает веб-приложение, которое выглядит как автономное нативное приложение.
* background\_color – цвет, который используется на заставке при запуске приложения.
* theme\_color – устанавливает цвет панели инструментов.
* orientation – усиливает определенную ориентацию экрана.
* scope – определяет набор URL-адресов, находящихся в приложении, и обычно используется для определения момента выхода пользователя из приложения.
* icons – когда пользователь добавляет сайт на домашний экран, можно определить набор используемых браузером изображений.

Таким образом, определив базовые возможности manifest.json можно добавить поддержку PWA и его ключевую возможность – «установку» веб-приложения. Установка веб-приложений — это добавление «закладки» на домашний экран или в программу запуска приложений. Есть некоторые довольно очевидные вещи, которые необходимо предоставить браузеру, чтобы тот мог считать сайт приложением: название, иконки, и т.д.

Спецификация манифеста предлагает стандартный способ добавления manifest.json в веб-сайт. Необходимо сослаться на файл манифеста в HTML-странице следующим образом: <link rel="manifest" href="/manifest.json"> [14].

Стоит отметить, что не все поля manifest.json являются обязательными. Для того, чтобы определить самый простой манифест необходимо определить имя приложения (name) и одну или нескольких иконок (Рисунок 3.3)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.3. Пример простого манифеста.

Как и другие веб-ресурсы, манифест веб-приложения должен быть доступен для любого веб-браузера или поискового робота. Если разработчик веб-приложения хочет известить поисковых роботов о запрете на сканирование файла, он может сделать это включив манифест веб-приложения в файл robots.txt или использовать HTTP-заголовок X-Robots-Tag.

Определим манифест для нашего приложения, указав имя, иконки, стартовый адрес, цвета панели инструментов и заставки, а также режим отображения UI (Рисунок 3.4).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.3. Пример простого манифеста.

## **Раздел 3.2. Использование Service workers**

Одной из важнейших проблем, от которой страдали пользователи веб-приложений, была работа в условиях потери связи: отсутствие хорошего механизма для управления кешем ресурсов и настраиваемыми сетевыми запросами.

Service worker один из типов web worker. По сути, это файл JavaScript, который запускается отдельно от основного потока браузера, перехватывает сетевые запросы, кэширует или извлекает ресурсы из кэша и доставляет push-сообщения. Используя service worker, можно без труда получить приложение, использующее в первую очередь кэшированные ресурсы, указывая тем самым поведение по умолчанию в автономном режиме, до того, как будут получены данные по сети. Так обычно работают нативные приложения, что часто является причиной выбора пользователя в их пользу.

Так MDN в документации по использованию service worker определяет основные шаги, которые необходимо сделать для базовой настройки service worker [15]:

1. URL service worker опрашивается и регистрируется посредством вызова метода ServiceWorkerContainer.register().
2. Если регистрация прошла успешно, то сервис-воркер начинает работать внутри ServiceWorkerGlobalScope (en-US); это, по сути, особый вид контекста воркера, работающий вне главного потока браузера, без доступа к DOM.
3. Теперь service worker может обрабатывать события.
4. Установка service worker начинается после того, как все контролируемые им страницы кэшированы и доступны для последующего использования. Событие install всегда посылается первым воркеру (оно может быть использовано для запуска начальной загрузки данных в IndexedDB, для кеширования ресурсов). Данный этап сродни процедуре установки нативного или FirefoxOS-приложения — все делается доступным для использования в офлайн-режиме.
5. Как только обработчик события oninstall завершит свою работу, сервис-воркер считается установленным.
6. Далее следует активация. После того как воркер установлен, он получает событие onactivate, которое обычно используется для очистки ресурсов, задействованных в предыдущей версии скрипта service worker.
7. Теперь service worker может контролировать страницы, но только в случае, если те открыты после успешного вызова register(). То есть документ может начать жизнь с service worker или даже без него и продолжать нормально работать. Поэтому документы должны быть перезагружены, чтобы действительно быть подконтрольными service worker.

Следовательно жизненный цикл service worker можно определить следующим образом:

* **Регистрация** – сообщает браузеру местонахождение воркера и начинает установку в фоновом режиме.
* **Установка** – запускает событие установки, в котором можно выполнять некоторые задачи при установке service worker.
* **Активация** – при наличии открытых страниц, контролируемых предыдущим service worker, новый service worker переходит в состояние ожидания и активируется только при отсутствии загружаемых страниц, которые все еще использует старый.

Для того, чтобы использовать service worker в приложении, для начала нужно его зарегистрировать. Пример регистрации service worker продемонстрирован на Рисунке 3.4

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.4. Пример регистрации service worker

В данном примере мы проверяем, допускает ли браузер работу воркера, и, если да, то регистрируем воркер в app.js/ main.js.

Одна из возможностей service worker является статический кэш. Он необходим, если нужно загрузить все ресурсы веб-сайта и кэшировать их при первом переходе на страницу. Для этого используется новый API хранилища service worker – cache – глобальный для всех service workers, который позволяет хранить результаты запросов, используя в качестве ключа для их получения сами запросы. Этот API работает аналогично стандартному кэшу браузера, но только для конкретного домена. Данные в кэше сохраняются до тех пор, пока мы их не захочем удалить.

Начнем с реализации статического кэша: укажем вручную, какие ресурсы помещать в кэш, например изображения, файлы CSS и js. Простой пример реализации статического кэша продемонстрирован на Рисунке 3.5.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.5. Пример реализации статического кэша.

Здесь мы добавляем обработчик события install к service worker (ссылаемся на него с помощью self), и затем вызываем метод ExtendableEvent.waitUntil() объекта события. Такая конструкция гарантирует, что service worker не будет установлен, пока код, переданный внутри waitUntil(), не завершится с успехом.

Внутри waitUntil() мы используем метод caches.open(), чтобы создать новый кэш, который назовём v1, это будет первая версия кеша ресурсов. Этот метод возвращает promise для созданного кэша; когда он выполнится, у объекта созданного кэша мы вызовем метод addAll(), который в качестве параметра ожидает получить массив origin-относительных URL всех ресурсов, которые мы хотим хранить в кэше.

Теперь, когда ресурсы сайта находятся в кэше, необходимо указать service worker, что делать с этим контентом. Это легко сделать, обработав событие fetch.

Событие fetch возникает каждый раз, когда запрашиваются любые подконтрольные service worker ресурсы, к которым относятся документы из области видимости и другие ресурсы, связанные с этими документами (например, если в index.html происходит кросс-доменный запрос для загрузки изображения, то он тоже попадёт в service worker). Пример настраиваемых ответов на запросы к ресурсам продемонстрирован на Рисунке 3.6

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.6. Пример настраиваемых ответов на запросы к ресурсам

Подключив к service worker обработчик события fetch, внутри него на объекте события мы вызываем метод respondWith(), чтобы заменить ответы. Метод caches.match(event.request) позволяет нам проверять сетевой запрос ресурса на соответствие какому-либо доступному в кэше ресурсу, если такой ресурс имеется, то мы его возвращаем.

Для тестирования service worker, можно воспользоваться Google DevTools. Так на Рисунке 3.7 показаны кэшированные ресурсы для нашего приложения «schedule.manager», полученные из Google DevTools

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.7. Сведения о кэшировании ресурсов приложения «schedule.manager», полученные из Google DevTools

Как видно, ресурсы были успешно кэшированы: стили, изображения, chuncks генерации; и при следующей загрузке веб-приложения, мы сможем получать ресурсы их хранилища.

## **Раздел 3.3. Возможность установки прогрессивных веб-приложений на устройства пользователей**

«Добавить на главный экран» или A2HS – это функция, доступная в современных браузерах, которая позволяет пользователю «установить» веб-приложение на свое устройство: добавить на главный экран ярлык, чтобы впоследствии получать доступ к приложению одним касанием.

Добавление приложения на главный экран возможно применить несколькими способами:

1. Использовать встроенные возможности браузера;
2. Использовать кнопку установки в самом веб-приложении, реализованную разработчиком.

В зависимости от браузера и операционной системы технические требования для установки PWA в системе могут различаться. Однако, если у веб-приложения имеется манифест и активный service worker, обрабатывающий входящие запросы fetch, пользователи смогут установить веб-приложение на всех поддерживаемых платформах.

После открытия веб-страницы в меню поддерживаемого браузера, как правило, находится опция: «Добавить на главный экран», а на desktop версиях браузеров соответствующая иконка, вызывающая диалоговое окно установки PWA на устройство (Рисунок 3.8).

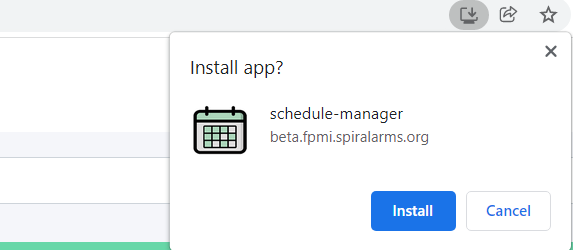


Рисунок 3.8. Процесса установки PWA встроенными возможностями браузера.

Подтвердив установку в появившемся баннере, на главном экране вашего устройства и/или в списке всех приложений должен появиться новый ярлык приложения. Данная процедура позволит PWA запускаться с домашнего экрана, док-станции или панели задач пользователя в отдельном окне вместо вкладки браузера. Теперь веб-приложение, также как и нативное, можно искать на устройстве в списке установленных программ и переключаться между ними с помощью многозадачности.

Однако, что, если мы хотим использовать собственную кнопку для установки приложения?

Браузер выполняет все шаги, связанные с установкой, от отображения баннера до добавления значка приложения на рабочий стол. Однако веб-приложение тоже может обрабатывать представление пользовательского интерфейса, которое приводит к запросу на установку PWA. В этом случае браузер определяет, должна ли отображаться кнопка и представление на установку приложения. Веб-приложение должно проверить, может ли оно отображать подсказку об установке, прослушивая beforeinstallprompt событие, и попросить браузер показать баннер установки приложения, если пользователь её запрашивает.

Чтобы открыть диалог установки, нам необходимо:

* Прослушать beforeinstallprompt событие у глобального объекта window документа DOM.
* Сообщить пользователю, что приложение может быть установлено с помощью кнопки или другого элемента, который будет генерировать событие жеста пользователя.
* Показать подсказку, вызвав метод prompt() объекта события beforeinstallprompt.

Пример простой реализации представления (кнопки) установки PWA продемонстрирован на Рисунке 3.9

В данном случае мы перехватываем событие beforeinstallprompt и сохраняем его, чтобы вызывать у него метод prompt() для отображения диалогового окна на установку приложения, когда пользователи нажимают на кнопку installButton.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.9. Пример реализации кнопки установки PWA

Таким образом, даже не продвигая установку приложений, пользователи смогут установить его на устройство. При этом стоит отметить, что не многие пользователи будут поступать подобным образом, в связи с чем продвижение установки приложения имеет важное значение.

Добавление файла манифеста веб-приложения и service worker действительно улучшает пользовательский опыт нашего традиционного веб-приложения. Это связано с тем, что PWA быстрые, безопасные, надежные и, что наиболее важно, поддерживают автономный режим. Рассмотрев процесс создания PWA, можно однозначно сказать, что добавление его поддержки в существующий проект и создание нового проекта с его поддержкой действительно является менее трудозатратым, чем разработка нативных приложений, а преимущества, предоставляемые PWA, заслуживают того, чтобы многие разработчики обратили на него внимание.

## **Раздел 3.4. Обновление прогрессивных веб-приложений на устройствах пользователей**

После публикации прогрессивного веб-приложения возникает необходимость его обновления. Одни пользователи используют его из браузера, другие устанавливают на свои устройства. Нативное приложение обновляется следующим образом. Оно скачивает свежий инсталлятор, удаляет старую версию и устанавливается заново. Только после этого пользователь получает новую версию приложения.

Схожим образом действует и браузер при обновлении service worker.

Браузер запускает алгоритм обновления при изменении service worker или его зависимостей. Браузер обнаруживает обновления, используя побайтовое сравнение кэшированных файлов и ресурсов, поступающих из сети. Затем он пытается установить новую версию service worker, и новый service worker будет находиться в состоянии waiting.

Всего у service worker три статуса: installing, waiting и active. Active – это статут активного, работающего service worker. Стадии installing и waiting service-worker проходит на пути к active. На стадии installing service worker нужно время, чтобы установиться, а стадии waiting ему нужна причина, чтобы заменить текущий service worker (обычно это закрытие всех вкладок приложения)[2].

Новая установка запустит событие установки для нового работника службы. Если вы кэшируете статические данные в этом обработчике событий, статические данные также будут повторно кэшированы.

Чтобы определить, что новый service worker готов и установлен, мы используем событие updatefound из объекта регистрации service worker. Это событие запускается, когда начинается установка нового worker. Чтобы определить, что service worker успешно обновлен, нам нужно дождаться, пока его состояние изменится на установленное с событием statechange (Рисунок 3.10).

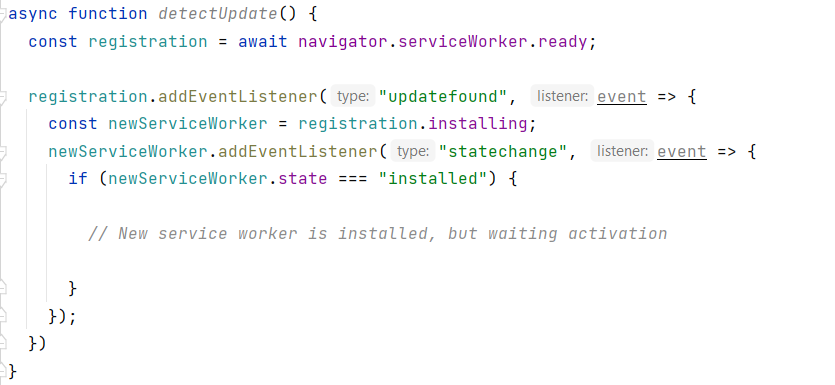


Рисунок 3.10. Пример реализации ослеживания состояния обновления service worker

Таким образом service worker будет установлен, но при наличии открытых страниц, контролируемых предыдущим service worker, новый service worker переходит в состояние ожидания и активируется только при отсутствии загружаемых страниц, которые все еще использует старый. Хотя это и не рекомендуется, но новый service worker cможет пропустить этот период ожидания и немедленно начать активацию. В обработчике прослушивания события install, необходимо вызвать метод skipWaiting(), а в обработчике прослушивания события activate предоставить новому service worker права на всех открытых клиентов, это могут быть standalone окна PWA или вкладка браузера (Рисунок 3.11).



Рисунок 3.11. Пример реализации пропуска ожидания активации service worker

Метод skipWaiting() означает, что новый service worker будет контролировать страницы, которые были загружены более старой версией. Это означает, что некоторые уже открытые страницы будут обрабатываться старым service worker, при этом новый service worker будет обрабатывать последующие открываемые страницы. Так как это может помешать работе приложения, данный подход не рекоммендуется к использованию.

Итак, соберем все воедино, и окончательный поток выглядит следующим образом: пользователь открывает PWA → PWA перехватывает событие изменения видимости и заставляет service worker обновиться → service worker проверяет, есть ли новая версия приложения. Если доступно → мы обрабатываем событие обновления, показывая пользователю панель обновления → Пользователь перезапускает приложение, тем самым активирует новый service worker.

# **ЗАКЛЮЧЕНИЕ**

Сегодня веб – это важнейшее средство предоставления простых, понятных и удобных интерфейсов для потребительских и корпоративных приложений. Он настолько распространен, что подавляющее большинство приложений, изначально создававшихся для мобильных и настольных платформ, сейчас разрабатываются с помощью веб-технологий, что требует от разработчиков постоянного наблюдения за развитием веб-технологий, иначе сделать свои проекты успешными просто не получится. Так неотъемлемой частью современной веб-разработки являются прогрессивные веб-приложения.

Progressive web applications (PWAs) – это веб-приложения, разработанные с использованием ряда конкретных технологий, таких как web API, service workers, манифесты веб-приложений; и стандартных шаблонов, позволяющих им использовать преимущества как веб-функций, поддерживаемых современными браузерами, так и функций нативных приложений. Такие приложения запускаются везде, их не нужно устанавливать, чтобы воспользоваться функционалом, обладают рядом характеристик, обеспечивающих пользователей новым опытом взаимодействия с контентом и позволяет им обновлять веб-приложения до прогрессивных веб-приложений независимо от их собственной операционной системы. Они сочетают в себе возможности веб-сайтов и мобильного программного обеспечения, что позволяет создать иммерсивный пользовательский интерфейс и повысить вовлеченность пользователей и коэффициент конверсии.

Прогрессивные веб-приложения позицианируют себя как производительные, безопасные и надежные. Вместе с тем они предоставляют пользователю опыт работы в браузере, похожий на нативное приложение. Несмотря на низкую стоимость разработки, дополнительные пути распространения, PWAs имеют и некоторые недостатки, где наиболее весомыми являются зависимость функционала приложения от конкретной платформы и браузера, а также ограничение возможностей пользовательского интерфейса функциональностью веб-приложений.

Хотя прогрессивное веб-приложение – это довольно новое явление, все большее число компаний уже извлекают выгоду из его обширных возможностей. Так приложения на базе PWA используют как основные или в дополнение к основному такие компании как Twitter, Tinder, Uber, Printerest, Telegram, Starbucks, Forbes, AliExpress, Aviasales.

PWA не требуют разных версий для разных устройств; одно прогрессивное приложение отвечает требованиям всех конечных устройств, на которых оно работает. Таким образом, значительно сокращается объем усилий, которые прилагают разработчики, и, как следствие, уменьшается стоимость создания приложения.

Для того, чтобы создать прогрессивное веб-приложение или внедрить его поддержку в уже существующее понадобятся только добавить manifest.json и service-worker. Так определив базовые возможности manifest.json и добавив service-worker можно реализовать поддержку PWA и его ключевую возможность – установку веб-приложения на устройство пользователей. Используя service worker, можно без труда получить приложение, использующее в первую очередь кэшированные ресурсы, указывая тем самым поведение по умолчанию в автономном режиме, до того, как будут получены данные по сети. Так обычно работают нативные приложения, что часто является причиной выбора пользователя в их пользу.

От внедрения прогрессивных веб-приложений некоторые компании могут выиграть в увеличении роста мобильного трафика и скорости загрузки приложения, сокращении использования памяти устройства, повышнии уровеня вовлеченности пользователей. Уже сейчас разработка PWA обходится в несколько раз дешевле разработки нативных приложений, а с учетом того, что крупные игроки, такие как Google и Microsoft, уделяют все большее и большее внимание поддержке PWA, меняя рынок разработки приложений и создавая инструменты для внедрения PWA, стоимость разработки может снизиться еще сильнее.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Progressive web apps (PWAs) [Electronic resource] / Mozilla Developer Network contributors, 2021. – Mode of access: https://developer.mozilla.org/en-US/docs/Web/Progressive\_web\_apps. – Date of access: 04.12.2021.

2. Introduction to progressive web apps [Electronic resource] / Mozilla Developer Network contributors, 2021. – Mode of access: https://developer.mozilla.org/en-US/docs/Web/Progressive\_web\_apps/Introduction. – Date of access: 04.12.2021.

3. Materials of PWA Summit 2021 [Electronic resource] / PWA Summit, 2021. – Mode of access: https://www.youtube.com/playlist?list=PLdwe-qCygjtEpL6LP3  
oyWxdJfPqKkHV2y. – Date of access: 04.12.2021.

4. Building a Progressive Web App and Deploying it [Electronic resource] / Prajwal KG. – dev.to, 2020. – Mode of access: https://dev.to/kgprajwal/building-a-progressive-web-app-and-deploying-it-10p7. – Date of access: 04.12.2021.

5. PWA для SSR приложения в 5 строк на Workbox 6 [Электронный ресурс] / Вадим Бударин. – medium.com, 2020. – Режим доступа: https://medium.com/p/  
dc89fc513361. – Дата доступа: 04.12.2021.

6. Make your PWA feel more like an app [Electronic resource] / Thomas Steiner. – web.dev, 2020. – Mode of access: https://web.dev/app-like-pwas. – Date of access: 04.12.2021.

7. Могут ли PWA конкурировать с нативными приложениями [Электронный ресурс] / WebbDEV. – medium.com, 2018. – Режим доступа: https://medium.com/webbdev/a4297bb58c27. – Дата доступа: 04.12.2021.

8. What is native app? Definition, tips, and best example [Electronic resource] / Le Pham, 2021. – Mode of access: https://magenest.com/en/native-app/. – Date of access: 05.12.2021.

9. Native app [Electronic resource] / Alexander S. Gillis. – TechTarget, 2020. – Mode of access: https://magenest.com/en/native-app/. – Date of access: 05.12.2021.

10. What are Progressive Web Apps? [Electronic resource] / Sam Richard, Pete LePage. – web.dev, 2020. – Mode of access: https://web.dev/what-are-pwas/. – Date of access: 04.12.2021.

11. All You Need to Know About Progressive Web App to Decide if Your Business Needs One? [Electronic resource] / Kateryna Khalimonchuk. – Fulcrum, 2020. – Mode of access: https://fulcrum.rocks/blog/progressive-web-app/. – Date of access: 04.12.2021.

12. Infographic: Progressive Web Apps. The Future of Mobile. [Electronic resource] / Jamie Fuller, 2017. – Mode of access: https://appinstitute.com/progressive-web-apps-infographic – Date of access: 04.12.2021.

13. Do You Need A Progressive Web App? PWA Benefits. [Electronic resource] / Patrycja Paterska, 2020. – Mode of access: https://www.elpassion.com/blog/when-to-consider-a-progressive-web-app-for-your-business. – Date of access: 04.12.2021.

14. Web Application Manifest. [Electronic resource] / W3C Working Draft, 2021. – Mode of access: https://www.w3.org/TR/appmanifest/. – Date of access: 13.12.2021.

15. Использование Service Worker. [Electronic resource] / MDN Web Docs, 2021. – Mode of access: https://developer.mozilla.org/ru/docs/Web/API/ Service\_Worker\_API/Using\_Service\_Workers. – Date of access: 13.12.2021.