**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра технологий программирования

**ГРУШЕВСКИЙ АНДРЕЙ АЛЕКСАНДРОВИЧ**

**ПРИМЕНЕНИЕ МЕТОДОЛОГИИ PWA ДЛЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ-ЕЖЕДНЕВНИКА**

Дипломная работа

студента 4 курса 8 группы

|  |  |
| --- | --- |
| «Допустить к защите»  **Зав. кафедрой** ТП  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2017 г | **Руководитель**  *Полойко Д. К.*  старший преподаватель  **Рецензент**  *Дробушевич Л. Ф.*  доцент, к.ф.-м.н. |

**Минск 2017**

**ПОСТАНОВКА ЗАДАЧИ**

1. Изучить методологию PWA.
2. Сравнить существующие веб и мобильные приложения-ежедневники, их возможности и особенности.
3. Создать приложение-ежедневник на основе методологии PWA и протестировать его.
4. По результатам тестирования сделать вывод о применимости данной методологии.

**АННОТАЦИЯ**

Грушевский А. А. Применение методологии PWA для разработки приложения-ежедневника: Дипломная работа/ Минск: БГУ, 2017. – 29 c.

Данная работа посвящена методологии прогрессивных веб-приложений разработки программного обеспечения.

**АНАТАЦЫЯ**

Грушэўскі А. А. Выкарыстанне метадалогіі PWA для распрацоўкі прыкладання-штодзённіка: Дыпломная работа/ Мінск: БДУ, 2017. – 29 c.

Дадзеная работа прысвечана метадалогіі прагрэсіўных вэб-прыкладанняў распрацоўкі праграмнага забеспячэння.

**ANNOTATION**

Hrusheuski A. А. The use of the PWA methodology to develop a daily journal application: Diploma work/ Minsk: BSU, 2017. – 29 p.

This work is devoted to the methodology of progressive web application development software.

**РЕФЕРАТ**

Дипломная работа, 29 страниц, 7 рисунков, 5 таблиц, 20 источников.

**Ключевые слова:** МЕТОДОЛОГИЯ, ПРОГРЕССИВНОЕ ВЕБ-ПРИЛОЖЕНИЕ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ТЕХНОЛОГИЯ.

**Объект исследования:** методология прогрессивных веб-приложений.

**Цель работы**: изучить методологию прогрессивных веб-приложений, создать приложение, использующее данную методологию, оценить зрелость данной методологии разработки программного обеспечения.

**Методы исследования:** экспериментальный анализ.

**Результатом работы** является вывод о применимости методологии прогрессивных веб-приложений.

**Областью применения** является разработка программного обеспечения.

**РЭФЕРАТ**

Дыпломная работа, 29 старонак, 7 малюнкаў, 5 табліц, 20 крыніц.

**Ключавыя словы:** МЕТАДАЛОГІЯ, ПРАГРЭСІЎНАЕ ВЭБ-ПРЫКЛАДАННЕ, ПРАГРАМНАЕ ЗАБЕСПЯЧЭННЕ, ТЭХНАЛОГІЯ.

**Аб'ект даследавання:** метадалогія прагрэсіўных вэб-прыкладанняў.

**Мэта работы:** даследаваць метадалогію прагрэсіўных вэб-прыкладанняў, стварыць прыкладанне, якое выкарыстоўвае дадзеную метадалогію, ацаніць сталасць дадзенай метадалогіі распрацоўкі праграмнага забеспячэння.

**Метады даследавання:** эксперыментальны аналіз.

**Вынікам працы** з'яўляецца вывад аб прымянімасці метадалогіі прагрэсіўных вэб-прыкладанняў.

**Вобласцю ўжывання** з'яўляецца праектаванне праграмнага забеспячэння.

**ABSTRACT**

Diploma work, 29 pages, 7 figures, 5 sheets, 20 sources.

**Keywords:** METHODOLOGY, PROGRESSIVE WEB APPLICATION, SOFTWARE, TECHNOLOGY.

**Object of research:** methodology of progressive web application.

**Purpose:** study the methodology of progressive web applications, create an application using this methodology, evaluate the maturity of this software development methodology.

**Methods:** experimental analysis.

**The result of work** is the conclusion about the applicability of the methodology of progressive web applications.

**The field of application** is software engineering.

СОДЕРЖАНИЕ

[ОСНОВНЫЕ ТЕРМИНЫ И ПОНЯТИЯ 6](#_Toc483564516)

[ВВЕДЕНИЕ 8](#_Toc483564517)

[1 ОБЩИЕ ТЕХНОЛОГИИ ДЛЯ МЕТОДОЛОГИИ ПВП 9](#_Toc483564518)

[1.1 Описание технологий 9](#_Toc483564519)

[1.2 Web App Manifest 10](#_Toc483564520)

[1.3 Service Worker 10](#_Toc483564521)

[1.4 Application Shell Architecture 11](#_Toc483564522)

[2 ПРИЛОЖЕНИЕ-ЕЖЕДНЕВНИК 13](#_Toc483564523)

[2.1 Сравнение существующих приложений, определение требуемого функционала 13](#_Toc483564524)

[2.2 Описание прогрессивного приложения-ежедневника 15](#_Toc483564525)

[2.3 Серверная часть 17](#_Toc483564526)

[2.4 Клиентская часть 17](#_Toc483564527)

[2.5 Анализ эффективности приложения 20](#_Toc483564528)

[ЗАКЛЮЧЕНИЕ 23](#_Toc483564529)

[СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ 24](#_Toc483564530)

[ПРИЛОЖЕНИЕ А 26](#_Toc483564531)

[ПРИЛОЖЕНИЕ Б 27](#_Toc483564532)

[ПРИЛОЖЕНИЕ В 28](#_Toc483564533)

# ОСНОВНЫЕ ТЕРМИНЫ И ПОНЯТИЯ

Прогрессивное веб-приложение (ПВП, Progressive web application, PWA) – тип приложений или методология разработки программного обеспечения. [1]

Пользовательский опыт (User experience, UX) – это восприятие и ответные действия пользователя, возникающие в результате использования и/или предстоящего использования продукции, системы или услуги. [2]

API – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. [3]

Технология push (push-нотификация) — один из способов распространения информации (контента) в Интернете, когда данные поступают от поставщика к пользователю на основе установленных параметров. Пользователь же в свою очередь либо отвергает, либо принимает данные. [4]

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript. [5]

BSON (Binary JavaScript Object Notation) – формат электронного обмена цифровыми данными, основанный на JavaScript. Является надмножеством JSON, включая дополнительно регулярные выражения, двоичные данные и даты. [6]

DOM (Document Object Model) – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов. [7]

XHR (XMLHttpRequest) – API, доступное в скриптовых языках браузеров, таких как JavaScript. Использует запросы HTTP или HTTPS напрямую к веб-серверу и загружает данные ответа сервера напрямую в вызывающий скрипт. [8]

Localstorage – это программный метод и протокол веб-приложения, используемый для хранения данных в веб-браузере. [9]

Операционная система (ОС) – комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем. [10]

Фреймворк (Framework) – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. [11]

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных. [12]

AJAX (Asynchronous JavaScript and XML) – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. [13]

# ВВЕДЕНИЕ

На текущий момент на мобильных платформах используются два типа приложений: мобильные и веб. У каждого из приведенных типов есть свои недостатки. К примеру, минусы мобильных приложений:

* Требуют установки через магазин приложений;
* Зависят от целевой операционной системы.

Недостатки веб-приложений:

* Требуют подключение к интернету;
* Зависят от качества сети.

При этом, приведенные минусы одного из типов приложений не являются минусами другого. Таким образом, объединение возможностей веб и мобильного приложения позволит избавиться от перечисленных выше недостатков. Методология прогрессивных веб-приложений (ПВП) разработана именно для решения данной проблемы.

Целью дипломной работы было исследовать практическую пригодность методологии прогрессивных веб-приложений ПВП для промышленной разработки. Для этого было решено разработать с её использованием простейшее приложение-ежедневник и по результатам выполнения данной практической задачи сделать соответствующие выводы о готовности методологии к использованию для выполнения реальных проектов.

# ОБЩИЕ ТЕХНОЛОГИИ ДЛЯ МЕТОДОЛОГИИ ПВП

## Описание технологий

Прогрессивные веб-приложение (Progressive web app) – термин, обозначающий новую методологию в разработке ПО. Технически, ПВП являются обычными веб-страницами, но выглядят для пользователя, как мобильное (нативное) приложение. Этот новый тип приложений комбинирует возможности, которые предоставляют современные браузеры с положительными сторонами мобильной разработки. Они могут быть добавлены на домашний экран устройства с помощью браузера и работать в независимости от подключения к интернету.

Также эта методология дает возможность, используя одни и те же средства, знания и языки программирования, создавать приложения, не зависящие от особенностей операционной системы, что сильно упрощает кроссплатформенную разработку.

Для работы веб-приложения могут быть использованы любые технологии, поддерживаемые браузером, однако некоторые из них обязательны, чтобы приложение считалось прогрессивным, а именно:

* Web App Manifest. Файл манифеста предоставляет разработчикам централизованное место для метаданных, относящихся к приложению. Он описывается в формате JSON. Цель его заключается в предоставлении данных для установки приложения на домашний экран устройства для быстрого доступа без посещения магазина приложений. [14]
* Service Worker. Service worker является событийно-управляемым web worker-ом (Объект, который запускает JavaScript файл. Он работает в отдельном потоке от остальной страницы), зарегистрированном для конкретного сайта. Он предоставляется в виде JavaScript файла, который может контролировать ресурсные запросы и запросы навигации страницы, к которой он привязан. Также он позволяет кэшировать ресурсы в очень детальном виде, что предоставляет полный контроль над поведением приложения. [15]
* Application Shell Architecture. Для быстрой загрузки страницы service worker кэширует базовый пользовательский интерфейс приложения. Он представляет собой минимально необходимое количество HTML, CSS, JavaScript кода для первоначальной загрузки страницы, контент в которую может быть загружен динамически. [16]

Далее рассмотрим каждую из технологий подробнее.

## Web App Manifest

Для подключения файла манифест требуется вставить в HTML-файл главной страницы файла в тег head следующий код:

<link rel="manifest" href="/manifest.json">

Здесь атрибут href указывает на расположение файла.

Давайте рассмотрим, какие поля обязательны для заполнения:

* name – предоставляет человеко-читаемое имя приложения, как оно должно быть показано пользователю.
* short-name – предоставляет короткое человеко-читаемое имя приложения, которое показывается, когда недостаточно места для отображения полного имени.
* icons – содержит массив объектов, которые описывают иконки приложения. Объекты состоят из поля src (путь к файлу), type (тип), size (размер в пикселях).
* start\_url – определяет ссылку, которая открывается, когда пользователь запускает приложение с устройства.
* display – определяет предпочтительный для разработчика вид приложения, в случае прогрессивного приложения часто используется значение “standalone”, тогда ПВП будет выглядеть, как нативное.

Также желательно указывать свойства background\_color и theme\_color. Они позволяют показывать анимацию плавного перехода, когда приложение не до конца запустилось. Также в зависимости от ОС значение этих полей иногда меняет вид приложения (например, фон в переключателе приложений android). [14]

## Service Worker

Как уже было сказано выше, service worker работает в отдельном потоке из-за чего не блокирует основной. Также он не имеет доступа к DOM. Service worker предназначен быть полностью асинхронным, поэтому такие API, как синхронный XHR, localstorage, не могут быть использованы внутри него.

Service worker работает только по протоколу HTTPS для обеспечения безопасности.

Для работы необходимо зарегистрировать service worker, после чего у него начинается жизненный цикл, который состоит из следующих шагов:

Загрузка.

Установка.

Активация.

Service worker немедленно загружается, как только пользователь впервые открывает сайт. После этого он загружается примерно каждые 24 часа для поддержания актуальности.

Установка начинается, если загруженная версия новее уже установленной или других версий не обнаружено.

При первом посещении service worker устанавливается и, в случае успеха, считается активированным.

Если service worker уже существует, то новая версия устанавливается в фоне, но не активируется сразу. Это состояние называется «в ожидании». Новая версия активируется только после того, как не осталось страниц, которые ссылаются на старую версию.

Назовем еще несколько функций service worker:

* Фоновая синхронизация данных;
* Кэширование запросов;
* Компиляция кода на стороне клиента;
* Оптимизации быстродействия, к примеру, предзагрузка ресурсов, которые могут понадобиться пользователю в ближайшее будущее. [15]

## Application Shell Architecture

Оболочка приложения – статическая часть приложения, которая должна быть доступна пользователю всегда.

Требования к оболочке:

* Должна быстро загружаться;
* Должна быть кэширована;
* Должна динамически отображать контент приложения.

Пример оболочки и контента приложения смотрите ниже (рис. 1). [16]

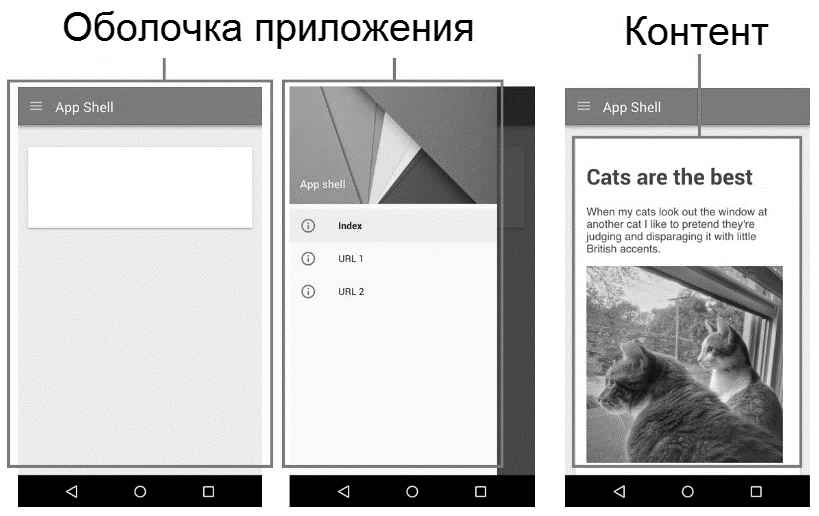


Рисунок 1

Далее приведены данные о времени и количестве байт, необходимом для загрузки страницы, в которой реализована технология Application Shell Architecture, в первый раз и повторно (табл. 1). Такой результат достигается за счет кэширования при использовании service worker. [17]

Таблица 1 – Статистика загрузки тестовой страницы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Время загрузки | Первый байт | Начало рендера | Размер документа | Полная загрузка | |
| Время | Размер |
| Первая загрузка | 3,779с | 1,879с | 2,273с | 52 Кбайт | 5,614с | 90 Кбайт |
| Повторная загрузка | 0,507с | 0,224с | 0,462с | 70 Кбайт | 0,737с | 83 Кбайт |

# ПРИЛОЖЕНИЕ-ЕЖЕДНЕВНИК

## Сравнение существующих приложений, определение требуемого функционала

Выделим необходимый функционал для ежедневника.

Для сравнения рассмотрим следующие веб-приложения, а также их аналоги для ОС android:

* Wunderlist (https://wunderlist.com);
* Todoist (https://ru.todoist.com).

Сравнение будем проводить по следующим критериям: для веб-приложений – время загрузки страницы и объем передаваемых данных при скорости сети 3G (табл. 2), для android-приложений – размер установленного приложения и размер пользовательских данных после запуска и входа в приложение (табл. 3)

Таблица 2 – Статистика загрузки веб-приложений

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Wunderlist | Todoist |
| Первая загрузка | Время загрузки | 3,39с | 2,73 с |
| Размер документа | 1,9 Мбайт | 188 Кбайт |
| Повторная загрузка | Время загрузки | 1,60с | 0,96с |
| Размер документа | 104 Кбайт | 1,9 Кбайт |

Таблица 3 – Размер android-приложений

|  |  |  |
| --- | --- | --- |
|  | Wunderlist | Todoist |
| Размер приложения | 29,75 Мбайт | 31,48 Мбайт |
| Размер данных | 492 Кбайт | 420 Кбайт |

Как видно из приведенных данных, во время первого посещения сайтов большая часть данных кэшируется и не требует повторной загрузки. Зачастую это медиа файлы и изображения. Однако, время загрузки не сократилось пропорционально объему данных. Это возникает из-за ряда причин. Во-первых, всегда есть минимальная задержка ответа сервера. Во-вторых, на стороне клиента выполняются скрипты, что может блокировать загрузку страницы. В-третьих, требуется время на прорисовку страницы.

Пока все страница не прорисовалась первый раз, пользователь видит белый экран. Чем дольше пользователь его видит, тем меньше хочет пользоваться данным приложением. Чтобы обойти данную проблему в приложения часто добавляют анимацию загрузки, что может обмануть пользователя, и ему покажется, что страница загрузилась быстрее.

Что же касается мобильных приложений, прекрасно видно, что их размер в десятки раз превосходит размер загружаемых данных для веб-приложений, хотя по функционалу они близки. Это связанно с особенностями разработки нативных приложений. Их размер растет, при равных возможностях, гораздо быстрее веб-приложений.

Теперь рассмотрим функционал.

Оба приложения имеют похожий интерфейс и возможности (рис. 2, рис. 3)

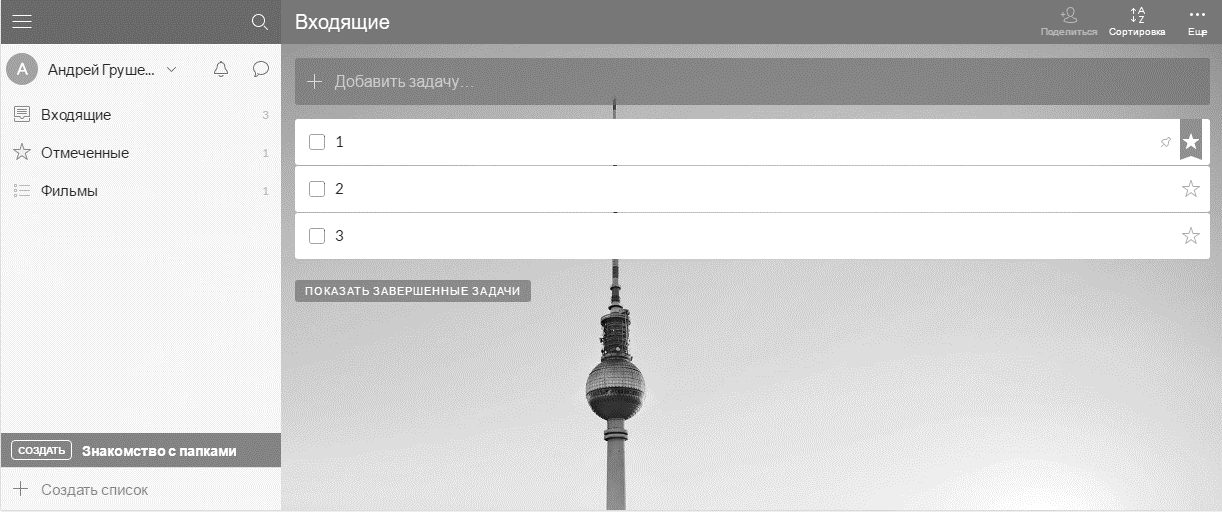


Рисунок 2 – Веб-приложение Wunderlist

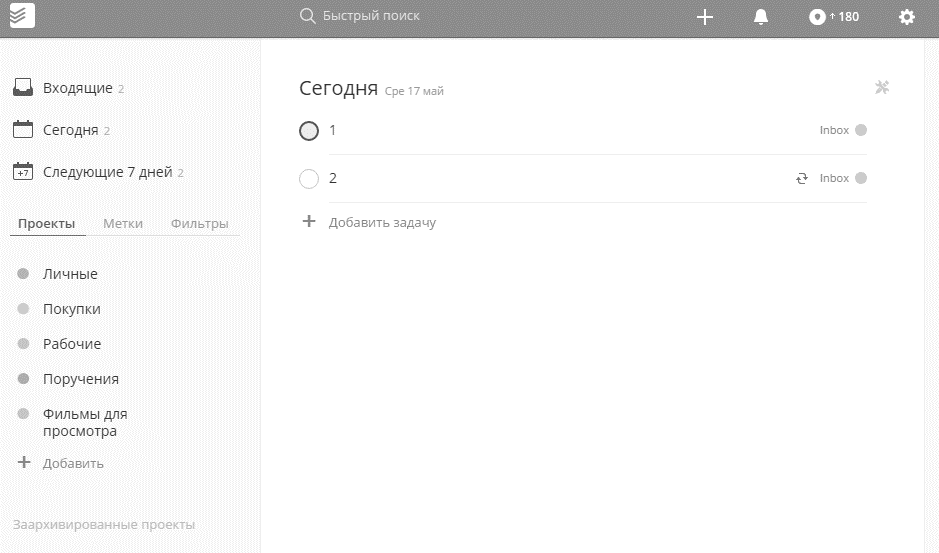


Рисунок 3 – Веб-приложение Todoist

Возможности обоих приложений:

* Регистрация и вход в приложение;
* Редактирование, завершение задач;
* Редактирование, завершение подзадач;
* Редактирование сроков завершения задач;
* Добавление комментариев к задачам;
* Добавление меток к задачам;
* Сортировка по сроку завершения задач;
* Создание списка задач.

Также в обоих приложениях есть возможность нотификации. Еще стоит отметить, что веб-приложение Wunderlist не работает для устройств с малым разрешением, вместо этого на сайте предлагают установить мобильное приложение.

Интерфейс android-аналогов этих сайтов совпадает за исключением того, что меню слева не может быть отображено одновременно с отображением основной части приложения. Функционал совпадает практически полностью.

## Описание прогрессивного приложения-ежедневника

Поскольку целью дипломной работы не является полное копирование всех возможностей схожих приложений, выделим несколько функций, которые должны быть реализованы в приложении (рис. 4):

* Регистрация и вход в приложение;
* Редактирование, завершение задач;
* Редактирование сроков завершения задач;
* Добавление меток к задачам;
* Сортировка по сроку завершения задач.

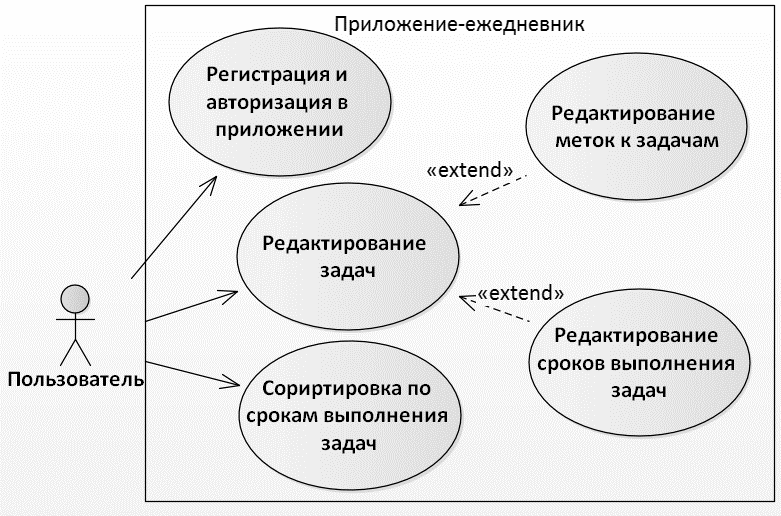


Рисунок 4 – Use-case диаграмма

Опишем, какие инструменты будут использоваться.

Для стороны сервера будут использованы NodeJS (серверная платформа, исполняющая JavaScript код), ExpressJS (фреймворк приложений NodeJS), MongoDB (NoSQL СУБД, инструкции к которой могут описываться JavaScript кодом). Для работы протокола HTTPS требуется сертификат безопасности. Чтобы решить эту проблему будем использовать облачную платформу приложений heroku (https://www.heroku.com). Она позволяет покрыть все требования для выполнения дипломной работы бесплатно. При этом она легко масштабируется для приложений разного размера.

Для стороны клиента будет использована библиотека jQuery для упрощения работы с DOM-элементами и AJAX запросами.

## Серверная часть

Опишем коллекции в базе данных MongoDB. Формат, используемый для их описания – BSON. Будет использована только одна коллекция users (пользователи), в каждой ячейке которой будут храниться задачи, относящиеся к конкретному пользователю. Модель коллекции пользователей:

User: {

first\_name: String,

last\_name: String,

email: String,

hash\_password: String,

tasks: Array,

\_id: ObjectId

}

Благодаря особенностям данной СУБД в поле tasks можно хранить любые объекты, однако в целях оптимизации быстродействия рекомендуется специфицировать их. Опишем, как должна выглядеть модель задачи:

Task: {

label: String,

done: Boolean,

important: Boolean,

last\_updated: Date,

date\_limitation: Date,

\_id: ObjectId

}

API серверной части для задач находится в приложении А.

## Клиентская часть

Файл манифеста имеет уже описанную структуру и приведен в приложении Б. Можно отметить, что количество и размер иконок может быть любым, однако важно понимать, какие разрешения иконок поддерживают разные браузеры.

Далее приведем service worker (прил. В). Вверху можно сразу увидеть названия для кэш-объектов, важно отметить, что удобно включать номер версии в них. Изменение этого номера принуждает перезаписать кэш для обновления приложения после загрузки более новой версии service worker. Далее идут названия файлов, которые необходимо кэшировать. Потом описано, непосредственно, поведение service worker, обрабатываются события установки, активации и посылания запросов. При посылании запроса страницей worker осуществляет один из двух сценариев. Первый – сеть доступна, тогда он посылает запрос и, при получении, кэширует его. Второй – сеть недоступна, тогда возвращается такой же ранее кэшированный запрос. Архитектура приложения представлена на рисунке 5.

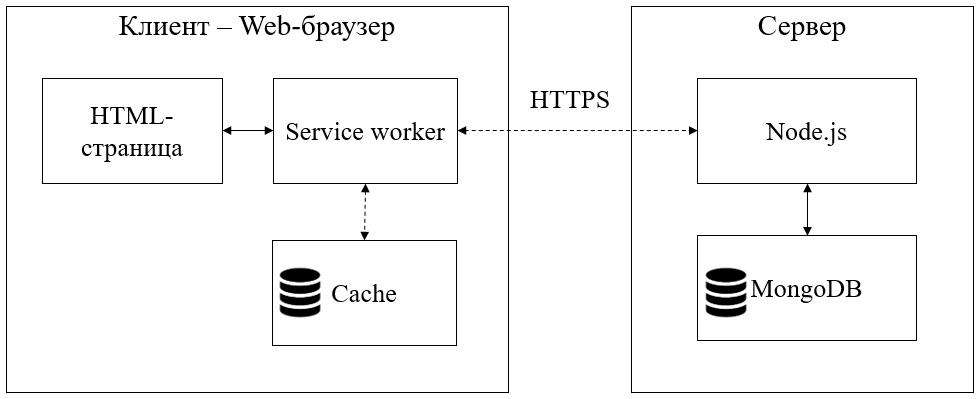


Рисунок 5 – Архитектура приложения

Для обеспечения большей надежности и стабильности, поместим часть кода, необходимой для активации service worker прямо в html-страницу, это гарантирует, что попытка загрузить service worker будет даже, если не загрузится основной JavaScript-файл:

<script>

window.onload = () => {

if ('serviceWorker' in navigator) {

navigator.serviceWorker

.register('/service-worker.js')

.then(function() {

console.log('Service Worker Registered');

})

.catch(function(error) {

console.error('Service Worker Failed', error);

});

}

};

</script>

Основной код JavaScript-программы использует возможности стандарта языка ECMAScript 6. Здесь описано взаимодействие с DOM-элементами, AJAX-запросами и основная логика. Также присутствует индикация сети. При загрузке приложение будет пытаться восстановить задачи, сохраненные ранее в localstorage, далее пошлет запрос на сервер за актуальной информацией и обновит имеющийся список задач.

Приведем скриншоты созданного приложения. На рисунке 6 показан вид основной части приложения, на рисунке 7 – боковое меню.

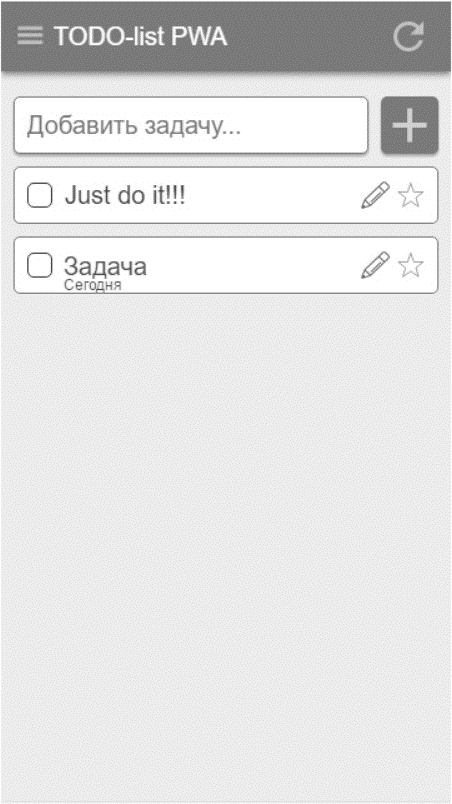


Рисунок 6 – Скриншот основной части приложения

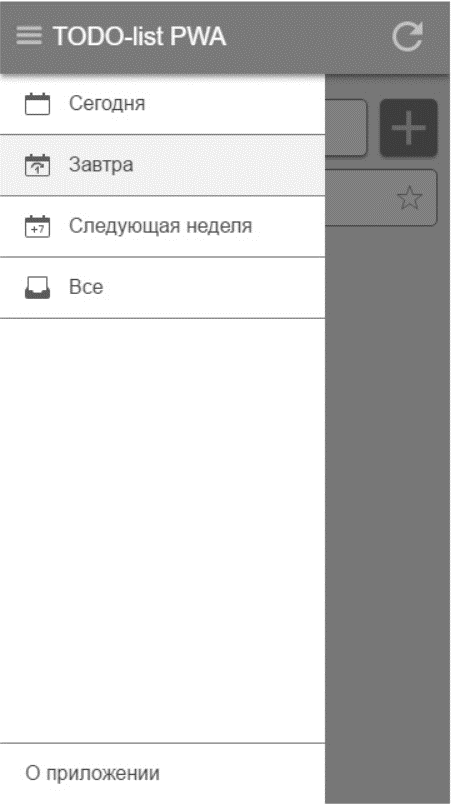


Рисунок 7 – Скриншот меню приложения

## Анализ эффективности приложения

Для начала, приведем данные о времени загрузке веб-страницы приложения при первом посещении и при повторном, а также количество загружаемой информации для обоих случаев, когда технологии service worker и manifest не включены (табл. 4). Данные получены с использованием браузера Google Chrome с эмуляциями разных скоростей сети.

Таблица 4 – Статистика загрузки страницы веб-приложения

|  |  |
| --- | --- |
|  | Результаты |
| Время первой загрузки | 0,52с |
| Объем данных при первой загрузке | 110 Кбайт |
| Время повторной загрузки | 0,35с |
| Объем данных при повторной загрузке | 4,40 Кбайт |
| Время повторной загрузки 3G | 0,47с |
| Время повторной загрузки без подключения к интернету | - |

Также приведем статику загрузки веб-приложения, когда технологии service worker и manifest включены (табл. 5). То есть, когда веб-приложение становится прогрессивным.

Таблица 5 – Статистика загрузки страницы прогрессивного веб-приложения

|  |  |
| --- | --- |
|  | Результаты |
| Время первой загрузки | 0,53с |
| Объем данных при первой загрузке | 113 Кбайт |
| Время повторной загрузки | 0,23с |
| Объем данных при повторной загрузке | 3,30 Кбайт |
| Время повторной загрузки 3G | 0,22с |
| Время повторной загрузки без подключения к интернету | 0,22с |

В первом случае объем передаваемых данных, а, следовательно, и время загрузки сокращается при повторной загрузке благодаря автоматическому кэшированию браузера. Но при изменении качества сети время загрузки увеличивается, а при отсутствии сети невозможно загрузить страницу.

Во втором случае объем данных при первой загрузке незначительно увеличился за счет загрузки файла service worker. При этом время сопоставимо с первым случаем. Все отличие видно во время повторной загрузке. Помимо сокращения объема данных (запрос создается только для пользовательского контента), сильно сокращается время загрузки. Также нет никакого влияния качества сети на работу приложения. При отсутствии сети вместо того, чтобы посылать запрос на сервер, service worker использует кэшированный результат предыдущего запроса, таким образом пользователь сможет использовать приложение без затруднений. Его данные будут синхронизированы при подключении к интернету.

Далее требуется протестировать добавление приложения на домашний экран. Для реализации этой возможности от разработчика не требуется ничего, кроме уже описанных шагов о файле манифеста и service worker, наличии HTTPS-сервера. При повторном открытии веб-сайта в течении пяти минут, пользователю будет показано сообщение о возможности добавления приложения на домашний экран. Если он подтвердит свое согласие, то будет выполнена установка. [18]

Скорость открытия приложения с домашнего экрана сложно оценить объективно. Ведь при запуске, сразу показывается та часть приложения, что описана в манифесте, а именно название, фон и иконка. Далее плавно загружается окно приложения (уже без адресной строки браузера) и появляется сохраненный пользовательский контент. Все это длится около одной секунды, что является высоким показателем даже для нативных приложений.

Стоит отметить негативные особенности данной методологии. Несмотря на то, что количество API, поддерживаемых браузерами постоянно растет, возможности нативных приложений все еще шире. Нужны ли они для конкретного приложения, решают его создатели.

Также есть еще один минус: технология Service Worker поддерживается только браузерами Google Chrome и Mozilla Firefox, технология Web App Manifest поддерживается только браузером Google Chrome [19]. Однако другие современные браузеры тоже смогут поддерживать эти технологии в скором времени. Несмотря на этот очевидный минус, пользуются Google Chrome более 75% пользователей, на втором по популярности месте идет Mozilla Firefox, что позволяет пренебречь некоторыми ограничениями для других пользователей.

# ЗАКЛЮЧЕНИЕ

В данной дипломной работе были достигнуты следующие результаты:

Была проверена практическая зрелость методологии создания прогрессивных веб-приложений.

Проанализирован функционал существующих веб и нативных приложений-ежедневников, и выработан функционал, требуемый для тестового приложения.

Было создано приложение, использующее технологии прогрессивных веб-приложений, а также проведен анализ его эффективности.

Ссылка на действующее приложение – https://todo-pwa.herokuapp.com.

Полный код приложения можно посмотреть на github – https://github.com/18089396a47/diplom.

Как видно из результатов тестирования, ПВП можно считать уже достаточно привлекательной альтернативой для мобильных и веб-приложений.

Для примера удачного применения можно привести сайт AliExpress. За достаточно короткое время после внедрения данной методологии, коэффициент конверсии новых пользователей вырос на 104%, количество посещений увеличилось в 2 раза, на 74% увеличилась длительность посещения сайта для всех браузеров. [20]

Все выше сказанное позволяет сделать вывод, что ближайшее будущее мобильной разработки стоит за прогрессивными веб-приложениями, ведь у данной методологии несомненно много плюсов, а количество минусов будет постепенно сокращаться.

# СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

1. Wikipedia [Electronic resource] / Progressive web application – Mode of access: <https://en.wikipedia.org/wiki/Progressive_web_app>. – Date of access: 22.04.2017
2. Wikipedia [Электрон. ресурс] / Опыт взаимодействия – Режим доступа: https://ru.wikipedia.org/wiki/Опыт\_взаимодействия. – Дата доступа: 07.05.2017
3. Wikipedia [Электрон. ресурс] / API – Режим доступа: https://ru.wikipedia.org/wiki/API. – Дата доступа: 07.05.2017
4. Wikipedia [Электрон. ресурс] / Технология push Режим доступа: https://ru.wikipedia.org/wiki/Технология\_push. – Дата доступа: 07.05.2017
5. Wikipedia [Электрон. ресурс] / JSON – Режим доступа: https://ru.wikipedia.org/wiki/JSON. – Дата доступа: 07.05.2017
6. Wikipedia [Электрон. ресурс] / BSON – Режим доступа: https://ru.wikipedia.org/wiki/BSON. – Дата доступа: 07.05.2017
7. Wikipedia [Электрон. ресурс] / Document Object Model – Режим доступа: https://ru.wikipedia.org/wiki/Document\_Object\_Model. – Дата доступа: 07.05.2017
8. Wikipedia [Электрон. ресурс] / XMLHttpRequest – Режим доступа: https://ru.wikipedia.org/wiki/XMLHttpRequest. – Дата доступа: 07.05.2017
9. Wikipedia [Электрон. ресурс] / Web Storage – Режим доступа: https://ru.wikipedia.org/wiki/Web\_Storage. – Дата доступа: 09.05.2017
10. Wikipedia [Электрон. ресурс] / Операционная система – Режим доступа: https://ru.wikipedia.org/wiki/Операционная\_система. – Дата доступа: 07.05.2017
11. Wikipedia [Электрон. ресурс] / Фреймворк – Режим доступа: https://ru.wikipedia.org/wiki/Фреймворк. – Дата доступа: 11.05.2017
12. Wikipedia [Электрон. ресурс] / Система управления базами данных – Режим доступа: https://ru.wikipedia.org/wiki/Система\_управления\_базами\_данных. – Дата доступа: 11.05.2017
13. Wikipedia [Электрон. ресурс] / AJAX – Режим доступа: https://ru.wikipedia.org/wiki/AJAX. – Дата доступа: 07.05.2017
14. Mozilla development network [Electronic resource] / Web App Manifest – Mode of access: https://developer.mozilla.org/en-US/docs/Web/Manifest. – Date of access: 15.04.2017
15. Mozilla development network [Electronic resource] / Service Worker API – Mode of access: https://developer.mozilla.org/en-US/docs/Web/API/Service\_Worker\_API. – Date of access: 15.04.2017
16. Google Developers [Electronic resource] / Addy Osmani, Matt Gaunt – Instant Loading Web Apps with an Application Shell Architecture – Mode of access: https://developers.google.com/web/updates/2015/11/app-shell. – Date of access: 15.04.2017
17. WebPagetest [Electronic resource] / Web Page Performance Test – Mode of access: http://www.webpagetest.org/result/151112\_HH\_11D0/. – Date of access: 29.04.2017
18. Google Developers [Electronic resource] / Matt Gaunt, Paul Kinlan – Web App Install Banners – Mode of access: https://developers.google.com/web/fundamentals/engage-and-retain/app-install-banners/. – Date of access: 16.05.2017
19. Can I use [Electronic resource] / Can I use – Mode of access: http://caniuse.com/. – Date of access: 20.04.2017
20. Google Developers [Electronic resource] / AliExpress – Mode of access: https://developers.google.com/web/showcase/2016/aliexpress. – Date of access: 16.05.2017

# ПРИЛОЖЕНИЕ А

Все запросы возвращают код 200 в случае успеха, 401 – в случае, если пользователь не авторизирован, 500 – в случае ошибки.

Получение задач:

* Метод – GET;
* Адрес – /data/get?{period}/, period может быть равен «today» (по умолчанию), «tomorrow», «week», «all»;
* Результат – коллекция задач за соответствующий период, отсортированная по времени окончания.

Добавление задачи:

* Метод – POST;
* Адрес – /data/add/;
* Параметры – label, important, done, date\_limitation.

Изменение задачи:

* Метод – PUT;
* Адрес – /data/edit/{id}/, id – идентификатор существующей задачи, если он не определен или не найден, то код 404;
* Параметры – label, important, done, date\_limitation.

Удаление задач:

* Метод – DELETE;
* Адрес – /data/delete/{id}/, id – идентификатор существующей задачи, если он не определен или не найден, то код 404.

# ПРИЛОЖЕНИЕ Б

{

"name": "TODO-list PWA",

"short\_name": "TODO-PWA",

"icons": [{

"src": "images/icons/icon-128x128.png",

"sizes": "128x128",

"type": "image/png"

}, {

"src": "images/icons/icon-144x144.png",

"sizes": "144x144",

"type": "image/png"

}, {

"src": "images/icons/icon-152x152.png",

"sizes": "152x152",

"type": "image/png"

}, {

"src": "images/icons/icon-192x192.png",

"sizes": "192x192",

"type": "image/png"

}, {

"src": "images/icons/icon-256x256.png",

"sizes": "256x256",

"type": "image/png"

}],

"start\_url": "/index.html",

"display": "standalone",

"background\_color": "#3E4EB8",

"theme\_color": "#2F3BA2"

}

# ПРИЛОЖЕНИЕ В

var dataCacheName = 'TODO-Data-v1';

var cacheName = 'TODO-PWA-v1';

var filesToCache = [

'/',

'/index.html',

'/scripts/jquery-3.2.1.min.js',

'/scripts/app.js',

'/styles/inline.css',

'/images/ic\_add\_white\_36px.svg',

'/images/ic\_refresh\_white\_36px.svg',

'/images/star.svg',

'/images/offline.svg',

'/manifest.json'

];

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

e.waitUntil(

caches.open(cacheName).then(function(cache) {

return cache.addAll(filesToCache)

.then(() => {

return self.skipWaiting();

});

})

);

});

self.addEventListener('activate', function(e) {

e.waitUntil(

caches.keys().then(function(keyList) {

return Promise.all(keyList.map(function(key) {

if (key !== cacheName && key !== dataCacheName) {

return caches.delete(key);

}

}));

})

);

return self.clients.claim();

});

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

var dataUrl = 'https://todo-pwa.herokuapp.com/data';

if (e.request.url.indexOf(dataUrl) > -1) {

e.respondWith(

caches.open(dataCacheName).then(function(cache) {

return fetch(e.request).then(function(response){

cache.put(e.request.url, response.clone());

return response;

});

})

);

} else {

e.respondWith(

caches.match(e.request).then(function(response) {

return response || fetch(e.request);

})

);

}

});