МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное

учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «СИТиС»

Отчет по проекту

на тему «Практика создания Progressive Web App»

по дисциплине «Технологии программирования Web-приложений»

Выполнил: Студент группы М092401(75)

Ладонов А.А.

Проверил: к.т.н., доцент

Гузеев А.В.

Содержание

[Введение 3](#_Toc198585798)

[Теоретические основы PWA 4](#_Toc198585799)

[Подготовка к разработке PWA 6](#_Toc198585800)

[Разработка PWA 7](#_Toc198585801)

[Демонстрация результатов работы 8](#_Toc198585802)

[Приложение 1. Код элементов приложения 14](#_Toc198585803)

# Введение

В современном мире веб-разработки всё большую популярность набирают Progressive Web Apps (PWA) — технология, которая объединяет преимущества веб-сайтов и мобильных приложений. PWA позволяют пользователям получать быстрый, надежный и интерактивный опыт, сравнимый с нативными приложениями, но без необходимости загрузки из магазинов. Это особенно актуально в условиях растущих требований к производительности, доступности и удобству использования на различных устройствах.

Цель данного реферата — изучить принципы создания PWA, рассмотреть ключевые технологии, лежащие в их основе, и разобрать практические аспекты разработки. В ходе работы будут рассмотрены такие вопросы, как использование Service Workers для оффлайн-режима, настройка Web App Manifest для добавления на домашний экран, а также методы оптимизации производительности. Кроме того, будут затронуты вопросы тестирования и публикации PWA, что позволит понять полный цикл разработки подобных приложений.

Структура реферата включает теоретическую часть, посвящённую основам PWA, практическое руководство по их созданию, а также раздел, посвящённый проверке и внедрению готовых решений. В заключении будут подведены итоги и обозначены перспективы дальнейшего развития этой технологии.

# Теоретические основы PWA

Progressive Web App (PWA) – технология веб-разработки, позволяющая создавать приложения, которые сочетают в себе преимущества классических веб-сайтов и нативных мобильных приложений. PWA работают в браузере, но поддерживают оффлайн-режим, push-уведомления и установку на устройство, что делает их более удобными для пользователей. PWA работают на любом устройстве и в любом браузере, корректно отображаются на всех типах экранов (ПК, смартфоны, планшеты). Благодаря технологии Service Workers данные приложения можно кешировать и работать с ним без интернета. Такие приложения поддерживают добавление на домашний экран через Web App Manifest и оптимизированы для мгновенного запуска даже при медленном соединении и не требуют публикации в App Store/Google Play, поскольку обновляются автоматически. PWA – это некая середина между веб-сайтами и нативными приложениями: они дешевле в разработке, но предлагают многие преимущества нативных решений. Однако для сложных задач (например, AR или фоновая работа) нативные приложения остаются более мощным вариантом. В таблице 1 приведено сравнение между нативными, классическими веб приложениями и PWA:

Таблица - Сравнение видов приложений

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Нативное приложение | Классическое веб приложение | PWA |
| Установка | Да (через магазины) | Нет | Да (через браузер) |
| Оффлайн-работа | Да | Нет (только с кешем) | Да (Service Workers) |
| Доступ к API устройства | Полный | Ограниченный | Ограниченный (камера, геолокация) |
| Производительность | Максимальная | Зависит от браузера | Высокая (близка к нативной) |
| Обновления | Через магазины | Автоматические | Автоматические |
| Распространение | Через App Store/Google Play | Через URL | Через URL, без модерации |
| Стоимость разработки | Высокая (отдельно под iOS/Android) | Низкая | Низкая (один код для всех ОС) |
| Примеры | Instagram, WhatsApp | Обычные сайты (Wikipedia, BBC) | Twitter Lite, Starbucks |

Одним из самых известных примеров успешного внедрения PWA является Twitter Lite. Эта версия социальной сети была разработана для пользователей с медленным интернетом и слабыми устройствами. Благодаря технологии PWA, Twitter Lite загружается почти мгновенно, потребляет меньше трафика и работает в оффлайн-режиме, показывая ранее загруженные твиты. Приложение занимает в 10 раз меньше места, чем нативная версия, и при этом сохраняет ключевые функции, такие как пуши и возможность публикации контента. Другой пример – Starbucks. Компания создала PWA для онлайн-заказов, чтобы охватить пользователей с ограниченным доступом к интернету. Приложение позволяет просматривать меню, выбирать продукты и оформлять заказы даже при нестабильном соединении.

Рассмотрим ключевые технологии PWA:

* Service Workers – это фоновые скрипты, которые работают отдельно от основного потока браузера. Они отвечают за кеширование ресурсов, что позволяет приложению загружаться мгновенно и работать без интернета. Service Workers также обрабатывают push-уведомления и синхронизируют данные при восстановлении соединения;
* Web App Manifest – JSON-файл, который содержит метаданные приложения. Благодаря манифесту PWA можно установить на домашний экран, где оно будет выглядеть как обычное нативное приложение;
* HTTPS – обязательное требование для PWA, обеспечивающее безопасность данных. Без защищённого соединения такие функции, как Service Workers, просто не будут работать. HTTPS предотвращает перехват информации и защищает пользователей от атак.
* Оффлайн-режим и кеширование – одна из главных особенностей PWA. При первом посещении приложение загружает необходимые ресурсы (HTML, CSS, JS, изображения) и сохраняет их в кеше с помощью Service Workers. При повторном открытии или отсутствии интернета данные берутся из локального хранилища, обеспечивая бесперебойную работу. Для управления кешем используются Cache API.

Эти технологии делают PWA мощным инструментом, сочетающим удобство веба с функциональностью нативных приложений.

# Подготовка к разработке PWA

Для создания PWA с использованием базовых веб-технологий потребовалась особая организация проекта. Основу составила классическая структура файлов: index.html как точка входа, директории для CSS и JavaScript, а также отдельные JSON-файлы для конфигурации.

Ключевым моментом подготовки стала настройка Web App Manifest (manifest.json). Этот файл был создан вручную с указанием всех необходимых метаданных: названия приложения, короткого имени, цветов темы, иконок различного размера и ориентации дисплея. Особое внимание уделилось правильному пути к иконкам и корректному указанию start\_url.

Для реализации оффлайн-функциональности был написан кастомный Service Worker (sw.js). Его регистрация выполнялась через отдельный JavaScript-файл (app.js), где проверялась поддержка браузером и осуществлялась корректная установка. В Service Worker были прописаны стратегии кеширования для основных ресурсов (HTML, CSS, JS) и статических данных.

Так как PWA требуют HTTPS-соединения, на этапе разработки использовался локальный сервер с поддержкой SSL. Для этого применялся инструмент http-server с соответствующими параметрами, что позволило тестировать все функции без необходимости развертывания на сервере.

Адаптивный дизайн реализовывался через медиа-запросы в CSS и flexible-верстку, что обеспечило правильное отображение на устройствах с разными размерами экранов.

Для оптимизации производительности были минимизированы все статические ресурсы. CSS и JavaScript файлы подвергались сжатию, изображения оптимизировались с сохранением приемлемого качества. Это было необходимо для уменьшения скорости первой загрузки.

# Разработка PWA

Разработка началась с тщательного проектирования структуры приложения. Были выделены ключевые разделы и какие из них должны быть доступны в оффлайн-режиме. На этом этапе мы создали базовую HTML-разметку с учетом требований PWA, включая обязательный мета-тег viewport и ссылку на манифест.

Следующим шагом стала разработка Web App Manifest. Был создан manifest.json файл, где подробно описаны метаданные приложения: его название, короткое имя для домашнего экрана, цветовую схему, ориентацию экрана и набор иконок для разных устройств. Особое внимание было уделено правильному указанию путей к иконкам, чтобы они корректно отображались после установки приложения.

Реализация Service Worker стала наиболее сложной частью проекта. Был написан отдельный sw.js файл, в котором реализована логику кэширования ресурсов при первой загрузке. Для этого использовался Cache API, чтобы создавать отдельный кэш для статических ресурсов и данных приложения. Была предусмотрена стратегия обновления кэша при выходе новых версий приложения, чтобы пользователи всегда получали актуальный контент.

Интеграция Service Worker с основным приложением потребовала особого внимания. В главном JavaScript-файле был добавлен код для регистрации Service Worker с проверкой поддержки браузером. Также была реализована систему уведомлений о готовности приложения к работе в оффлайн-режиме.

Код приложения и его компонентов указан в Приложении 1.

# Демонстрация результатов работы

В качестве тестового приложения была использована концепция списка дел (To-Do листа). В качестве образца был взят существующий сервис Trello. В процессе разработки было разработано простое приложение с функционалом интерактивной доски. Приложение отображено на рисунке 1:

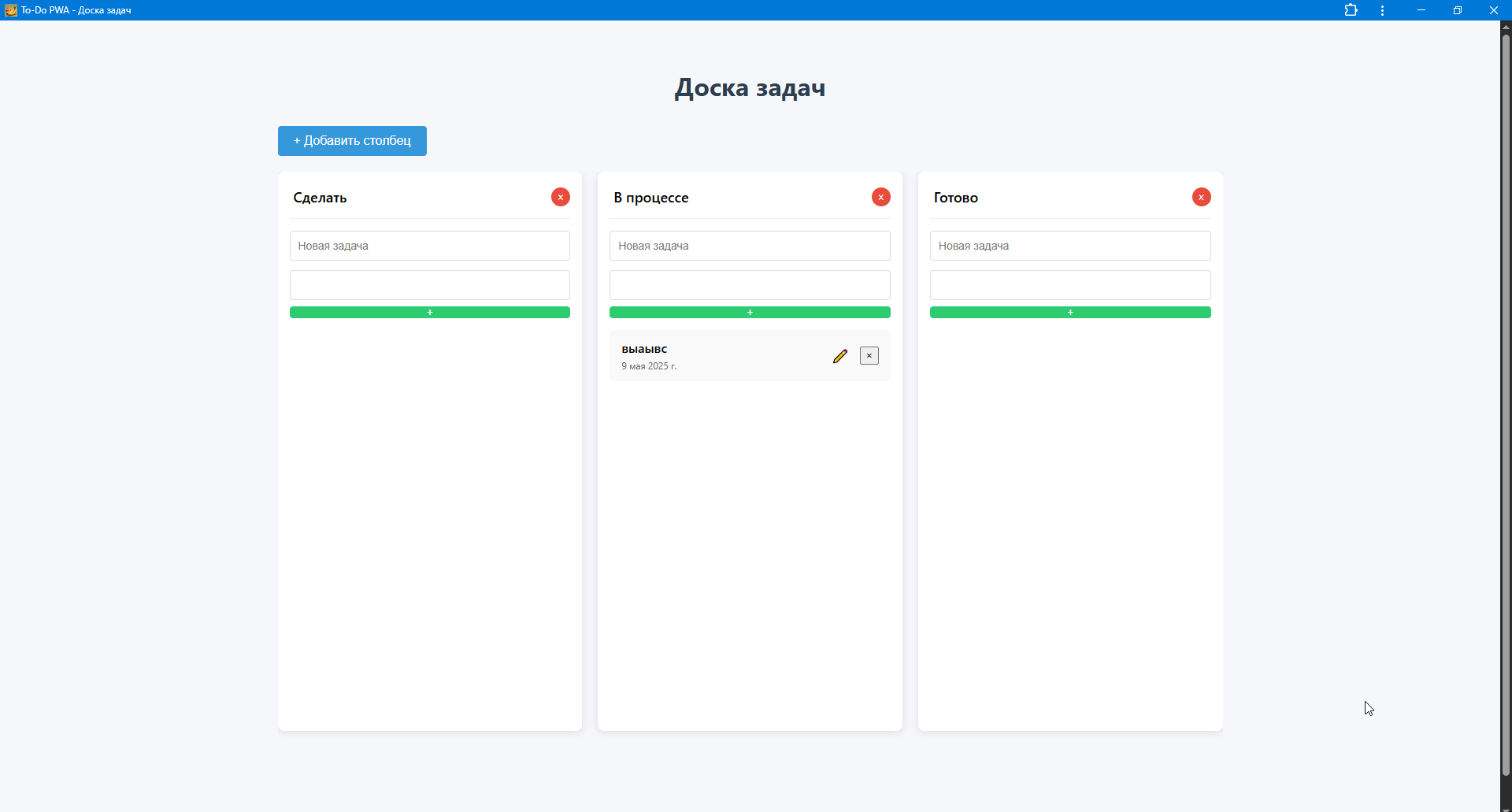


Рисунок – Внешний вид разработанного PWA

Изначально приложение открывается в браузере, после чего становится возможным его установка через соответствующие элементы интерфейса браузера. На рисунке 2 показано окно с предложением установить приложение:

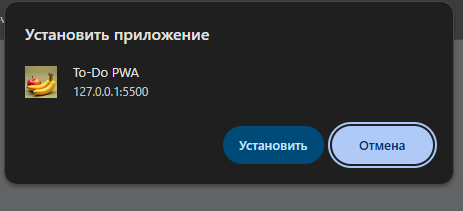


Рисунок - Предложение установить приложение

После установки приложение перемещается на рабочий стол в виде ярлыка. Это видно на рисунке 3:



Рисунок - Ярлык PWA

Приложение позволяет добавлять и удалять столбцы задач. Кроме того, эти столбцы можно переименовать. На рисунках 4, 5 и 6 показаны добавление, удаление и переименование столбца соответственно:

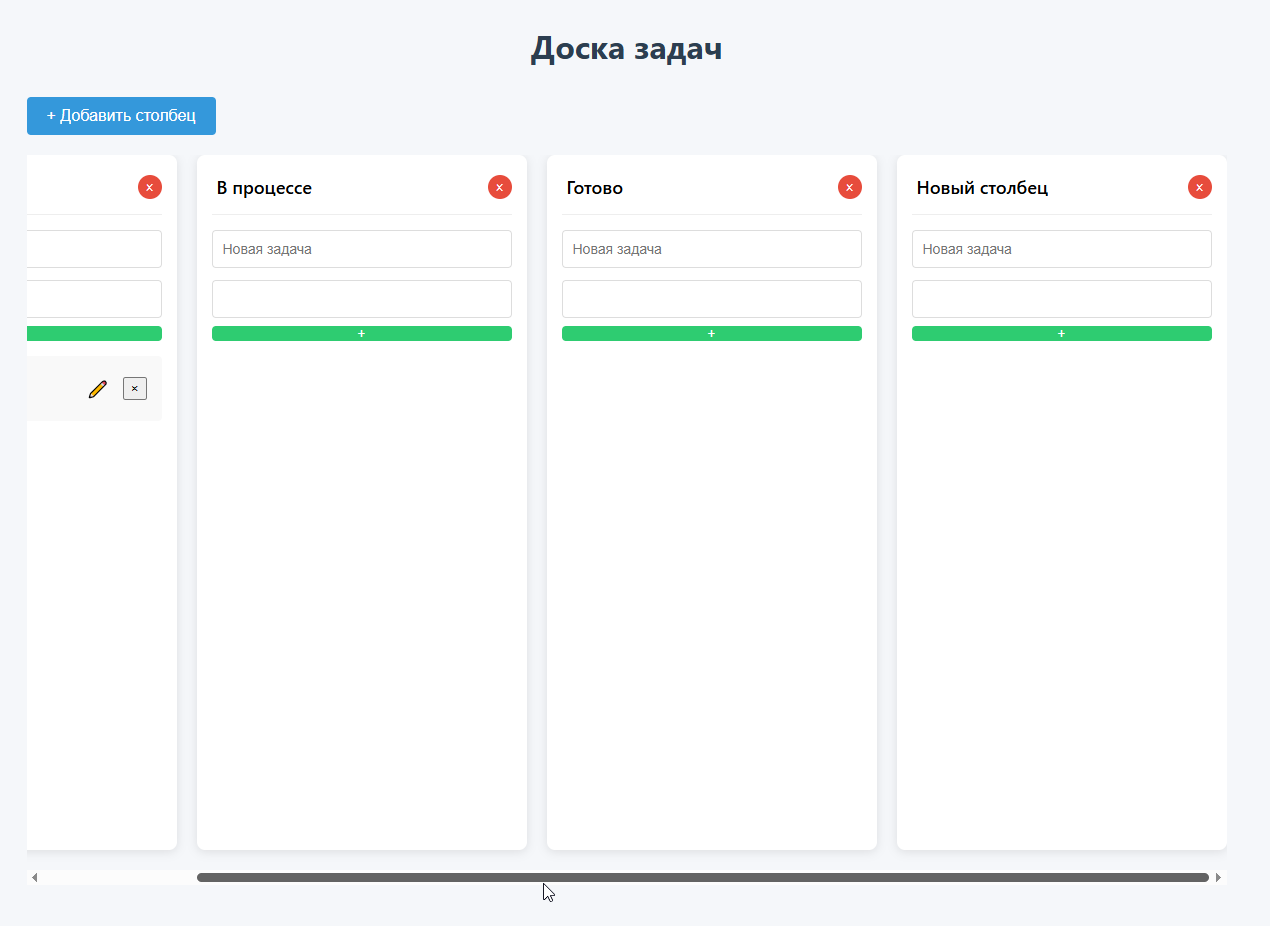


Рисунок - Добавлен новый столбец

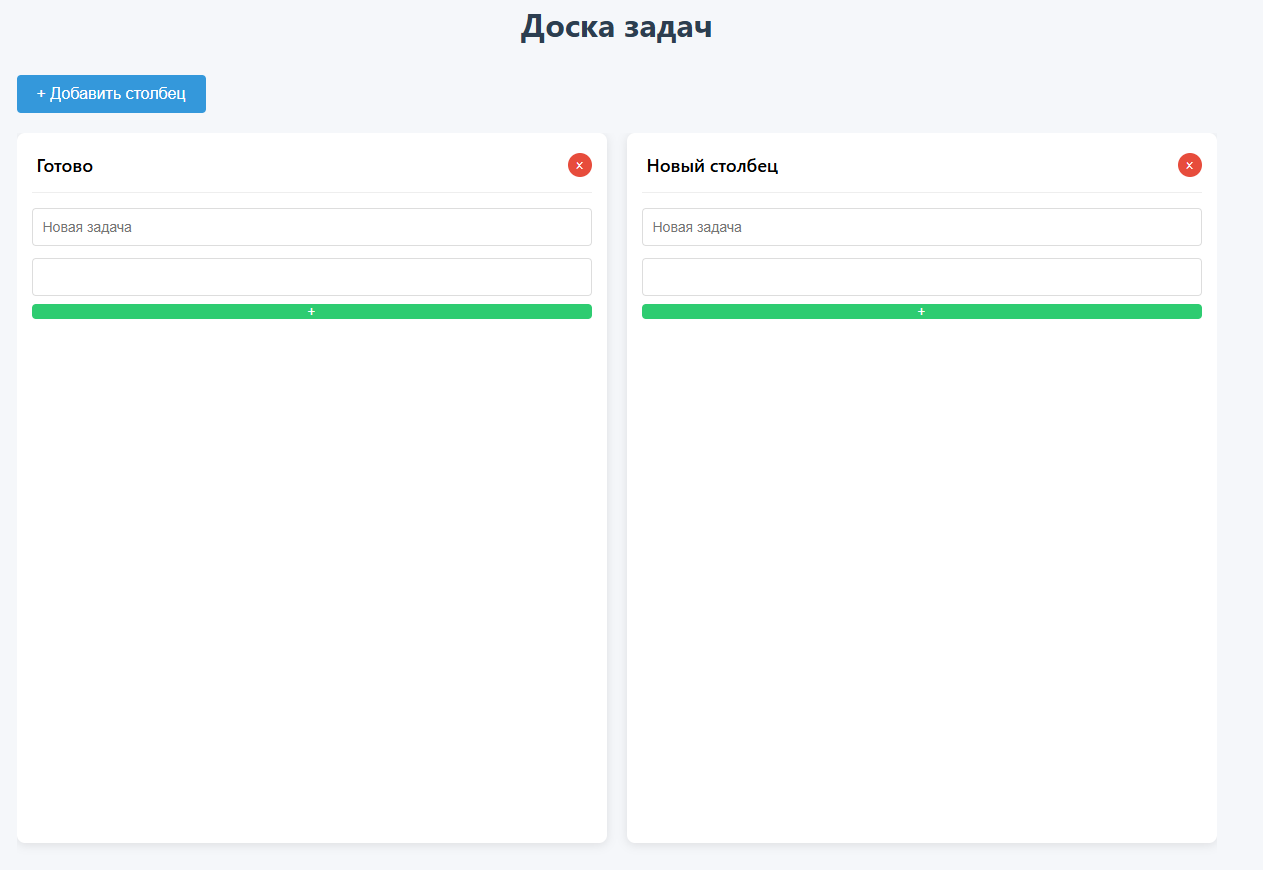


Рисунок - Удалено два существующих столбца

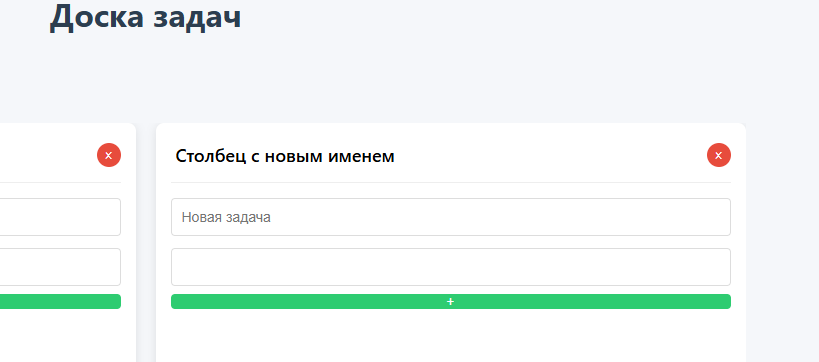


Рисунок - Переименован столбец

В приложении можно создавать задачи. Чтобы добавить задачу следует указать название и дату задачи, после чего требуется нажать зеленую кнопку со знаком «+». После создания задачи её можно отредактировать, изменив название или дату. Кроме того, задачу можно передвинуть в другой столбец или вовсе удалить. На рисунках 7, 8, 9, 10 и 11 показана работа указанных функций.

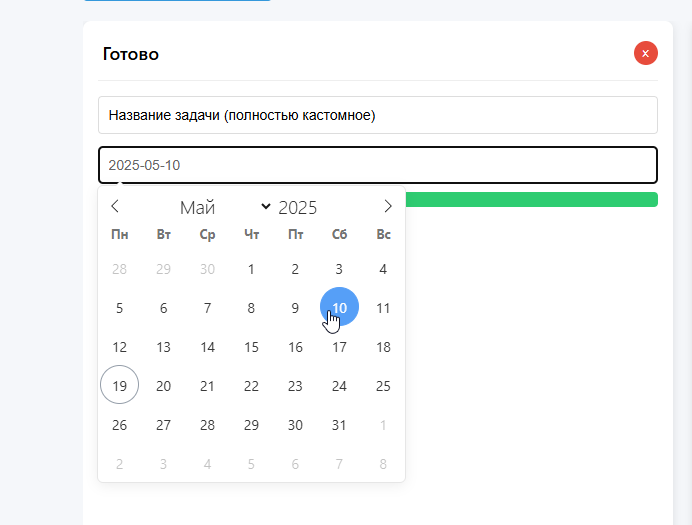


Рисунок - Выбор даты для создания задачи

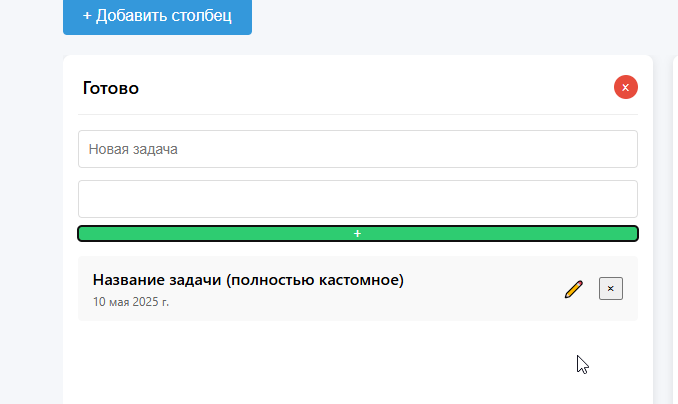


Рисунок - Созданная задача

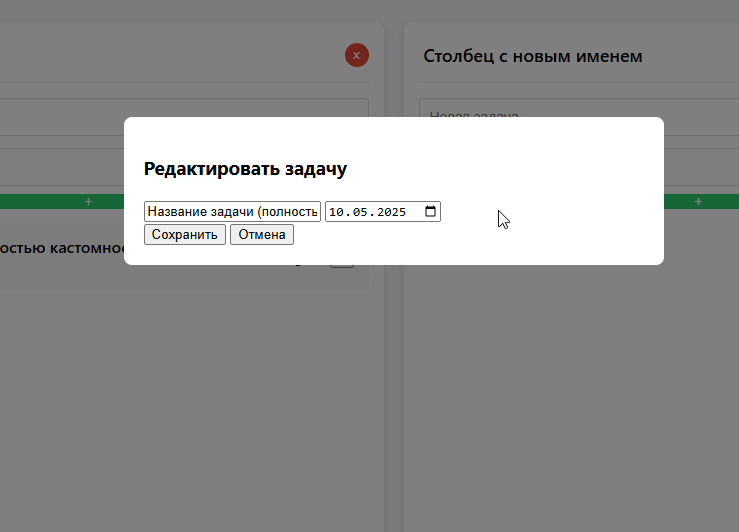


Рисунок - Окно редактирования задачи

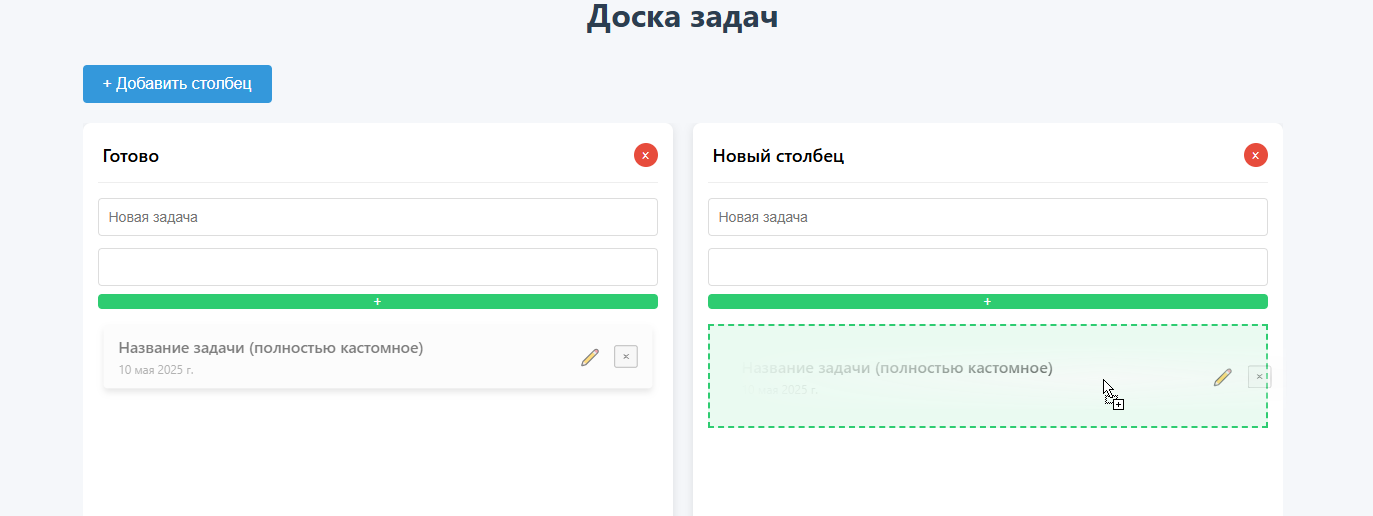


Рисунок - Процесс переноса задачи из одного столбца в другой

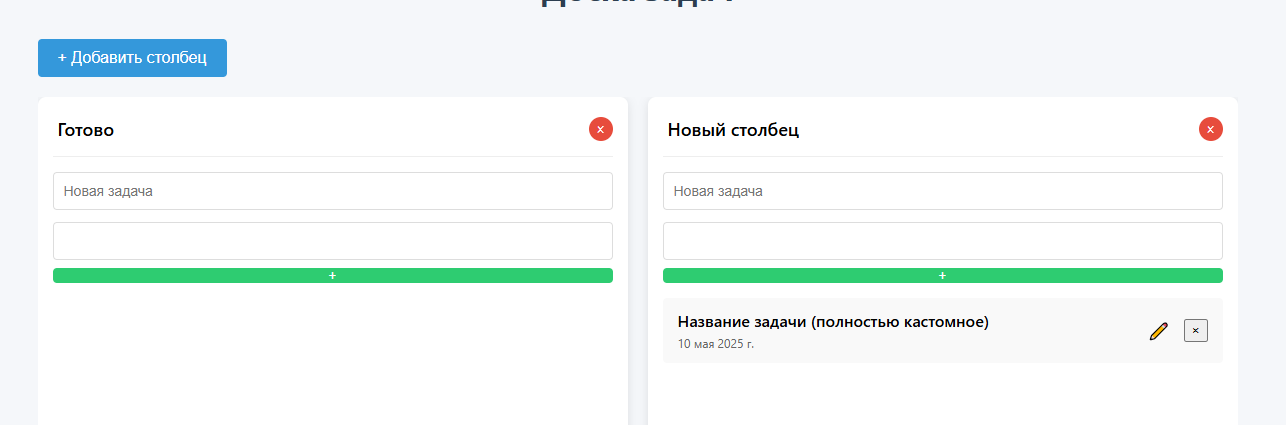


Рисунок - Задача перенесена в другой столбец

# Приложение 1. Код элементов приложения

Код «index.html»:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Доска задач</title>

<link rel="stylesheet" href="styles.css">

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

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/flatpickr/dist/flatpickr.min.css">

</head>

<body>

<div class="container">

<h1>Доска задач</h1>

<button id="add-column">+ Добавить столбец</button>

<div id="columns" class="columns"></div>

</div>

<div id="edit-modal" class="modal">

<div class="modal-content">

<h3>Редактировать задачу</h3>

<form id="edit-form">

<input type="text" id="edit-text" required>

<input type="date" id="edit-date" class="flatpickr">

<input type="hidden" id="edit-task-id">

<input type="hidden" id="edit-column-id">

<div class="modal-buttons">

<button type="submit">Сохранить</button>

<button type="button" id="cancel-edit">Отмена</button>

</div>

</form>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/flatpickr"></script>

<script src="https://npmcdn.com/flatpickr/dist/l10n/ru.js"></script>

<script src="app.js"></script>

<script>

if ('serviceWorker' in navigator) {

navigator.serviceWorker.register('sw.js', { scope: './' })

.then(() => console.log('SW зарегистрирован'))

.catch(err => console.log('Ошибка SW:', err));

}

</script>

</body>

</html>

Код styles.css:

body {

font-family: 'Segoe UI', Arial, sans-serif;

margin: 0;

padding: 20px;

background-color: #f5f7fa;

min-height: 100vh;

}

.container {

margin: 0 auto;

padding: 20px;

}

h1 {

color: #2c3e50;

text-align: center;

margin-bottom: 30px;

}

#add-column {

background-color: #3498db;

color: white;

border: none;

padding: 10px 20px;

font-size: 16px;

border-radius: 4px;

cursor: pointer;

margin-bottom: 20px;

transition: background-color 0.3s;

}

#add-column:hover {

background-color: #2980b9;

}

.columns {

display: flex;

gap: 20px;

overflow-x: auto;

padding-bottom: 20px;

min-height: 70vh;

}

.column {

min-width: 300px;

background: white;

padding: 15px;

border-radius: 8px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

flex: 1;

}

.column-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 15px;

padding-bottom: 10px;

border-bottom: 1px solid #eee;

}

.column-title {

font-weight: 600;

font-size: 18px;

cursor: pointer;

padding: 5px;

flex-grow: 1;

margin-right: 10px;

}

.column-title:focus {

outline: none;

background-color: #f0f7ff;

border-radius: 4px;

}

.delete-column {

background: #e74c3c;

color: white;

border: none;

width: 24px;

height: 24px;

border-radius: 50%;

cursor: pointer;

display: flex;

align-items: center;

justify-content: center;

font-size: 14px;

}

.delete-column:hover {

background: #c0392b;

}

.task-form {

display: flex;

margin-bottom: 15px;

gap: 10px;

}

.task-form input {

flex: 1;

padding: 10px;

border: 1px solid #ddd;

border-radius: 4px;

font-size: 14px;

}

.task-form button {

background: #2ecc71;

color: white;

border: none;

padding: 0 15px;

border-radius: 4px;

cursor: pointer;

}

.task-form button:hover {

background: #27ae60;

}

.task-list {

list-style: none;

padding: 0;

margin: 0;

min-height: 100px;

}

.task-list li {

background: #f9f9f9;

padding: 12px 15px;

margin-bottom: 8px;

border-radius: 4px;

display: flex;

justify-content: space-between;

align-items: center;

transition: transform 0.2s;

}

.task-list li:hover {

transform: translateX(5px);

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

.task-list li span {

flex-grow: 1;

}

.delete-task {

background: #e74c3c;

color: white;

border: none;

width: 22px;

height: 22px;

border-radius: 50%;

cursor: pointer;

display: flex;

align-items: center;

justify-content: center;

font-size: 12px;

margin-left: 10px;

}

.delete-task:hover {

background: #c0392b;

}

@media (max-width: 768px) {

.columns {

flex-direction: column;

}

.column {

min-width: auto;

}

}

.task-list li {

cursor: grab;

transition: transform 0.2s, box-shadow 0.2s;

}

.task-list li.dragging {

opacity: 0.5;

transform: scale(0.98);

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

}

.task-list.drag-over {

background-color: rgba(46, 204, 113, 0.1);

border: 2px dashed #2ecc71;

transition: background-color 0.3s;

}

.task-list li:active {

cursor: grabbing;

}

@keyframes fadeIn {

from { opacity: 0; transform: translateY(10px); }

to { opacity: 1; transform: translateY(0); }

}

.task-list li {

animation: fadeIn 0.3s ease-out;

}

.modal {

display: none;

position: fixed;

z-index: 100;

left: 0;

top: 0;

width: 100%;

height: 100%;

background-color: rgba(0,0,0,0.5);

}

.modal-content {

background: white;

margin: 15% auto;

padding: 20px;

width: 80%;

max-width: 500px;

border-radius: 8px;

}

.task-content {

flex-grow: 1;

}

.task-date {

display: block;

font-size: 12px;

color: #666;

margin-top: 4px;

}

.task-actions {

display: flex;

gap: 8px;

}

.edit-btn {

background: none;

border: none;

cursor: pointer;

font-size: 16px;

}

.task-item {

padding: 12px;

margin-bottom: 8px;

background: #fff;

border-radius: 4px;

box-shadow: 0 1px 3px rgba(0,0,0,0.1);

}

.task-text {

display: block;

margin-bottom: 4px;

font-weight: 500;

}

.task-date {

font-size: 12px;

color: #666;

}

.task-form {

display: flex;

flex-direction: column;

gap: 8px;

margin-top: 10px;

}

.task-form input,

.task-form button {

width: 100%;

box-sizing: border-box;

}

Код «app.js»:

let columns = JSON.parse(localStorage.getItem('columns')) || [

{

id: 1,

title: 'Сделать',

tasks: [

{

id: Date.now(),

text: 'Пример задачи',

date: new Date().toISOString().split('T')[0]

}

]

},

{

id: 2,

title: 'В процессе',

tasks: []

},

{

id: 3,

title: 'Готово',

tasks: []

}

];

function saveToLocalStorage() {

localStorage.setItem('columns', JSON.stringify(columns));

}

function renderColumns() {

const columnsContainer = document.getElementById('columns');

columnsContainer.innerHTML = '';

columns.forEach(column => {

const columnElement = document.createElement('div');

columnElement.className = 'column';

columnElement.dataset.id = column.id;

columnElement.innerHTML = `

<div class="column-header">

<div class="column-title" contenteditable="true">${column.title}</div>

<button class="delete-column">×</button>

</div>

<form class="task-form">

<input type="text" class="task-input" placeholder="Новая задача" required>

<input type="date" class="task-date">

<button type="submit">+</button>

</form>

<ul class="task-list"></ul>

`;

flatpickr(columnElement.querySelector('.task-date'), {

dateFormat: "Y-m-d",

locale: "ru"

});

columnsContainer.appendChild(columnElement);

renderTasks(column.id);

const taskList = columnElement.querySelector('.task-list');

taskList.addEventListener('dragover', handleDragOver);

taskList.addEventListener('dragleave', handleDragLeave);

taskList.addEventListener('drop', handleDrop);

});

}

function renderTasks(columnId) {

const column = columns.find(c => c.id == columnId);

const taskList = document.querySelector(`.column[data-id="${columnId}"] .task-list`);

taskList.innerHTML = '';

column.tasks.forEach(task => {

const taskElement = document.createElement('li');

taskElement.dataset.id = task.id;

taskElement.draggable = true;

taskElement.innerHTML = `

<div class="task-content">

<span class="task-text">${task.text}</span>

<span class="task-date">${formatDate(task.date)}</span>

</div>

<div class="task-actions">

<button class="edit-btn">✏️</button>

<button class="delete-btn">×</button>

</div>

`;

taskElement.addEventListener('dragstart', handleDragStart);

taskList.appendChild(taskElement);

});

}

function formatDate(dateString) {

if (!dateString) return 'Без срока';

const date = new Date(dateString);

return date.toLocaleDateString('ru-RU', {

year: 'numeric',

month: 'long',

day: 'numeric'

});

}

document.getElementById('add-column').addEventListener('click', () => {

const newColumn = {

id: Date.now(),

title: 'Новый столбец',

tasks: []

};

columns.push(newColumn);

saveToLocalStorage();

renderColumns();

});

document.addEventListener('click', (e) => {

if (e.target.classList.contains('delete-column')) {

const columnId = parseInt(e.target.closest('.column').dataset.id);

columns = columns.filter(c => c.id !== columnId);

saveToLocalStorage();

renderColumns();

}

if (e.target.classList.contains('delete-btn')) {

const taskElement = e.target.closest('li');

const columnId = parseInt(taskElement.closest('.column').dataset.id);

const taskId = parseInt(taskElement.dataset.id);

const column = columns.find(c => c.id === columnId);

column.tasks = column.tasks.filter(t => t.id !== taskId);

saveToLocalStorage();

renderColumns();

}

if (e.target.classList.contains('edit-btn')) {

const taskElement = e.target.closest('li');

const taskId = parseInt(taskElement.dataset.id);

const columnId = parseInt(taskElement.closest('.column').dataset.id);

const column = columns.find(c => c.id === columnId);

const task = column.tasks.find(t => t.id === taskId);

document.getElementById('edit-text').value = task.text;

document.getElementById('edit-date').value = task.date || '';

document.getElementById('edit-task-id').value = taskId;

document.getElementById('edit-column-id').value = columnId;

document.getElementById('edit-modal').style.display = 'block';

}

if (e.target.id === 'cancel-edit') {

document.getElementById('edit-modal').style.display = 'none';

}

});

document.addEventListener('submit', (e) => {

if (e.target.classList.contains('task-form')) {

e.preventDefault();

const input = e.target.querySelector('.task-input');

const dateInput = e.target.querySelector('.task-date');

const taskText = input.value.trim();

if (taskText) {

const columnId = parseInt(e.target.closest('.column').dataset.id);

const column = columns.find(c => c.id === columnId);

column.tasks.push({

id: Date.now(),

text: taskText,

date: dateInput.value || null

});

saveToLocalStorage();

renderTasks(columnId);

input.value = '';

dateInput.value = '';

}

}

if (e.target.id === 'edit-form') {

e.preventDefault();

const taskId = parseInt(document.getElementById('edit-task-id').value);

const columnId = parseInt(document.getElementById('edit-column-id').value);

const newText = document.getElementById('edit-text').value.trim();

const newDate = document.getElementById('edit-date').value;

const column = columns.find(c => c.id === columnId);

const task = column.tasks.find(t => t.id === taskId);

if (task) {

task.text = newText;

task.date = newDate || null;

saveToLocalStorage();

renderColumns();

}

document.getElementById('edit-modal').style.display = 'none';

}

});

function handleDragStart(e) {

e.dataTransfer.setData('text/plain', JSON.stringify({

taskId: parseInt(e.target.dataset.id),

sourceColumnId: parseInt(e.target.closest('.column').dataset.id)

}));

e.target.classList.add('dragging');

}

function handleDragOver(e) {

e.preventDefault();

if (e.target.classList.contains('task-list') || e.target.closest('.task-list')) {

const taskList = e.target.classList.contains('task-list')

? e.target

: e.target.closest('.task-list');

taskList.classList.add('drag-over');

}

}

function handleDragLeave(e) {

if (e.target.classList.contains('task-list')) {

e.target.classList.remove('drag-over');

}

}

function handleDrop(e) {

e.preventDefault();

const taskList = e.target.closest('.task-list');

taskList.classList.remove('drag-over');

const data = JSON.parse(e.dataTransfer.getData('text/plain'));

const { taskId, sourceColumnId } = data;

const targetColumnId = parseInt(taskList.closest('.column').dataset.id);

moveTask(sourceColumnId, targetColumnId, taskId);

}

function moveTask(sourceColumnId, targetColumnId, taskId) {

const sourceColumn = columns.find(c => c.id === sourceColumnId);

const targetColumn = columns.find(c => c.id === targetColumnId);

const taskIndex = sourceColumn.tasks.findIndex(t => t.id === taskId);

if (taskIndex === -1) return;

const [task] = sourceColumn.tasks.splice(taskIndex, 1);

targetColumn.tasks.push(task);

saveToLocalStorage();

renderColumns();

}

document.addEventListener('DOMContentLoaded', () => {

renderColumns();

if ('serviceWorker' in navigator) {

navigator.serviceWorker.register('sw.js', { scope: './' })

.then(() => console.log('SW зарегистрирован'))

.catch(err => console.log('Ошибка SW:', err));

}

});

Код «sw.js»:

const CACHE\_NAME = 'pwa-kanban-cache-v1';

const urlsToCache = [

'./',

'./index.html',

'./styles.css',

'./app.js',

'./icon-192x192.png',

'./icon-512x512.png'

];

self.addEventListener('install', (event) => {

event.waitUntil(

caches.open(CACHE\_NAME)

.then(cache => cache.addAll(urlsToCache))

.catch(err => console.log('Ошибка кеширования:', err))

);

});

self.addEventListener('fetch', (event) => {

event.respondWith(

caches.match(event.request)

.then(response => {

return response || fetch(event.request);

})

);

});

Код «manifest.json»:

{

"name": "To-Do PWA",

"short\_name": "Todo",

"start\_url": "/",

"display": "standalone",

"background\_color": "#ffffff",

"theme\_color": "#0078d7",

"icons": [

{

"src": "icon-192x192.png",

"sizes": "192x192",

"type": "image/png"

},

{

"src": "icon-512x512.png",

"sizes": "512x512",

"type": "image/png"

}

]

}