**Документация для HTML-страницы списка дел**

Этот HTML-файл предназначен для веб-приложения списка дел с интерактивными функциями, такими как добавление, сортировка и управление задачами. Ниже приведено подробное описание структуры кода:

**Структура файлов**

* **index.html**: Основной файл, содержащий структуру приложения списка дел.
* **style.css**: Внешний файл CSS для кастомных стилей.
* **script.js**: Внешний файл JavaScript для динамической функциональности.

**Структура HTML**

**1. Раздел <head>**

* **Мета-теги**:
  + <meta charset="UTF-8">: Устанавливает кодировку символов.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Обеспечивает адаптивность страницы на мобильных устройствах.
* **Заголовок**: Устанавливает заголовок страницы как "To-Do List".
* **Ссылки на CSS**:
  + Bootstrap CSS для макета и компонентов.
  + style.css для кастомных стилей.

**2. Раздел <body>**

**Экран приветствия**

* **ID**: welcomeScreen.
* **Описание**:
  + Полноэкранный блок, который приветствует пользователя в приложении.
  + Содержит заголовок и кнопку для перехода к созданию задач.
* **Элементы**:
  + <h1>: Заголовок с текстом "Welcome to Your To-Do List".
  + <button>: Кнопка с текстом "Start Creating Tasks" (ID: startBtn).

**Приложение списка дел**

* **ID**: taskApp.
* **Описание**:
  + Главный интерфейс для работы с задачами. Изначально скрыт (через style="display: none;").
* **Элементы**:
  + **Форма добавления задач**:
    - Поле для ввода текста задачи (id="taskInput").
    - Поле выбора даты дедлайна (id="deadlineInput").
    - Флажок для пометки задачи как приоритетной (id="priorityInput").
    - Кнопка для добавления задачи (id="addTaskBtn").
  + **Активные задачи**:
    - Заголовок "Active Tasks".
    - Блок с сообщением о пустом списке задач (id="emptyTaskMessage") с изображением и текстом.
    - Список активных задач (id="taskList") — изначально скрыт.
  + **Завершенные задачи**:
    - Блок для завершенных задач (id="completedTasksSection") — изначально скрыт.
    - Список завершенных задач (id="completedTaskList").
    - Кнопка для очистки завершенных задач (id="clearCompletedBtn").

**Футер**

* Размещен внизу страницы, содержит текст:
  + © Adiya ;). Made with ❤️.

**Подключаемые ресурсы**

1. **Bootstrap**:
   * CSS и JavaScript для использования стандартных стилей и компонентов.
2. **Внешние файлы**:
   * style.css — для стилизации приложения.
   * script.js — для добавления интерактивности.

**Дополнительные заметки**

* **Кастомные стили** (через style.css):
  + Включают розовую тему кнопок (btn-pink) и элементы с анимацией.
* **Функциональность JavaScript** (через script.js):
  + Управляет видимостью экранов (приветствие или список задач).
  + Обрабатывает добавление, редактирование, удаление и сортировку задач.
  + Обеспечивает отображение завершенных задач и очищение их списка.

**Документация для JavaScript-кода списка дел**

Этот JavaScript-код реализует интерактивные функции для управления задачами в приложении списка дел. Приведены описания каждой функции, структуры и логики.

**Основные функции**

**Инициализация**

Код внутри DOMContentLoaded выполняется после полной загрузки HTML-документа:

* Объявление переменных для элементов DOM.
* Настройка начального экрана (welcomeScreen) и переключение на основной интерфейс (taskApp) с анимацией.

**Основные переменные**

* tasks: Массив для хранения активных задач.
* completedTasks: Массив для завершенных задач.
* **Элементы DOM**:
  + welcomeScreen, taskApp: Экран приветствия и основной интерфейс.
  + taskInput, deadlineInput, priorityInput: Поля ввода задачи, даты дедлайна и флажок приоритета.
  + addTaskBtn: Кнопка добавления задачи.
  + taskList, completedTaskList: Списки активных и завершенных задач.
  + clearCompletedBtn: Кнопка очистки завершенных задач.

**Функциональность**

**1. Переключение экранов**

startBtn.addEventListener('click', () => { ... });

* При нажатии на кнопку "Start Creating Tasks":
  + Скрывается экран приветствия.
  + Отображается основной интерфейс.
  + Выполняется плавная прокрутка с анимацией.

**2. Добавление задач**

function addTask() { ... }

addTaskBtn.addEventListener('click', addTask);

* Проверяет, заполнены ли все поля (taskTitle, deadline).
* Создает объект задачи с уникальным id, заголовком, дедлайном и флагом приоритета.
* Добавляет задачу в массив tasks, сортирует его и перерисовывает список.

**3. Сортировка задач**

function sortTasks() { ... }

* Сортирует задачи по следующим критериям:
  1. Задачи с флажком приоритета отображаются выше.
  2. Задачи сортируются по дате дедлайна.

**4. Отображение задач**

function renderTasks() { ... }

* Если массив задач пуст, показывает сообщение о пустом списке.
* В противном случае перерисовывает задачи:
  + Высокоприоритетные задачи выделяются классом list-group-item-danger.
  + Задачи с дедлайном "завтра" выделяются дополнительным эффектом (glow).
* Генерирует HTML-код для каждой задачи:
  + Кнопки "Delete" и "Edit".
  + Индикатор завершения задачи.

**5. Редактирование задачи**

window.editTask = function (taskId) { ... }

* Находит задачу по id.
* Позволяет редактировать заголовок задачи через prompt.

**6. Завершение задачи**

window.markComplete = function (taskId) { ... }

* Перемещает задачу из массива tasks в completedTasks.
* Удаляет задачу из списка активных задач и обновляет интерфейс.

**7. Удаление задачи**

window.deleteTask = function (taskId) { ... }

* Удаляет задачу из массива tasks по id.
* Обновляет список задач.

**8. Отображение завершенных задач**

function renderCompletedTasks() { ... }

* Если массив завершенных задач пуст, скрывает секцию.
* В противном случае отображает задачи с отметкой "Completed".

**9. Очистка завершенных задач**

clearCompletedBtn.addEventListener('click', () => { ... });

* Очищает массив completedTasks и обновляет интерфейс.

**Дополнительные функции**

* **Сортировка после изменений**: Все изменения в задачах (добавление, завершение, удаление) вызывают функцию sortTasks для обновления порядка.
* **Стилизация**:
  + Высокоприоритетные задачи и задачи с дедлайном "завтра" выделяются визуально.
  + Скрытие/показ секций реализовано с использованием стилей и JavaScript.

**Ключевые моменты**

* Весь функционал завязан на динамическое изменение DOM через JavaScript.
* Код использует события и функции для разделения задач:
  + addTask — добавление.
  + sortTasks — сортировка.
  + renderTasks и renderCompletedTasks — отображение.
  + editTask, markComplete, deleteTask — управление задачами.

**Документация для CSS-стилей**

Этот CSS-файл отвечает за оформление веб-приложения списка дел. Рассмотрим основные стили, их назначение и ключевые моменты реализации.

**1. Общие стили**

**Шрифты и цвета**

body {

font-family: 'Comic Sans MS', cursive, sans-serif;

background-color: #ffe6f2;

color: #333;

}

* **font-family**: Используется шрифт Comic Sans MS для создания игривого и дружелюбного интерфейса.
* **background-color**: Фон страницы — нежно-розовый оттенок (#ffe6f2).
* **color**: Основной цвет текста — темно-серый (#333).

**Заголовки**

h1, .welcome-title {

color: #ff66a3;

}

* Заголовки (<h1> и .welcome-title) окрашены в яркий розовый (#ff66a3).

**Кнопки**

.btn-pink {

background-color: #ff66a3;

color: #fff;

border: none;

}

.btn-pink:hover {

background-color: #ff4d94;

}

* **Класс .btn-pink**:
  + Основной фон: насыщенно-розовый (#ff66a3).
  + Текст белого цвета.
  + При наведении цвет становится темнее (#ff4d94).

**2. Стили начального экрана**

**Экран приветствия**

#welcomeScreen {

background-color: #ffe6f2;

color: #ff66a3;

transition: opacity 0.5s ease, transform 0.5s ease;

}

#welcomeScreen.hidden {

opacity: 0;

transform: scale(0.9);

pointer-events: none;

}

* Начальный экран (#welcomeScreen):
  + **Фон:** тот же розовый оттенок, что и у тела страницы.
  + **Анимация скрытия**:
    - Снижение прозрачности (opacity).
    - Уменьшение масштаба (transform: scale(0.9)).

**Основной контент приложения**

#taskApp {

opacity: 0;

transform: translateY(20px);

transition: opacity 0.5s ease, transform 0.5s ease;

}

#taskApp.visible {

opacity: 1;

transform: translateY(0);

}

* Элементы приложения (#taskApp):
  + Изначально скрыты (прозрачность 0, смещение вниз).
  + При добавлении класса .visible появляются с анимацией подъема.

**3. Карточки задач**

**Список задач**

.list-group-item {

background-color: #ffe6f2;

position: relative;

transition: all 0.3s;

}

.list-group-item:hover {

background-color: #ffd6e7;

}

* **Задачи (.list-group-item)**:
  + Фон: светло-розовый (#ffe6f2).
  + При наведении фон становится темнее (#ffd6e7).

**Кнопки внутри задач**

.list-group-item .btn {

opacity: 0;

transition: all 0.3s ease, opacity 0.3s;

}

.list-group-item:hover .btn {

opacity: 0.8;

}

* Кнопки ("Delete", "Edit"):
  + Изначально скрыты (opacity: 0).
  + Появляются при наведении на задачу.

**Завершенные задачи**

.list-group-item.completed {

text-decoration: line-through;

color: #82d876;

background-color: #f4f4f4;

}

* Завершенные задачи:
  + Перечеркнутый текст.
  + Цвет текста — зеленоватый (#82d876).
  + Светлый фон (#f4f4f4).

**4. Секция пустого списка**

#emptyTaskMessage img {

max-width: 150px;

}

#emptyTaskMessage {

color: #888;

}

* **Изображение**: ограничено в ширине до 150px.
* **Текст**: серый цвет (#888) для ненавязчивого отображения.

**5. Анимации**

**Подсветка задачи**

.glow {

animation: glowEffect 1s infinite alternate;

}

@keyframes glowEffect {

from {

box-shadow: 0 0 10px #ff99cc;

}

to {

box-shadow: 0 0 20px #ff66a3;

}

}

* Класс .glow:
  + Используется для подсветки задач с дедлайном "завтра".
  + Анимация плавного увеличения/уменьшения теней.

**6. Нижний колонтитул**

footer {

background-color: #ff99cc;

color: white;

}

footer a {

color: white;

text-decoration: underline;

}

footer a:hover {

color: #ff66a3;

}

* Фон: розовый (#ff99cc).
* Ссылки:
  + Подчеркивание по умолчанию.
  + При наведении меняют цвет на ярко-розовый (#ff66a3).

**Итог**

Этот CSS-файл создает визуально привлекательный и дружественный интерфейс с акцентом на розовых тонах. Используются плавные анимации, чтобы улучшить взаимодействие пользователя с приложением.

**Документация для Node.js сервера**

Этот сервер использует модуль http для создания веб-сервера, а также модули fs и path для работы с файловой системой и формирования путей к файлам. Сервер обслуживает статичные файлы, такие как HTML, CSS, JavaScript, изображения, а также возвращает страницу 404, если запрашиваемый файл не найден.

**Описание работы сервера**

1. **Импорт модулей**:
   * http: для создания сервера.
   * fs: для чтения файлов.
   * path: для работы с путями файлов в операционной системе.

const http = require('http');

const fs = require('fs');

const path = require('path');

1. **Порт и создание сервера**: Сервер запускается на порту 3000. Когда сервер получает запрос, он выполняет определённые действия в зависимости от URL.

const PORT = 3000;

const server = http.createServer((req, res) => {

console.log(`Server request: ${req.url}`);

1. **Функция createPath()**: Эта функция формирует полный путь к файлу, который требуется отдать клиенту, исходя из текущего каталога и имени файла.

const createPath = (filePath) => path.resolve(\_\_dirname, 'endterm', filePath);

1. **Логика обработки запросов**: В зависимости от запрашиваемого URL, сервер определяет, какой файл нужно отправить пользователю и с каким типом содержимого (например, HTML, CSS, изображение).
   * **Корневой путь (/)**: Загружает файл endterm.html.
   * **CSS файл**: Если запрашивается файл с расширением .css, сервер отдаёт его с типом text/css.
   * **JavaScript файл**: Для файлов с расширением .js сервер устанавливает тип text/javascript.
   * **Изображения**: Для файлов с расширением .gif, .jpg, или .jpeg сервер устанавливает тип содержимого как соответствующее изображение (image/gif, image/jpeg и т.д.).
   * **Страница 404**: Если запрашиваемый файл не найден, сервер отдаёт страницу с ошибкой 404.

switch (true) {

case req.url === '/':

basePath = createPath('endterm.html'); // Главная страница

break;

case req.url.endsWith('style.css'):

basePath = createPath(req.url.slice(1)); // Путь к CSS

contentType = 'text/css';

break;

case req.url.endsWith('script.js'):

basePath = createPath(req.url.slice(1)); // Путь к JS

contentType = 'text/javascript';

break;

case req.url.endsWith('.gif') || req.url.endsWith('.jpg') || req.url.endsWith('.jpeg'):

basePath = createPath(req.url.slice(1)); // Путь к изображениям

contentType = `image/${path.extname(req.url).slice(1)}`;

break;

default:

basePath = createPath('404.html'); // Страница 404

contentType = 'text/html';

break;

}

1. **Чтение и отправка файлов**: Используется fs.readFile(), чтобы прочитать содержимое файла и отправить его клиенту. Если файл не найден или произошла ошибка, сервер вернёт страницу 404.

fs.readFile(basePath, (err, data) => {

if (err) {

console.error(err);

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end('<h1>Page not found</h1>');

} else {

res.writeHead(200, { 'Content-Type': contentType });

res.end(data);

}

});

1. **Запуск сервера**: Сервер слушает порт 3000 и выводит сообщение в консоль о том, что он запущен.

server.listen(PORT, 'localhost', (error) => {

error ? console.log(error) : console.log(`Server is listening on http://localhost:${PORT}`);

});

**Основные особенности**

1. **Обработка различных типов файлов**:
   * Статичные файлы (HTML, CSS, JS, изображения) обслуживаются через сервер.
   * Используется правильный Content-Type для каждого типа файла.
2. **Перехват 404 ошибок**:
   * Когда файл не найден, сервер возвращает страницу 404 с текстом "Page not found".
3. **Гибкость для добавления других ресурсов**:
   * Логика обработки URL может быть расширена для обслуживания других типов ресурсов, если это необходимо.

**Примечания**

* Чтобы сервер работал корректно, все файлы (например, endterm.html, style.css, script.js) должны находиться в каталоге endterm.
* Сервер поддерживает работу с изображениями и стандартными веб-ресурсами, такими как HTML, CSS и JavaScript.