Проект «Список дел»

*#Todo List*

Требования к проекту (*бизнес-логика*): (вопрос «Что?» (не «Как?» и не «Почему?»).

1. Получение данных с сервера, отрисовка их на странице.
2. Через форму создавать новые задачи, с отправкой запроса на сервер.
3. Выбор пользователя из списка доступных.
4. Изменять статус завершённости по чекбоксу.
5. Удалять задачи

UI:

1. Шапка с названием проекта
2. Форма: текстовое поле, select для выбора ползователя и кнопка
3. Список всех дел
4. Задача:

* Текст
* ЧекБокс
* Крестик и удаление

Шаги:

1. Базовая разметка и стили
2. Получение задач и пользователей с сервера
3. Отрисовать задачи на странице динамически
4. Добавить пользователей в выпадающий список
5. Логика добавление задачи
6. Логика изменения статуса
7. Логика удаления
8. Информирование об ошибках

***Await*** - как правило лежит внутри *асихронных функций*. Запускает функцию после остальных синхронных функций.

***`fetch`*** используется для инициирования сетевых запросов и возвращает `Promise`, который разрешается, когда запрос завершен.

*Немного о методах:*

1. **\*\*GET\*\*:** *Используется для запроса данных* с сервера. Параметры могут передаваться в URL. Этот метод является \*\*идемпотентным\*\*, что означает, что многократные одинаковые запросы должны возвращать одинаковый результат.

2. **\*\*POST\*\*:** Предназначен для *отправки данных на сервер* для создания нового ресурса. Данные отправляются в теле запроса (request body). Этот метод не идемпотентный, то есть повторяющиеся запросы могут возвращать разные результаты, например, при каждом запросе создается *новая запись*.

3. **\*\*PUT\*\*:** *Используется для обновления существующего ресурса на сервере*. Если ресурс не существует, метод PUT может быть использован для его создания. Подобно методу POST, данные также отправляются в теле запроса.

4. **\*\*DELETE\*\*:** Предназначен *для удаления ресурса*, указанного в URL запроса.

5. **\*\*PATCH\*\*:** *Используется для частичного обновления* ресурса на сервере. Это означает, что PATCH обновляет только те данные, которые указаны в запросе, в отличие от PUT, который заменяет данные полностью.

const todo = todoList.querySelector(`**[data-id**="${todoId}"**]**`) – через квадратные скобки мы обращаемся к атрибутам.

**`createDocumentFragment()`** *создает фрагмент в памяти*, но не добавляет его непосредственно в DOM. После того как все необходимые узлы (элементы) добавлены в фрагмент, фрагмент может быть добавлен в реальный узел DOM, например, с помощью метода `appendChild()`.

**Типы основных ошибок в js:**

* *ReferenceError*

Создает экземпляр, представляющий ошибку, возникающую при отмене ссылок на недопустимую ссылку.

* *SyntaxError*

Создает экземпляр, представляющий синтаксическую ошибку.

* *TypeError*

Создает экземпляр, представляющий ошибку, возникающую, когда переменная или параметр не имеют допустимого типа

Частный случай: **Объект ReferenceError** представляет ошибку при ссылке на переменную, которая не существует (или еще не инициализирована) в текущей области.

*todos?\_limit=15* – установка определённого лимита получаемых дел.

Замыкание. Пример замыкания в JavaScript:

```javascript

function createCounter() {

let count = 0;

return ***function() {***

***count++;***

***console.log(count);***

***};***

}

const counter = createCounter();

counter(); // Выведет: 1

counter(); // Выведет: 2

Это один из примеров замыкания в JavaScript, где внутренняя функция сохраняет доступ к переменным из внешней области видимости (в данном случае, `count`).

1. ***\*\*Каррирование\*\**** - это процесс преобразования функции с множеством аргументов в последовательность функций, которые принимают по одному аргументу.

function add(x) {

return function(y) {

return x + y;

};

}

2. ***\*\*Колбек функции\*\**** - это функции, которые передаются в качестве аргументов другой функции и вызываются внутри этой функции.

А как понять, что функция сохраняет значение, или не сохраняет. Почему в данном случае она запоминает контекст, хотя не должна? Это связано с замыканием?

* Отличный вопрос! Да, это связано с механизмом \*\*замыкания\*\*. Когда функция использует переменные из внешней области видимости (скопированные значения) внутри себя, ***она создает замыкание и сохраняет ссылку на эти переменные***, даже после завершения выполнения внешней области видимости.

js

(**function**() {

*//Globals (/locals for app.js (for developer))*

**const** ul = document.getElementById('todo-list');

**const** selector = document.getElementById('user-todo');

**const** form = document.querySelector('form');

**let** todos = [];

**let** users = [];

*//Attach Events*

    document.addEventListener('DOMContentLoaded', initializationApp); *//При загрузке страницы*

    form.addEventListener('submit', handleSubmit)

*//Basic Logic*

**function** getUserName(userId) {

**const** someUser = users.find(user **=>** user.id === userId); */\*На выходе получаем объект,*

*соответствующий данному идентификатору \*/*

        return someUser.name;

    }

**function** printTodos({id, userId, title, completed}) {

**const** li = document.createElement('li'); *//Создание list item*

        li.className = 'todo-item';

        li.dataset.id = id; *//Присвоение элементу id согласно данным пользователя*

        li.innerHTML = `<span> ${title} <i> by </i> <b> ${getUserName(userId)} </b> </span>`;

**const** status = document.createElement('input'); *//Добавление status*

        status.type = 'checkbox';

        status.checked = completed; *//Статус выполнения задачи*

        status.addEventListener('change', handleTodoChange);

**const** closeSpn = document.createElement('span'); *//Добавление cross*

        closeSpn.innerHTML = '&times'; *//Спец.символ "крестик"*

        closeSpn.className = 'close';

        closeSpn.addEventListener('click', handleClose);

        li.prepend(status);

        li.append(closeSpn);

        ul.prepend(li); *//Добавление элемента в начало списка (для прямого отображения)*

    }

**function** createUserOption(user){ *//Функция создания опций списка (пользователи)*

**const** option = document.createElement('option');

        option.value = user.id;

        option.innerText = user.name;

        selector.append(option);

    }

**function** removeTodo (todoId) {

        todos = todos.filter(todo **=>** todo.id !== todoId);

**const** todo = ul.querySelector(`[data-id="${todoId}"]`);

*//Удаление обработчика событий*

        todo.querySelector('input').removeEventListener('change', handleTodoChange);

        todo.querySelector('.close').removeEventListener('click', handleClose);

*//Удаление самого элемента !*

        todo.remove();

    }

**function** alertError(someError) {

        alert(someError.message);

    }

*//Event Logic*

**function** initializationApp() { *//Выполнение операций при загрузке страницы*

        Promise.all([getAllTodos(), getAllUsers()]).then(values **=>** {

            [todos, users] = values;

*//Отправление элементов в разметку*

            todos.forEach(element **=>** {

                printTodos(element);

            });

            users.forEach((user) **=>** createUserOption(user));

        })

    }

**function** handleSubmit(Event) { *//Отправка формы, добавление данных на сервер*

        Event.preventDefault();

        createTodo({

            "userId": Number(form.user.value),

            "title": form.todo.value,

            "completed": false

        })

    }

**function** handleTodoChange(){

**const** todoId = this.parentElement.dataset.id;

**const** completed = this.checked;

        toggleTodoComplete(todoId, completed);

    }

**function** handleClose() {

**const** todoId = this.parentElement.dataset.id; *//Получаем id родительского элемента*

        deleteTodo(todoId);

    }

*//Async Logic*

**async** **function** getAllTodos(){

        try {

**const** response = await fetch('https://jsonplaceholder.typicode.com/todos?\_limit=15');

**const** data = await response.json();

            if(!response.ok) {

                throw new Error('Failed to connect.');

            }

            return data

        } catch (error) {

            alertError(error);

        }

    }

**async** **function** getAllUsers(){

        try {

**const** response = await fetch('https://jsonplaceholder.typicode.com/users');

**const** data = await response.json();

            return data

        } catch (error) {

            alertError(error);

        }

    }

**async** **function** createTodo(todo) {

        try {

**const** response = await fetch('https://jsonplaceholder.typicode.com/todos', {

                method: 'POST',

                body: JSON.stringify(todo),

                headers: {

                    'Content-Type': 'application/json',

                },

*/\*. Указание заголовка `Content-Type: application/json`*

*информирует сервер о том, что данные будут представлены в формате JSON,*

*что позволяет ему корректно обработать полученные данные. \*/*

            });

**const** newTodo = await response.json(); *//Ждём ответа на запрос с сервера. Получаем юзера*

            printTodos(newTodo); *//Печатаем юзера с его запросами*

*// console.log(todoId);*

*// printTodos({id: todoId, ...todo});*

        } catch (error) {

            alertError(error);

        }

    }

**async** **function** toggleTodoComplete(todoId, completed) {

        try {

**const** response = await fetch(`https://jsonplaceholder.typicode.com/todos/${todoId}`,

            {

                method: 'PATCH', *//Добавление элемента, дополнение существующего*

                body: JSON.stringify({completed}),

                headers: {

                    'Content-Type': 'application/json',

                },

            })

**const** data = await response.json();

            console.log(data);

            if(!response.ok) {

                throw new Error('Failed to connect with the server.');

            }

        } catch (error) {

            alertError(error);

        }

    }

**async** **function** deleteTodo(id){

        try {

**const** response = await fetch(`https://jsonplaceholder.typicode.com/todos/${id}`,{

                method: 'DELETE',

                headers: {

                    'Content-Type': 'application/json',

                    },

                }

            );

**const** data = await response.json();

            console.log(data);

            if(response.ok) { *//Если запрос выполнился*

                removeTodo(id);

            }

            else {

                throw new Error('Failed to connect with the server.');

            }

        } catch (error) {

            alertError(error);

        }

    }

})()

Html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>ToDo List</title>

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

</head>

<body>

    <header>

        <h1>ToDo List</h1>

    </header>

    <main>

        <form autocomplete="off">

            <label>

                <input id="new-todo" name="todo" type="text" placeholder="new todo" />

            </label>

            <select id="user-todo" name="user">

                <option disabled selected>select user</option>

            </select>

            <button>Add Todo</button>

        </form>

        <ul id="todo-list"></ul>

    </main>

    <script src="app.js"></script>

</body>

</html>

Css

@import url('https://fonts.googleapis.com/css2?family=Space+Grotesk:wght@500&display=swap');

@import url('https://fonts.googleapis.com/css2?family=Raleway:wght@500&display=swap');

\*{

  margin: 0;

  padding: 0;

}

header { */\* by A.G. \*/*

  background-color: #959CE91F;

  color: #483B78;

  width: 100vw;

  height: 88px;

  display: flex;

  align-items: center;

}

body {

  background-color: #959CE91F;

}

.img1 {

  width: 100%;

  height: 100%;

  background-size: cover;

}

.img2 {

  position:absolute;

  margin: 4%;

  left: 5%;

  width: 509px;

  height: 764px;

  background-size:contain;

}

.img3 {

  position: relative;

  margin: 10%;

  left: 5%;

  width: 216px;

  height: auto;

  background-size:contain;

  float: right;

  top: 480px;

}

.main-text {

  color: #FFFFFF;

  width: 100%;

  height: 1239px;

  position: absolute;

  text-align: center;

  font-size: 26px;

  top: 80%;

  font-family:'Space Grotesk', sans-serif;

  font-weight: bolder;

}

.text-color{

  color: #959CE9;

  font-weight: 900;

}

.text-1{

*/\* background-color: #483B78; \*/*

  margin: 5%;

  float: right;

  overflow: hidden;

  position: relative;

  left: 530px;

  height: 560px;

  width: 923px;

  max-width: 1005px;

  max-height: 610px;

  min-width: 835px;

  min-height: 500px;

}

.text-1 p {

*/\* background-color: #959CE91F; \*/*

  color:black;

*/\* height: 523px; \*/*

*/\* width: 899px; \*/*

  font-size: 23px;

*/\* position: absolute; \*/*

*/\* left: 45%; \*/*

  font-family: 'Raleway', sans-serif;

  line-height: 140%;

}

.title{

  color: #6251A2;

  font-size: 54px;

  font-weight: 700;

  font-family: 'Space Grotesk', sans-serif;

}

.text-arrow{

  color: #6251A2;

  font-size: 39px;

  float: right;

  position: relative;

  right: 22%;

  font-weight: 600;

  font-family: 'Space Grotesk', sans-serif;

}

*/\* .block2 h3{ Почему слово Поиск не поднимается вверх?*

*top: 100px;*

*} \*/*

.block1 {

  width: 637px;

  height: 348px;

*/\* margin-left: 10%; \*/*

  background-color: rgb(255, 255, 255);

  border: 1px solid #483B78 ;

  border-radius: 66px;

  float: left;

  position: relative;

  left: 44px;

  top: 274px;

}

.block1 h3{

  color: #BDAEF4;

  font-size: 45px;

  font-family: 'Space Grotesk', sans-serif;

  font-weight: 700;

  text-shadow: -2px 2px 3px #483B78;

  padding: 10% 7% 1% 7%

}

.block1 p {

  font-size: 23px;

  font-family: 'Raleway', sans-serif;

  font-weight: 600;

  line-height: 27px;

  color: black;

  opacity: 0.75;

  padding: 5% 7% 15% 7%;

}

.block2 {

  width: 839px;

  height: 348px;

  background-color: rgb(255, 255, 255);

  border: 1px solid #483B78 ;

  border-radius: 66px;

  float: left;

  position: relative;

  left: 90px;

  top: 220px;

}

.block2 h3{

  color: #BDAEF4;

  font-size: 45px;

  font-family: 'Space Grotesk', sans-serif;

  font-weight: 700;

  text-shadow: -2px 2px 3px #483B78;

  padding: 8% 7% 1% 7%

}

.block2 p {

  font-size: 23px;

  font-family: 'Raleway', sans-serif;

  font-weight: 600;

  line-height: 27px;

  color: black;

  opacity: 0.75;

  padding: 3% 7% 15% 7%;

}

.block3 {

  width: 1521px;

  height: 303px;

  background-color: rgb(255, 255, 255);

  border: 1px solid #483B78 ;

  border-radius: 66px;

  float: left;

  position: relative;

  left: 50px;

  top: 280px;

*/\* overflow: hidden; \*/*

}

.block3 h3{

  color: #BDAEF4;

  font-size: 45px;

  font-family: 'Space Grotesk', sans-serif;

  font-weight: 700;

  text-shadow: -2px 2px 3px #483B78;

  padding: 3% 4% 1% 4%

}

.block3 p {

  font-size: 23px;

  font-family: 'Raleway', sans-serif;

  font-weight: 600;

  line-height: 27px;

  color: black;

  opacity: 0.75;

  padding: 2% 3% 10% 4%;

}

#page4{

  width: 660px;

  height: 668px;

  float: left;

  position: relative;

  top: 390px;

  left: 50px;

}

#page4 p{

  font-size: 21px;

  font-family: 'Raleway', sans-serif;

  font-weight: 600;

  line-height: 24.65px;

  color: black;

  opacity: 0.75;

  padding: 5% 6% 5% 6%;

}

#page4 h3{

  color: #6251A2;

  font-size: 54px;

  font-family: 'Space Grotesk', sans-serif;

  font-weight: 700;

  line-height: 68.9px;

  text-shadow: -2px 2px 3px #100b21;

  padding: 2% 3% 1% 5%

}

.img4{

  width: 900px;

  height: 600px;

  float: right;

  position: relative;

  top: 300px;

  margin: 3%;

}

*/\* #page3 {*

*width:600px;*

*height:400px;*

*color:#ffa9a9;*

*background-color: aquamarine;*

*} \*/*