Молдавский Государственный Университет
Факультет Математики и Информатики
Департамент Информатики

Отчет по индивидуальной работе №2 по курсу JavaScript & TypeScript

Проверил: Nartea Nichita
Ответ составил: студент 1-го курса
группы I2302 Slavov Constantin

Кишинев, 2024

Цель индивидуальной работы:

Ознакомить студентов с продвинутыми функциями JavaScript, включая асинхронный JavaScript, модули и обработку ошибок.

Теоретическая часть:

Задача данного проекта: написать функцию, которая будет делать запрос и получать данные со стороннего ресурса. Таким образом, программа каждый раз будет отображать чем должен заняться Капитан Смит в этот раз.

Описание цели и основные этапы работы:

- Создание функции, которая будет делать запрос и получать данные со стороннего ресурса.
- Добавление в функцию getRandomActivity() обработку ошибок. В случае ошибки на экран должен выводиться текст: “К сожалению, произошла ошибка”.
- Изменение функции getRandomActivity() для того, чтобы она использовала ключевые слова async/await.
- Добавление функционала для обновления данных каждую минуту при помощи функции setTimeout().
- Изменить функцию getRandomActivity() так, чтобы она возвращала данные, и добавить функцию updateActivity(), которая будет отображать полученные данные.

Краткое описание особенностей реализации:

В своей программе, я создал все файлы, которые были расписаны в задании и еще добавил несколько CSS-свойств для текста, чтобы он выглядел на экране более приятно.

- Функция `getRandomActivity()` отправляет запрос к стороннему API (https://www.boredapi.com/api/activity/) с помощью функции `fetch()`. После получения ответа, она преобразует его в формат JSON с помощью `response.json()`. Затем функция возвращает

случайную активность из полученных данных. Если во время запроса произойдет ошибка, функция перехватывает ее в блоке `catch`, выводит сообщение об ошибке в консоль и возвращает строку `"К сожалению, произошла ошибка"`.

- Функция `updateActivity()` вызывает функцию `getRandomActivity()`, чтобы получить случайную активность. Затем она обновляет текстовое содержимое HTML-элемента с идентификатором `'activity'` значением полученной активности. После этого функция устанавливает таймер с помощью `setTimeout`, чтобы вызвать саму себя через 60000 миллисекунд (то есть каждую минуту) для обновления активности.

Пример использования проекта:

Вывод:

Исходя из проделанной работы, я научился работать со сторонними ресурсами для получения от них данных, попрактиковался с работой на асинхронном JavaScript, научился обрабатывать ошибки и понял в чем различие между async/await. Данный опыт безусловно пригодится мне в будущей работе на языке программирования JavaScript.

Ссылка на репозиторий GitHub:

https://github.com/kraaddys/JS_and_TS/tree/main

Ответы на контрольные вопросы:

- Какое значение возвращает функция fetch?

Ответ: Функция `fetch` возвращает объект типа Promise. Этот объект Promise представляет результат асинхронной операции запроса на сервер. Promise - это объект, который представляет успешное завершение или неудачу асинхронной операции. Когда запрос завершается, Promise может быть разрешен (fulfilled) с объектом Response (если запрос завершился успешно) или отклонен (rejected) с ошибкой (если запрос завершился с ошибкой).

- Что представляет собой Promise?

Ответ: Промис в JavaScript - это объект-обертка, который обеспечивает возможность асинхронного выполнения функций, переданных в него. Промисы были созданы для организации последовательного выполнения асинхронного кода. Он может иметь 3 состояния: Pending (ожидание): Исходное состояние, не выполнено и не отклонено. Fulfilled (выполнено): Означает, что операция завершилась успешно. Rejected (отклонено): Означает, что операция завершилась неудачей.

- Какие методы доступны у объекта Promise?

Ответ: У объекта Promise доступны следующие методы:

1. `then`: Используется для добавления обработчиков успешного выполнения и ошибки асинхронной операции. Возвращает новый Promise.
2. `catch`: Используется для добавления обработчика ошибки асинхронной операции. Возвращает новый Promise.
3. `finally`: Используется для добавления обработчика, который будет вызван после завершения выполнения Promise, независимо от его исхода. Возвращает новый Promise.

- **Каковы основные различия между использованием async / await и Promise?**

Ответ: Основные различия между использованием `async/await` и `Promise`:

1. Синтаксис: `async/await` предоставляет более линейный и читаемый синтаксис по сравнению с использованием цепочек `then/catch`. Он позволяет писать асинхронный код в структуре, похожей на синхронный.
2. Обработка ошибок: В `async/await` используется блок `try/catch` для обработки ошибок, что делает код более легким для чтения и написания. В то время как в Promise для обработки ошибок часто приходится использовать метод `catch`.
3. Возврат значений: `async/await` позволяет легко возвращать значения из асинхронных функций, просто используя `return`. В Promise, для возврата значений часто приходится использовать конструкцию `resolve()` и `reject()`.
4. Читаемость кода: Код с использованием `async/await` обычно более понятен и легко читаем, так как он выглядит ближе к синхронному коду, в то время как цепочки Promise могут быть более громоздкими и трудными для понимания.

Использование `async/await` обычно предпочтительнее, особенно при написании более сложного асинхронного кода, благодаря улучшенной читаемости и управляемости.

Список источников информации:

https://github.com/MSU-Courses/javascript_typescript/tree/main/docs
https://www.google.com/