**МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет Математики и Информатики**

**Департамент Информатики**

Индивидуальная работа № 2

***Javascript and Typescript***

Проверил: Nartea N.

Выполнил: Iudina Maria grupa IA2304

Кишинев, 2024

Оглавление

[I. Введение. 3](#_Toc166709352)

[II. Практическая часть 4](#_Toc166709353)

[Выводы. 6](#_Toc166709354)

[Контрольные вопросы 7](#_Toc166709355)

[Библиография 8](#_Toc166709356)

# Введение.

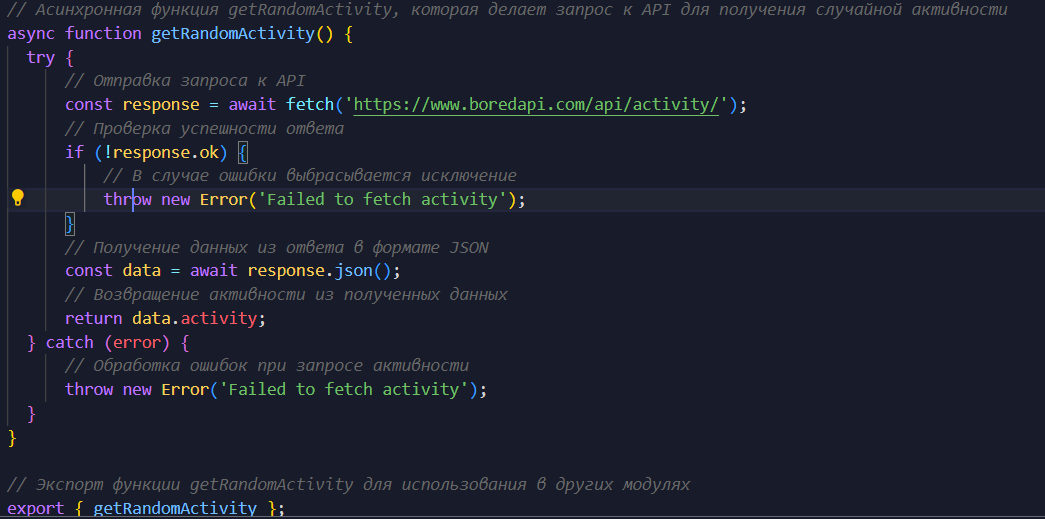
Главной ***целью*** данного проекта является ознакомление и развитие практических навыков в продвинутых функциями JavaScript, включая асинхронный JavaScript, модули и обработку ошибок.

В современном мире веб-разработки, понимание продвинутых функций JavaScript становится все более важным для разработчиков. Одном из ключевых моментов этого понимания является умение работать с асинхронным JavaScript, который позволяет эффективно управлять выполнением кода в среде веб-приложений. Кроме того, знание о модулях JavaScript и методах их использования открывает новые возможности для организации и структурирования кодовой базы проекта. Важным аспектом развития навыков программирования на JavaScript является также умение эффективно обрабатывать ошибки, что позволяет создавать более надежные и устойчивые приложения. В этой работе мы погрузимся в мир продвинутых функций JavaScript, изучая асинхронный JavaScript, модули и обработку ошибок, чтобы подготовить вас к созданию мощных и инновационных веб-приложений на примере нашего Капитана и его путешествиях.

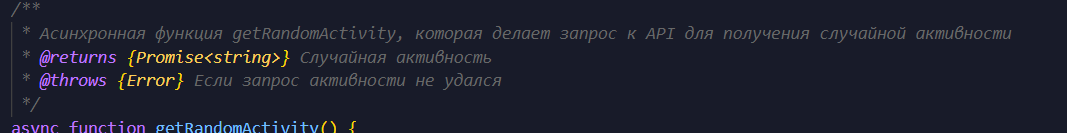
1. Практическая часть

Этот код иллюстрирует применение асинхронной функции в сочетании с оператором await для отправки запроса к API и обработки данных, полученных в ответ. Мы воспользуемся оператором try/catch для обработки возможных ошибок, возникающих при выполнении запроса.

Функция getRandomActivity использует fetch для отправки асинхронного запроса к API и получения случайной активности. Мы применяем оператор await для ожидания ответа от сервера, после чего осуществляем обработку полученных данных. Если происходит ошибка, мы используем исключение для ее перехвата и обработки в блоке catch.



Далее необходимо аккуратно и полно задокументировать код в соответствии со стандартом JSDoc. Каждая функция и метод должны быть описаны, включая указание их входных параметров, выходных данных и описания функционала. Комментарии должны быть ясными, четкими и информативными, чтобы обеспечить понимание работы кода другим разработчикам.



Этот участок кода демонстрирует использование асинхронной функции для периодического обновления активности на странице с интервалом в одну минуту. Мы используем функцию setTimeout для установки таймера, который запускает функцию обновления активности через минуту после завершения текущего запроса.

Функция updateActivity применяет оператор await для вызова функции getRandomActivity и получения случайной активности. Затем мы используем setTimeout для запуска функции updateActivity повторно через минуту. Это обеспечивает периодическое обновление активности на странице каждую минуту, предпочтительно перед использованием setInterval. Этот подход более надежный и предотвращает перекрытие запросов, если обработка запроса занимает больше времени, чем одна минута.



Далее нам снова необходимо аккуратно и полно задокументировать код в соответствии со стандартом JSDoc. Каждая функция и метод должны быть описаны, включая указание их входных параметров, выходных данных и описания функционала. Комментарии должны быть ясными, четкими и информативными, чтобы обеспечить понимание работы кода другим разработчикам.



# Выводы.

Использование асинхронных функций и оператора await в JavaScript позволяет эффективно управлять асинхронными операциями, такими как запросы к API, без необходимости использования колбэков или промисов. Это делает код более читаемым и понятным.

Кроме того, использование функции setTimeout для создания периодических задач, таких как обновление данных на странице, предпочтительнее использования setInterval в некоторых случаях. Это позволяет избежать перекрытия запросов и обработки данных, если операция занимает больше времени, чем заданный интервал.

Объединение этих методов позволяет создавать понятный, эффективный и надежный код для работы с асинхронными задачами в JavaScript.

## Контрольные вопросы

1. Какое значение возвращает функция fetch?

Функция fetch возвращает объект типа Promise.

1. Что представляет собой Promise?

Объект Promise в JavaScript представляет собой асинхронную операцию, которая может быть в процессе выполнения, завершена успешно или завершена с ошибкой. Он используется для обработки асинхронных операций и представляет результат или ошибку, которые могут быть доступны в будущем.

1. Какие методы доступны у объекта Promise?

У объекта Promise доступны следующие методы:

then(onFulfilled, onRejected): Этот метод используется для добавления обработчиков для успешного выполнения или отклонения промиса.

catch(onRejected): Этот метод используется для добавления обработчика ошибок к промису.

finally(onFinally): Этот метод позволяет добавить обработчик, который будет вызван независимо от того, был ли промис разрешен или отклонен.

1. Каковы основные различия между использованием async / await и Promise?

Основные различия между использованием async/await и Promise:

Синтаксис:

С использованием Promise используется цепочка методов then() и catch(), что может привести к так называемому "стэк трэйсу", когда несколько обработчиков вызываются один за другим.

С использованием async/await используется более линейный и читаемый синтаксис, который выглядит как последовательный код.

Обработка ошибок:

В Promise ошибки обрабатываются с помощью метода catch(), что может усложнить логику при наличии нескольких асинхронных операций.

С async/await ошибки обрабатываются с помощью блока try...catch, что делает код более понятным и управляемым.

Читаемость кода:

async/await делает код более легким для чтения и понимания, особенно когда есть несколько асинхронных операций, которые должны быть выполнены последовательно.

В случае Promise цепочка методов может стать слишком длинной и сложной для понимания.

Управление асинхронными операциями:

С async/await управление асинхронными операциями становится более прямолинейным и интуитивно понятным, так как выражается последовательным выполнением.

Promise позволяет более гибко управлять асинхронными операциями, но зачастую может требовать больше усилий для поддержания читаемости и понимания кода.

## Библиография

1. Материалы лабораторной №1 на сайт курса (https://github.com/MSU-Courses/javascript\_typescript/blob/main/lab/LI2/JS02.md).

2. https://developer.mozilla.org/ru/docs/Learn/JavaScript/Asynchronous/Introducing

3. https://developer.mozilla.org/ru/docs/Learn/JavaScript/Asynchronous

4. <https://doka.guide/js/async-in-js/>

5. https://learn.javascript.ru/async-await