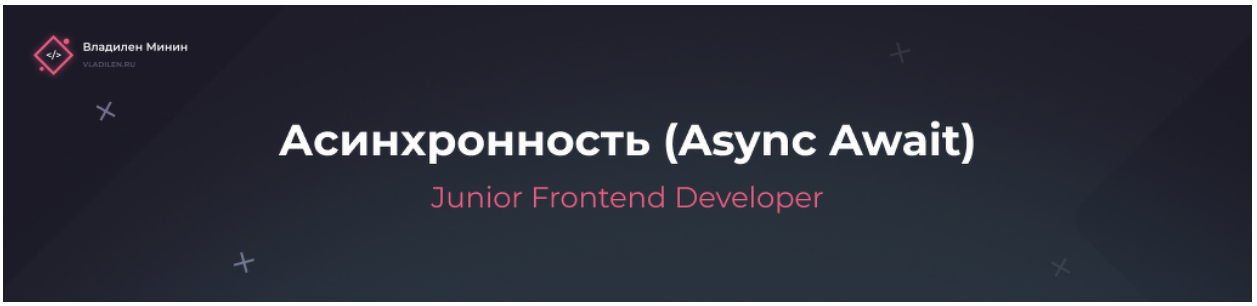

Цель урока:

Вы узнаете, в чем состоит проблема then, catch, finally и как ее решить с помощью конструкции async await.

Содержание урока:

1. Проблема then, catch и finally
2. Как работает async await?

Дополнительные материалы:

1. async await в JavaScript: https://learn.javascript.ru/as...
2. Урок 8. JavaScript. Как работает Async, Await. Работа с сервером с fetch: https://youtu.be/SHiUyM_fFME

При работе с **then, catch и finally** можно избавиться от вложенности, чтобы код не рос “вправо”.
Например, с помощью кода ниже мы получаем сначала пользователя, а затем все комментарии для этого пользователя:

```
fetch(USERS_URL, {
 method: 'GET',
 headers: {},
})
 .then(response => response.json())
 .then(users => {
   const firstUserId = users[0].id
   fetch(`${COMMENT_URL}?postId=${firstUserId}`, {
     method: 'GET',
     headers: {},
   })
     .then(response => response.json())
     .then(result => {
       console.log('result', result)
     })
     .catch(error => {
       console.log('error', error)
     })
 })
 .catch(error => {
   console.log('error', error)
 })
```

Здесь видно, что код растет “вправо”. Чтобы от этого избавиться, можно внутри **then** просто вернуть асинхронный запрос при помощи **fetch**.

```
fetch(USERS_URL, {
 method: 'GET',
 headers: {},
})
 .then(response => response.json())
 .then(users => {
   const firstUserId = users[0].id
   return fetch(`${COMMENT_URL}?postId=${firstUserId}`, {
     method: 'GET',
     headers: {},
   })
 })
 .then(response => response.json())
 .then(result => {
   console.log('result', result)
 })
 .catch(error => {
   console.log('error', error)
 })
```

Но данный способ не отменяет полезности **async await**, так как данная конструкция убирает использование колбэков и помогает писать асинхронный код как синхронный. Поэтому **старайтесь применять async await** на практике при использовании сложных запросов.

Задание #1

Требуется переписать данный код, который использует **then**, **catch** и **finally**, в код, который использует исключительно **async await и try, catch, finally.**

```
const POSTS_URL = 'https://jsonplaceholder.typicode.com/posts';
let isLoading = false;
const createNewPost = () => {
isLoading = true;
fetch(POSTS_URL, {
  method: 'POST',
})
  .then((response) => response.json())
  .then((result) => {
    console.log('result', result)
  })
  .catch((error) => {
    console.log('error', error)
  })
  .finally(() => {
    isLoading = false;
  });
};
createNewPost();
```

Задание #2

Требуется переписать данный код, который использует **then и catch**, в код, который использует исключительно **async await и try, catch**.

```
const TODOS_URL = 'https://jsonplaceholder.typicode.com/todos';
const getTodosByIds = (ids) => {
  const requests = ids.map((id) => fetch(`${TODOS_URL}/${id}`));
  Promise.all(requests)
      .then((responses) => {
          const dataResults = responses.map((data) => data.json());
          return Promise.all(dataResults)
      })
      .then((allTasks) => {
          console.log(allTasks);
      })
      .catch((error) => {
          console.log(error);
      })
}
getTodosByIds([43, 21, 55, 100, 10]);
```

Задание #3

Вам необходимо создать функцию **renderAlbums**, которая будет отображать данные об альбомах в DOM-дереве. Для получения данных используйте следующий url: https://jsonplaceholder.typicode.com/albums

HTML-шаблон для элемента альбома выглядит следующим образом:

```
<li>название альбома</li>
```

Итоговые HTML-элементы с информацией об имени альбома **поместите в тег ul** с классом равным “**data-container**”.

Для удобства **необходимо добавить** элемент **span** с текстом “**Загрузка...**” перед загрузкой альбомов, и спрятать этот элемент после загрузки данных.

Используйте исключительно async await и try, catch, finally. Если при загрузке альбомов произошла ошибка, то внутрь тега с классом равным “**data-container**” поместите текст “**Произошла ошибка в получении данных об альбомах...**”.