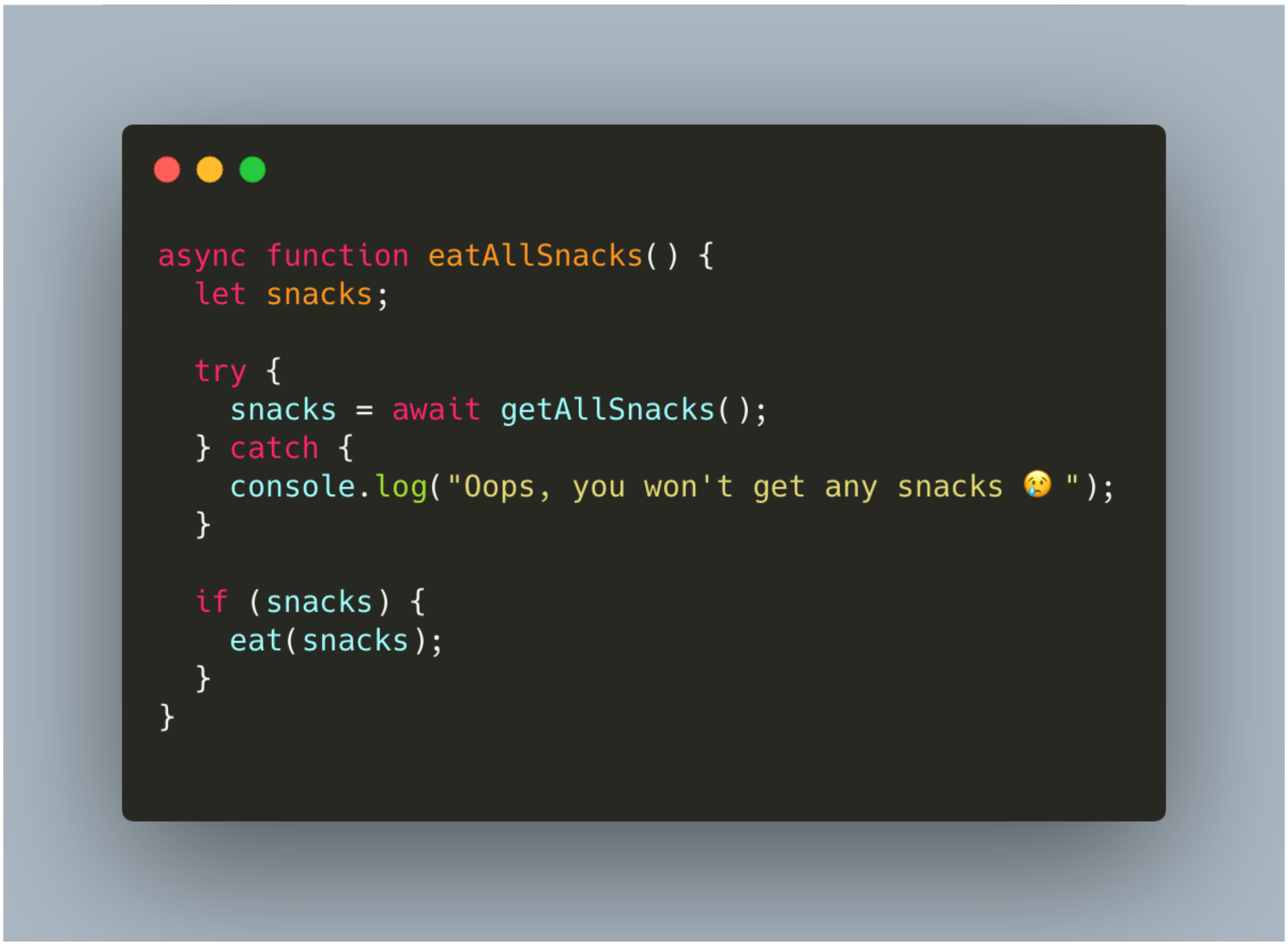
Async/await without try/catch in JavaScript

Dzmitry Bayarchyk
Oct 2, 2019 · 2 min read ★



```
async function eatAllSnacks() {
  let snacks;

  try {
    snacks = await getAllSnacks();
  } catch {
    console.log("Oops, you won't get any snacks 😢 ");
  }

  if (snacks) {
    eat(snacks);
  }
}
```

“How to be sure that you handle async errors correctly”

When *async/await* was announced it became a game-changer in JavaScript development. It allows writing code in a synchronous way and we don’t need to have chained promise handlers:

```
function fetchAndUpdatePosts() {
  fetchPosts()
    .then((posts) => {
      updatePosts(posts)
        .catch((err) => {
          console.log('error in updating posts');
        });
    })
    .catch(() => {
      console.log('error in fetching posts');
    });
}
```

oldFetchAndUpdatePosts.js hosted with ♡ by GitHub view raw

That how this code can be refactored with the *async/await* syntax:

```
async function fetchAndUpdatePosts() {
  let posts;

  try {
    posts = await fetchPosts();
  } catch {
    console.log('error in fetching posts');
  }

  if (!posts) {
    return;
  }

  try {
    await updatePosts();
  } catch {
    console.log('error in updating posts');
  }
}
```

regularAsyncAwaitExample.js hosted with ♡ by GitHub view raw

Now it is easier to follow the code. But there are still some improvements we can do with that code to keep it cleaner.

Since we wrap our async function with *try/catch* we expect to handle all errors from this function in the following *catch* block. But what if we continue our code inside this *try* block? That means that all errors will land into the closest *catch* block.

```
async function fetchAndUpdatePosts() {
  let posts;

  try {
    posts = await fetchPosts();

    doSomethingWithPosts(posts); // throws an error
  } catch {
    // Now it handles errors from fetchPosts and doSomthingWithPosts.
    console.log('error in fetching posts');
  }

  // ...
}
```

combineCatchHandler.js hosted with ♡ by GitHub   view raw

The more code we add the less clear and more generic that *catch* block becomes. The only way how we can prevent this is by being careful at code reviews. But how can we do that error handling explicitly to the *fetchPosts* function and nothing else?

Even if our code doesn't look like we work with promises don't forget that we still do. Every function that is declared with an *async* keyword returns a promise even if we don't explicitly have a return statement. Since we have a promise we have access to all its methods. The one we're interested in is Promise.prototype.catch. This method also returns a promise so we can combine it with an *await* keyword.

```
async function fetchAndUpdatePosts() {
  const posts = await fetchPosts().catch(() => {
    console.log('error in fetching posts');
  });

  if (posts) {
    doSomethingWithPosts(posts);
  }
}
```

promiseCatchWithAwait.js hosted with ♡ by GitHub   view raw

This allows to explicitly catch errors for the async function and when something happens in *doSomethingWithPosts* it won't land into the *fetchPosts catch* block and we can handle errors separately.

Conclusion

When working with *async/await* we can use all functions of a regular promise to avoid *try/catch* noise in our code, helps to explicitly handle errors and keeps your variables as constants.

JavaScript Asynchronous Asyncawait Software Development Error Handling