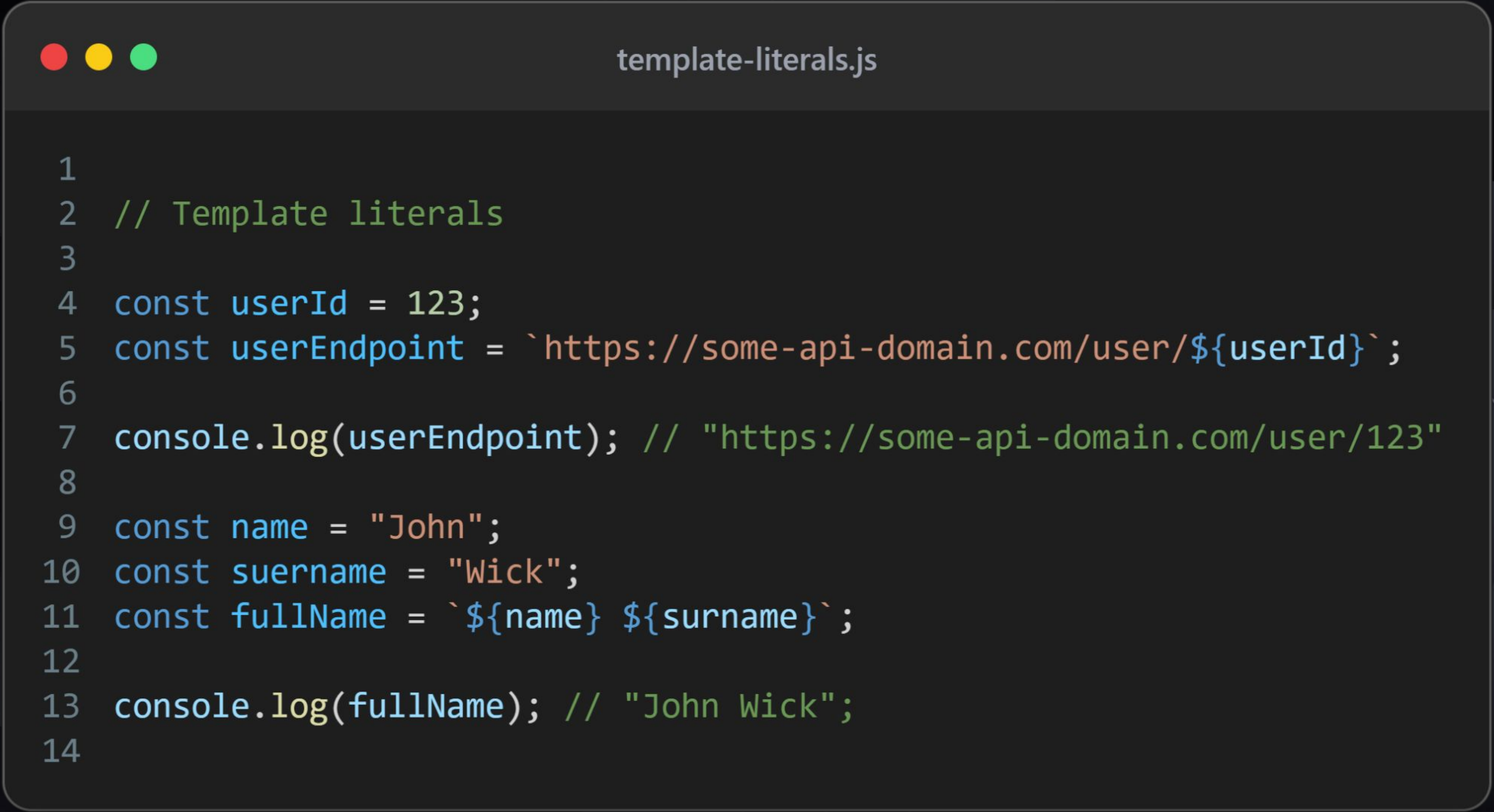
JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA

Homework

Template string

- Allows to easily concat strings or insert values into a premade string
- Use ` ` (key ~ `)
- To insert a variable into such a string - use ${variableName}

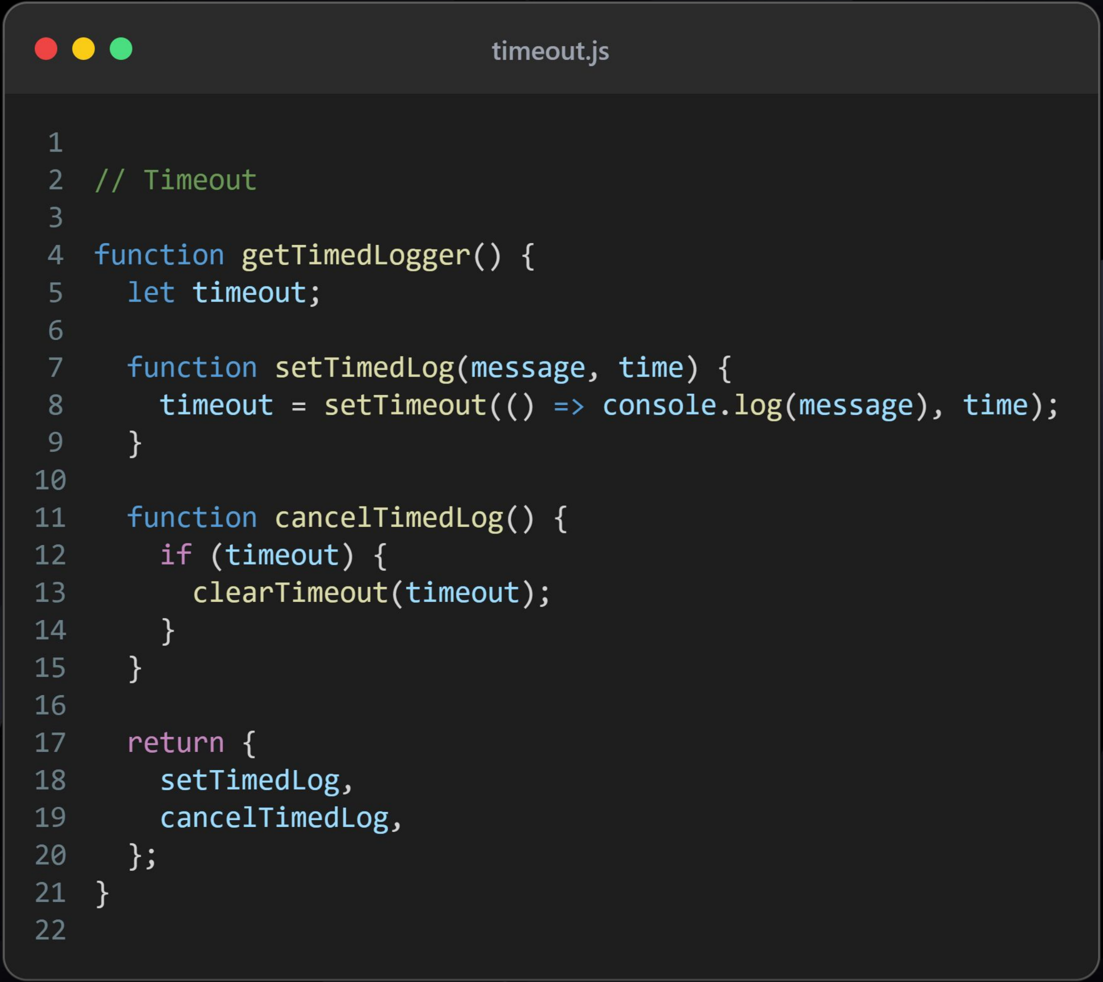

template-literals.js

```javascript

// Template literals

const userId = 123;
const userEndpoint = `https://some-api-domain.com/user/${userId}`;

console.log(userEndpoint); // "https://some-api-domain.com/user/123"

const name = "John";
const surname = "Wick";
const fullName = `${name} ${surname}`;

console.log(fullName); // "John Wick";

```

ASYNCHRONOUS JAVASCRIPT

Timeout

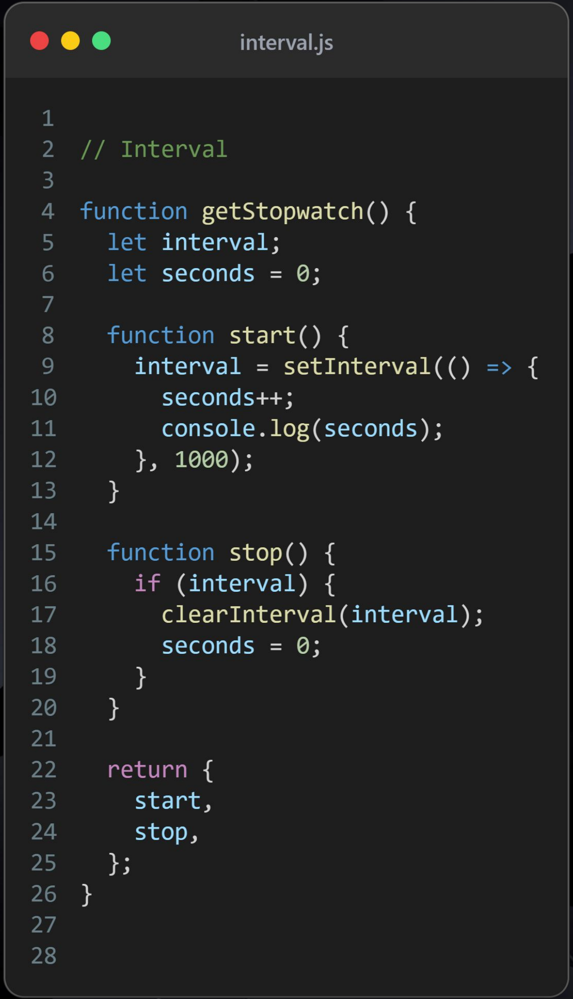
- Create a function that will return us an object with 2 methods
- The first method will allow us to log a message in the console after a certain number of milliseconds
- The second one will allow us to cancel the message log before its timeout


timeout.js

```

// Timeout

function getTimedLogger() {
  let timeout;

  function setTimedLog(message, time) {
    timeout = setTimeout(() => console.log(message), time);
  }

  function cancelTimedLog() {
    if (timeout) {
      clearTimeout(timeout);
    }
  }

  return {
    setTimedLog,
    cancelTimedLog,
  };
}

```

Interval

- Create a stopwatch - function that will return 2 methods - start and stop
- Start stopwatch will count seconds in the console
- Stop will finish running our stopwatch

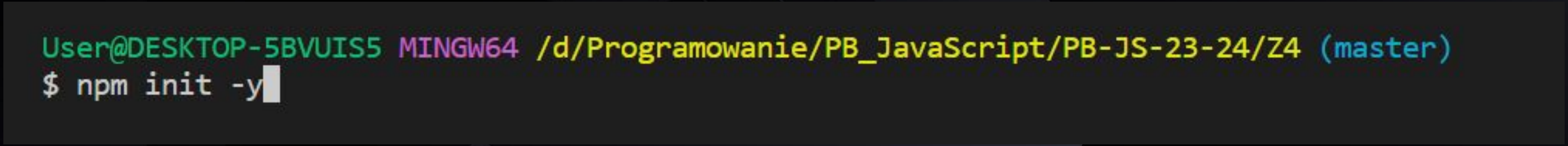


interval.js

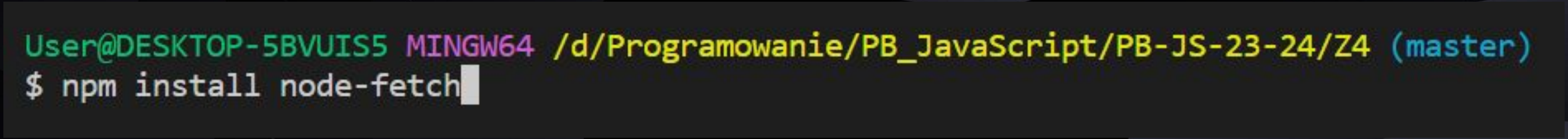
```javascript

// Interval

function getStopwatch() {
  let interval;
  let seconds = 0;

  function start() {
    interval = setInterval(() => {
      seconds++;
      console.log(seconds);
    }, 1000);
  }

  function stop() {
    if (interval) {
      clearInterval(interval);
      seconds = 0;
    }
  }

  return {
    start,
    stop,
  };
}


```

fetch

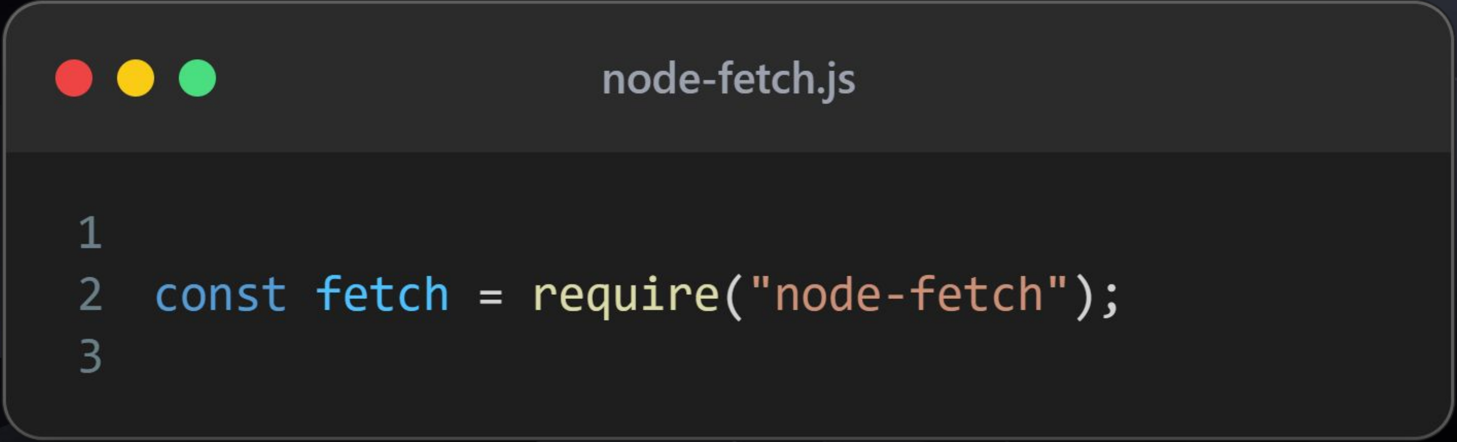
- Been in the browser for quite some time
- In Node since version 18 (April 2022)
- For versions < 18 we'll use 'node-fetch' npm package



```
User@DESKTOP-5BVUIS5 MINGW64 /d/Programowanie/PB_JavaScript/PB-JS-23-24/Z4 (master)
$ npm init -y
```



```
User@DESKTOP-5BVUIS5 MINGW64 /d/Programowanie/PB_JavaScript/PB-JS-23-24/Z4 (master)
$ npm install node-fetch
```

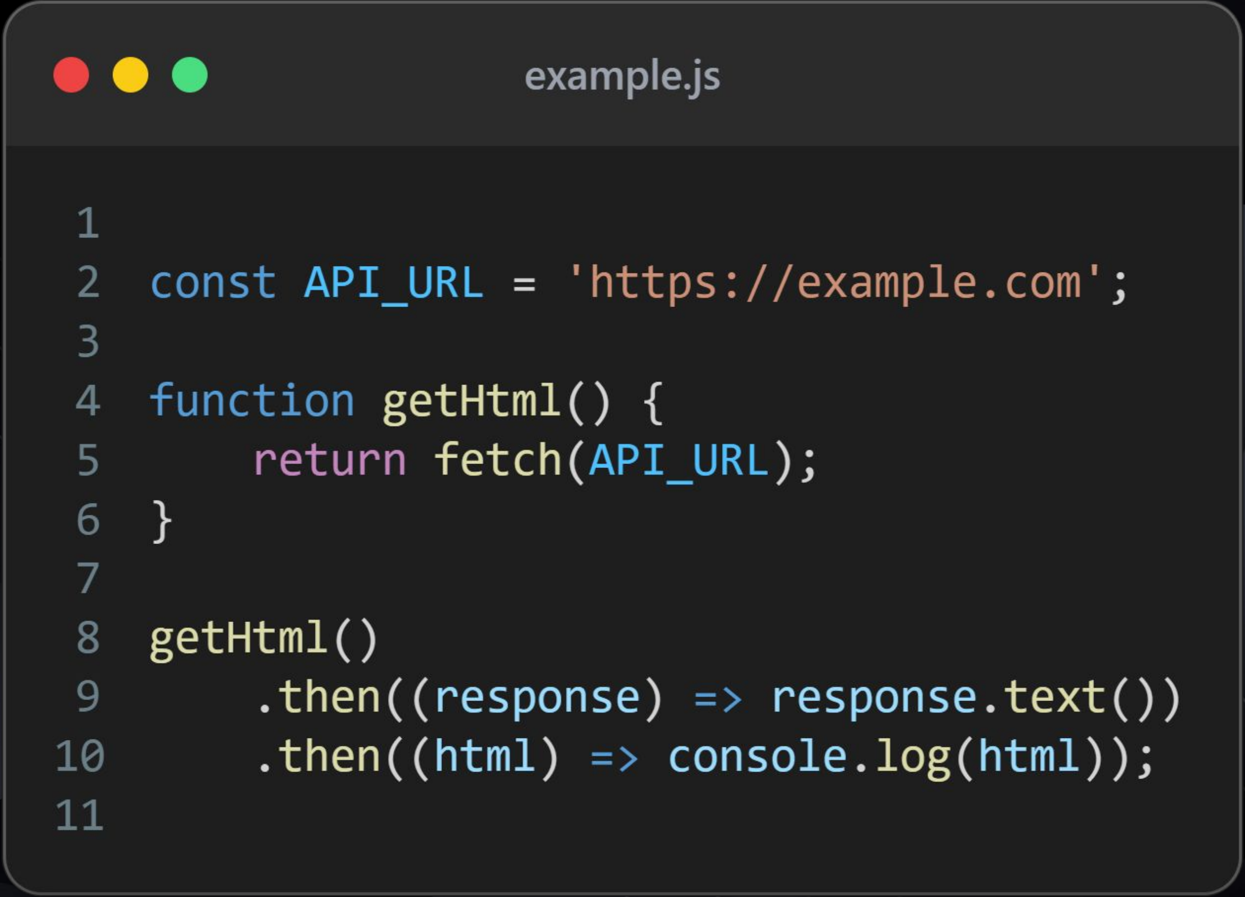


node-fetch.js

```

const fetch = require("node-fetch");

```

fetch – simple

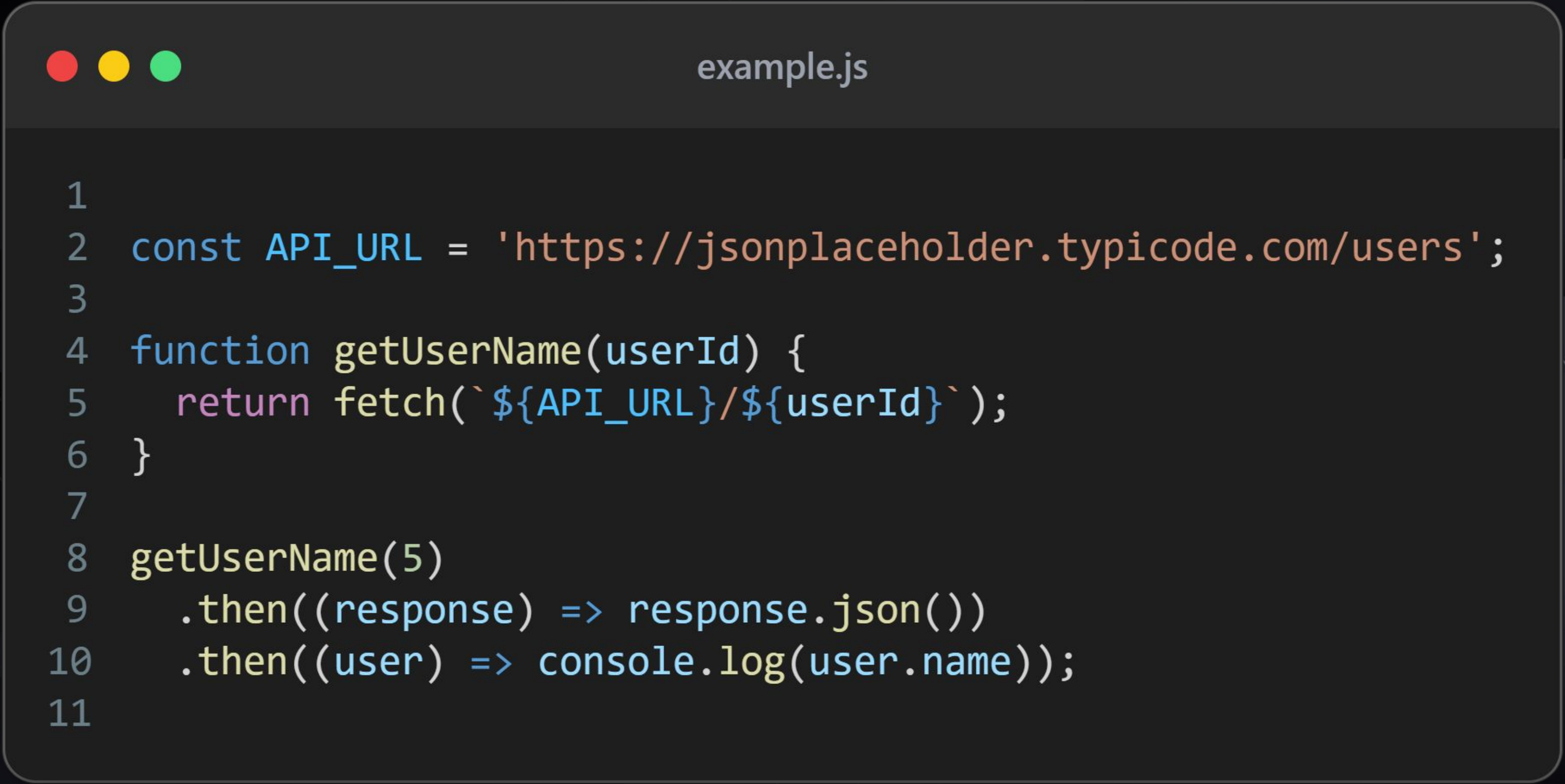
- Get HTML document
- Use “https://example.com”
- console log HTML document


example.js

```
const API_URL = 'https://example.com';

function getHtml() {
    return fetch(API_URL);
}

getHtml()
    .then((response) => response.text())
    .then((html) => console.log(html));
```

fetch - simple

- Get user with ID 5
- Use “https://jsonplaceholder.typicode.com/users” endpoint
- console log the name


example.js

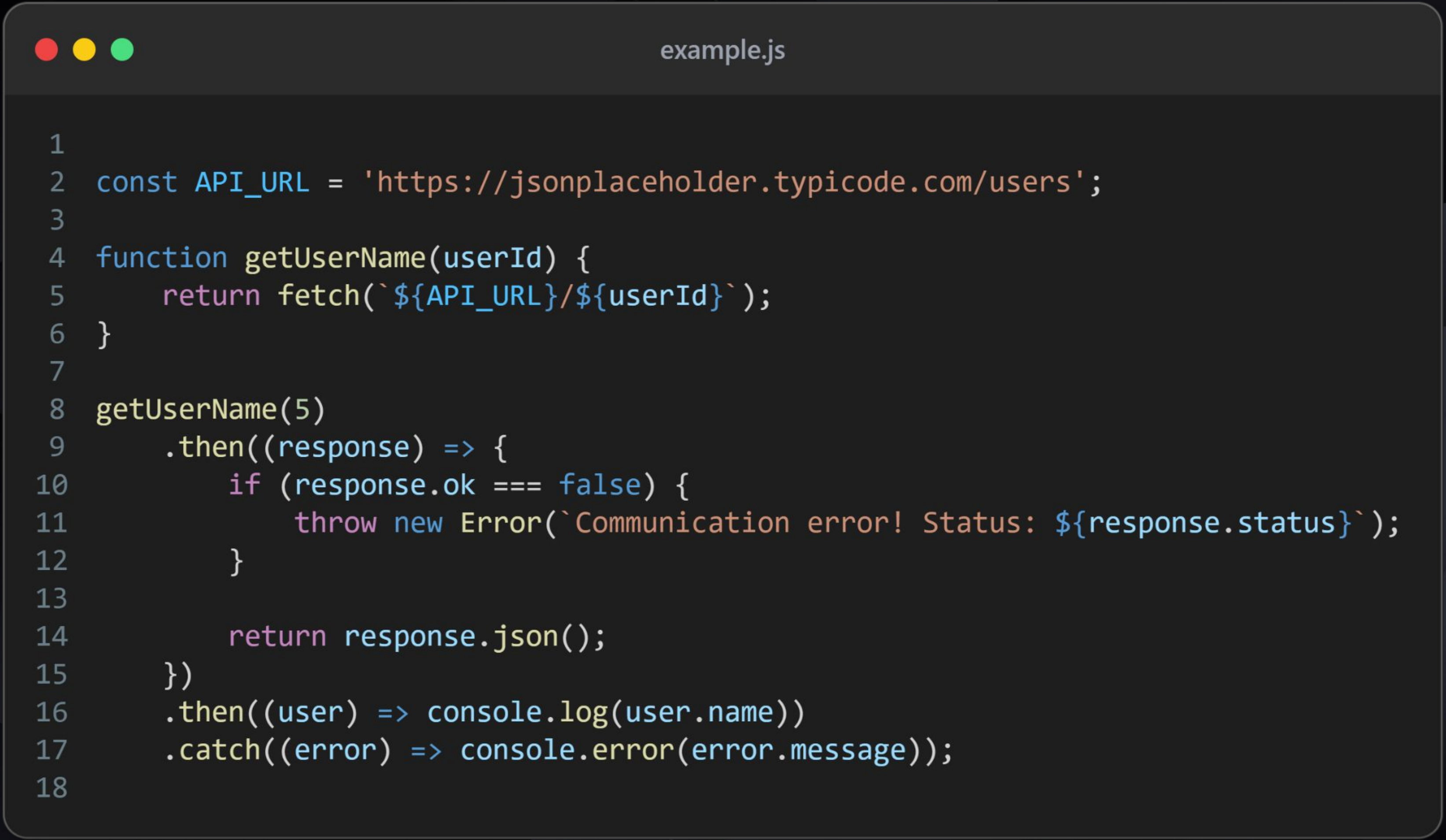
```js

const API_URL = 'https://jsonplaceholder.typicode.com/users';

function getUserName(userId) {
  return fetch(`${API_URL}/${userId}`);
}

getUserName(5)
  .then((response) => response.json())
  .then((user) => console.log(user.name));

```

fetch - error

- Add error handling to the previous exercise
- Use “https://jsonplaceholder.typicode.com/users” endpoint
- Include network errors
- Include server errors

fetch - handling errors

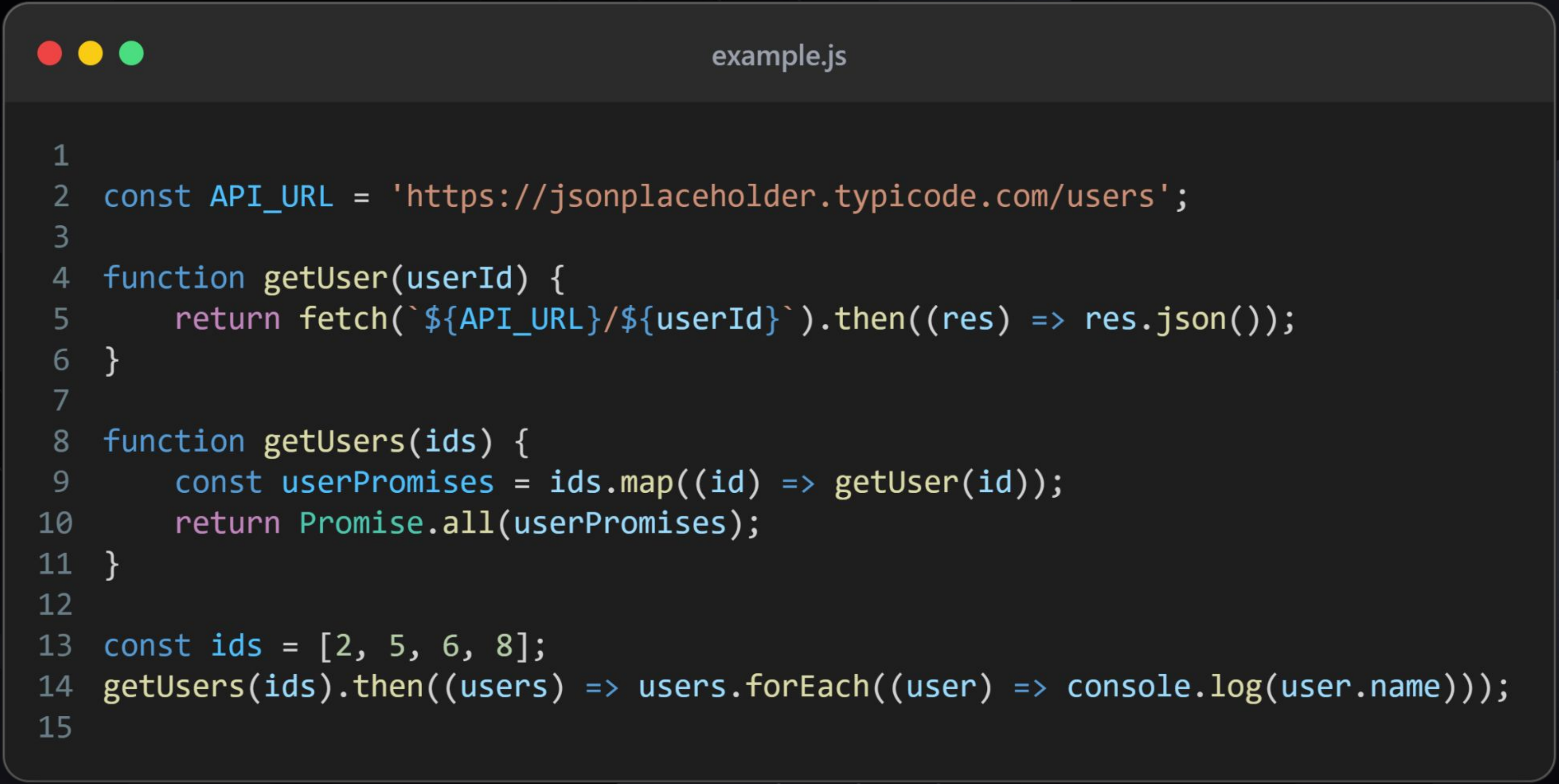
- .catch() is not enough when using fetch
- fetch promise returns a Response object
- MDN Response
- this object has an 'ok' property
- if response.ok === false we got and error from the server - need to check in .then() - not .catch()


example.js

```javascript

const API_URL = 'https://jsonplaceholder.typicode.com/users';

function getUserName(userId) {
    return fetch(`${API_URL}/${userId}`);
}

getUserName(5)
    .then((response) => {
        if (response.ok === false) {
            throw new Error(`Communication error! Status: ${response.status}`);
        }

        return response.json();
    })
    .then((user) => console.log(user.name))
    .catch((error) => console.error(error.message));

```

fetch - multiple data

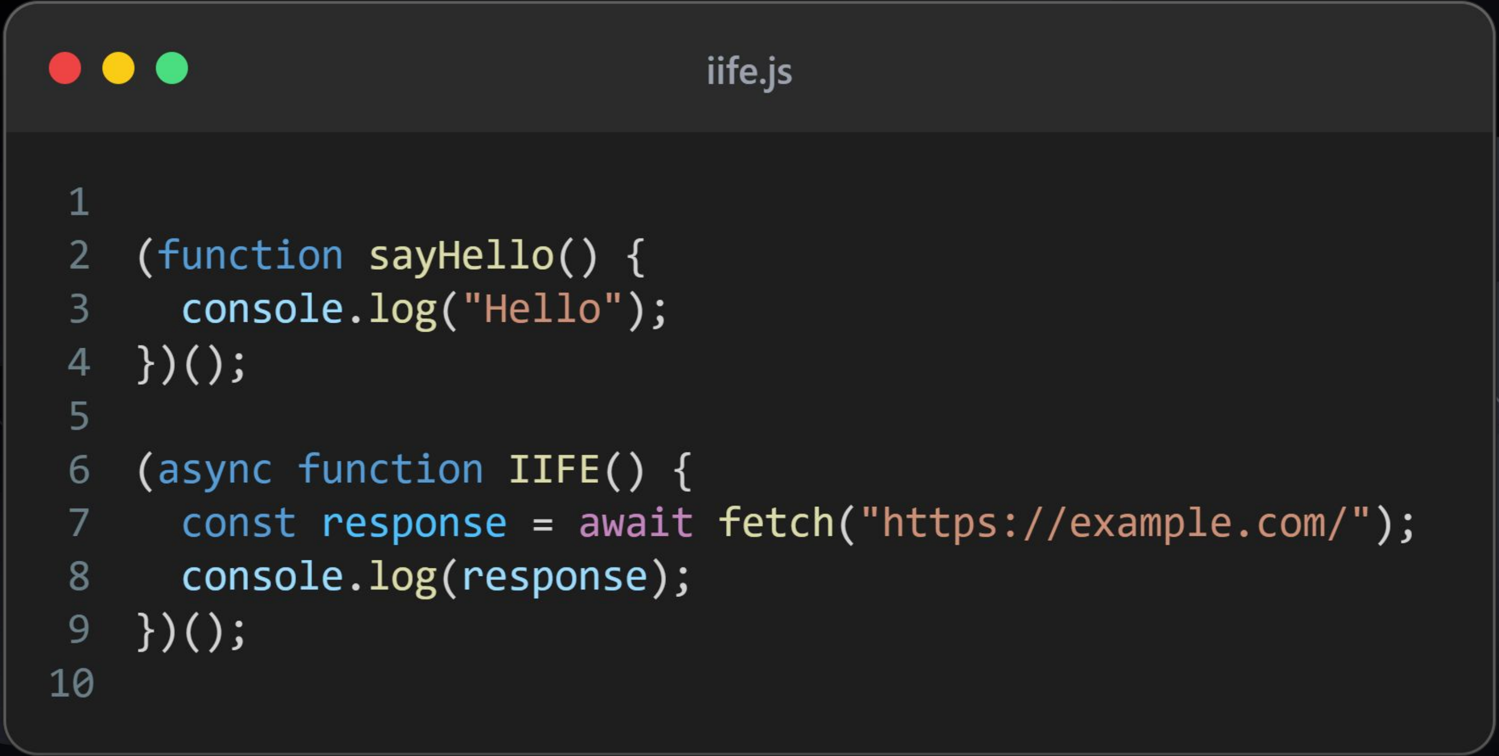
- For a given array of ids (ex. [2,5,6,8])
- Fetch all users with corresponding ids
- console log all fetched user names
- Use "https://jsonplaceholder.typicode.com/users" endpoint


example.js

```javascript

const API_URL = 'https://jsonplaceholder.typicode.com/users';

function getUser(userId) {
    return fetch(`${API_URL}/${userId}`).then((res) => res.json());
}

function getUsers(ids) {
    const userPromises = ids.map((id) => getUser(id));
    return Promise.all(userPromises);
}

const ids = [2, 5, 6, 8];
getUsers(ids).then((users) => users.forEach((user) => console.log(user.name)));

```

IIFE

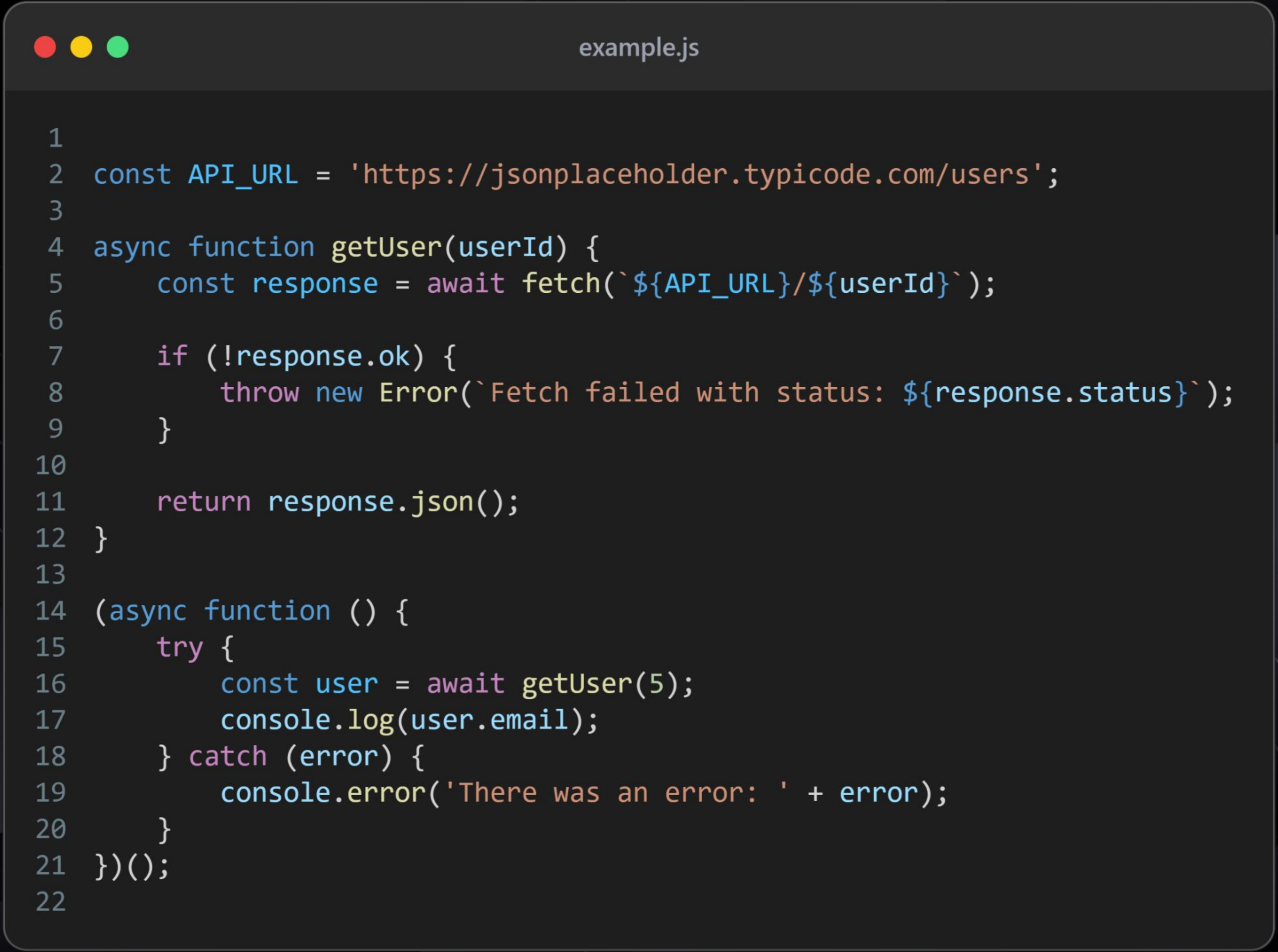
- Immediately Invoked Function Expression
- Allows to write and run a function in the same place
- Was used to force closure in some patterns (when there were no native modules in JS)
- Can be used to use AWAIT


iife.js

```javascript

(function sayHello() {
  console.log("Hello");
})();

(async function IIFE() {
  const response = await fetch("https://example.com/");
  console.log(response);
})();

```

fetch - simple async/await

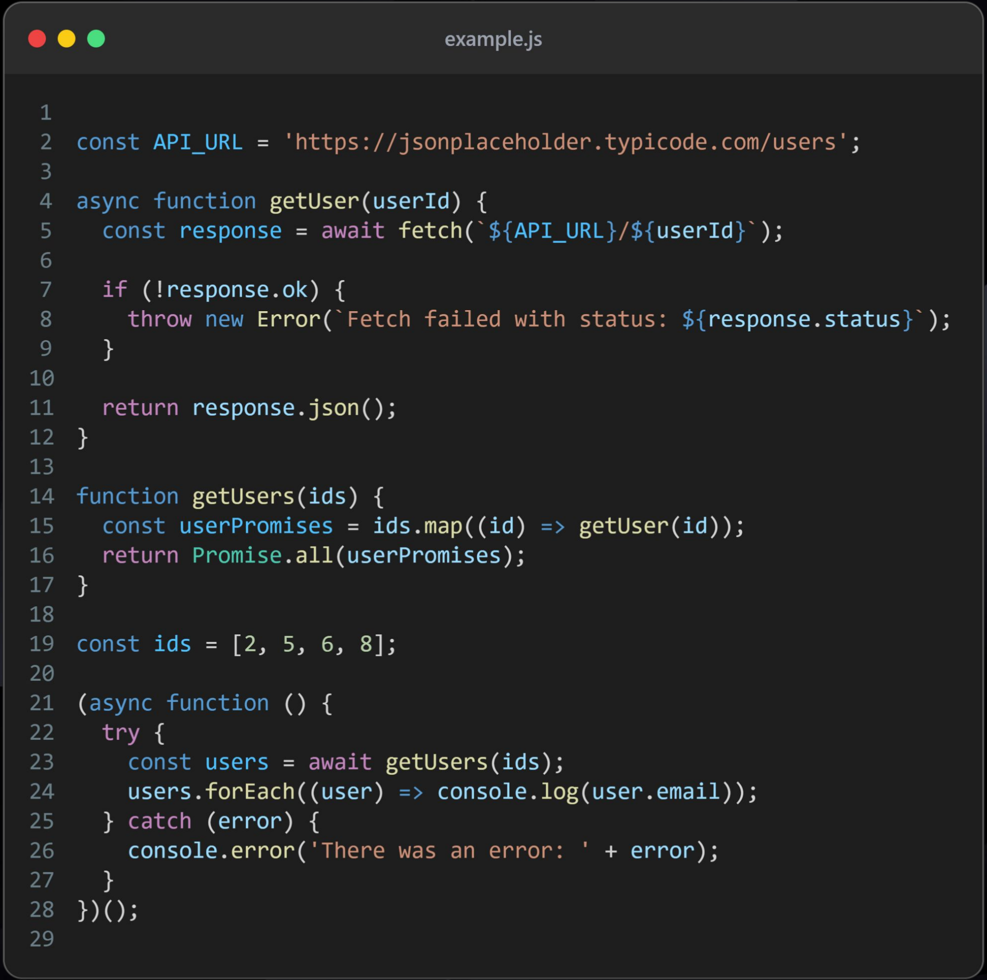
- Get user name with ID 5
- Use “https://jsonplaceholder.typicode.com/users” endpoint
- Remember that this endpoint returns JSON!
- console log user’s email
- Handle errors
- Use async/await


example.js

```javascript

const API_URL = 'https://jsonplaceholder.typicode.com/users';

async function getUser(userId) {
    const response = await fetch(`${API_URL}/${userId}`);

    if (!response.ok) {
        throw new Error(`Fetch failed with status: ${response.status}`);
    }

    return response.json();
}

(async function () {
    try {
        const user = await getUser(5);
        console.log(user.email);
    } catch (error) {
        console.error('There was an error: ' + error);
    }
})();

```

fetch - multiple async/await

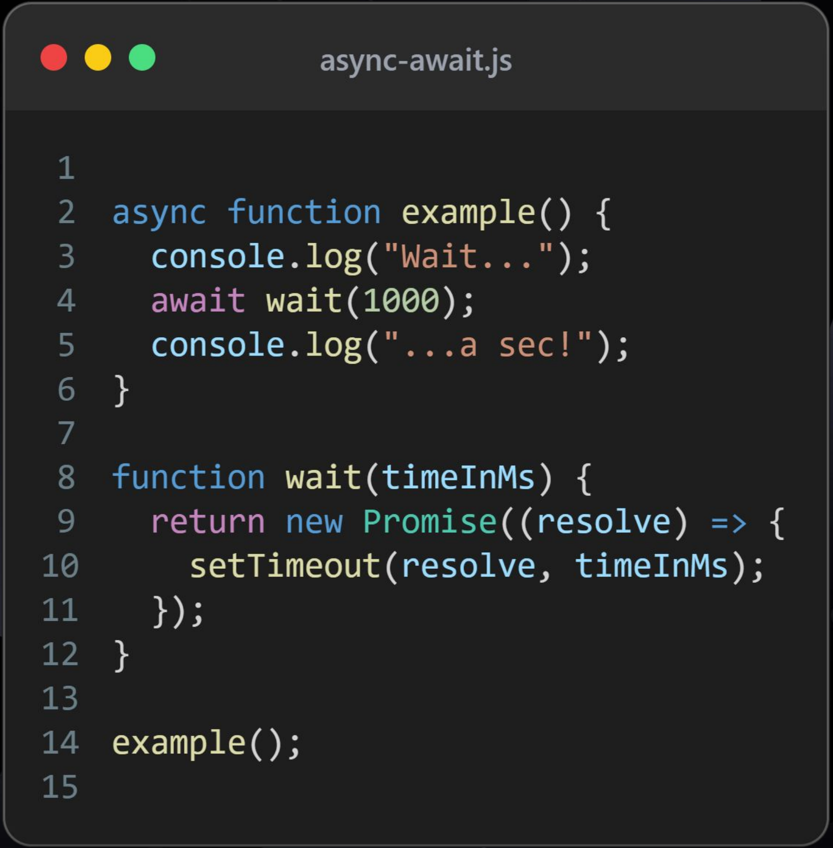
- For a given array of ids (ex. [2,5,6,8])
- Fetch all users with corresponding ids
- console log user emails
- Use “https://jsonplaceholder.typicode.com/users” endpoint
- Handle errors (try/catch)
- Use async/await


example.js

```javascript

const API_URL = 'https://jsonplaceholder.typicode.com/users';

async function getUser(userId) {
  const response = await fetch(`${API_URL}/${userId}`);

  if (!response.ok) {
    throw new Error(`Fetch failed with status: ${response.status}`);
  }

  return response.json();
}

function getUsers(ids) {
  const userPromises = ids.map((id) => getUser(id));
  return Promise.all(userPromises);
}

const ids = [2, 5, 6, 8];

(async function () {
  try {
    const users = await getUsers(ids);
    users.forEach((user) => console.log(user.email));
  } catch (error) {
    console.error('There was an error: ' + error);
  }
})();

```

wait

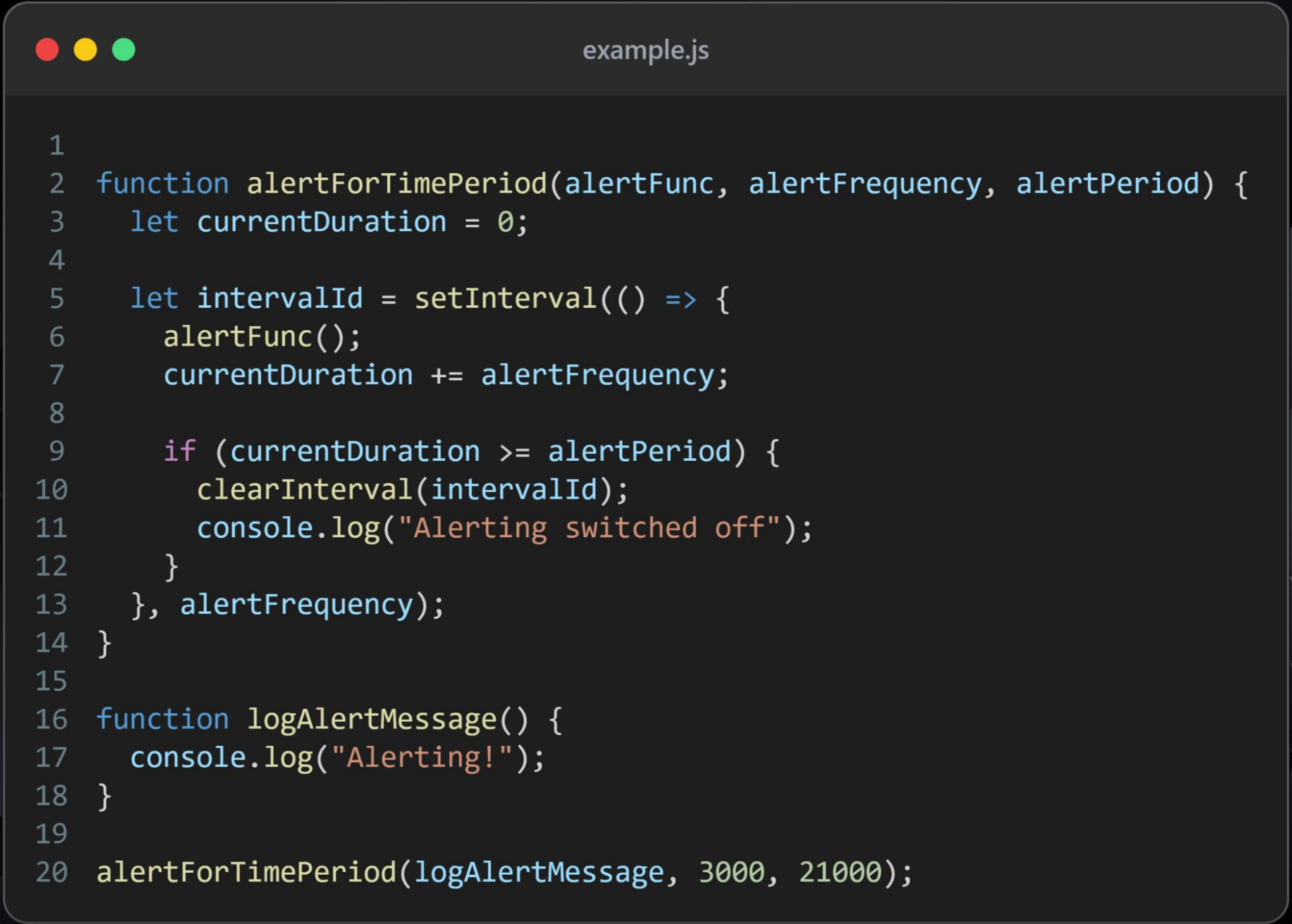
- Create a function that will allow us to wait for a certain amount of milliseconds before running the next taks


async-await.js

```javascript

async function example() {
  console.log("Wait...");
  await wait(1000);
  console.log("...a sec!");
}

function wait(timeInMs) {
  return new Promise((resolve) => {
    setTimeout(resolve, timeInMs);
  });
}

example();

```

alert

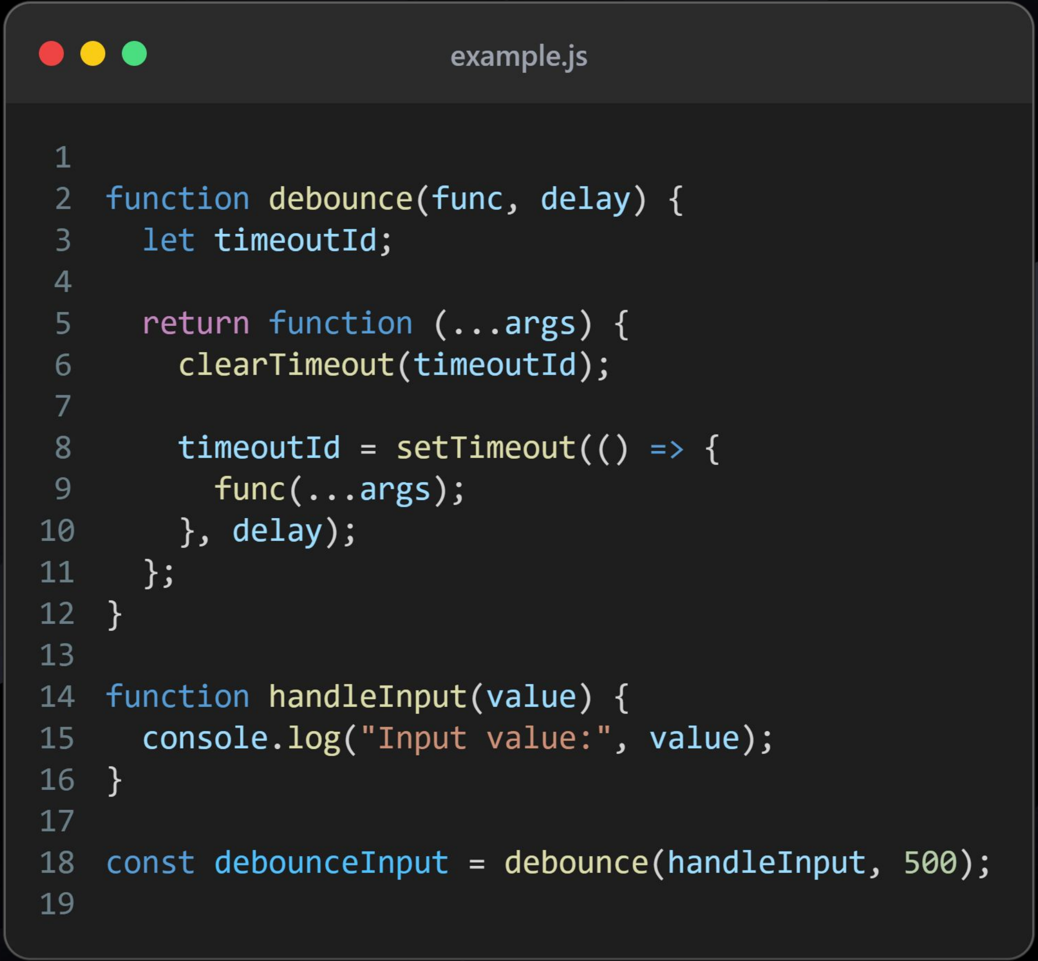
- Create a function that will display a message in the console at specified intervals for a given duration


example.js

```javascript

function alertForTimePeriod(alertFunc, alertFrequency, alertPeriod) {
  let currentDuration = 0;

  let intervalId = setInterval(() => {
    alertFunc();
    currentDuration += alertFrequency;

    if (currentDuration >= alertPeriod) {
      clearInterval(intervalId);
      console.log("Alerting switched off");
    }
  }, alertFrequency);
}

function logAlertMessage() {
  console.log("Alerting!");
}

alertForTimePeriod(logAlertMessage, 3000, 21000);
```

debounce

- Create a debounce function that allows efficient handling of time-consuming tasks



```javascript

function debounce(func, delay) {
  let timeoutId;

  return function (...args) {
    clearTimeout(timeoutId);

    timeoutId = setTimeout(() => {
      func(...args);
    }, delay);
  };
}

function handleInput(value) {
  console.log("Input value:", value);
}

const debounceInput = debounce(handleInput, 500);

```

HOMEWORK

- Random Promise generator
- Fetch from API
- IIFE closure
- 2 EXAM tasks