

Web Developer

Programmazione - Javascript e Typescript

Docente: Shadi Lahham

Browser APIs

interfacing with browsers

Shadi Lahham - Web development

Browser APIs Overview

Browser APIs, also known as web APIs, are native features within the browser that can be utilized in web applications, allowing for the implementation of features with minimal code, such as network requests

Limitations:

- Not fully standardized yet so support may differ across different browsers
- Developers need to be aware of differences and implement fallbacks
- Standardization and cross-browser support has improved over the years

Important categories of browser APIs

1. DOM manipulation APIs
2. Fetch API
3. Device APIs
4. Graphics APIs
5. Storage APIs
6. Audio and Video APIs
7. Interface APIs
8. Communication APIs
9. Web Workers API

DOM manipulation APIs

1.DOM manipulation APIs

The way web pages are organized and controlled with JavaScript is through the Document Object Model (DOM), while the Browser APIs enable the selection, addition, removal, and modification of web page elements

The DOM API falls into this group of APIs, allowing you to create, delete, and dynamically apply new styles to your page's HTML and CSS

DOM manipulation APIs

// Getting elements

// 1. getElementById(): this method selects an element by its id attribute

```
let element = document.getElementById('myElement');
```

// 2. getElementsByClassName(): selects elements by their class name

```
let elements = document.getElementsByClassName('myClass');
```

// 3. getElementsByTagName(): selects elements by their tag name

```
let elements = document.getElementsByTagName('p');
```

// 4. querySelector(): selects the first matching element that matches the specified CSS selector

```
let element = document.querySelector('.myClass');
```

// 5. querySelectorAll(): selects all matching elements that match the specified CSS selector

```
let elements = document.querySelectorAll('.myClass');
```

DOM manipulation APIs

// Adding elements:

// 1. createElement(): creates a new HTML element

```
let newElement = document.createElement('p');
```

// 2. innerHTML or textContent: sets the content of an element

```
newElement.innerHTML = '<strong>Hello World!</strong>';
```

// 3. appendChild(): adds an element to the end of the selected element

```
element.appendChild(newElement);
```


DOM manipulation APIs

// Removing elements:

// 1. removeChild(): removes a child element from the selected element
`element.removeChild(childElement);`

// 2. remove(): removes the selected element
`element.remove();`

// 3. innerHTML: sets the content of the selected element to an empty string
`element.innerHTML = '';`

Fetch API

2.Fetch API

The Browser APIs offer a way to send HTTP requests from the front-end of a web app, enabling live updates of dynamic content without the need to refresh the page

This functionality enables communication with a web server and allows for responses in JSON, plain text, or XML format

The Fetch API, a contemporary substitute for XHR, was introduced in modern browsers to simplify asynchronous HTTP requests

Fetch API

```
const endpoint = 'https://run.mocky.io/v3/fake';

fetch(endpoint)
  .then(response => {
    if (response.ok) {
      return response.json();
    } else {
      throw new Error('Network response was not ok.');
```



```
    }
  })
  .then(data => {
    // do something with data
    console.log(data);
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
```

Using xhr

```
// without fetch, using xhr
const xhr = new XMLHttpRequest();
const endpoint = 'https://run.mocky.io/v3/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx';
xhr.open('GET', endpoint);
xhr.onload = function () {
  if (xhr.status === 200) {
    try {
      const data = JSON.parse(xhr.responseText);
      console.log(data);
    } catch (error) {
      console.error('Error parsing response JSON:', error);
    }
  } else {
    console.error('Request failed. Returned status of ' + xhr.status);
  }
};
xhr.send();
```

Fetch API support

The `fetch()` function is supported by all modern web browsers:

- Chrome 42+
- Firefox 39+
- Safari 10.1+
- Edge 14+
- Opera 29+

Fetch API support

Older browsers that do not support `fetch()`:

- Internet Explorer (all versions)
- Opera Mini (all versions)
- Android Browser (before version 4.4)
- UC Browser (before version 11.4)

In such cases, a polyfill can be utilized to provide the `fetch()` function for browsers that lack native support such as the [whatwg-fetch](#) polyfill

Fetch an image

```
const resource = 'https://picsum.photos/400/200';
fetch(resource)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.blob();
  })
  .then(blob => {
    const imageUrl = URL.createObjectURL(blob);

    // create a new image element
    const img = new Image();
    img.onload = () => document.body.appendChild(img);
    img.onerror = () => console.error('Failed to create image element.');
```

imageUrl

```
    img.src = imageUrl;
  })
  .catch(error => console.error('Error fetching image:', error));
```


Request an image

```
// without fetch, using xhr
const xhr = new XMLHttpRequest();
xhr.onload = () => {
  if (xhr.status === 200) {
    const blob = xhr.response;
    const imageUrl = URL.createObjectURL(blob); // can use the url now
    // create a new image element
    const img = new Image();
    img.onload = () => document.body.appendChild(img);
    img.onerror = () => console.error('Failed to create image element.');
```

// create a new image element

```
    img.src = imageUrl;
  }
};
xhr.onerror = () => console.error('Error fetching image.');
```

// create a new image element

```
const resource = 'https://picsum.photos/400/200';
xhr.open('GET', resource);
xhr.responseType = 'blob';
xhr.send();
```

Put using Fetch

```
const endpoint = 'https://fake.pipedream.net';

fetch(endpoint, {
  method: 'PUT',
  body: JSON.stringify({ name: 'John', age: 30 }),
  headers: {
    'Content-Type': 'application/json'
  }
})
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
.then(data => console.log(data))
.catch(error => console.error('Error making PUT request:', error));
```

Async post using fetch

```
async function makePostRequest(endpoint, payload) {  
  const headers = new Headers();  
  headers.append('Content-Type', 'application/json; charset=UTF-8');  
  headers.append('Accept', 'application/json'); // client wants JSON data in response  
  
  try {  
    const options = {method: 'POST', headers: headers, body: JSON.stringify(payload)};  
  
    const response = await fetch(endpoint, options);  
    if (!response.ok) {  
      throw new Error('Network response was not ok.');    }  
    const data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.error('Error making POST request:', error);  
  }  
}
```

Async post using fetch

```
const endpoint = 'https://xxxxxxxxxxxxx.x.pipedream.net';

const payload = {
  name: 'Jerry Duncan',
  email: 'jerry.duncan@mail.com'
};

makePostRequest(endpoint, payload);
console.log(`rest of code doesn't wait for the request`);
```

Device APIs

3.Device APIs

APIs for mobile devices enable web applications to access and utilize the features of modern devices

One such example is the Geolocation API, which utilizes the device's capacity to determine the user's geographic location, making it possible to provide location-based services within web applications

This is done using various methods such as GPS, Wi-Fi, and IP address
With this API, developers can query the device's location, watch the device's location in real-time, and handle location-related errors

Geolocation API

```
// note: the following code will be blocked if not triggered by a user event  
//      it also need to run in a web server and the user has to allow location in the browser  
  
if (navigator.geolocation) {  
  // even if geolocation is supported, it won't work if the user doesn't give permission  
  navigator.geolocation.getCurrentPosition(position => {  
    console.log(`Latitude: ${position.coords.latitude} Longitude: ${position.coords.longitude}`);  
  });  
} else {  
  console.log('Geolocation is not supported by this browser.');}
```

Geolocation API

`Geolocation.getCurrentPosition()`

used to fetch the present location of a device

`Geolocation.watchPosition()`

used to set up a function to handle location updates automatically whenever the device's position changes. The function returns the updated location

Both of these methods can take up to 3 parameters:

1. `successCallback`

specifies a function to execute when the current position is successfully retrieved

2 `errorCallback`

specifies a function to execute if there was an error while retrieving the current position

3 `options`

An optional parameter that specifies additional options for retrieving the current position

Geolocation API examples

Mapping and Navigation: Google Maps and Waze use geolocation to track the user's location and provide directions

Local Search: Yelp and Foursquare identify the user's location and suggest nearby places to eat, drink, and shop

Weather Reports: Weather applications use geolocation to provide forecasts for the user's current location

Social Media: Snapchat, Instagram, and Facebook use geolocation to add location-based filters and tags to user-generated content

Graphics APIs

4.Graphics APIs

Within web applications, the canvas API offered by browser APIs enables the creation of interactive and dynamic graphics

Nowadays, modern web browsers support the production of graphics on the internet using the Canvas API

Canvas API

The Canvas API is a JavaScript and HTML interface for creating dynamic graphics. It allows you to produce a wide range of shapes, objects, and styles on a web page.

At present, the Canvas API is limited to two-dimensional graphics. The HTML canvas element provides a designated area on a web page where you can create dynamic graphics using JavaScript.

It is a powerful tool for creating animations, games, charts, graphs, and other visual effects, and is compatible with all major web browsers.

Canvas API

html

```
<canvas id="myCanvas"></canvas>
```

Javascript

```
// get the canvas element from the HTML document
```

```
const canvas = document.getElementById('myCanvas');
```

```
// set the canvas dimensions
```

```
canvas.width = 500;
```

```
canvas.height = 500;
```

```
// get the canvas context, which is used for drawing
```

```
const ctx = canvas.getContext('2d');
```

```
// Set the background color to Light blue
```

```
ctx.fillStyle = 'lightblue';
```

```
ctx.fillRect(0, 0, canvas.width, canvas.height);
```

Canvas API

// Draw other shapes or graphics on top of the background

// draw a rectangle

```
ctx.fillStyle = '#bada55';  
ctx.fillRect(10, 10, 50, 50);
```

// draw text

```
ctx.font = '30px Arial';  
ctx.fillStyle = '#7555da';  
ctx.fillText('Hello World', 20, 80);
```

Canvas animate example

The following is a more complex example of how to use Canvas API for 2D animations

Canvas animate example

```
// Get the canvas element from the HTML document  
const canvas = document.getElementById('myCanvas');  
  
// Set the canvas dimensions  
canvas.width = 500;  
canvas.height = 500;  
  
// Get the canvas context, which is used for drawing  
const ctx = canvas.getContext('2d');  
  
// Set the initial ball position, size, and velocity  
let x = 50;  
let y = 50;  
let radius = 20;  
let dx = 5;  
let dy = 5;
```


Canvas animate example

```
// Set the initial fill color and trail color
```

```
let fillColor = 'red';
```

```
let trailColor = 'rgba(255, 0, 0, 0.2)';
```

```
// Define the animation loop
```

```
function animate() {
```

```
// Clear the canvas before drawing the next frame
```

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

```
// Draw the trail behind the ball
```

```
ctx.fillStyle = trailColor;
```

```
ctx.beginPath();
```

```
ctx.arc(x, y, radius, 0, 2 * Math.PI);
```

```
ctx.fill();
```

```
// Update the ball position
```

```
x += dx;
```

```
y += dy;
```

Canvas animate example

```
// Check if the ball has hit the edges of the canvas
if (x + radius > canvas.width || x - radius < 0) {
    dx = -dx;
    fillColor = getRandomColor();
}

if (y + radius > canvas.height || y - radius < 0) {
    dy = -dy;
    fillColor = getRandomColor();
}

// Draw the ball
ctx.fillStyle = fillColor;
ctx.beginPath();
ctx.arc(x, y, radius, 0, 2 * Math.PI);
ctx.fill();
```

Canvas animate example

```
// Request the next animation frame  
requestAnimationFrame(animate);  
}  
  
// Helper function to generate a random color  
function getRandomColor() {  
  const r = Math.floor(Math.random() * 256);  
  const g = Math.floor(Math.random() * 256);  
  const b = Math.floor(Math.random() * 256);  
  return `rgb(${r}, ${g}, ${b})`;  
}  
  
// Start the animation Loop  
animate();
```

Storage APIs

5.Storage APIs

Browser APIs offer functionalities for saving data within a web browser, known as Web Storage. This feature is helpful in preserving user preferences, app data, or any other relevant information that needs to be stored locally

Web Storage API enables the client-side storage of data and has two modes:

- `sessionStorage`
- `localStorage`

The former stores data temporarily during a session, while the latter keeps it even after the browser is closed

sessionStorage

```
// Session Storage  
// stores data temporarily  
// data is lost when closing the browser  
  
// Set data in session storage  
sessionStorage.setItem('key', 'value');  
  
// Retrieve data from session storage  
let data = sessionStorage.getItem('key');  
console.log(data); // Output: 'value'
```

localStorage

```
// Local Storage  
// stores data with no expiration date  
// data persists even after closing the browser or shutting down the PC  
  
// Set data in Local storage  
localStorage.setItem('key', 'value');  
  
// Retrieve data from Local storage  
let data = localStorage.getItem('key');  
console.log(data); // Output: 'value'
```

Storage APIs

// Web storage can be used to store objects in the form of JSON strings

// Set object in Local storage

```
let object = { name: 'John', age: 30 };  
localStorage.setItem('obj', JSON.stringify(object));
```

// Retrieve object from Local storage

```
let data = JSON.parse(localStorage.getItem('obj'));  
console.log(data.name); // Output: 'John'
```


Cookies vs Local Storage

- Both are mechanisms for storing data on the client-side
- Both can be used to store
 - user preferences
 - login information
 - other types of data
- Cookies are small text files stored on the user's computer by the browser
- Local Storage is a key-value store that stores large amounts of data
- Limitations of cookies
 - Store up to 4KB of data
 - Sent to the server with every HTTP request
 - Have an expiration date set by the server
- Local Storage never expires. Remains until removed by user or developer

Setting and Getting a Cookie

```
// set a cookie with a key of "username" and a value of "John Doe" that expires in 7 days
document.cookie =
  'username=John Doe; expires=' + new Date(Date.now() + 7 * 24 * 60 * 60 * 1000).toUTCString();

// get the value of the "username" cookie
const cookieValue = document.cookie.replace(
  /(?:(?:^|.*;)\s*)username\s*\=\s*([^;]*).*$)|^.*$/,
  '$1'
);
console.log(cookieValue); // outputs "John Doe"

// note:
// some browsers don't store cookies with the file:/// protocol. Use a webserver
```

Audio and Video APIs

6.Audio and Video APIs

Web browser APIs offer functionalities for playing and managing audio and video, enabling web applications to deliver immersive multimedia experiences

These APIs are capable of handling, showcasing, and generating various media formats

In terms of JavaScript manipulation, the `<video>` and `<audio>` elements are essentially the same. They both have similar APIs for controlling playback, adjusting volume, seeking to specific times, and handling events

Audio and video elements

```
<audio controls>  
  <source src="example.mp3" type="audio/mpeg">  
  Your browser does not support the audio element.  
</audio>
```

```
<video controls>  
  <source src="example.mp4" type="video/mp4">  
  Your browser does not support the video element.  
</video>
```

Note

Using the API we can manipulate audio and video without the controls attribute

Audio API

```
<body>
  <audio>
    <source src="./song.mp3" type="audio/mpeg">
    Your browser does not support the audio element.
  </audio>

  <button>Seek and play</button>

  <!-- end of body -->
  <script src="./main.js"></script>
</body>
```

Audio API

```
// get the first audio element first button  
const audio = document.querySelector('audio');  
const btn = document.querySelector('button');  
  
// Play audio  
const playAudio = () => audio.play();  
  
// Pause audio  
const pauseAudio = () => audio.pause();  
  
// Seek to a specific time in the audio (in seconds)  
const seekAudio = time => (audio.currentTime = time);  
  
// Set the volume of the audio (0.0 to 1.0)  
const setVolume = volume => (audio.volume = volume);  
  
// Get the duration of the audio (in seconds)  
const getDuration = () => audio.duration;
```

Audio API

```
// Get the current playback time of the audio (in seconds)  
const getCurrentTime = () => audio.currentTime;  
  
// Listen for when the audio has ended  
audio.onended = () => console.log('Audio has ended');  
  
// Listen for when the audio playback position has changed  
audio.ontimeupdate = () => console.log(`Current time:${audio.currentTime}`);  
  
// Add event listeners to the buttons  
btn.addEventListener('click', () => {  
  seekAudio(180); // 3 minutes  
  setVolume(0.5);  
  playAudio();  
});
```


Interface APIs

7.Interface APIs

Drag and drop is an example of Interface APIs

With the drag and drop APIs, users can perform drag and drop operations on a web page, facilitating the creation of advanced user interface designs like file managers, image galleries, and task management tools

This feature allows users to drag files or images from their computer and drop them on a web page or move elements within a web page to alter their position or perform other actions

Drag and drop API

HTML5 drag and drop API

- Users can move images, text, and files by clicking and dragging
- Web developers can create user-friendly interfaces
- Several events to handle drag and drop events
 - dragstart, dragover, dragenter, dragleave, drop, and dragend
- Provides a set of methods to handle data in drag and drop interactions

Simple drag and drop

HTML

```
<div id="drag-item" class="draggable">Drag me</div>  
<div id="target" class="droppable">Drop here</div>
```

CSS

```
.draggable {  
  width: 100px;  
  background-color: orange;  
  cursor: move;  
}
```

```
.droppable {  
  width: 200px;  
  height: 100px;  
  border: 1px solid black;  
}
```

Simple drag and drop

JAVASCRIPT

```
const dragElement = document.getElementById("drag-item");
const dropElement = document.getElementById("target");

dragElement.setAttribute("draggable", true); // assign "draggable" attribute to element

// set the data that will be carried during the drag operation
dragElement.ondragstart = event => event.dataTransfer.setData("text/plain", event.target.id);

dropElement.ondragover = event => event.preventDefault();

dropElement.ondrop = event => {
  event.preventDefault();
  const data = event.dataTransfer.getData("text/plain");
  event.target.appendChild(document.getElementById(data));
};
```

Drag and drop example

The following is a more complex example but uses the same methods and events as before

This example changes the image that appears below the cursor during the drag operation. This can be set to any image

The drag image works in Chrome, but the code requires changes to work in Firefox and other browsers

Drag and drop example

```
<!DOCTYPE html>
<html>
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href="./drag.css">
  <link rel="icon"
    href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22 viewBox=%220 0 20 19%22><text
y=%2215px%22><img alt="apple icon" data-bbox="178 491 198 521"/></text></svg>">
</head>
<body>
  <!-- note the 'draggable' attribute below -->
  <div id="tree" class="box draggable" draggable="true"><img alt="tree icon" data-bbox="541 641 558 668"/><img alt="tree icon" data-bbox="561 641 578 668"/><img alt="tree icon" data-bbox="581 641 598 668"/><img alt="tree icon" data-bbox="601 641 618 668"/><img alt="tree icon" data-bbox="621 641 638 668"/><img alt="tree icon" data-bbox="641 641 658 668"/><img alt="tree icon" data-bbox="661 641 678 668"/></div>
  <div id="truck" class="box drop-target">
    <p><img alt="truck icon" data-bbox="131 718 148 745"/></p>
  </div>
  <script src="./drag.js"></script>
</body>
</html>
```

Drag and drop example

```
body { font-size: 36px; }
```

```
.box, #truck > p {  
  border: 1px dashed #000000;  
  padding: 10px;  
  border-radius: 6px;  
}
```

```
.box {  
  width: 350px;  
  margin-bottom: 10px;  
  background-color: #f5f5f5;  
}
```

```
#tree { background-color: #bada55; }
```


Drag and drop example

```
#truck {  
  display: flex;  
  align-content: flex-start;  
  flex-wrap: wrap;  
}  
  
#truck > p {  
  pointer-events: none; /* important to prevent drag leave on children */  
  background-color: #dbc7c0;  
  margin: 4px;  
}  
  
.draggable { cursor: move; }  
.highlighted { background-color: #d9d954; }  
.semi-transparent { opacity: 0.5; }
```

Drag and drop example

```
// get draggable and target elements
const draggable = document.querySelector('.draggable');
const dropTarget = document.querySelector('.drop-target');

// toggleable classnames for styling
const highlightClass = 'highlighted';
const transparentClass = 'semi-transparent';

// drag payload and format - what gets carried
const dragFormat = 'text/plain';
const getDragPayload = () => (Math.random() < 0.5 ? '🍎' : '🍏'); // can be a simple string

// optional drag image; pre-loaded since loading during drag takes too long
// using a code image here, but could also have been dragImg.src = './example.jpg'
const dragImg = new Image();
dragImg.src = `data:image/svg+xml,
<svg xmlns=%22http://www.w3.org/2000/svg%22 viewBox=%220 0 20 19%22>
  <text y=%2215px%22>🍎</text>
</svg>`;
```

Drag and drop example

```
// implement drag event handlers
const handleDragStart = event => {
  // set the drag image to the preloaded image; this is optional
  event.dataTransfer.setDragImage(dragImg, 0, 0);

  // set the drag content; what actually gets carried during the drag
  event.dataTransfer.setData(dragFormat, getDragPayload());
  event.currentTarget.classList.add(transparentClass);
};

const handleDragEnd = event => event.currentTarget.classList.remove(transparentClass);

// prevent browser default behavior doesn't allow the element to be dropped
const handleDragOver = event => event.preventDefault();

const handleDragEnter = event => event.currentTarget.classList.add(highlightClass);

const handleDragLeave = event => event.currentTarget.classList.remove(highlightClass);
```

Drag and drop example

```
const handleDrop = event => {  
  event.preventDefault(); // prevent browser default action which opens some elements as a link  
  const text = event.dataTransfer.getData(dragFormat);  
  const elem = document.createElement('p');  
  elem.textContent = text;  
  event.currentTarget.appendChild(elem);  
  event.currentTarget.classList.remove(highlightClass);  
};  
  
// setup event handlers  
draggable.addEventListener('dragstart', handleDragStart);  
draggable.addEventListener('dragend', handleDragEnd);  
dropTarget.addEventListener('dragover', handleDragOver);  
dropTarget.addEventListener('dragenter', handleDragEnter);  
dropTarget.addEventListener('dragleave', handleDragLeave);  
dropTarget.addEventListener('drop', handleDrop);
```

Communication APIs

8.Communication APIs

WebSocket:

WebSockets is a persistent protocol for real-time communication that allows the server to push data to the client, making it ideal for applications that require real-time updates like gaming, financial trading, and chat apps

WebRTC: (Real-time communication technology)

WebRTC facilitates real-time audio and video streaming and data transfer directly between two peers over the internet, without an intermediary, enabling instant communication in web applications

WebSocket

- protocol for real-time communication between client and server
- enables server to push data to client without need for active data requests
- establishes persistent connection between server and client
- both client and server can send messages at any time
- ideal for apps requiring real-time updates
 - gaming
 - financial trading
 - chat apps
 - collaborative work tools

WebSocket

WebSocket connection establishment and communication

- Client sends handshake request to server to establish WebSocket connection
- Server sends acknowledgment message to client
- Client and server can send messages to each other
 - text, JSON, or other data format
- Client updates UI in real-time upon receiving a message from server
- WebSocket connection remains open until
 - client closes it
 - or server terminates it

WebSocket - client

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Websocket example</title>
</head>

<body>
  <input type="text">
  <button>send</button>
  <script src="./client.js"></script>
</body>

</html>
```

WebSocket - client

```
const input = document.querySelector('input');
const btn = document.querySelector('button');

// create a new WebSocket object
const socket = new WebSocket('ws://localhost:8080');

// event handler for when the WebSocket connection is established
socket.onopen = () => {
  console.log('WebSocket connection established');

  const doSend = () => {
    // send a message to the server
    socket.send(input.value);
    input.value = '';
  };
  btn.onclick = doSend;
  input.onkeypress = e => (e.key === 'Enter' ? doSend() : null);
};
```

WebSocket - client

// event handler for incoming messages from the server

```
socket.onmessage = event => {  
  document.body.appendChild(  
    document.createElement('p')  
  ).textContent = `Received message from server: ${event.data}`;  
};
```

// event handler for when the WebSocket connection is closed

```
socket.onclose = event => console.log(`WebSocket connection closed with code ${event.code}`);
```

WebSocket - server

To run the server you need to

- install ws with the command
 - **npm install ws**
- put the server code in a file
 - server.js
- run the program
 - node server.js

WebSocket - server

```
// create a new websocket server - NodeJS code
const WebSocket = require('ws');
const wss = new WebSocket.Server({ port: 8080 });

// event handler - new websocket connection is established
wss.on('connection', ws => {
  console.log('New WebSocket connection established');

  // event handler - incoming messages from client
  ws.on('message', message => {
    // Log and send message back to client
    console.log(`Received message from client: ${message}`);
    ws.send(`You said: ${message}`);
  });

  // event handler for when the websocket connection is closed
  ws.on('close', () => console.log('WebSocket connection closed'));
});
```

Web Workers API

Web Workers API

Web applications can perform intricate tasks without disrupting the main thread through the utilization of web workers provided by browser APIs

Overall, web workers are a powerful tool for frontend developers to improve the performance and responsiveness of web applications

By offloading complex or time-consuming tasks to a separate background thread, web developers can create faster and more efficient web applications that provide a better user experience

Web Workers API

Web workers can be used in frontend development for tasks such as:

- Image processing
- Computationally-intensive tasks
- Real-time communication
- Game development
- Data visualization
- Offline functionality
- Background tasks

Web Worker - HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Worker example</title>
  <link rel="stylesheet" href="./style.css">
</head>

<body>
  <div id="prime-list"></div>
  <script src="./main.js"></script>
</body>

</html>
```

Web Worker - CSS

```
body { font-size: 40px; max-width: 1000px; margin: 0 auto; }
```

```
.tortoise, .hare {  
  padding: 5px;  
  border: 1px dashed black;  
  margin: 5px;  
  border-radius: 5px;  
}
```

```
.hare { background-color: #ffb74d; }  
.tortoise { background-color: #cddc39; }
```

```
#prime-list {  
  display: flex;  
  justify-content: flex-start;  
  flex-wrap: wrap;  
}
```

Web Worker - main.js

```
const primeList = document.getElementById('prime-list');
```

```
// utility function
```

```
const createElem = (content, className) => {  
  const elem = document.createElement('div');  
  elem.textContent = content;  
  elem.classList.add(className);  
  return elem;  
};
```

Web Worker - main.js

```
// create and start workers  
const worker = new Worker('./worker.js');  
worker.postMessage({ start: 0, end: 100 });  
  
const worker2 = new Worker('./worker.js');  
worker2.postMessage({ start: 0, end: 100 });  
  
worker.addEventListener('message', ({ data }) => {  
  if (data === 'end') worker.terminate();  
  primeList.appendChild(createElem(`🐰 ${data}`, 'hare'));  
});  
  
worker2.addEventListener('message', ({ data }) => {  
  if (data === 'end') worker2.terminate();  
  primeList.appendChild(createElem(`🐢 ${data}`, 'tortoise'));  
});
```

// the rest of the code can continue to do other things while the workers work

Web Worker - worker.js

```
const isPrime = n => {  
  if (n < 2) return false;  
  for (let i = 2; i <= Math.sqrt(n); i++) {  
    if (n % i === 0) return false;  
  }  
  return true;  
};
```

Web Worker - worker.js

```
async function findPrimesInRange(start, end) {
  const delayFactor = 1.05;
  for (let i = start; i <= end; i++) {
    // waste time
    await new Promise(resolve =>
      setTimeout(resolve, Math.floor(Math.random() * delayFactor) * 1000)
    );

    // check if prime
    if (isPrime(i)) self.postMessage(i);
  }
  self.postMessage('end');
}

self.addEventListener('message', ({ data }) => {
  const { start, end } = data;
  findPrimesInRange(start, end);
});
```

Prime testing

To test if a number is prime, check if it has any divisors other than 1 and itself. When factors of a composite number are examined, if both factors are greater than the square root of the number, their product would exceed the number, which contradicts the definition of a composite number. So at least one of the factors must be less than or equal to the square root of the number

If no divisors are found up to the square root, there won't be any divisors beyond it because any additional divisor would result in a product greater than the original number

Thus it can be concluded that the number is prime

Your turn

1.My album

Create a page with 10 random images that are dynamically loaded using requests when the user opens the page

The images should be of the same height and width. There should be 4 images per row, so a 4x3 grid that fits the 10 images

The user can click buttons to load a new image in real time, or remove an image

The user can also arrange the images in the gallery using drag and drop

The gallery should remember the loaded images and their order so that if the user returns to the page, the same images are loaded and keep the same order

References

[What are browser APIs?](#)

[Fetch API](#)

[Geolocation API](#)

[Canvas API](#)

[How to render 3D in 2D canvas](#)

[A basic introduction to Canvas API](#)

References

[Web Storage API](#)

[Storage - Web APIs](#)

[The Embed Audio element](#)

[Web Audio API](#)

[Video and Audio APIs](#)

[The Video Embed element](#)

References

[Drag operations](#)

[HTML Drag and Drop API](#)

[DataTransfer: setData\(\) method](#)

[DataTransfer: getData\(\) method](#)

[DataTransfer: setDragImage\(\) method](#)

References

[WebSocket](#)

[Web Workers API](#)