Node.JS & Express.JS

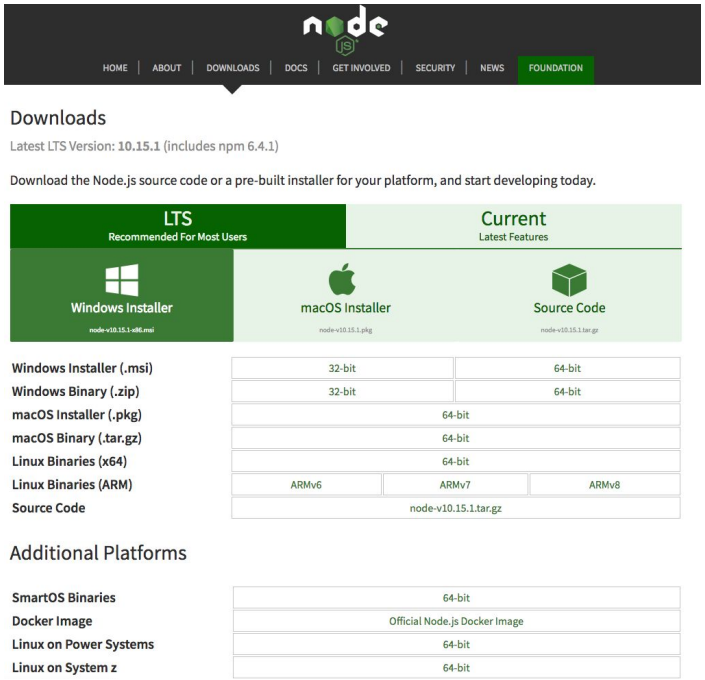
What is Node.JS ?

Node (or more formally Node.js) is an open-source, cross-platform, runtime environment that allows developers to create all kinds of server-side tools and applications in JavaScript.

- **Great performance!** Node was designed to optimize throughput and scalability in web applications and is a good solution for many common web-development problems (e.g. real-time web applications).
- Code is written in **"plain old JavaScript"**, which means that less time is spent dealing with "context shift" between languages when you're writing both client-side and server-side code.
- **JavaScript is a relatively new programming language** and benefits from improvements in language design when compared to other traditional web-server languages (e.g. Python, PHP, etc.) Many other new and popular languages compile/convert into JavaScript so you can also use TypeScript, CoffeeScript, ClojureScript, Scala, LiveScript, etc.
- **The node package manager (NPM)** provides access to hundreds of thousands of reusable packages. It also has best-in-class dependency resolution and can also be used to automate most of the build toolchain.
- **Node.js is portable.** It is available on Microsoft Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS, and NonStop OS. Furthermore, it is well-supported by many web hosting providers, that often provide specific infrastructure and documentation for hosting Node sites.
- **It has a very active third party ecosystem and developer community**, with lots of people who are willing to help.

Install Node.JS

https://nodejs.org/en/download/



Hello World

Create index.js file and put the following code:

```
console.log('Hello World');
```

Run "node index.js" in command line on the same directory.

Blocking vs. Non Blocking

Blocking

```
const fs = require('fs');
const data = fs.readFileSync('./file.txt', 'utf8'); // blocks here until file is read
console.log(data);

moreWork(); // will run after console.log
```

Non Blocking

```
const fs = require('fs');
fs.readFile('./file.md', 'utf8' , (err, data) => {
  if (err) throw err;
  console.log(data);
});

moreWork(); // will run before console.log
```

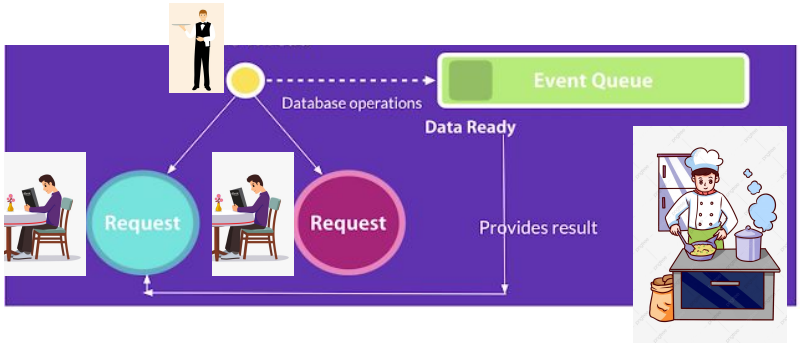
Think Non Blocking as a restaurant...

[Article Link]

When a person comes to the restaurant and calls the waiter and orders a dish to be prepared.

The waiters goes to the cook and tells the order.

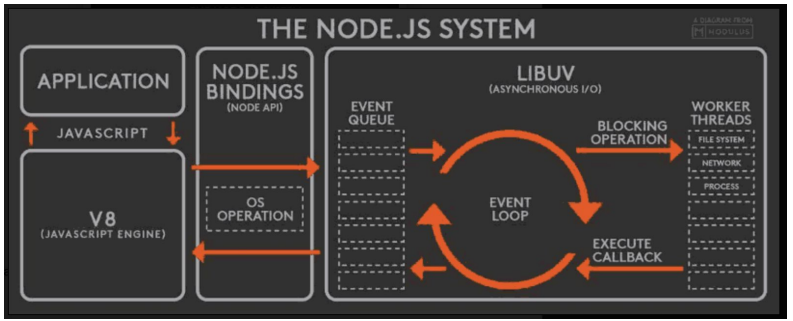
Then the waiter is now free to serve others guest and repeat the same process.



Here the **order** is the **request** and the **waiter** is the **event loop**.

He puts all the request inside a queue (**event queue**), assigns (**callback**) it to the cook (**libuv**, which provides the event loop and the non blocking i/o capabilities to NodeJS.) which is managed by the OS.

The waiter (**event loop**) continuously checks the **queue** that which process has been completed and returns back it to the customer with the **response** if its request has completed.



And Blocking as a troublesome guest...

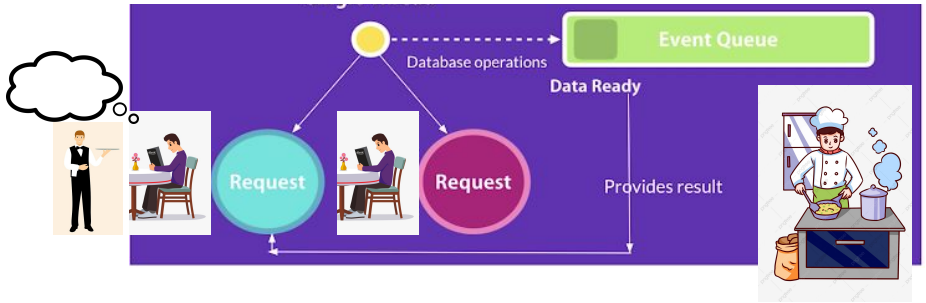
[Article Link]

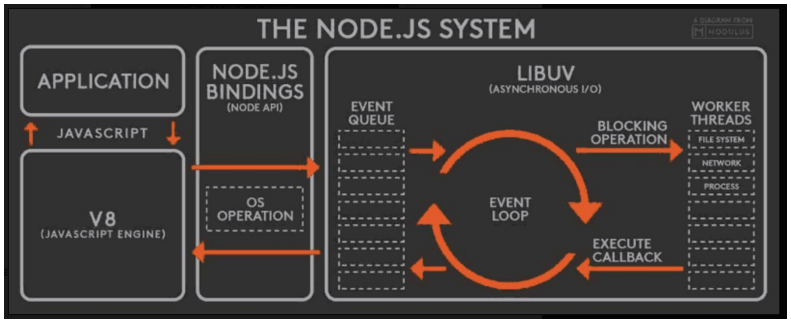
Suppose a guest comes and enquires the waiter for more information about the dish or wants it to be customise...

It would hold the waiter for few minutes and in that interval other customers have to wait.

Here the **request (guest)** is **blocking** and it could only be processed only when the required jobs has been done (The waiter gives the information about the dishes or customise it accordingly).

This can be a example of a big file to be read and then execute the other tasks after it.





readFile Example

```
const fs = require("fs");

// Read users.json file
fs.readFile("users.json", function(err, data) {

    // Check for errors
    if (err) throw err;

    // Converting to JSON
    const users = JSON.parse(data);

    console.log(users); // Print users
});
```

writeFile Example

```
const fs = require("fs");

// STEP 1: Reading JSON file
const users = require("./users");

// Defining new user
let user = {
        name: "New User",
        age: 30,
        language: ["PHP", "Go", "JavaScript"]
};

// STEP 2: Adding new data to users object
users.push(user);

// STEP 3: Writing to a file
fs.writeFile("users.json", JSON.stringify(users), err => {

        // Checking for errors
        if (err) throw err;

        console.log("Done writing"); // Success
});
```

Get Node Args - process.argv

```
console.log(process.argv);
```

```
$ node test.js one two three four five
```

```
[ 'node',
  'C:\elad\argvdemo\test.js',
  'one',
  'two',
  'three',
  'four',
  'five' ]
```

We can use this to get just the args:

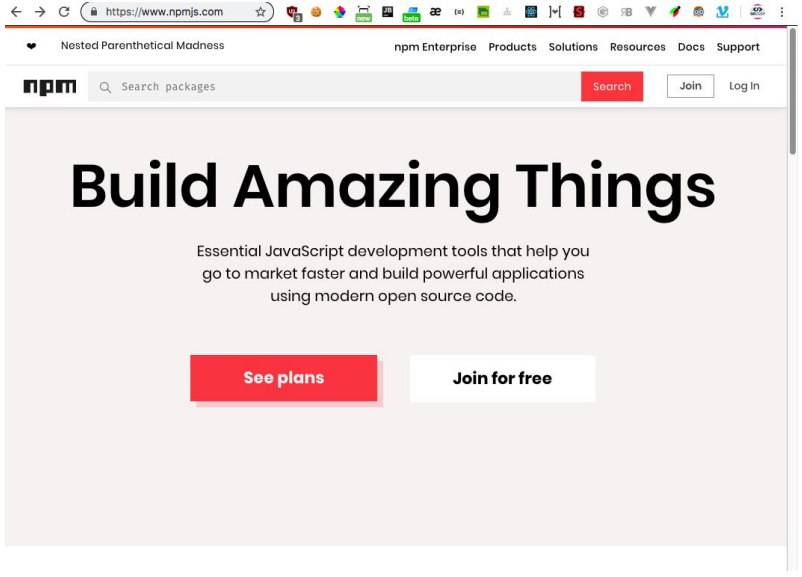
```
let myArgs = process.argv.slice(2);
console.log('myArgs: ', myArgs);
```

Hello World with response from server

```
// Load HTTP module
const http = require("http");

// Create HTTP server and listen on port 8000 for requests
http.createServer((request, response) => {

   // Set the response HTTP header with HTTP status and Content type
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body "Hello World"
   response.end('Hello World\n');
}).listen(8000);

// Print URL for accessing server
console.log('Server running at http://127.0.0.1:8000/');
```

NPM

NPM is the most important tool for working with *Node* applications. NPM is used to fetch any packages (JavaScript libraries) that an application needs for development, testing, and/or production, and may also be used to run tests and tools used in the development process.



package.json

package.json file lists all the dependencies for a specific JavaScript "package", including the package's name, version, description, initial file to execute, production dependencies, development dependencies, versions of *Node* it can work with, etc.

The **package.json** file should contain everything NPM needs to fetch and run your application (if you were writing a reusable library you could use this definition to upload your package to the npm repository and make it available for other users).

use "**npm init**" to create a package.json file for your project

npm install

This command **installs a package on node_modules** directory, and any packages that it depends on.

npm install (with no args, in package dir)
Installs all packages described in package.json

npm install [<@scope>/]<name>@<tag>

alias: **npm i**

Use
  npm i -D package-name
  or **npm i --save-dev package-name**

To install a package and put its name and version on devDependencies section on package.json
If a dependency is only used during development, you should instead save it as a "development dependency"
(so that your package users don't have to install it in production)

Use
  npm i -g package-name
To install a package globally in order to use it everywhere in your system and not specific into a project.

Use **nodemon** to auto restart server on code change

Either through cloning with git or by using npm (the recommended way):

npm install -g nodemon
And nodemon will be installed globally to your system path.

You can also install nodemon as a development dependency:

npm install --save-dev nodemon

Express.js

Express is the most popular *Node* web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

- Write **handlers** for requests with different HTTP verbs at different URL paths (routes).
- Integrate with **"view" rendering engines** in order to generate responses by inserting data into templates.
- **Set common web application settings** like the port to use for connecting, and the location of templates that are used for rendering the response.
- **Add** additional request processing "**middleware**" at any point within the request handling pipeline.

Install and use express.js library

Install express in your project directory:

```
npm install express
```

Change index.js into this:

```
const express = require('express')
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!')
});

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

Explanation for a basic express app

```
const express = require('express')
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!')
});

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

method

path

callback function with request, response objects

port

use express module

create a new express instance

Add a get route on '/' path and define a function with the request and response for this route

Send a response with the string 'Hello World!'

Listen to port 8000 for getting requests

REST API

Todos Example:

```
[
  {
    id: 1,
    title: 'Throw garbage',
    completed: false
  },
  {
    id: 2,
    title: 'Wash the dishes',
    completed: false
  }
];
```

GET /todos GET /todos/:id	Get todos list Get a specific todo
POST /todos	Add a new todo
PUT /todos/:id	Update existing todo
DELETE /todos/:id	Delete existing todo

How to call REST API and check the responses?

Man, Use PostMan!

https://www.getpostman.com/

DB - From file

```
const initialTodos = [
  {
    id: 1,
    title: 'Throw garbage',
    completed: false
  },
  {
    id: 2,
    title: 'Wash the dishes',
    completed: false
  }
];
```

```
const store = {
  async read() {
    try {
      await fs.access(fileName);
      this.todos = JSON.parse((await fs.readFile(fileName)).toString());
    } catch (e) {
      this.todos = initialTodos;
    }

    return this.todos;
  },

  async save() {
    await fs.writeFile(fileName, JSON.stringify(this.todos));
  },

  async getIndexById(id) {
    try {
      const todos = await this.read();
      return todos.findIndex(todo => todo.id === +id);
    } catch (e) {
      console.log(e);
    }
  },

  async getNextTodoId() {
    let maxId = 1;
    const todos = await this.read();
    todos.forEach(todo => {
      if (todo.id > maxId) maxId = todo.id;
    });
    return maxId + 1;
  },

  todos: []
};
```

REST API - GET /todos /todos/:id

```
app.get('/todos', (req, res) => {
  res.json(store.read());
});
```

/todos
Response
Example:

```
[
  {
    id: 1,
    title: 'Throw garbage',
    completed: false
  },
  {
    id: 2,
    title: 'Wash the dishes',
    completed: false
  }
];
```

req.params.id

```
app.get('/todos/:id', async (req, res) => {
  const todos = await store.read();
  const todo = todos.find(todo => todo.id === +req.params.id);
  res.json(todo);
});
```

/todos/1
Response
Example:

```
{
  id: 1,
  title: 'Throw garbage',
  completed: false
}
```

REST API - POST /todos

```
app.post('/todos', async (req, res) => {

  const todo = req.body;
  todo.id = await store.getNextTodoId();
  store.todos.push(todo);
  await store.save();
  res.json('ok');

});
```

To get the req.body we need to install body-parser:

npm i body-parser

and use it:

```
const bodyParser = require('body-parser');
...
const app = express();
...
app.use(bodyParser.json());
```

Called also express.js "middleware"

/todos Request Body Example:

```
{
  title: 'Email my boss',
  completed: false
}
```

todos array after adding todo:

```
[
  {
    id: 1,
    title: 'Throw garbage',
    completed: false
  },
  {
    id: 2,
    title: 'Wash the dishes',
    completed: false
  },
  {
    id: 3,
    title: 'Email my boss',
    completed: false
  }
];
```

REST API - PUT /todos/:id

req.params.id

```
app.put('/todos/:id', async (req, res) => {

  const index = await store.getIndexById(req.params.id);
  store.todos[index] = req.body;
  await store.save();
  res.json('ok');

});
```

/todos/3 Request Body Example:

```
{
  title: 'Email my boss',
  completed: true
}
```

todos array after updating todo:

```
[
  {
    id: 1,
    title: 'Throw garbage',
    completed: false
  },
  {
    id: 2,
    title: 'Wash the dishes',
    completed: false
  },
  {
    id: 3,
    title: 'Email my boss',
    completed: true
  }
];
```

REST API - DELETE /todos/:id

req.params.id

```
app.delete('/todos/:id', async (req, res) => {

  const index = await store.getIndexById(req.params.id);
  store.todos.splice(index, 1);
  await store.save();
  res.json('ok');

});
```

DELETE /todos/3

todos array after deleting todo 3:

```
[
  {
    id: 1,
    title: 'Throw garbage',
    completed: false
  },
  {
    id: 2,
    title: 'Wash the dishes',
    completed: false
  },
  {
    id: 3,
    title: 'Email my boss',
    completed: true
  }
];
```

Add CORS support to use this server from outside

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin.

A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, and port) than its own origin.

npm i cors

and use it:

```
const cors = require('cors');
...
const app = express();
...
app.use(cors());      Called also express.js "middleware"
```

Use node-fetch to test our routes

```
npm i node-fetch
```

and use it:

```
const fetch = require('node-fetch');
```

```
fetch(url)
  .then((resp) => resp.json()) // Transform the data into json
  .then((data) => {
    // Create and append the li's to the ul
    })
})
```

Use node-fetch to test our routes

```
app.get('/testGet', async (req, res) => {
  const fetchResp = await fetch('http://localhost:3000/todos');
  const json = await fetchResp.json();
  res.send(json);
});

app.get('/testPost', async (req, res) => {
  try {
    const fetchResp = await fetch('http://localhost:3000/todos', {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        title: 'check',
        completed: false
      })
    });
    const json = await fetchResp.json();
    res.send(json);
  } catch (e) {
    res.send(e);
  }
});
```

Upload a file

On Server:

```
app.post("/upload", (req, res) => {
 req.pipe(fs.createWriteStream(`images/${req.query.filename}`));
  res.send("WOW!");
});
```

On Client:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <script src="https://unpkg.com/axios@0.20.0-0/dist/axios.min.js"></script>
    <input type="file" id="uploadedFile" />
    <button onclick="uploadFile()">Upload File</button>
    <script>
      function uploadFile() {
        const uploadedFile = document.querySelector("#uploadedFile");
        axios.post("http://localhost:8000/upload", uploadedFile.files[0], {
          params: { filename: uploadedFile.files[0].name },
        });
      }
    </script>
  </body>
</html>
```

If our form is a little more complex, it is recommended to use **Multer**:

https://github.com/expressjs/multer

Upload a file - Add upload progress

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <script src="https://unpkg.com/axios@0.20.0-0/dist/axios.min.js"></script>
    <input type="file" id="uploadedFile" />
    <button onclick="uploadFile()">Upload File</button>
    <script>
      function uploadFile() {
        const uploadedFile = document.querySelector("#uploadedFile");

        axios.post("http://localhost:8000/upload", uploadedFile.files[0], {
          params: { filename: uploadedFile.files[0].name },
          onUploadProgress: (progressEvent) => {
            const percentCompleted = Math.round(
              (progressEvent.loaded * 100) / progressEvent.total
            );
            console.log(percentCompleted);
          },
        });
      }
    </script>
  </body>
</html>
```

Upload a file - With React as client

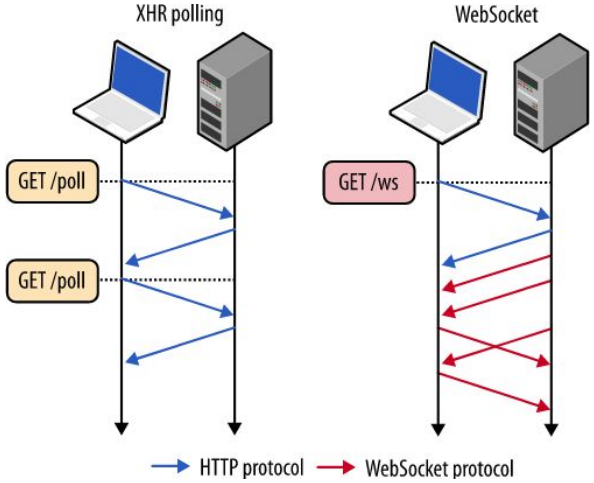
```
import React from "react";
import "./App.css";
import axios from "axios";
import { useRef } from "react";

const App = () => {
  const fileInput = useRef();
  const uploadImage = () => {
    const uploadedFile = fileInput.current;

    axios.post("http://localhost:8000/upload", uploadedFile.files[0], {
      params: { filename: uploadedFile.files[0].name },
      onUploadProgress: (progressEvent) => {
        const percentCompleted = Math.round(
          (progressEvent.loaded * 100) / progressEvent.total
        );
        console.log(percentCompleted);
      },
    });
  };
```

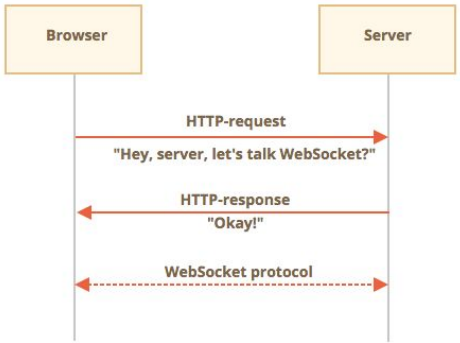
```
  return (
    <div className="App">
      <input type="file" ref={fileInput} />
      <br />
      <br />
            <button onClick={uploadImage}>Upload  Image</button>
    </div>
  );
};
export default App;
```

WebSocket

The **WebSocket** protocol, provides a way to exchange data between browser and server via a persistent connection. The data can be passed in both directions as “packets”, without breaking the connection and additional HTTP-requests.

WebSocket is especially great for services that require continuous data exchange, e.g. online games, real-time trading systems and so on.

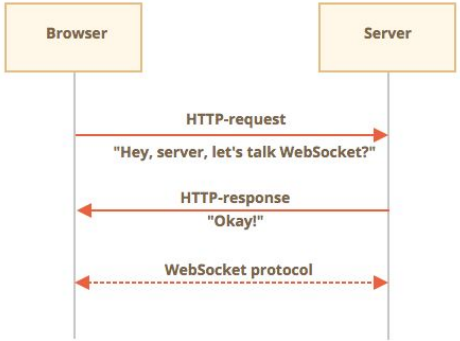




From:
https://coconauts.net/blog/2017/11/20/websocket-vs-rest/
https://javascript.info/websocket

WebSocket - Native JS Web API

```
let socket = new WebSocket("wss://javascript.info/article/websocket/demo/hello");

socket.onopen = function(e) {
  alert("[open] Connection established");
  alert("Sending to server");
  socket.send("My name is John");
};

socket.onmessage = function(event) {
  alert(`[message] Data received from server: ${event.data}`);
};

socket.onclose = function(event) {
  if (event.wasClean) {
    alert(`[close] Connection closed cleanly, code=${event.code} reason=${event.reason}`);
  } else {
    // e.g. server process killed or network down
    // event.code is usually 1006 in this case
    alert('[close] Connection died');
  }
};

socket.onerror = function(error) {
  alert(`[error] ${error.message}`);
};
```



For demo purposes, there's a small server `server.js` written in Node.js, for the example above, running. It responds with "Hello from server, John", then waits 5 seconds and closes the connection.

So you'll see events `open` → `message` → `close`.

WebSocket - Easy Server-Client with Socket.IO

Server

```
const http = require("http");
const socketIo = require("socket.io");

const server = http.createServer(app);

const io = socketIo(server);

const getApiAndEmit = (socket) => {
  const response = Date.now();
  // Emitting a new message. Will be consumed by the client
  socket.emit("FromAPI", response);
};

let interval;

io.on("connection", (socket) => {
  console.log("New client connected");
  if (interval) {
    clearInterval(interval);
  }
  interval = setInterval(() => getApiAndEmit(socket), 1000);
  socket.on("disconnect", () => {
    console.log("Client disconnected");
    clearInterval(interval);
  });
});

server.listen(PORT, () => {
  console.log(`Example app listening on port ${PORT}!`);
});
```

Client

```
   <div id="time"></div>
    <script
src="https://cdn.jsdelivr.net/npm/socket.io-client@2/dist/socket.io.js"></script>
    <script>
      const socket = io("http://localhost:8000");
      socket.on("FromAPI", (data) => {
        console.log("data", data);
        const time = document.getElementById("time");
        const dateTimeFormat = new Intl.DateTimeFormat("en-u-ca-hebrew", {
          year: "numeric",
          month: "short",
          day: "2-digit",
          hour: "numeric",
          minute: "numeric",
          second: "numeric",
        });
        time.innerHTML = dateTimeFormat.format(data);
      });
    </script>
```

WebSocket - Easy Server-Client with Socket.IO (React)

Server

```
const http = require("http");
const socketIo = require("socket.io");

const server = http.createServer(app);

const io = socketIo(server);

const getApiAndEmit = (socket) => {
  const response = Date.now();
  // Emitting a new message. Will be consumed by the client
  socket.emit("FromAPI", response);
};

let interval;

io.on("connection", (socket) => {
  console.log("New client connected");
  if (interval) {
    clearInterval(interval);
  }
  interval = setInterval(() => getApiAndEmit(socket), 1000);
  socket.on("disconnect", () => {
    console.log("Client disconnected");
    clearInterval(interval);
  });
});

server.listen(PORT, () => {
  console.log(`Example app listening on port ${PORT}!`);
});
```

Client

```
import React, { useEffect, useState } from "react";
import "./App.css";

import socketIOClient from "socket.io-client";

const App = () => {
  useEffect(() => {
    const socket = socketIOClient("http://localhost:8000");
    socket.on("FromAPI", (data) => {
      const dateTimeFormat = new Intl.DateTimeFormat("en-u-ca-hebrew", {
        year: "numeric",
        month: "short",
        day: "2-digit",
        hour: "numeric",
        minute: "numeric",
        second: "numeric",
      });
      setTime(dateTimeFormat.format(data));
    });
  }, []);

  const [time, setTime] = useState("");

  return (
    <div style={{ padding: "30px" }}>
      <div>{time}</div>
    </div>
  );
};

export default App;
```

WebSocket - Listen to new Todo (React)

Server

```
const http = require("http");
const socketIo = require("socket.io");
const server = http.createServer(app);
const io = socketIo(server);

io.on("connection", (socket) => {
  console.log("New client connected");
  socket.on("disconnect", () => {
    console.log("Client disconnected");
    clearInterval(interval);
  });
});

app.post("/todos", (req, res) => {
  fs.readFile(JSON_FILE, (err, data) => {
    const todos = JSON.parse(data);
    const title = req.body.title;
    const todoToAdd = {
      id: todos.length + 1,
      title: title,
    };
    todos.push(todoToAdd);
    fs.writeFile(JSON_FILE, JSON.stringify(todos), (err) => {
      // console.log(err);
      res.send("YOU SUCCEED!!!");
    });
    io.emit("FromAPI", todoToAdd);
  });
});

server.listen(PORT, () => {
  console.log(`Example app listening on port ${PORT}!`);
});
```

```
import React, { useEffect, useState } from "react";
import "./App.css";

import socketIOClient from "socket.io-client";

const App = () => {
  useEffect(() => {
    const socket = socketIOClient("http://localhost:8000");
    socket.on("FromAPI", (data) => {
      setTodo(data);
      setTimeout(() => setTodo({}), 3000);
    });
  }, []);

  const [todo, setTodo] = useState({});

  return (
    <div style={{ padding: "30px" }}>
      {todo && todo.title && <div>NEW TODO ARRIVED! {todo.title}</div>}
    </div>
  );
};

export default App;
```

```
npm install socket.io-client
```

Todo list app server side - Exercise

Prepare a todo-list application server side:

1. Create a new express.js server includes a get route to response with a todo list array.
   Each item in this array will contain: id (number), title (string) and a completed (boolean) fields
2. Create all needed routes like described in the presentation.
3. Add a route to delete all todos: **DELETE /todos**
4. Add a filter feature to get only completed todos using **GET /todos?completed=true**
   Tip: use req.query to get the query param.

More Info

1. https://javascript.info
2. https://eloquentjavascript.net