

 _Home_

  RAD

 <https://github.com/TheMIU/mongo-crud>



Node.js

- JavaScript use කරලා server-side development කරගන්න දෙන JavaScript runtime environment එකක්.
- එකම language එකක් use කරලා web applications වල server-side components (APIs & server logic) build කරන්න ඉඩසලසයි.
(මේකේදී client side, server side 2කම language එක JS)

What is Node.js

- Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to execute JavaScript code server-side.*
- It is built on the V8 JavaScript runtime engine, which is the same engine that powers the Chrome web browser's JavaScript execution.
- Node.js is commonly used for building server-side applications, APIs, and real-time applications like chat applications or online gaming platforms.
- It has gained widespread adoption in the web development community and is considered a versatile and powerful technology for building scalable and efficient server-side applications using JavaScript.

Key features and Characteristics of Node.js

- JavaScript Everywhere:
 - Node.js enables developers to use JavaScript for both client-side and server-side development.
 - This allows for the sharing of code and skills between the front end and back end of web applications.
- Asynchronous and Event-Driven:
 - One of the main strengths of Node.js is its non-blocking, asynchronous I/O model.
 - It uses an event-driven architecture that makes it well-suited for handling concurrent operations, such as processing multiple requests simultaneously.
- V8 JavaScript Engine:
 - Node.js leverages the V8 JavaScript engine developed by Google, which is known for its high performance in executing JavaScript code.
 - This engine compiles JavaScript code into machine code for faster execution.
- NPM (Node Package Manager):
 - Node.js comes with npm, a powerful package manager that simplifies the process of installing, managing, and sharing third-party libraries and tools.
 - The Node Package Manager is a crucial component of the Node.js ecosystem.

- **Single-Threaded, Event Loop:**
 - While Node.js is single-threaded, it uses an event loop to handle multiple concurrent connections.
 - This architecture allows it to efficiently handle a large number of connections without the need for threads.
 - **Community and Ecosystem:**
 - Node.js has a vibrant and active community, and it has a vast ecosystem of libraries and frameworks that simplify various aspects of web development.
 - Popular frameworks like Express.js are commonly used for building web applications with Node.js.
-
- **Scalability:**
 - Due to its non-blocking, event-driven nature, Node.js is well-suited for building scalable applications that can handle a large number of simultaneous connections.
 - It is commonly used in scenarios where high concurrency is essential, such as real-time applications.

Express.js

- Node.js වලට තියෙන web application framework එකිනි.
- web applications, API build කරන එක පහසු කරයි.
- routing, middleware, server-side logic handle කරන tools සහ features සපයයි.

What is Express.js?

- Express.js is a web application framework for Node.js.
 - It simplifies the process of building web applications and APIs by providing a set of features and tools for handling routes, middleware, and HTTP requests and responses.
 - Express.js is known for its minimal and flexible design, allowing developers to structure their applications in a way that makes sense for their specific use cases.
 - It is widely used for building both small and large-scale web applications.
 - Key features of Express.js include routing, middleware support, templating engines, and an ecosystem of extensions and plugins.
-

Client - Server side 2☺

එක project එකක තියන එකේ වාසි

Single Codebase:

Having both the client and server sides in the same project allows you to maintain a single codebase for your entire application. (Easy Maintenance)

Seamless Integration:

React is designed to work seamlessly with server-side logic. By integrating both sides in the same project, you can easily share code, configurations, and dependencies between the frontend and backend components. This can lead to a more cohesive and integrated development experience.

Code Organization:

Keeping client and server code together makes it easier to organize and manage the overall structure of your application using a modular approach. (Making it easier to navigate and understand the codebase)

Development Efficiency:

During development, having the client and server sides together allows for faster iteration. Changes made on one side can be quickly tested and validated against the other side. This can be especially beneficial when implementing features that require coordination between the frontend and backend.

Deployment and Hosting:

Deploying and hosting a unified project is often more straightforward than managing separate deployments for the client and server. Many hosting platforms and tools are designed to handle full-stack applications, simplifying the deployment process.

Reduced Latency:

By having both the client and server sides in the same project, you can minimize latency between them. This is important for real-time applications where quick communication between the frontend and backend is crucial.

Server-Side Rendering (SSR) and Isomorphic Applications:

React supports server-side rendering, which can be beneficial for performance and SEO. By having both client and server code in the same project, you can implement server-side rendering more seamlessly, creating isomorphic applications that share code between the client and server.

Middleware

"middleware" - code that runs before the final route call back. They are in the middle of the beginning of the route and the callback function.

What is Middleware in Express?

- Middleware plays a crucial role in building web applications by allowing you to perform various tasks during the different stages of a request lifecycle
- In the context of web frameworks like Express.js, middleware is used to add functionality to the application. There are several types of middleware:

```
// middleware
app.use(logger('dev'));
// convert req body to JSON format
app.use(express.json());
// convert form-data to JSON format
app.use(express.urlencoded({ extended: false }));
// req.cookies
app.use(cookieParser());
// static path define
app.use(express.static(path.join(__dirname, 'public')));

// Enable CORS for all routes
app.use(cors());
```

Sync vs Async



Java EE

Synchronous ← එකක් run වලො ඉවර වනෙකල අනිත් line එක blocking (one by one)

Asynchronous ← Time consuming, non blocking task එකක්. (no order)

Client > Home.tsx

```
fetchData = async () => {
  try {
    const response = await fetch('./product-data.json'); // pause execution
    const jsonData = await response.json(); // pause execution
  }
}
```

await ← execution එක pause කරයි. ඒ line එකේ වනෙ වැඩේ ඉවර වුනාම ඊළඟ line එකට යනවා.

await keyword එක use කරනවනම් ඒක Asynchronous function එකක් වන්නේ ඕන.

Sync vs Async

- The terms "**async**" and "**sync**" refer to programming paradigms related to the handling of **asynchronous** and **synchronous** operations in software development.
- These concepts are particularly relevant in languages and frameworks that deal with I/O operations, such as network requests, file system operations, or database queries.

Synchronous (Sync) Example

javascript

Copy code

```
// Synchronous code
function syncOperation() {
  console.log('Start');
  console.log('Middle');
  console.log('End');
}

syncOperation();
```

Asynchronous (Async) Example

```
// Asynchronous code using setTimeout
function asyncOperation() {
  console.log('Start');

  setTimeout(function() {
    console.log('Middle (Async)');
  }, 1000);

  console.log('End');
}

asyncOperation();
```

Features of Synchronous (Sync)

- **Blocking Behavior:**
 - In synchronous programming, tasks are executed sequentially, one after the other.
 - When a function or operation is called, the program waits for it to complete before moving on to the next task.
 - If a task takes time to complete (e.g., reading a file, making a network request), the entire program is blocked until that task finishes.
- **Straightforward Execution Flow:**
 - Synchronous code has a more straightforward and predictable execution flow.
 - Code is executed line by line, making it easier to reason about the program's state.
- **Potential for Slower Performance:**
 - Synchronous code can result in slower performance, especially in scenarios where I/O operations or other time-consuming tasks are involved.
 - The program might spend a significant amount of time waiting.

- **Straightforward Execution Flow:**
 - Synchronous code has a more straightforward and predictable execution flow.
 - Code is executed line by line, making it easier to reason about the program's state.
- **Potential for Slower Performance:**
 - Synchronous code can result in slower performance, especially in scenarios where I/O operations or other time-consuming tasks are involved.
 - The program might spend a significant amount of time waiting.

Features of Asynchronous (Async)

- **Non-Blocking Behavior:**
 - In asynchronous programming, tasks are initiated but do not block the execution of the program. Instead of waiting for a task to complete, the program can continue with other tasks.
 - Callback functions, promises, or async/await syntax are common mechanisms for handling asynchronous operations.
- **Efficient Use of Resources:**
 - Asynchronous code allows for more efficient use of resources because the program doesn't have to wait for each task to complete.
 - It can initiate multiple tasks concurrently.

Install Express

```
npx express-generator
```

```
npm install
```

nodemon

Server එකේ change එකක් වුනම, auto restart කරයි. නිතරම npm run start ගන්න ඔබ නෑ.

```
npm install nodemon
```

- Install nodemon globally on your machine
npm install -g nodemon
- Install nodemon on your project as dev-dependency
npm install nodemon -D

Add new script to package.json

```
"dev" : "nodemon ./bin/www"
```

```
npm run dev
```

Axios

<https://www.npmjs.com/package/axios>

```
npm install axios (frontend)
```

Axios is a promise-based HTTP library that lets developers make requests to either their own or a third-party server to fetch data. It offers different ways of making requests such as GET , POST , PUT/PATCH , and DELETE .

CORS

<https://www.npmjs.com/package/cors>

```
npm install cors
```

Web browser එකේ නිසල security feature එකක්.

Backend එකට වෙත unauthorized access වලේවයි. (Browser එකේ block කරන්නේ, postman වල cros අවලේ නෑ)

- CORS stands for Cross-Origin Resource Sharing.
- It is a security feature implemented by web browsers to control how web pages in one domain can request and interact with resources from another domain.
- CORS is designed to prevent unauthorized access to resources on a different origin (domain) than the one that served the original web page.

Same-Origin Policy (SOP)

- **By default, web browsers enforce the Same-Origin Policy**, which means that **web pages can only make requests to the same origin (protocol, domain, and port)** from which the page originated.
- Security Measure: **SOP is a security measure to prevent malicious websites from making unauthorized requests** on behalf of the user to another domain where the user is authenticated.

Need for using Cross-Origin Requests


- **Modern Web Applications:** With the rise of web applications that **consume APIs and services from different domains**, the need for **making cross-origin requests became essential**.
- **Cross-Origin Requests:** A cross-origin request occurs when a **web page hosted on one domain** which **makes a request to a different domain (server)**, for example, when a **front-end application** (JavaScript in the browser) **requests data from a server on a different domain**.

CORS Implementation

- Browser Enforcement:
 - CORS is enforced by web browsers.
 - When a cross-origin request is made, the browser sends an HTTP request with an Origin header to the target server.
 - Server Response:
 - The server can include specific HTTP headers in its response to indicate whether the cross-origin request is permitted.
-
- The primary CORS headers include:
 - **Access-Control-Allow-Origin:** Specifies which origins are permitted to access the resource.
 - **Access-Control-Allow-Methods:** Specifies the HTTP methods (e.g., GET, POST) that are allowed when accessing the resource.
 - **Access-Control-Allow-Headers:** Specifies which HTTP headers can be used when making the actual request.

Example CORS Headers:

plaintext

 Copy code

```
Access-Control-Allow-Origin: https://allowed-domain.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: Content-Type, Authorization
```

CORS Implementation

javascript

 Copy code

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

const app = express();

// Enable CORS for all routes
app.use(cors());

// Your routes and other middleware go here

const port = 3000;
app.listen(port, () => {
  console.log(`Server is listening on port ${port}`);
});
```

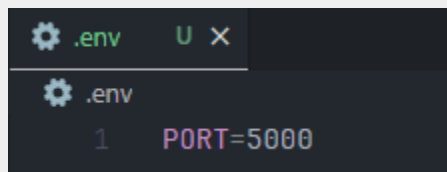
.env

```
npm install dotenv
```

server>www

```
var dotenv = require('dotenv');
dotenv.config();
```

server>.env



Server will run on port 5000



MongoDB

- NoSQL database එකකි
- JSON වලට සමාන format එකකට, data flexible විදියට store කරයි.
- සාමාන්යයෙන් web applications වල data handle කරන්න use කරයි.

MongoDB Setup:

- If using the MongoDB driver (low-level):

```
npm install mongodb
```

- If using Mongoose (high-level Object Data Modeling library):

```
npm install mongoose
```

Create DB

And allow for any ip (not recommended)

<https://youtu.be/7jKjBKeJdrs>

Browse collections

<https://youtu.be/yiks0VooAww>

Delete DB

<https://youtu.be/2LadFYG5DGc>

CRUD Operations

<https://github.com/TheMIU/mongo-crud>