Dependencies to install :-

npm i express cors mongoose jsonwebtoken bcryptjs cookie-parser dotenv morgan helmet

The **Morgan** package in Node.js is a popular middleware used for logging HTTP requests in web applications, particularly those built with Express.js.

In production, you often want to log requests to a file rather than just to the console. This allows you to store logs over time and analyze them later.

Combined format:-

const fs = require('fs'); const path = require('path'); const accessLogStream = fs.createWriteStream(path.join(\_\_dirname, 'access.log'), { flags: 'a' }); app.use(morgan('combined', { stream: accessLogStream }));

common format:-

app.use(morgan('common'));

custorm format:-

app.use(morgan(':method :url :status :response-time ms - :res[content-length]'));

This will store your logs in a file called access.log in the server's directory. Using the combined format ensures the logs are detailed and can be parsed for further analysis.

**Helmet** is a security middleware for Express.js applications in Node.js that helps secure your app by setting various HTTP headers. It does not make your app immune to attacks, but it adds important layers of security to prevent some common vulnerabilities.

**What does Helmet do?**

Helmet helps protect your app from several web security issues by setting HTTP headers. Some of these include:

1. **Content Security Policy (CSP)**: Helps prevent cross-site scripting (XSS) attacks by specifying which domains the browser should allow content to be loaded from.
2. **Strict-Transport-Security (HSTS)**: Ensures that browsers only communicate with your site over HTTPS.
3. **X-Content-Type-Options**: Prevents browsers from interpreting files as a different MIME type, which could lead to XSS attacks.
4. **X-Frame-Options**: Protects your site from clickjacking attacks by preventing it from being embedded in an iframe.
5. **X-XSS-Protection**: Enables cross-site scripting filter built into most browsers to stop reflected XSS attacks.
6. **Cache-Control**: Helps control caching policies to prevent sensitive data from being cached.
7. **Referrer-Policy**: Controls how much referrer information (information about the address of the previous web page) is sent along with requests.

**How to use Helmet in Node.js**

To get started with Helmet, first, install it via npm:

bash

Copy

npm install helmet

Then, in your Express app, you can use it like this:

javascript

Copy

const express = require('express');

const helmet = require('helmet');

const app = express();

// Use helmet middleware for security headers

app.use(helmet());

// Example route

app.get('/', (req, res) => {

res.send('Hello, secure world!');

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

**Helmet Features (Detailed)**

1. **helmet.contentSecurityPolicy()**: Enables the Content Security Policy (CSP) header, which helps prevent XSS attacks by controlling where resources (scripts, images, etc.) can be loaded from. You can customize it as needed:

javascript

Copy

app.use(helmet.contentSecurityPolicy({

directives: {

defaultSrc: ["'self'"],

scriptSrc: ["'self'", 'trusted.com'],

styleSrc: ["'self'", 'trusted.com'],

}

}));

1. **helmet.frameguard()**: Prevents your site from being embedded in an iframe to avoid clickjacking. The default setting is DENY:

javascript

Copy

app.use(helmet.frameguard({ action: 'sameorigin' }));

1. **helmet.xssFilter()**: Adds the X-XSS-Protection header to block some types of reflected XSS attacks:

javascript

Copy

app.use(helmet.xssFilter());

1. **helmet.hsts()**: Forces the use of HTTPS by setting the HTTP Strict Transport Security (HSTS) header. It's recommended to use it only if your site supports HTTPS:

javascript

Copy

app.use(helmet.hsts({ maxAge: 31536000, includeSubDomains: true, preload: true }));

1. **helmet.noCache()**: Prevents caching of sensitive data:

javascript

Copy

app.use(helmet.noCache());

1. **helmet.referrerPolicy()**: Controls the Referrer-Policy header to restrict how much referrer information is sent when navigating between websites:

javascript

Copy

app.use(helmet.referrerPolicy({ policy: 'no-referrer' }));

1. **helmet.dnsPrefetchControl()**: Controls browser DNS prefetching. It can be set to either allow or disallow this feature:

javascript

Copy

app.use(helmet.dnsPrefetchControl({ allow: false }));

**When to Use Helmet:**

* **Use Helmet in all production applications**. It provides an essential layer of security by configuring the security-related HTTP headers.
* For most use cases, you can simply use app.use(helmet()), as it applies a set of sensible defaults for common security headers.
* You can further fine-tune security by customizing the middleware (like helmet.contentSecurityPolicy() or helmet.hsts()), especially for specific needs of your app.

**Conclusion:**

Helmet is an easy-to-use security middleware for Express.js applications, providing an additional layer of protection through HTTP headers. It's highly recommended for any Node.js app, especially for production environments, as it helps to protect your app from common vulnerabilities such as XSS, clickjacking, and content sniffing attacks.

**Why Use crossOriginResourcePolicy: false?**

* **Disabling the CORP header** might be useful if:
  + You want to **allow resources to be shared freely** across all origins, and do not want the server to enforce restrictions based on cross-origin access.
  + You are **developing** a public API or resource that is meant to be accessed cross-origin by any site, and you do not want to restrict this.
  + For compatibility reasons, you might want to **disable CORP** if you are working with older browsers or systems that don’t fully support CORP.

*//credentials: true:*

*//Purpose: This option indicates that the server allows credentials (cookies, HTTP authentication, client-side SSL certificates, etc.) to be included in cross-origin requests.*

app.use(cors({

    credentials:true,

    origin: process.env.FRONTEND\_URL

}));

orderHistory: [

{

type: mongoose.Schema.ObjectId,

ref: 'order'

}

]

is creating an **array** called orderHistory that will store references to **other documents** in the database. Here's a simple explanation:

**What It Means:**

1. **type: mongoose.Schema.ObjectId**: This indicates that the orderHistory array will store **ObjectId values**. In MongoDB, an ObjectId is a unique identifier for a document in a collection.
2. **ref: 'order'**: The ref is used to tell **Mongoose** that these ObjectId values are **references** to documents in another collection, specifically the 'order' collection. So, each ObjectId in the orderHistory array will point to an order document in the database.