TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS (502070)

LAB4 – NODEJS

TDTU

How to read environment variables from Node.js (nodejs.dev)

The process core module of Node.js provides the env property which hosts all the environment variables that were set at the moment the process was started.

```
USER_ID=239482 USER_KEY=foobar node app.js
```

```
# .env file
USER_ID="239482"
USER_KEY="foobar"
NODE_ENV="development"
```

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

process.env.USER_ID // "239482"
process.env.USER_KEY // "foobar"
process.env.NODE_ENV // "development"
```

[ExpressJS Tutorial (tutorialspoint.com)]

[Express - Node.js web application framework (expressjs.com)]

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is an open source framework developed and maintained by the Node.js foundation.

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

app.get('/', function(req, res){
   res.send("Hello world!");
});

app.listen(3000);
```

Template Engines (expressjs.com)

A ***template engine*** enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page.

```
app.set('view engine', 'pug')
```

```
html
  head
    title= title
  body
    h1= message
```

```
app.get('/', (req, res) => {
  res.render('index', { title: 'Hey', message: 'Hello there!' })
})
```

EJS -- Embedded JavaScript templates

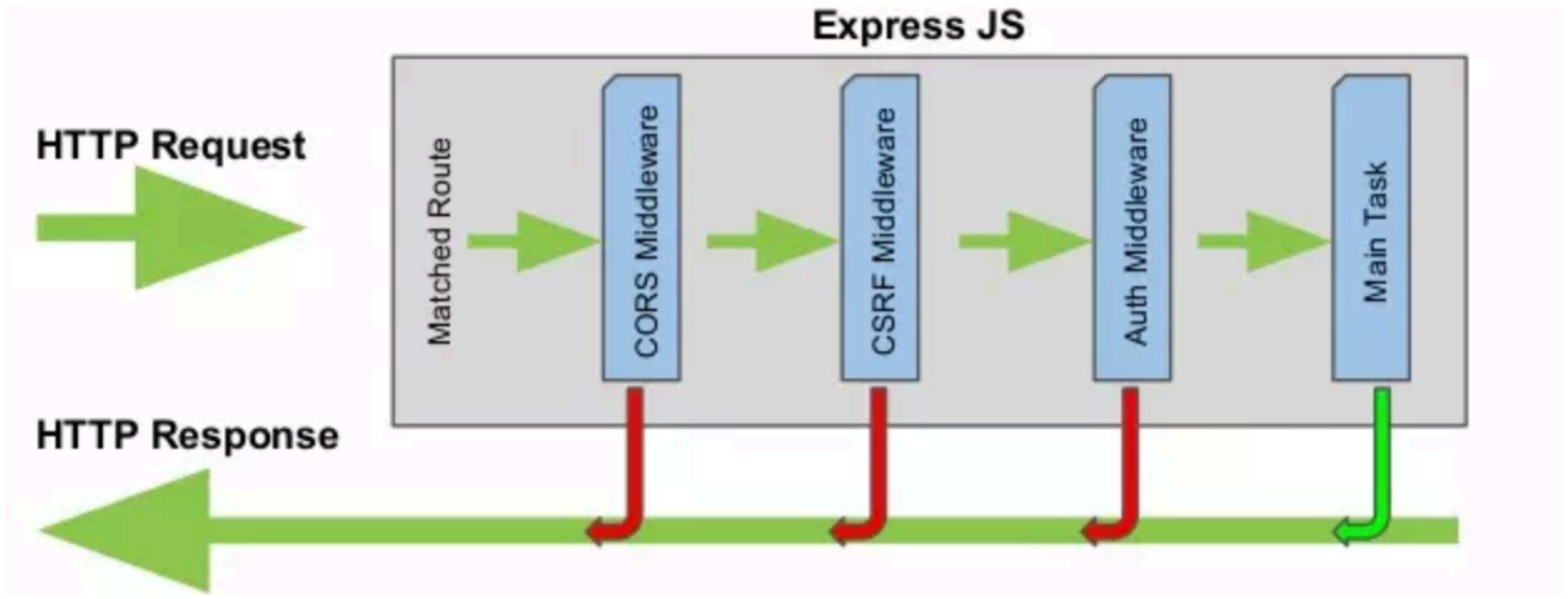
```
app.set('view engine', 'ejs')
```

```
app.get('/', (req, res) => {
    res.render('pages/index', {
        user: user
    })
})
```

```
<h1>Hi, I am <%= user.firstName  %></h1>
```

- Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of middleware function calls.

- ***Middleware*** functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named **next.**



Using Express middleware (expressjs.com)

Application-level

Bind application-level middleware to an instance of the app object by using the app.use() and app.METHOD() functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.

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

app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

[Using Express middleware (expressjs.com)](https://expressjs.com)

Router-level

Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

```
const express = require('express')
const app = express()
const router = express.Router()

// a middleware function with no mount path. This code is executed for every request to the router
router.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

Using Express middleware (expressjs.com)

Error-handling middleware

Define error-handling middleware functions in the same way as other middleware functions, except with four arguments instead of three, specifically with the signature (err, req, res, next)):

```
app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

Using Express middleware (expressjs.com)

Built-in middleware

- **express.static** serves static assets such as HTML files, images, and so on.
- **express.json** parses incoming requests with JSON payloads. NOTE: Available with Express 4.16.0+
- **express.urlencoded** parses incoming requests with URL-encoded payloads. NOTE: Available with Express 4.16.0+

[Using Express middleware (expressjs.com)](https://expressjs.com)

Third-party middleware

Use third-party middleware to add functionality to Express apps.
Install the Node.js module for the required functionality, then load it in your app at the application level or at the router level.

```
const express = require('express')
const app = express()
const cookieParser = require('cookie-parser')

// load the cookie-parsing middleware
app.use(cookieParser())
```

Express session middleware (expressjs.com)

HTTP is stateless; in order to associate a request to any other request, you need a way to store user data between HTTP requests. Cookies and URL parameters are both suitable ways to transport data between the client and the server. But they are both readable and on the client side. Sessions solve exactly this problem.

```
app.use(cookieParser());
app.use(session({secret: "Shh, its a secret!"}));

app.get('/', function(req, res){
   if(req.session.page_views){
      req.session.page_views++;
      res.send("You visited this page " + req.session.page_views + " times");
   } else {
      req.session.page_views = 1;
      res.send("Welcome to this page for the first time!");
   }
});
```

Express multer middleware (expressjs.com)

Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.

```
app.post('/profile', upload.single('avatar'), function (req, res, next) {
  // req.file is the `avatar` file
  // req.body will hold the text fields, if there were any
})

app.post('/photos/upload', upload.array('photos', 12), function (req, res,
next) {
  // req.files is array of `photos` files
  // req.body will contain the text fields, if there were any
})
```

ExpressJS - Form data (tutorialspoint.com)

To get started with forms, we will first install the *body-parser*(for parsing JSON and url-encoded data) and multer(for parsing multipart/form data) middleware.

```
app.post('/', function(req, res){
   console.log(req.body);
   res.send("recieved your request!");
});
```

```
 form(action = "/", method = "POST")
    div
       label(for = "say") Say:
       input(name = "say" value = "Hi")
    br
    div
       label(for = "to") To:
       input(name = "to" value = "Express forms")
    br
    button(type = "submit") Send my greetings
```

Thank you