

React Advance

Bài 4: Xây dựng middleware và API



Nội dung

1. Cài đặt Middleware
2. Sử dụng middleware
3. Xử lý lỗi - Error handling
4. Debugging
5. Kiến trúc MVC trong Express
6. Xây dựng API



1. Cài đặt Middleware



Cài đặt Middleware

- Middleware là hàm cho phép truy cập tới request object (**req**) và response (**res**) và hàm next trong request-response circle của ứng dụng.
- Next function là một hàm trong Express router để gọi và thực thi các hàm middleware kế tiếp.
- Middleware function có công dụng:
 - Thực thi một đoạn code.
 - Thay đổi thông tin của các Request và Response objects.
 - Kết thúc một vòng request-response cycle.
 - Gọi tiếp một hàm middleware trong stack.



Cài đặt middleware

JS index.js > ...

```
1  const express = require('express');
2  const app = express();
3
4
5  app.get('/', function (req, res, next) {
6    next();
7  })
8
9
10 app.listen(3000);
```

The diagram illustrates the execution flow of an Express.js application. It starts with an **HTTP request** (blue arrow) entering the **app.get('/')** middleware function (green box). Inside the function, the **next()** callback is executed (purple arrow), which then triggers the **Callback argument** (purple arrow) to the next middleware function in the stack. This leads to the **HTTP response** (red arrow) being sent back to the client. The code snippets show the setup of this stack with the `express()` constructor and the `listen` method.

- HTTP Method
- Path (route)
- Middleware function
- Callback argument tới middleware function
- HTTP response
- HTTP request



Cài đặt middleware

- Ví dụ ở demo “hello world” ở bài đầu tiên của express.
 - Khi có một request gửi đến chỉ đơn giản là phản hồi lại chuỗi “Hello world”

```
JS index.js > ...
1  const express = require('express');
2  const app = express();
3
4  app.get('/', (req, res) => {
5    |  res.send('Hello world!!!');
6  })
7
8  app.listen(3000);
```

- Tuy nhiên giờ muốn log lại thông tin như requestTime thì sẽ phải sử dụng middleware để làm chuyện này

```
const myLogger = (req, res, next) => {
  |  console.log('Logged');
  |  next();
}
```



Cài đặt Middleware – sử dụng đối tượng *app.use()*

```
JS index.js > ...
1  const express = require('express');
2  const app = express();
3
4  const myLogger = (req, res, next) => {
5      console.log('Logged');
6      next();
7  }
8
9  app.use(myLogger);
10
11 app.get('/', (req, res) => {
12     res.send('Hello world!!!');
13 })
14
15 app.listen(3000);
16
```



Middleware function - requestTime

- Cài đặt hiển thị thời gian mà request gửi đến server

```
JS index.js > ...
1  const express = require('express');
2  const app = express();
3
4  const requestTime = (req, res, next) => {
5    req.requestTime = Date.now();
6    next();
7  }
8
9  app.use(requestTime);
10
11 app.get('/', (req, res) => {
12   var responseText = `Hello World!<br />Request at: ${req.requestTime}`;
13   res.writeHead(200, {'Content-Type': 'text/html'});
14   res.end(responseText);
15 })
16
17 app.listen(3000);
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET' selected, a URL field containing 'localhost:3000', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Cookies' tab is also visible. Under the 'Body' tab, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'HTML'. The status bar at the bottom right indicates a 'Status: 200 OK' response with a time of '17 ms' and a size of '222 B'. The main content area displays the response body: '1 Hello World!
Request at: 1627502549147'.



Middleware function - validateCookies

```
> npm install cookie-parser --save
```

```
up to date, audited 53 packages in 737ms
```

```
found 0 vulnerabilities
```

```
js cookieValidator.js > ...
1 const cookieValidator = async (cookies) => {
2   try {
3     if(cookies.testCookie != null)
4       console.log(`⇒ Cookies.testCookies = ${cookies.testCookie}`);
5     else
6       throw new Error('Invalid cookies');
7   } catch (e) {
8     throw e;
9   }
10 }
11 module.exports = cookieValidator;
```

```
js index.js > ...
1 const express = require('express');
2 → const cookieParser = require('cookie-parser');
3 const cookieValidator = require('./cookieValidator');
4
5 const app = express();
6
7 const validateCookies = async (req, res, next) => {
8   try {
9     await cookieValidator(req.cookies);
10    next();
11  } catch (e){
12    next(e);
13  }
14 }
15
16 app.use(cookieParser());
17 app.use(validateCookies);
18
19 // error handler
20 app.use(function(error, req, res, next){
21   res.status(400).end(error.message);
22 })
23
24 app.get('/', (req, res) => {
25   res.writeHead(200, {'Content-Type': 'text/html'});
26   res.end(`Hello World! Cookies.testCookie = ${req.cookies.testCookie}`);
27 })
28
29 app.listen(3000);
```



Middleware function – validateCookies – Không có cookies

The screenshot shows a Postman request configuration for a GET request to localhost:3000. The 'Headers' tab is selected, showing six headers. The 'Body' tab is selected, displaying the response content: "1 Invalid cookies". The status bar at the bottom indicates a 400 Bad Request status with a time of 19 ms and a size of 170 B.

GET localhost:3000

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (5) Test Results

Status: 400 Bad Request Time: 19 ms Size: 170 B Save Response

Pretty Raw Preview Visualize Text

1 Invalid cookies



Middleware function – validateCookies – set biến cookies

The screenshot illustrates the process of setting a cookie using a middleware function like `validateCookies`.

Postman Request:

- Method: GET
- URL: localhost:3000
- Params tab is selected.
- Query Params table:

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Cookies tab is also visible.

Interceptor - Manage Cookies:

- Domain: localhost
- Cookie list:
 - Cookie_4 (testCookie=123456; Path=/; Expires=Thu, 28 Jul 2022 21:23:54 GMT)
- Add button is highlighted with a green arrow.
- Save button is highlighted with a blue arrow.

Annotations:

- An orange arrow points from the "Value" column in the Postman Query Params table to the "Value" field in the Interceptor's Add cookie dialog.
- A yellow bracket highlights the cookie value in the Interceptor's cookie list.



Middleware function – validateCookies – với cookies

The screenshot shows the Postman application interface. At the top, it displays a GET request to 'localhost:3000'. Below the request, there are tabs for 'Params' (which is selected), 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. To the right of these tabs is a 'Cookies' tab. Under the 'Params' tab, there is a table with one row containing 'Key' (Value: 'Key') and 'Value' (Value: 'Value'). In the 'Body' section, there are tabs for 'Body', 'Cookies (1)', 'Headers (6)', and 'Test Results'. The 'Body' tab is selected. It contains buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'HTML' (which is currently selected). Below these buttons is a code editor window showing the response body: '1 Hello World! Cookies.testCookie = 123456'. At the top right of the interface, it shows the status: 'Status: 200 OK', 'Time: 24 ms', 'Size: 219 B', and a 'Save Response' button.



Configurable middleware

```
JS middleware.js > ...
1  module.exports = (options) => {
2    return (req, res, next) => {
3      //? Code hàm xử lý middleware ở đây
4      next();
5    }
6  }
```



```
const mid = require('./middleware');

app.use(mid({option_1: '1', option_2: '2'}));
```



2. Sử dụng Middleware



Sử dụng Middleware

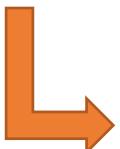
- Ứng dụng Express có thể sử dụng các loại middleware sau
 - Application-level middleware
 - Router-level middleware
 - Error-handling middleware
 - Build-in middleware
 - Third-party middleware
- Thường dùng application-level và router-level middleware với option mounth path để xây dựng router cho các ứng dụng Express



Application-level middleware

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

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



```
app.use('/user/:id', (req, res, next) => {
  console.log('Request Type:', req.method);
  next();
})
```



```
app.get('/user/:id', (req, res, next) => {
  res.send('USER');
})
```



Application-level middleware

```
app.use('/user/:id', (req, res, next) => {
  console.log('Request URL:', req.originalUrl);
  next();
}, (req, res, next) => {
  console.log('Request Type:', req.method);
  next();
})
```



```
app.get('/user/:id', (req, res, next) => {
  console.log('ID:', req.params.id);
  next();
}, (req, res, next) => {
  res.send('User Information');
})

//? handler for the user/:id path, which prints the user ID
app.get('/user/:id', (req, res, next) => {
  res.send(req.params.id);
})
```



Application-level middleware – middleware sub-stack

```
app.get('/user/:id', (req, res, next) => {
  //? if user ID is 0, skip to the next route
  if (req.params.id === '0')
    next('route');
  else
    //? otherwise pass the control to the next middleware function in this stack
    next();
}, (req, res, next) => {
  //? send a regular response
  res.send('Regular');
})  
  
//? handler for the /user/:id path, which sends a special response
app.get('/user/:id', (req, res, next) => {
  res.send('Special');
})
```



Application-level middleware – array for reusability

```
const logOriginalUrl = (req, res, next) => {
  console.log('Request URL:', req.originalUrl);
  next();
}

const logMethod = (req, res, next) => {
  console.log('Request Type:', req.method);
  next();
}

var logStuff = [logOriginalUrl, logMethod];
app.get('/user/:id', (req, res, next) => {
  res.send('User Information');
})
```



Router-level middleware – sử dụng `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();
})

//? a middleware sub-stack shows request info for any type of HTTP request to the /user/:id path
router.use('/user/:id', (req, res, next) => {
  console.log('Request URL:', req.originalUrl);
  next();
}, (req, res, next) => {
  console.log('Request Type:', req.method);
  next();
})

//? a middleware sub-stack that handles GET requests to the /user/:id path
router.get('/user/:id', (req, res, next) => {
  //? if the user ID is 0, skip to the next router
  if (req.params.id === '0')
    next('route');
  else
    //? otherwise pass control to the next middleware function in this stack
    next();
}, (req, res, next) => {
  //? render a regular page
  res.render('Regular');
})

//? handler for the /user/:id path, which renders a special page
router.get('/user/:id', (req, res, next) => {
  console.log(req.params.id);
  res.render('Special');
})

//? mount the router on the app
app.use('/', router);
```



Router-level middleware – sử dụng `express.Router()`

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

const app = express();
const router = express.Router();

//? predicate the router with a check and bail out when needed
router.use((req, res, next) => {
  if(!req.header['x-auth'])
    return next('router');
  next();
})

router.get('/user/:id', (req, res) => {
  res.send('Hello, user!');
})

//? use the router and 401 anything falling through
app.use('/admin', router, (req, res) => {
  res.sendStatus(401);
})
```



Error – handling middleware

```
JS Error.handling.js > ...
1  const express = require('express');
2
3  const app = express();
4  const router = express.Router();
5
6  app.use((err, req, res, next) => {
7    console.error(err.stack);
8    res.status(500).send('Something broken!');
9  })
10
11 app.listen(3000);
12
```



Build-in middleware

- Bắt đầu từ version 4.x của Express
- Express hỗ trợ các hàm build-in middleware sau:
 - express.static – để cấu hình các static assets như HTML file, CSS, image ...
 - express.json – để parse dữ liệu của request với định dạng JSON
 - express.urlencoded – parse dữ liệu của request với URL-encoded



Third-party middleware

- Third-party middleware cho phép gắn thêm các hàm, thư viện hỗ trợ express
- Ví dụ module ***cookie-parser***

```
JS Third.party.js > ...
1  const express = require('express');
2  const cookieParser = require('cookie-parser');
3
4  const app = express();
5
6  //? load the cookie-parsing middleware
7  app.use(cookieParser());
8
9  app.listen(3000);
10
```



3. Xử lý lỗi – Error Handling



Error handling – Catching Errors

```
app.get('/', (req, res) => {
  //? Express will catch this on its own
  throw new Error('BROKEN');
})
```

```
app.get('/user/:id', async (req, res, next) => {
  var user = await getUserById(req.params.id);
  res.send(user);
})
```

```
app.get('/', (req, res, next) => {
  fs.readFile('/file-does-not-exist', (err, data) => {
    if(err) {
      //? Pass errors to Express
      next(err);
    } else {
      res.send(data);
    }
  })
})
```

```
app.get('/', [
  (req, res, next) => {
    fs.writeFile('/inaccessible-path', 'data', next);
  },
  (req, res) => {
    res.send('OK')
  }
])
```



Error handling – Catching Errors

```
app.get('/', (req, res, next) => {
  setTimeout(() => {
    try {
      throw new Error('BROKEN');
    } catch (err) {
      next(err);
    }
  }, 100);
})
```

```
app.get('/', (req, res, next) => {
  Promise.resolve().then(() => {
    throw new Error('BROKEN');
  }).catch(next);           //? Errors will be passed to Express
})
```

```
app.get('/', [
  (req, res, next) => {
    fs.readFile('/maybe-valid-file', 'utf-8', (err, data) => {
      res.local.data = data;
      next(err);
    })
  },
  (req, res) => {
    res.locals.data = res.locals.data.split(',')[1];
    res.send(res.locals.data);
  }
])
```



Default error handler

- Thường cần phải set *res.statusCode* và *res.statusMessage*

```
const errorHandler = (err, req, res, next) => {
  if (res.headersSent) {
    return next(err);
  }
  res.statusCode(500);
  res.send('Error', {error: err});
}
```



Writing Error handlers

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

```
JS Error.hander.js > ...
1  const express = require('express');
2  const app = express();
3
4  app.use((err, req, res, next) => {
5    console.log(err.stack);
6    res.status(500).send('Something broke!');
7  })
8
9  const bodyParser = require('body-parser');
10 const methodOverride = require('method-override');
11
12 app.use(bodyParser.urlencoded({
13   extended: true
14 }))
15 app.use(bodyParser.json())
16 app.use(methodOverride())
17 app.use((err, req, res, next) => {
18   //? code logic here
19 })
```



Writing error handlers

```
const express = require("express");
const bodyParser = require("body-parser");
const methodOverride = require("method-override");

const app = express();

app.use(
  bodyParser.urlencoded({
    extended: true,
  })
);

app.use(bodyParser.json());
app.use(methodOverride());
app.use(logErrors);
app.use(clientErrorHandler);
app.use(errorHandler);
```



```
const logErrors = (err, req, res, next) => {
  console.error(err.stack);
  next(err);
};

const clientErrorHandler = (err, req, res, next) => {
  if (req.xhr) res.status(500).send({ error: "Something failed!" });
  else next(err);
};

const errorHandler = (err, req, res, next) => {
  res.status(500);
  res.render("Error", { error: err });
};
```

```
app.get(
  "/a_route_behind_paywall",
  function checkIfPaidSubscriber(req, res, next) {
    if (!req.user.hadPaid) {
      //? continue handling this request
      next("route");
    } else {
      next();
    }
  },
  function getPaidContent(req, res, next) {
    PaidContent.find((err, doc) => {
      if (err) return next(err);
      res.json(doc);
    });
  }
);
```



4. Debugging



Debugging trong Express

- Chạy câu lệnh sau để vào chế độ debugging trong Express
- Thông start server các modules

```
› DEBUG=express:* node index.js
```

```
express:application set "x-powered-by" to true +0ms
express:application set "etag" to 'weak' +2ms
express:application set "etag fn" to [Function: generateETag] +0ms
express:application set "env" to 'development' +1ms
express:application set "query parser" to 'extended' +0ms
express:application set "query parser fn" to [Function: parseExtendedQueryString] +0ms
express:application set "subdomain offset" to 2 +0ms
express:application set "trust proxy" to false +0ms
express:application set "trust proxy fn" to [Function: trustNone] +0ms
express:application booting in development mode +0ms
express:application set "view" to [Function: View] +0ms
express:application set "views" to '/Lab03/views' +0ms
express:application set "jsonp callback name" to 'callback' +1ms
express:router use '/' query +0ms
express:router:layer new '/' +0ms
express:router use '/' expressInit +1ms
express:router:layer new '/' +0ms
express:router use '/' cookieParser +0ms
express:router:layer new '/' +0ms
express:router use '/' validateCookies +0ms
express:router:layer new '/' +0ms
express:router use '/' <anonymous> +0ms
express:router:layer new '/' +0ms
express:router:route new '/' +0ms
express:router:layer new '/' +0ms
express:router:route get '/' +0ms
express:router:layer new '/' +1ms
```



Debugging trong Express

- Khi có request gửi tới server

```
express:router dispatching GET / +11ms
express:router query : / +1ms
express:router expressInit : / +1ms
express:router cookieParser : / +1ms
express:router validateCookies : / +1ms
=> Cookies.testCookies = 123456
express:router <anonymous> : / +6ms
```

- Nếu sử dụng Application generated của Express thì các câu lệnh sau đổi lại như sau

```
> DEBUG=Lab03:* node ./bin/www
```

```
> DEBUG=http,mail,express:* node index.js
```



Debugging trong Express

- Các option nâng cao – Advanced options

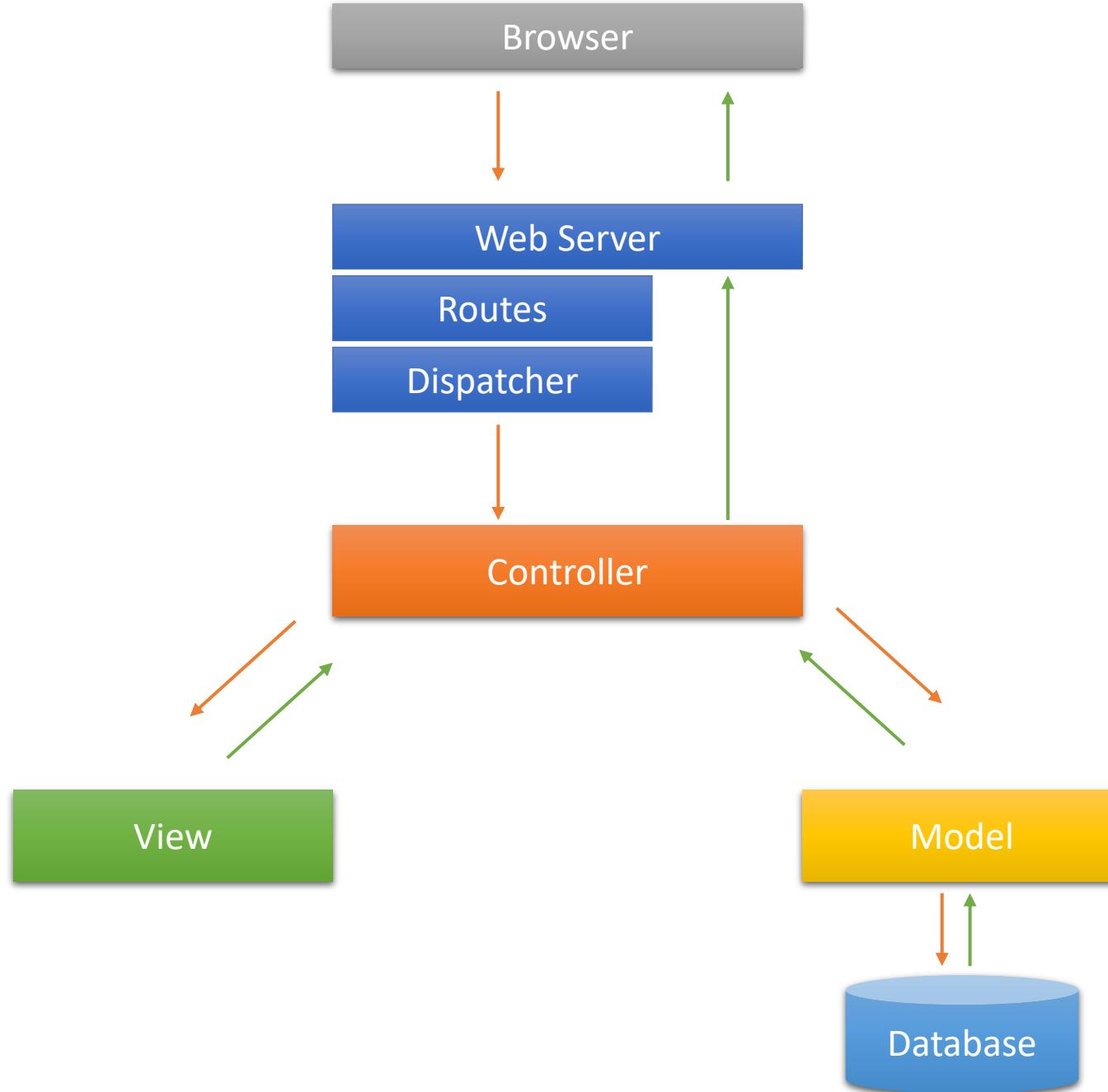
Tên	Mục đích
DEBUG	Enables/disables chế độ debugging
DEBUG_COLORS	Bật/tắt chế độ hiển thị màu trong console debugging
DEBUG_DEPTH	Độ sâu của đối tượng debugging
DEBUG_FD	Mô tả file log thông tin debugging
DEBUG_SHOW_HIDDEN	Hiển thị các thuộc tính ẩn trong các đối tượng khi debugging



5. Kiến trúc MVC trong ExpressJS



Kiến trúc MVC





6. Xây dựng API



Xây dựng API – Express-generator

- Cài đặt express-generator

```
> npm install -g express-generator
```

- Khởi tạo project express application với express-generator không có View

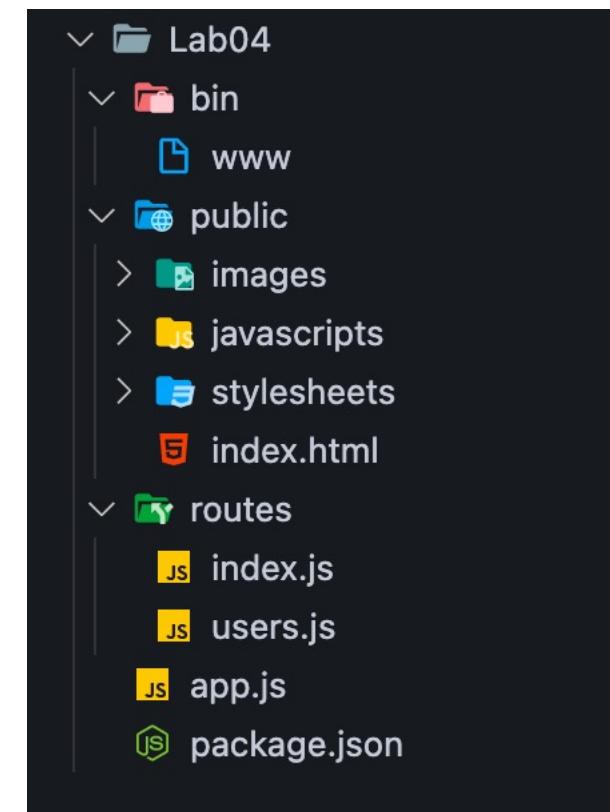
```
> express --no-view Lab04

  create : Lab04/
  create : Lab04/public/
  create : Lab04/public/javascripts/
  create : Lab04/public/images/
  create : Lab04/public/stylesheets/
  create : Lab04/public/stylesheets/style.css
  create : Lab04/routes/
  create : Lab04/routes/index.js
  create : Lab04/routes/users.js
  create : Lab04/public/index.html
  create : Lab04/app.js
  create : Lab04/package.json
  create : Lab04/bin/
  create : Lab04/bin/www

  change directory:
    $ cd Lab04

  install dependencies:
    $ npm install

  run the app:
    $ DEBUG=lab04:* npm start
```





Xây dựng API – Express-generator

- Khởi tạo project express application với express-generator với view engine EJS

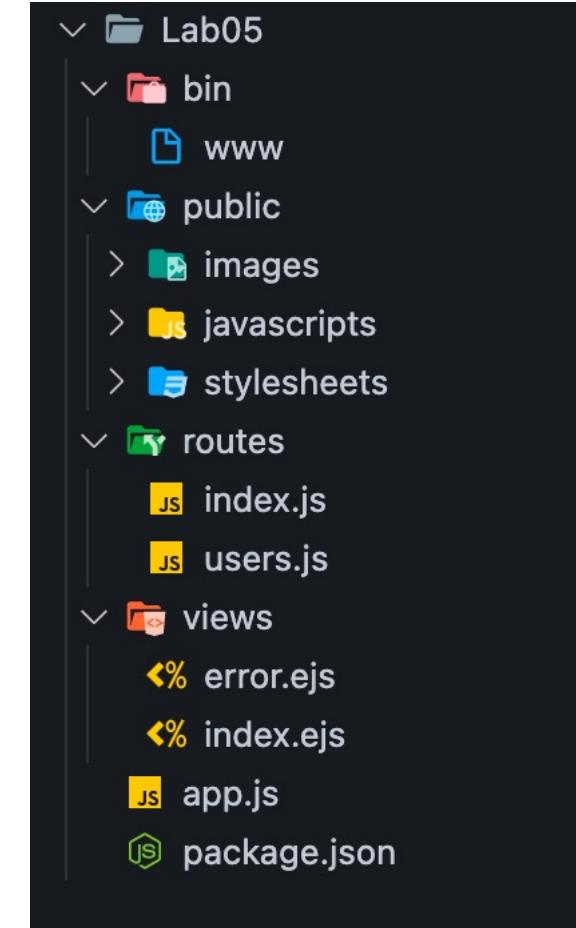
```
> express --view=ejs Lab05

  create : Lab05/
  create : Lab05/public/
  create : Lab05/public/javascripts/
  create : Lab05/public/images/
  create : Lab05/public/stylesheets/
  create : Lab05/public/stylesheets/style.css
  create : Lab05/routes/
  create : Lab05/routes/index.js
  create : Lab05/routes/users.js
  create : Lab05/views/
  create : Lab05/views/error.ejs
  create : Lab05/views/index.ejs
  create : Lab05/app.js
  create : Lab05/package.json
  create : Lab05/bin/
  create : Lab05/bin/www

  change directory:
    $ cd Lab05

  install dependencies:
    $ npm install

  run the app:
    $ DEBUG=lab05:* npm start
```





Bài tập về nhà

- Xây dựng một ứng dụng ExpressJS thực hiện các yêu cầu sau:
 - Xây dựng middleware Authentication thực hiện việc kiểm tra (5 điểm)
 - Nếu request có Cookies.isLogin = true thì cho phép chuyển tiếp vào middleware '/' tiếp theo
 - Nếu request không có Cookies.isLogin hoặc Cookies.isLogin = false thì sẽ thông báo lỗi "Access denied"
 - Xây dựng route.get('/') để xử lý in ra thông tin của isLogin (5 điểm)
 - Nếu request có Cookies.UID thì in UID ra (với UID là User ID)
 - Nếu không có thì trả về thông báo "You are not Admin"

