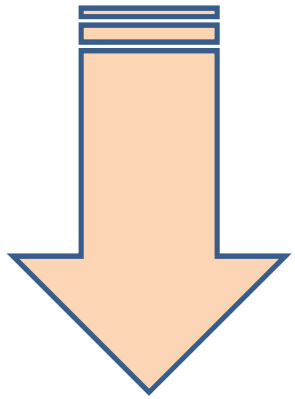
Express.js



Express.js

1. Routes, Request and Response objects, Template Engine
2. Understating Middleware, Query Middleware, Serving Static Files, Handling POST body data, Cookies, Sessions and Authentication

(chapters 18,19 in Node.js,Mongodb and Anguar Web Dev – by Caleb Daley , Brad Dayley, and Brendan Dayley text book)

Express.js

Introduction:

- ✓ Express is a lightweight module that wraps the functionality of the Node.js http module in a simple to use interface.
- ✓ **Express.js** is a fast, flexible and minimalist web framework for Node.js. It's effectively a tool that simplifies building web applications and APIs using JavaScript on the server side.
- ✓ Express is an open-source that is developed and maintained by the Node.js foundation.
- ✓ Express also extends the functionality of the http module to make it easy for you to handle server routes, responses, cookies, and statuses of HTTP requests.
- ✓ **To use Express in Node.js projects, install 'express' from the root dir of the project.**

npm install --save express

- ✓

Express.js

Key Features:

Middleware and Routing: Express.js makes it easy to organize your application's functionality using middleware and routing. Middleware functions allow you to handle tasks like authentication, logging, and error handling. Routing ensures that incoming requests are directed to the appropriate handlers.

Minimalistic Design: Express.js follows a simple and minimalistic design philosophy. This simplicity allows you to quickly set up a server, define routes, and handle HTTP requests efficiently. It's an excellent choice for building web applications without unnecessary complexity.

Flexibility and Customization: Express.js doesn't impose a strict application architecture. You can structure your code according to your preferences. Whether you're building a RESTful API or a full-fledged web app, Express.js adapts to your needs.

Scalability: Designed to be lightweight and scalable, Express.js handles a large number of requests asynchronously. Its event-driven architecture ensures responsiveness even under heavy loads.

Active Community Support: With a thriving community, Express.js receives regular updates and improvements. You'll find ample documentation, tutorials, and plugins to enhance your development experience.

Express.js

Key Features:

- ✓ Fast server side development
- ✓ Middleware
- ✓ Routing
- ✓ Templating
- ✓ Debugging
- ✓ Express allows dynamic rendering of HTML Pages based on passing arguments to templates.

Limitations of Express.js:

- Sometimes, there is no structural way to organize things, and the code becomes non-understandable.
- There are so many issues with callbacks.
- The error messages that will come are challenging to understand.

Companies using Express.js:

- ❑ Netflix
- ❑ eBay
- ❑ Uber
- ❑ IBM

Express.js

Server:

```
var express = require('express');
var app = express();
app.get('/', function(req, res){
  res.send("Hello world!");
});
app.listen(3000);
```

Syntax of defining routes: app.METHOD(PATH, HANDLER);

method can be: GET, PUT, POST, DELETE etc.

- **app.get(route, callback)**
- This function tells what to do when a **get** request at the given route is called. The callback function has 2 parameters, ***request(req)*** and ***response(res)***. The request **object(req)** represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc. Similarly, the response object represents the HTTP response that the Express app sends when it receives an HTTP request.

- The express module provides a series of functions that allow you to implement routes for the Express server. These functions all use the following syntax: app.(path, [callback . . .], callback)

Express.js

Configuring Express Settings :

var express = require('express') //imports 'express' module

var app = express(); //creates instance of 'express'

✓The express object provides the set(setting, value) and enable(setting) and disable(setting) methods to set the value of the application settings.

For example, the following lines of code enable the trust proxy setting and set the view engine to pug:

```
app.enable('trust proxy');
app.disable('strict routing');
app.set('view engine', 'pug');
```

✓To get the value of a setting, you can use the get(setting)

✓enabled(setting), and disabled(setting) → enables and disables the setting.

```
For example: app.enabled('trust proxy'); \\true
app.disabled('strict routing'); \\true
app.get('view engine'); \\pug
```

express application settings

Sl. No.	Setting	Description
1	Env	Defines the environment mode string, such as development, testing, and production. The default is the value of process.env.NODE_ENV.
2	trust proxy	Enables/disables reverse proxy support. The default is disabled
3	jsonp callback name	Defines the default callback name of JSONP requests. The default is ?callback=.
4	json replacer	Defines the JSON replacer callback function. The default is null.
5	json spaces	Specifies the number of spaces to use when formatting JSON response. The default is 2 in development, 0 in production.
6	case sensitivity routing	Enables/disables case sensitivity. For example, /home is not the same as /Home. The default is disabled.
7	strict routing	Enables/disables strict routing. For example, /home is not the same as /home/. The default is disabled.

express application settings

Sl. No.	Setting	Description
8	view cache	Enables/disables view template compilation caching, which keeps the cached version of a compiled template. The default is enabled.
9	view engine	Specifies the default template engine extension that is used when rendering templates if a file extension is omitted from the view.
10	views	Specifies the path for the template engine to look for view templates. The default is ./views.

Configuring Routes

- Routing refers to how an server side application responds to a client request to a particular endpoint. This endpoint consists of a URI (a path such as / or /books) and an HTTP method such as GET, POST, PUT, DELETE, etc.
- Before the server can begin accepting requests, you need to define routes. A route is simply a definition that describes how to handle the path portion of the URI in the HTTP request to the Express server.

Implementing Routes:

- There are two parts when defining the route.
  - First is the HTTP request method (typically GET or POST). Each of these methods often needs to be handled completely differently.
  - Second, is the path specified in the URL—for example, / for the root of the website, /login for the login page, and /cart to display a shopping cart.

The express module provides a series of functions that allow you to implement routes for the Express server. These functions all use the following syntax: app.(path, [callback . . .], callback) The portion of the syntax actually refers to the HTTP request method, such as GET or POST.

For example:

Routing – Express.js

- HTTP methods provide additional functionality to your data using the POST, PUT, and DELETE requests.
- The **POST** request method creates new data in your server
- **PUT** updates existing information.
- The **DELETE** request method removes the specified data.

Serving static files

- **app.use('/static', express.static('assets'));**
- With the above code, static files located in the ‘assets’ directory are now served at the ‘/static’ path prefix. For example, ‘assets/css/style.css’ is accessible at ‘/static/css/style.css’.
- **Example:**
- const express = require('express');
- const app = express();
- const PORT = 3000;
- app.use('/static', express.static('assets'));
- app.get('/', (req, res) => {
- res.send('Hello World!');
- });
- app.listen(PORT, () => console.log(`Server listening on port: ${PORT}`));

- **From browser:**
- localhost:3000/static/temp.html sends as response ‘temp.html’ stored in assets folder .

Express.js - Sending and Receiving Cookies

-

Express.js - Sending and Receiving Cookies

-