Developing Back-End Apps with Node.js and Express

Module 3 Cheat Sheet: Express Web Application Framework

Package/Method	Description	Code Example
Dependencies in `package.json`	A dependency of **express** version between **4.0 to 5.0** will be declared as:	`"dependencies":{"express":"4.x"}`
new express()	Creates an express object which acts as a server application.	`const express = require("express");` `const app = new express();`

express.listen()	The listen method is invoked on the express object with the port number on which the server listens. The function is executed when the server starts listening.	`app.listen(3333, () => {`
		`  console.log("Listening at`
		`  http://localhost:3333)`
		`})`

express.get();	This method is meant to serve the retrieve requests to the server. The get() method is to be implemented with two parameters; the first parameter defining the end-point and the second parameter is a function taking the request-handler and response-handler.	`// handles GET queries to end point /user/about/id.`
		`app.get("user/about/:id", (req,res)=>{`
		`  res.send("Response about user "`
		`  +req.params.id)`
		`})`

express.post();	This method is meant to serve the create requests to the server. The post() method is to be implemented with two parameters: the first parameter defines the end-point and the second parameter is a function taking the request-handler and response-handler.	`// handles POST queries to the same end point.`
		`app.post("user/about/:id", (req,res)=>{`
		`  res.send("Response about user "`
		`  +req.params.id)`
		`})`

express.use()	This method takes middleware as a parameter. Middleware acts as a gatekeeper in the same order that it is used, before the request reaches the get() and post() handlers. The order in which the middleware is chained depends on the order in which the .use() method is used to bind them. The middleware myLogger() function takes three parameters, which are request, response, and next. You can define a method that takes these three parameters and then bind it with express.use() or router.use(). Here, you are creating middleware named myLogger and making the application use it. The output rendered includes the time the request is received.	`const express = require("express");`
		`const app = new express();`
		`function myLogger(req, res, next){`
		`  req.timeReceived = Date();`
		`  next();`
		`}`
		`app.get("/", (req, res)=>{`
		`  res.send("Request received at`
		`  "+req.timeReceived+" is a success!")`
		`})`

express.Router()	Router-level middleware is not bound to the application. Instead, it is bound to an instance	`const express = require("express");`
		`const app = new express();`
		`let userRouter = express.Router();`
		`let itemRouter = express.Router();`
		`userRouter.use(function (req, res, next){`

	of express.Router(). You can use specific middleware for a specific route instead of having all requests going through the same middleware. Here, the route is /user and you want the request to go through the user router. Define the router, define the middleware function that the router will use and what happens next, and then you bind the application route to the router.	`  console.log(“User quert time:”, Date());`
		`  next();`
		`})`
		`userRouter.get(“/:id”, function (req, res,`
		`  next) {`
		`  res.send(“User ”+req.params.id+ “ last`
		`  successful login ”+Date())`
		`})`
		`app.listen(3333, () => {`
		`  console.log(“Listening at http://localhost:3333)`
		`})`

express.static()	This is an example of static middleware that is used to render static HTML pages and the images from the server side. At the application level, the static files can be rendered from the cad220_staticfiles directory. Notice that the URL has only the server address and the port number followed by the filename.	`const express = require(“express”);`
		`const app = new express();`
		`app.use(express.static("cad220_staticfiles"))`
		`app.listen(3333, () => {`
		`  console.log("Listening at  http://localhost:3333")`
		`})`

jsonwebtoken.sign()	Used for signing-in based on a generated JWT (JSON Web token)	`if (uname === "user" && pwd === "password") {`
		`  return res.json({`
		`  token: jsonwebtoken.sign({ user: "user" }, JWT_SECRET),`
		`  });`
		`  }`

jsonwebtoken.verify()	Verifies a JWT by passing the token value & the JWT secret as arguments.	`const verificationStatus =  jsonwebtoken.verify(tokenValue, "aVeryVerySecretString");`

Project folder strucure	A fairly established project structure for API's built using Express.js is:	`test-project/`
		`  node_modules/`
		`  config/`
		`  db.js  //Database connection and configuration`
		`  credentials.js  //Passwords/API keys for external services used by your app`
		`  models/  //For mongoose schemas`
		`  items.js`
		`  prices.js`
		`  routes/  //All routes for different entities in different files`
		`  items.js`
		`  prices.js`
		`  app.js`
		`  routes.js  //Require all routes in this and then require this file in`
		`  package.json`

Skills Network