Open-Source Report

Proof of knowing your stuff in CSE312

Guidelines

Provided below is a template you must use to write your reports for your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we need to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

[Axios]

General Information & Licensing

Code Repository	https://github.com/axios/axios
License Type	MIT
License Description	This type of license allows users to reuse code for any purpose, even if it is part of proprietary software. Users are free to modify the code as long as they include the original copy of the MIT license in their distributions. The MIT License allows user to use the software for : ● Commercial use ● Modify

	• Distribute • Sublicense • Private User The user of the Software with the MIT License must include: • Copyright • License
License Restrictions	Software that contains an MIT license restricts its user to hold the author of the software liable. This means any mistakes can not hold the author of the software for any mistakes.

[Bodyparser]

General Information & Licensing

Code Repository	https://github.com/expressjs/body-parser
License Type	MIT
License Description	This type of license allows users to reuse code for any purpose, even if it is part of proprietary software. Users are free to modify the code as long as they include the original copy of the MIT license in their distributions. The MIT License allows user to use the software for : • Commercial use • Modify • Distribute • Sublicense • Private User The user of the Software with the MIT License must include: • Copyright • License
License Restrictions	Software that contains an MIT license restricts its user to hold the author of the software liable. This means any mistakes can not hold the author of the software for any mistakes.

[Cross-Origin Resource Sharing]

General Information & Licensing

Code Repository	https://github.com/OpenLiberty/guide-cors
License Type	MIT
License Description	This type of license allows users to reuse code for any purpose, even if it is part of proprietary software. Users are free to modify the code as long as they include the original copy of the MIT license in their distributions. The MIT License allows user to use the software for : ● Commercial use ● Modify ● Distribute ● Sublicense ● Private User The user of the Software with the MIT License must include: ● Copyright ● License
License Restrictions	Software that contains an MIT license restricts its user to hold the author of the software liable. This means any mistakes can not hold the author of the software for any mistakes.

[Express.js]

General Information & Licensing

Code Repository	https://github.com/expressjs/express
License Type	MIT
License Description	This type of license allows users to reuse code for any purpose, even if it is part of proprietary software. Users are free to modify the code as long as they include the original copy of the MIT license in their distributions.

	The MIT License allows user to use the software for : ● Commercial use ● Modify ● Distribute ● Sublicense ● Private User The user of the Software with the MIT License must include: ● Copyright ● License
License Restrictions	Software that contains an MIT license restricts its user to hold the author of the software liable. This means any mistakes can not hold the author of the software for any mistakes.

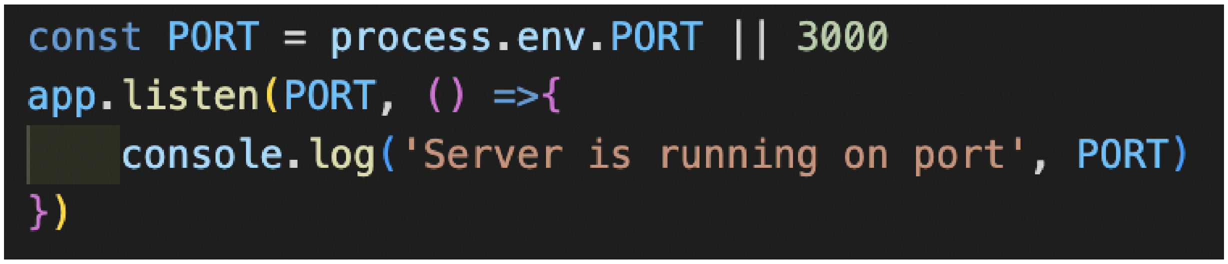
Magic ★★⁺₊˚‧˚☽˚⁀➴₊˚★⚞✧༄

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section will likely grow beyond the page

TCP CONNECTION



```
const PORT = process.env.PORT || 3000
app.listen(PORT, () =>{
    console.log('Server is running on port', PORT)
})
```

This portion of the code is setting up the HTTP server using the Express framework in Node.js which will allow it to handle incoming requests from clients and send responses back to them.

The app.listen() method is used to start the server and listen for incoming requests on a particular port.

The const PORT = process.env.PORT || 3000 line is defining a constant named PORT and setting its value to the value of the PORT environment variable, or 3000 if the PORT environment variable is not set. This means that the server will listen on port 3000 by default, but it can be configured to listen on a different port by setting the PORT environment variable.

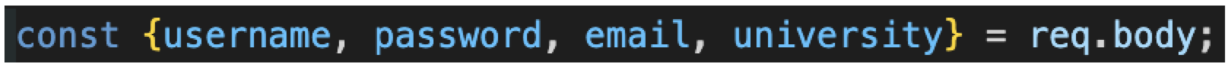
The app.listen() method is then called with the PORT constant as its first argument and a callback function as its second argument. The callback function will be called when the server has started and is listening for incoming requests. The console.log() statement inside the callback function is used to log a message to the console indicating that the server is running on the specified port.

PARSING HTTP HEADERS

Axios is a popular HTTP client library for JavaScript that can be used in the browser and in Node.js. CORS (Cross-Origin Resource Sharing) is a mechanism that allows a web page to make requests to a server on a different origin (domain, protocol, or port) than the page itself. Axios can be used in conjunction with CORS to make HTTP requests to a server on a different origin. By default, Axios automatically sets the Origin header of the request to the page's origin, and the server can use this header to determine whether to allow the request. If the server allows the request, it sends a Access-Control-Allow-Origin header in the response, indicating that the request is allowed.

Router.post is a method that is used for HTTP POST requests. It is used in some of the pictures to submit data to a server for processing. When the request is sent to the server. Router.post is used to specify what should happen with the data that is included in the request. Router.get is used for HTTP GET requests when you want to retrieve data from the server. When a GET is sent over the server, router.get is used to specify what should happen in response to the request.

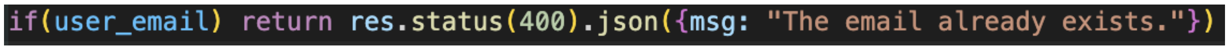
Req.body (loginController.js line 10)



```
const {username, password, email, university} = req.body;
```

This portion uses destructuring assignment to extract the username, password, email, university from the req.body object. The code uses destructuring assignment to extract the values of the username, password, email, and university properties from the req.body object. This object is expected to contain the request data submitted by the client, in the form of a JSON object. The destructuring assignment statement extracts the values of the specified properties from the req.body object and assigns them to variables with the same names.

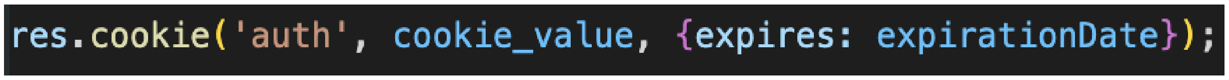
Res.status (loginController.js line 12)



```
if(user_email) return res.status(400).json({msg: "The email already exists."})
```

the res.status() method is used to set the HTTP status code of the response. It takes a single argument, which is the numeric status code that should be set. For example, the code res.status(400) sets the status code of the response to 400. The res.json() method is used to send a JSON response to the client. It takes a single argument, which is the JavaScript object that should be converted to JSON and sent as the response body.

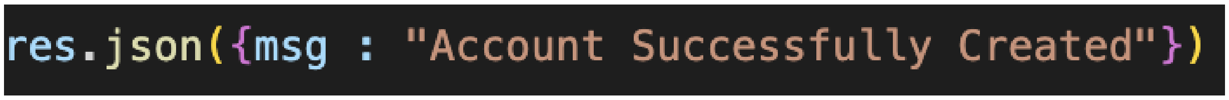
Res.cookie (loginController.js line 19)



```
res.cookie('auth', cookie_value, {expires: expirationDate});
```

The res.cookie() method is used to set a cookie in the HTTP response. It takes three arguments: the name of the cookie, the value of the cookie, and an options object that specifies additional details about the cookie.

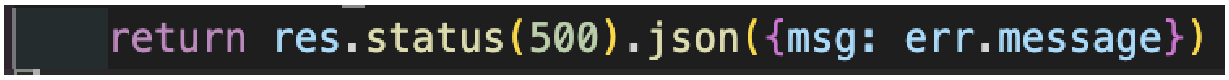
Res.json (loginController.js line 25)



```
res.json({msg : "Account Successfully Created"})
```

The res.json() method is used to send a JSON response to the client with a message indicating that the user's account was successfully created. The client can then use this information to display a success message to the user.

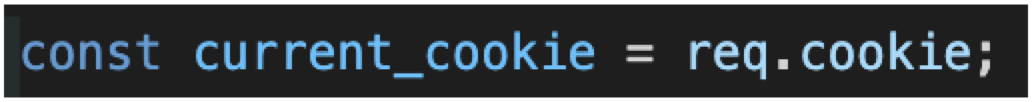
Res.status (loginController.js line 28)



```
return res.status(500).json({msg: err.message})
```

The code is executed inside a try block. If an error occurs while executing the code in the try block, it will be caught by the catch block and the res.status() and res.json() methods will be used to send an error response to the client. The err.message property contains the error message, which is then sent to the client in the JSON response.

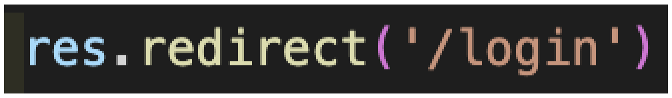
Req.cookie (loginController.js line 60)



```
const current_cookie = req.cookie;
```

The req.cookie property is used to get the cookie from the request. If the cookie is falsy, the code redirects the client to the /login page and returns a JSON response with an error message.

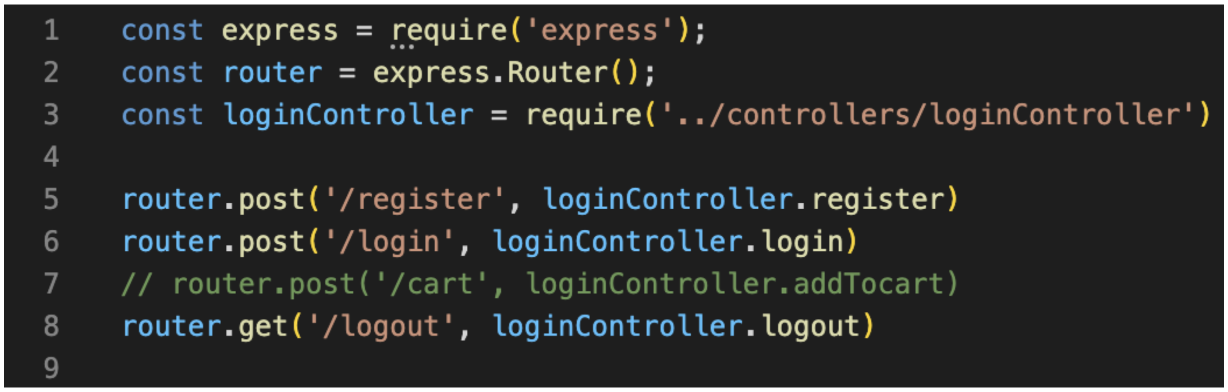
Req.redirect (loginController.js line 62)



```
res.redirect('/login')
```

The res.redirect() method is used to redirect the client to the /login page. This can be used to redirect the client to a login page if they are not logged in or do not have the necessary permissions to access a protected resource.

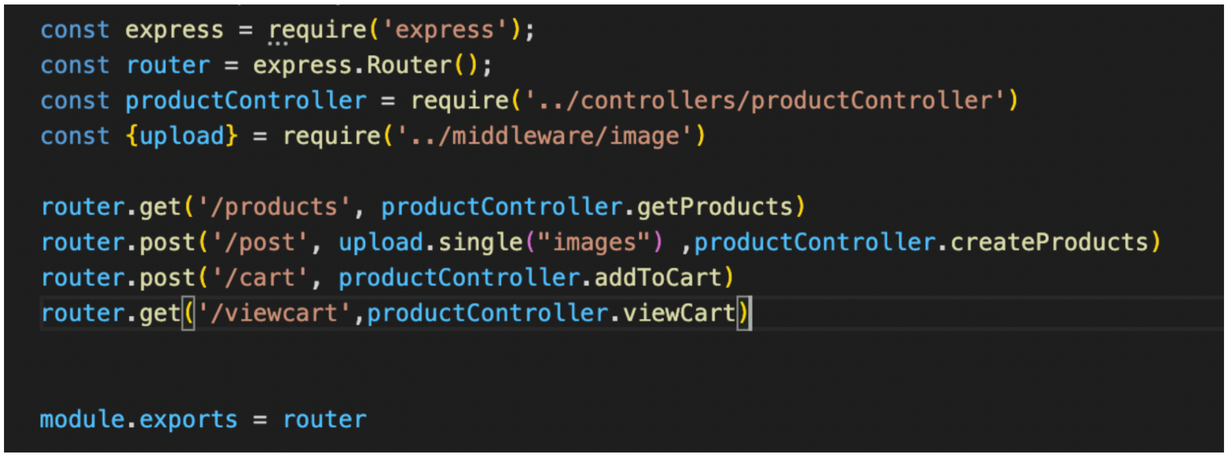
Login.js



```
1   const express = require('express');
2   const router = express.Router();
3   const loginController = require('../controllers/loginController')
4
5   router.post('/register', loginController.register)
6   router.post('/login', loginController.login)
7   // router.post('/cart', loginController.addTocart)
8   router.get('/logout', loginController.logout)
9
```

This portion consists of 3 routes, one for handling user registration, one for user login, and one for user logout.
The first route in line 5 handles HTTP POST requests to the /register path. When the request is received at this path, the loginController.register method is used to handle the request and generate a response. It is the method that is responsible for registering a new user.
The second route in line 6 handles the request to the login path. This method is responsible for authenticating the user and logging them into the application. It could do this by checking the user's credentials against a database and creating a session token.

Product.js

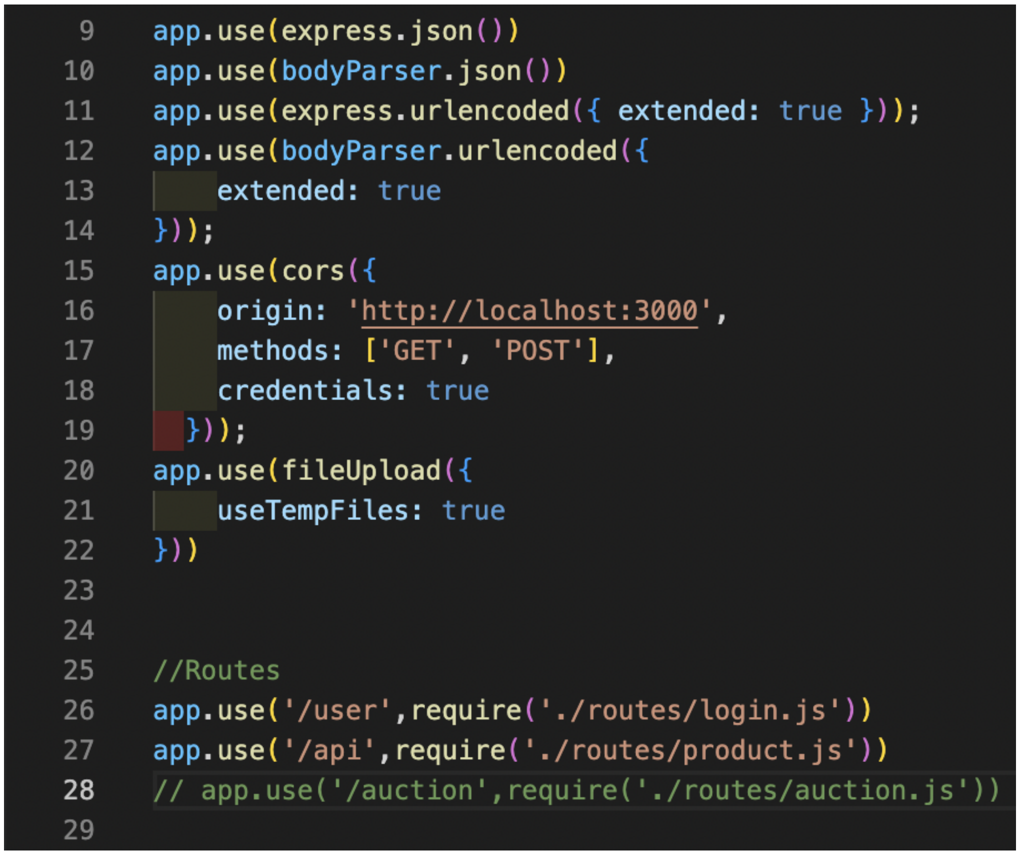


```
const express = require('express');
const router = express.Router();
const productController = require('../controllers/productController')
const {upload} = require('../middleware/image')

router.get('/products', productController.getProducts)
router.post('/post', upload.single("images") ,productController.createProducts)
router.post('/cart', productController.addToCart)
router.get('/viewcart',productController.viewCart)


module.exports = router
```

This part of the code uses the get and post paths. On line 3, we specify the folder and file from where we call the getProducts, createProducts and the viewCart functions.
On line 5, the getProduct function is called using the get request which acquires the product information that users inputs. On line 6, the post request is used by calling the createProducts function that handles the image, description, price and the item that the user input. And lastly, line 7 uses the get request to call viewCart that displays the products on the UI.

This code creates a router using the express.Router() method and defines some routes for the router. The router.get() and router.post() methods are used to handle HTTP GET and POST requests, respectively. The router.get('/products') route will handle requests to the /products endpoint and will return a list of products by calling the getProducts method from the productController module. The router.post('/post') route will handle requests to the /post endpoint and will create a new product by calling the createProducts method from the productController module. The router.get('/viewcart') route will handle requests to the /api/viewcart endpoint and will return the user's products in the cart by calling the viewCart method from the productController module.
The router.post('/cart') route will handle post requThe router.post('/cart') route will handle post requests to the /api/cart endpoint and will return the user's products in the cart by calling the viewCart method from the productController module.

Server.js



```
app.use(express.json())
app.use(bodyParser.json())
app.use(express.urlencoded({ extended: true }));
app.use(bodyParser.urlencoded({
    extended: true
}));
app.use(cors({
    origin: 'http://localhost:3000',
    methods: ['GET', 'POST'],
    credentials: true
  }));
app.use(fileUpload({
    useTempFiles: true
}))


//Routes
app.use('/user',require('./routes/login.js'))
app.use('/api',require('./routes/product.js'))
// app.use('/auction',require('./routes/auction.js'))

```

This portion includes multiple middleware functions that handle a variety of aspects of the application. Some of these are parsing JSON and URL-encoded data, enabling CORS (Cross-Origin Resource Sharing) for cross-domain requests, and allowing file uploads.

The app.use() method is used to add these middleware functions to the application. The express.json() and bodyParser.json() middleware functions enable the application to parse JSON data included in HTTP requests. The express.urlencoded() and bodyParser.urlencoded() middleware functions enable the application to parse URL-encoded data included in HTTP requests. The cors() middleware function enables the application to handle cross-domain requests, allowing the application to accept requests from the specified origin (in this case, http://localhost:3000) and using the specified HTTP methods (in this case, GET and POST). The fileUpload() middleware function enables the application to handle file uploads.

<u>WEBSOCKET</u>

Socket.io is a library that can be used with Express to add functionality to the application. It uses WebSockets to establish a connection between the client and server, allowing them to send and receive data and for efficient and low-latency communication. On the client-side, the code connects to the server using the socket.io-client library and listens for connect and message events.