Week 4 Workshop

NodeJS Express MongoDB

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes
Check-In	10 minutes
Week Recap	40 minutes
Task 1 & 2	60 minutes
BREAK	15 minutes
Tasks 2 & 3 - Leave time for testing!	90 minutes
Check-Out	15 minutes

Check-In

- How was this week? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- You must complete all Exercises before beginning the Workshop Assignment.

Week 4 Recap - Overview

New Concepts This Week

• HTTPS
  • SSL/TLS
  • Symmetric-Key Cryptography
  • Public-Key Cryptography
  • SSL Certificates & OpenSSL
  • TLS Handshaking
	• Uploading Files with Multer
	• Cross-Origin Resource Sharing
	• OAuth

Next slides will review these concepts

HTTPS

- Hyper Text Protocol Secure
- Uses TLS (Transport Layer Security) for secure channel
- Secure channel: Privacy + Integrity
- Privacy: Prevent data being read by third parties
- Integrity: Prevent data from being changed by third parties

Transport Layer Security (TLS) Protocol

- Often still called SSL/TLS or even just SSL though SSL protocol has been deprecated – TLS is its successor
- TLS is official cryptographic protocol for the web
- Uses public-key cryptography to generate session keys for symmetric-key cryptography using a "handshaking" process

Symmetric Key Cryptography

- Both sender and receiver sides have the same secret key
- They use it along with an encryption algorithm to encrypt and decrypt messages to each other
- Even if a message is intercepted and the encryption algorithm is known, cannot be decrypted easily without the secret key
- Brute force techniques could guess it, but would take a lot of time and processing power

Public-Key Cryptography

- A receiver (such as a server) obtains a private/public key pair
- Public key can be distributed freely
- Private key is held by receiver only
- Public key and private key are crytographically related but mathematically impossible to figure out private key from public key
- **Discuss:** Between the public key and the private key, which one encrypts, and which one decrypts? (student to answer)

Public-Key Cryptography

- Answer: Public key is used by a sender (client) to encrypt message, public key cannot decrypt. Private key held by receiver (server) is used to decrypt message encrypted with public key, not used to encrypt.
- **Discuss next:**
  - What is the advantage of public-key cryptography over symmetric key cryptography?
  - Why don't we use public-key cryptography for everything?

Public-Key Cryptography

- Answers:
  - The advantage of public-key cryptography is that you don't have to send the decryption key between server and receiver, so it's not vulnerable to being intercepted by a third party
  - We don't use it all the time because it's an expensive process

SSL Certificate

- A public key must be certified by a Certificate Authority (CA), or browsers will not accept it
- To obtain a public/private key pair, go through certification process with a CA, then they will grant you a SSL certificate that contains a public key, along with the correponding private key
- You can then send your SSL certificate to clients who ask to communicate securely, and the client will be able to verify you are who you say you are through the certificate
- For development purposes, use OpenSSL to generate a self-signed certificate

TLS Handshaking

- Process by which server and client establish secure channel for communication. In a nutshell:
  - Client requests to communicate securely, server sends back SSL certificate
  - Client verifies certificate, sends "premaster secret" random string to server, encrypted with server's public key
  - Server decrypts premaster secret using private key, then client and server both generate session keys using premaster secret and other identical inputs/steps
  - Both sides then use session key for symmetric key cryptography

Uploading Files

- Files typically uploaded through form input
- In HTML forms, through `<input type="file" name="myFile" />`
- The form should supply an endpoint that handles the file upload on the server side through the action attribute, along with post method and enctype of multipart/form-data, e.g.:

```
<form action="/imageUpload" method="post" enctype="multipart/form-data">
```

Uploading Files with Multer

- Server must be able to process an uploaded file and handle it appropriately
- Multer middleware is commonly used to help with this
- Multer lets you configure where and how to save the file, and apply a filter if needed to accept or reject the upload
- Multer sets two objects on the request message: **body** and **file** (or **files**, if multiple files are being uploaded)
- **req.body** object will contain text from form text inputs, if any
- **req.file** object will contain file + additional info
- **req.files** object will contain array of files + additional info

Same-Origin Policy

- All modern browsers implement a same-origin policy to prevent against malicious scripts gaining access to protected resources
- Same origin means:
  - same URI scheme (protocol)
  - same hostname
  - same port

Origin

- Given this URL: https://my.nucampsite.com/

- Say whether the following URLs are of the same origin or not:
  - https://www.nucampsite.com
  - https://my.nucampsite.com/welcome
  - http://my.nucampsite.com/messages
  - https://my.nucampsite.com:99

Origin

- Answer:
  - Only https://my.nucampsite.com/welcome is of the same origin.

Cross-Origin Resource Sharing

- CORS standards allow us to explicitly allow resources from another origin to be shared that would otherwise be blocked due to browser's same-origin policy, using headers
- CORS HTTP response headers:
  - Access-Control-Allow-Origin is the most common, lets browsers know if certain origins are allowed to access a resource
  - Other headers include Access-Control-Allow-Credentials, Access-Control-Allow-Headers, etc

CORS Requests: Simple

- Two main types: Simple and Preflighted
- Simple requests are safe, do not need preflighting
  - GET
  - HEAD
  - POST if meeting following criteria:
    - Content-Type can only be application/x-www-form-urlencoded, multipart/form-data, or text/plain
    - No custom headers, only certain safe headers permitted
- Server can set endpoints to accept simple requests regardless of origin
- Server can set whitelist of origins to accept, or * to allow all requests to a particular endpoint

CORS Requests: Preflighted

- All requests that don't meet criteria for simple requests are preflighted: unsafe requests, can possibly cause changes in existing server data
- Must send preflight request to server before sending actual request
  - OPTIONS request including CORS request headers such as Origin, Access-Control-Request-Method, etc to describe to server what client wishes to actually request, from what origin, etc
- Server responds with status code 200 if it's OK to send the actual request, along with CORS response headers about what server will accept, such as Access-Control-Allow-Origin, Access-Control-Allow-Methods
- Client then decides whether or not to send actual request
- Npm module **cors** helps us to configure all this

OAuth

- OAuth 1 evolved from Twitter, now deprecated
- OAuth 2 is its successor
- OpenID is an alternative authentication protocol
- Open standards to allow websites to use third party authentication providers for users to log in
- Third party "resource servers" hold user profile/account data
- Our server must go through OAuth 2 server to gain access to resources

OAuth & Facebook

- We looked at an example of using an implicit grant flow to authenticate users via Facebook's OAuth server over HTTPS
- Register with Facebook to get App ID & App Secret
- User clicks Log In button for Facebook, grants our app access
- Front end app contacts Facebook with App ID to get access token, then passes access token to back end Express server app
- Express app sends user's access token back to Facebook OAuth server along with App ID & App Secret, OAuth server validates and sends back user profile information
- Express app then creates new user account for user if there isn't one, and sends user a JSON Web Token to use for access to our server

Workshop Assignment

- It's time to start the workshop assignment!
- Task 2 will be the most time-taking and involve additional JavaScript on top of what you have learned.
- Leave yourselves time for testing with Postman after Task 3. Watch the assignment video for instructions on how to test.
- Break out into groups of 2-3. Sit near your workshop partner(s).
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - 10-minute rule does *not* apply to talking to your partner(s). You should consult each other throughout.
- Follow the workshop instructions very closely.
  - Both the video and written instructions. Pay careful attention to any screenshots in the written instructions.
- Talk to your instructor if any of the instructions are unclear to you.

Check-Out

- Submit to the learning portal one of the following options:
  - Either: a zip file of your entire **nucampsiteServer** folder with your updated files, *excluding* the node_modules folder,
  - Or: a text file that contains the link to a public online Git repository for the **nucampsiteServer** folder.
- Wrap up – Retrospective
  - What went well
  - What could improve
- Congratulations on making it to the end of the Full Stack Bootcamp!