Secure Software Development
Project 02

Matthew Byrne

GitHub Repo

Introduction

The goal of this project was to take the initial guidelines set out in the JIRA board created in Project 1 and apply the security techniques discussed to a simple implementation of the web application described in the project spec. The technology chosen to implement this was Node.js with Express and MongoDB as a datastore, these technologies were chosen both because of their familiarity, and the large number of robust security focused packages that are available on NPM. This project heavily relies on NPM packages for the implementation of security features that are known to be secure due to the open-source and highly scrutinised nature of these packages. Packages such as Crypto, Passport, CSURF, Multer, Helmet, Express-session, CORS, Winston, and Morgan were used throughout development.

User stories outlined in the JIRA board were used as a reference during the development to track progress. During the project development there were several occurrences where packages initially specified in user stories were not suitable for the implementation of the given feature, and a different package had to be discovered and investigated for development progress to continue. Some of the initial elications from the JIRA board were not implemented due to them being highly complex, out of scope, or severely time consuming. However, if this project were to eventually be used as a production application these steps would be required to ensure the security of the application. The entire development of the application will be discussed further below, outlining exactly where issues arose, what was left un-implemented from the initial design, how secure design was implemented, and the security testing that was performed.

Authentication

Passport.js was used to handle user authentication with "local" strategy to handle user registration and login locally with a username and password. Passwords are securely hashed using the Crypto package with the PBKDF2 algorithm and then salted to ensure proper storage of user secrets and to prevent the use of rainbow tables by attackers. Included in the initial JIRA board was TOTP based MFA, automated account lockouts, and third party OAuth service login options such as Google sign-in. If this application were to become a production application these would be implemented to improve the application's security.

Authorisation

Authorisation is implemented using a custom middleware that calls passport's built-in function isAuthenticated(), which checks whether or not the incoming request has a valid login. When a user first registers a flag is set in the user record to record whether or not this account belongs to an administrator, this allows the same custom middleware to confirm whether or not the request is coming from a person with admin privileges. Currently this admin flag is toggled manually in the code, but this would of course be changed for a more secure solution for a production version of the application. With the use of isAuthenticated() & isAdmin() the server can implement role-based access control for certain routes. In the case of this project there are two protected routes; the merchant file upload portal, and the admin panel that shows meta-data about all the files uploaded. Throughout development of the authN & authZ processes OWASP guidelines such as RBAC, PoLP, proper password validation & handling and several others were implemented to build a secure application.

Session Management

Sessions are handled using Express-Session, session data is stored on the Mongodb database. Sessions tokens are configured with security in mind and follow OWASP's Session Management guidelines, Session ID's have a length of 192 bits and use a strong CSPRNG, session tokens are regenerated on auth, cookie headers such as "secure", "SameSite",“ HTTPOnly“ are used with HSTS configured, session cookies have a maxAge of 24 hours, a new session ID is generated for each new visit to the site, and session tokens are destroyed on logout.

CSRF

CSRF tokens are used on the merchant file-upload page to verify incoming requests as being from the authenticated user. The CSURF package is used to generate and validate tokens from the user's session, when the form is submitted the server checks if the CSRF token matches. In this case tokens had to be sent to the server using a URL parameter due to the way the package Multer (used for form submission/validation) handles multi-part forms. In a production version of this application proper implementation with CSRF tokens embedded as a hidden input inside the form would be preferred as the current implementation does risk exposing tokens to bad actors through logs and browser history. Discussion of this bug is linked in appendix B. OWASP's CSRF prevention cheat sheet was used in the implementation of CSRF tokens.

Data Validation & Sanitisation

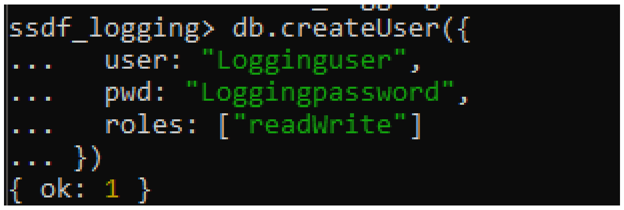
Multer is used to handle form submission to ensure proper validation of the file-upload form, more specifically, to restrict file type and size. Multer also handles processing the file upload. General text based input validation is done with express-validator on inputs such as usernames and passwords to ensure they meet minimum complexity requirements and follow OWASP guidelines on Authentication. Usernames must at minimum be 3 characters long and passwords must be between 8 and 30 characters long, contain at least 1 uppercase letter, 1 lowercase letter, and 1 number. Merchant files are restricted to only be .PDFs and less than 1MB in size. For input sanitization packages XSS & express-mongo-sanitise are used to automatically escape & encode any potentially malicious inputs before being stored on the database or shown on the page. Fortunately, in the case of this basic application only usernames and filenames are user-controlled data. OWASP's cheat sheets on XSS & SQL Injection prevention were used in the implementation of input sanitisation. An ODM is also used to further mitigate the chance of a successful SQL injection attack.

Data Handling & Storage

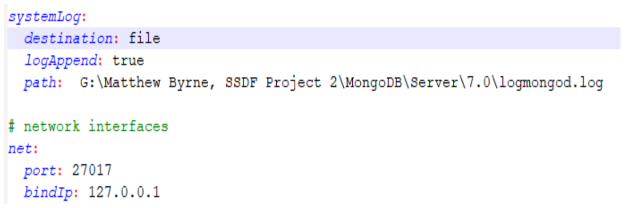
The ODM Mongoose is used to implement schemas for another layer of protection against SQL injection attacks as it allows for the abstraction of user input in database queries. When a user first submits a file it is stored in the file system as temporary storage with a new name given to it by the server. As incorporating a full virus scan API is out of scope for this simple implementation a simulated version is used instead to randomly decide if a file is marked as safe or unsafe. In a real-world scenario this RNG function would be replaced with sending the file to a third-party API to perform a virus scan, and a CAPTCHA system would be implemented to prevent automated scripts.

If the file is found to be safe then it is encrypted at rest to AES256 bit level using the Crypto package, and is then stored in the MongoDB as an encrypted binary. If a file is marked as unsafe it is removed from local storage and an event log is made. Regardless of outcome the temporary files are removed from the local file system once the operation is complete. OWASP's file handling cheat sheet was used as a guide to securely implement file-submission handling & storage.

User auth data, session data, files, and logs are stored securely with Mongoose schemas. The tables used in the db require authentication. User accounts were created with simple read and write permissions so as to follow the principle of least privilege. Mongodb has been configured to block all requests originating from outside IP's and to log all activity.



```
ssdf_logging> db.createUser({
...    user: "Logginguser",
...    pwd: "Loggingpassword",
...    roles: ["readWrite"]
... })
{ ok: 1 }
```



```
systemLog:
  destination: file
  logAppend: true
  path:  G:\Matthew Byrne, SSDF Project 2\MongoDB\Server\7.0\logmongod.log

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1
```

In a production application Mongodb enterprise edition would be used to enable disk level encryption at rest and TLS encryption in transit. Field level encryption would also be performed on all sensitive user data rather than just the merchant files as has been done in this example. Finally, regular database backups would also be scheduled to occur on a regular basis.

Logging & Error Handling

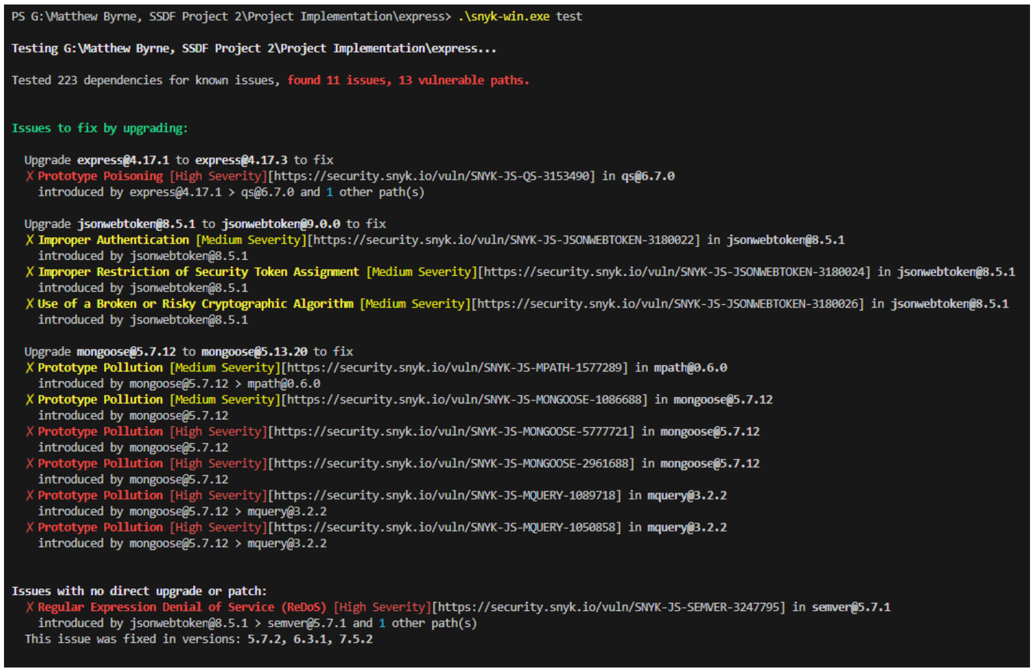
Package's Winston and Morgan are used for logging. Morgan logs all incoming HTTP requests to the console so they can be viewed live. Winston logs are created when an operation occurs that requires a record of an incident, examples such as a file failing the virus scan, if someone fails a login, if there is a server error, or if an unauthenticated user tries to navigate to a protected route will trigger a log event to occur. Information such as the URL, status, user agent, and originating IP address are recorded with a reason for the record. Logs are stored securely on the MongoDB database. Whenever an error occurs, the event is caught and a basic error message is returned to the user whilst a more detailed log of the event is triggered.

Secure Configuration

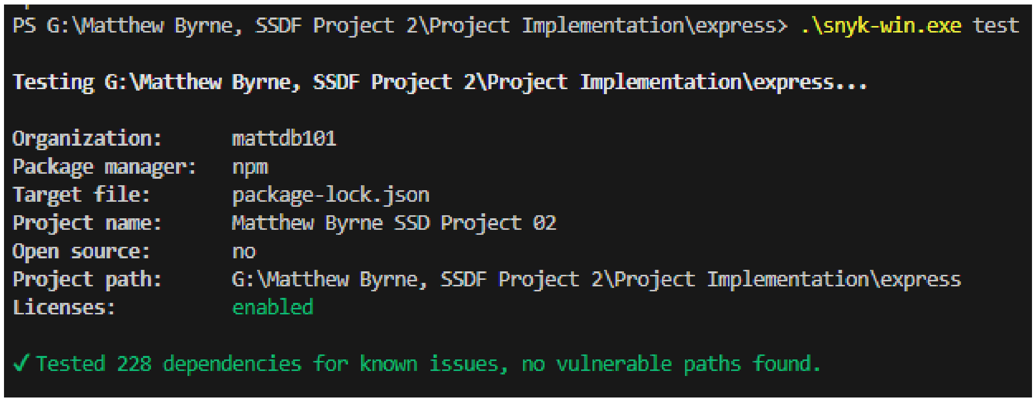
All secrets are stored in and called from environment variables to prevent exposure of sensitive information. If this project were running on a public domain rather than localhost the communication between the client and the server would be encrypted using HSTS. As stated, this project makes use of RBAC, CSRF Protection, and Data Sanitisation. It also makes use of rate limiting, secure session cookies, secure headers, CSP, CORS, feature policies, and other security configuration measures. If this were a production application a CDN with DDOS protection would be implemented as well as firewall protection.

Secure Dependencies

Snyk was used to test for security vulnerabilities in the dependencies used.



```
PS G:\Matthew Byrne, SSDF Project 2\Project Implementation\express> .\snyk-win.exe test

Testing G:\Matthew Byrne, SSDF Project 2\Project Implementation\express...

Tested 223 dependencies for known issues, found 11 issues, 13 vulnerable paths.


Issues to fix by upgrading:

  Upgrade express@4.17.1 to express@4.17.3 to fix
  ✗ Prototype Poisoning [High Severity][https://security.snyk.io/vuln/SNYK-JS-QS-3153490] in qs@6.7.0
    introduced by express@4.17.1 > qs@6.7.0 and 1 other path(s)

  Upgrade jsonwebtoken@8.5.1 to jsonwebtoken@9.0.0 to fix
  ✗ Improper Authentication [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-JSONWEBTOKEN-3180022] in jsonwebtoken@8.5.1
    introduced by jsonwebtoken@8.5.1
  ✗ Improper Restriction of Security Token Assignment [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-JSONWEBTOKEN-3180024] in jsonwebtoken@8.5.1
    introduced by jsonwebtoken@8.5.1
  ✗ Use of a Broken or Risky Cryptographic Algorithm [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-JSONWEBTOKEN-3180026] in jsonwebtoken@8.5.1
    introduced by jsonwebtoken@8.5.1

  Upgrade mongoose@5.7.12 to mongoose@5.13.20 to fix
  ✗ Prototype Pollution [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-MPATH-1577289] in mpath@0.6.0
    introduced by mongoose@5.7.12 > mpath@0.6.0
  ✗ Prototype Pollution [Medium Severity][https://security.snyk.io/vuln/SNYK-JS-MONGOOSE-1086688] in mongoose@5.7.12
    introduced by mongoose@5.7.12
  ✗ Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-MONGOOSE-5777721] in mongoose@5.7.12
    introduced by mongoose@5.7.12
  ✗ Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-MONGOOSE-2961688] in mongoose@5.7.12
    introduced by mongoose@5.7.12
  ✗ Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-MQUERY-1089718] in mquery@3.2.2
    introduced by mongoose@5.7.12 > mquery@3.2.2
  ✗ Prototype Pollution [High Severity][https://security.snyk.io/vuln/SNYK-JS-MQUERY-1050858] in mquery@3.2.2
    introduced by mongoose@5.7.12 > mquery@3.2.2


Issues with no direct upgrade or patch:
  ✗ Regular Expression Denial of Service (ReDoS) [High Severity][https://security.snyk.io/vuln/SNYK-JS-SEMVER-3247795] in semver@5.7.1
    introduced by jsonwebtoken@8.5.1 > semver@5.7.1 and 1 other path(s)
  This issue was fixed in versions: 5.7.2, 6.3.1, 7.5.2
```

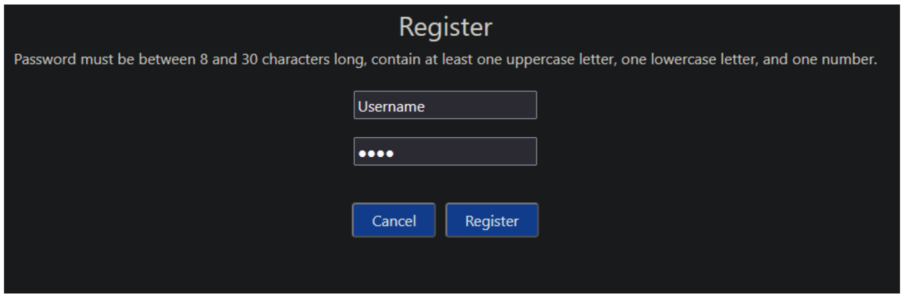


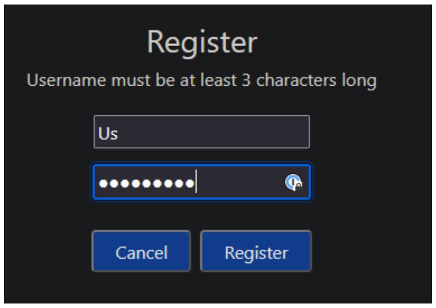
```
PS G:\Matthew Byrne, SSDF Project 2\Project Implementation\express> .\snyk-win.exe test

Testing G:\Matthew Byrne, SSDF Project 2\Project Implementation\express...

Organization:      mattdb101
Package manager:   npm
Target file:       package-lock.json
Project name:      Matthew Byrne SSD Project 02
Open source:       no
Project path:      G:\Matthew Byrne, SSDF Project 2\Project Implementation\express
Licenses:          enabled

✓ Tested 228 dependencies for known issues, no vulnerable paths found.
```

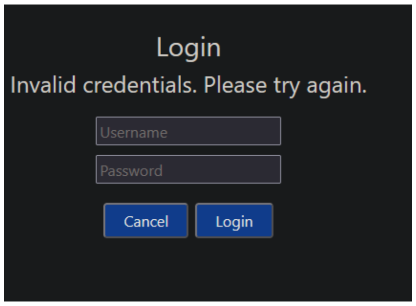
After updating packages there are no known security issues with the dependencies used.

Testing

Testing input validation

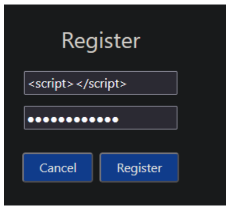

Register
Password must be between 8 and 30 characters long, contain at least one uppercase letter, one lowercase letter, and one number.

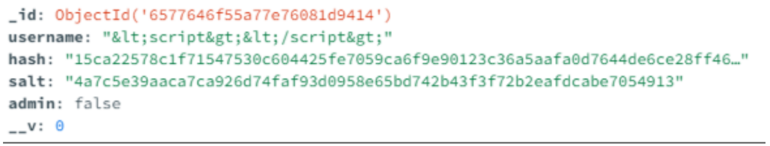

Register
Username must be at least 3 characters long


Login
Invalid credentials. Please try again.

Invalid credentials error is generic and does not give away excess details such as username was invalid or password was invalid.

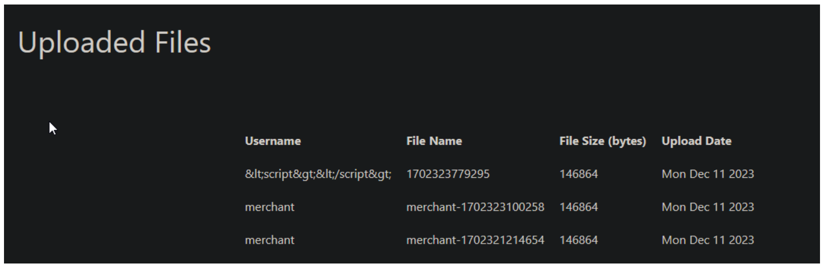
Testing input sanitisation





```
_id: ObjectId('6577646f55a77e76081d9414')
username: "&lt;script&gt;&lt;/script&gt;"
hash: "15ca22578c1f71547530c604425fe7059ca6f9e90123c36a5aafa0d7644de6ce28ff46…"
salt: "4a7c5e39aaca7ca926d74faf93d0958e65bd742b43f3f72b2eafdcabe7054913"
admin: false
__v: 0
```

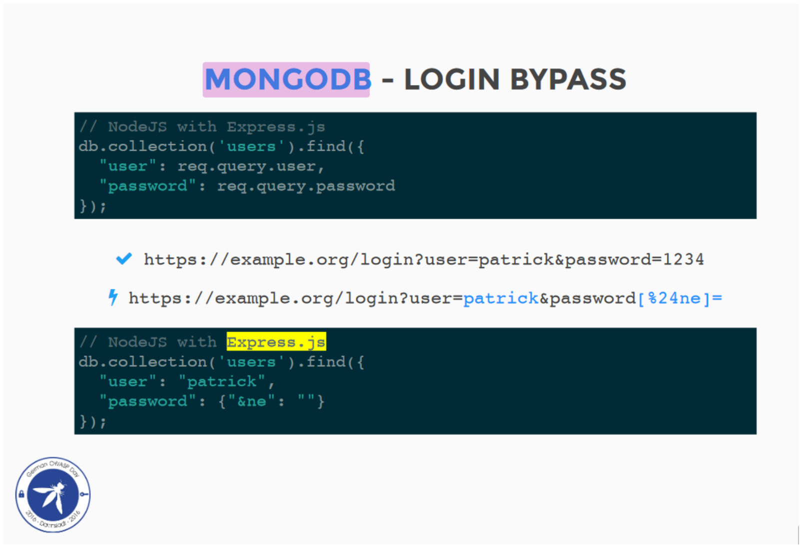
Dangerous username stored safely in the DB



Uploaded Files

Username	File Name	File Size (bytes)	Upload Date
<script></script>	1702323779295	146864	Mon Dec 11 2023
merchant	merchant-1702323100258	146864	Mon Dec 11 2023
merchant	merchant-1702321214654	146864	Mon Dec 11 2023

Testing NoSQL Injection



MONGODB - LOGIN BYPASS

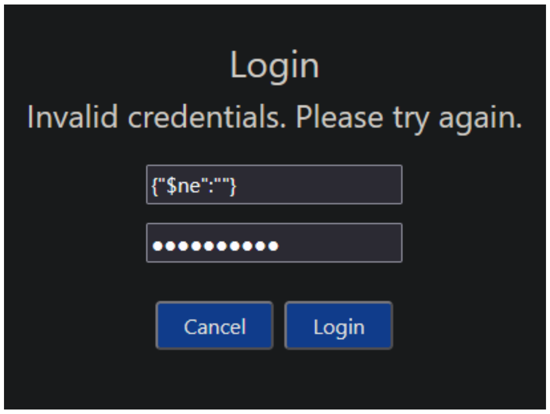
```
// NodeJS with Express.js
db.collection('users').find({
  "user": req.query.user,
  "password": req.query.password
});
```

✓ https://example.org/login?user=patrick&password=1234

ϟ https://example.org/login?user=patrick&password[%24ne]=

```
// NodeJS with Express.js
db.collection('users').find({
  "user": "patrick",
  "password": {"&ne": ""}
});
```

(Taken from OWASP guide on NoSQLInjection)

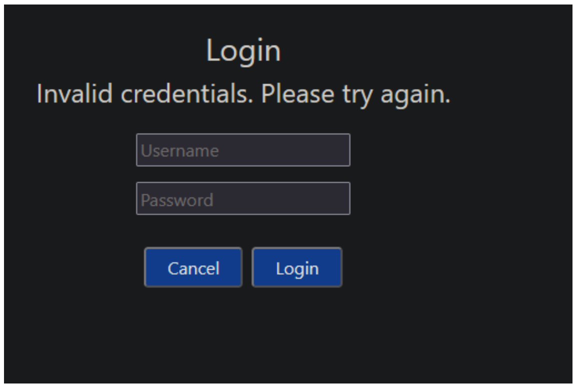


Login

Invalid credentials. Please try again.

{"$ne":""}

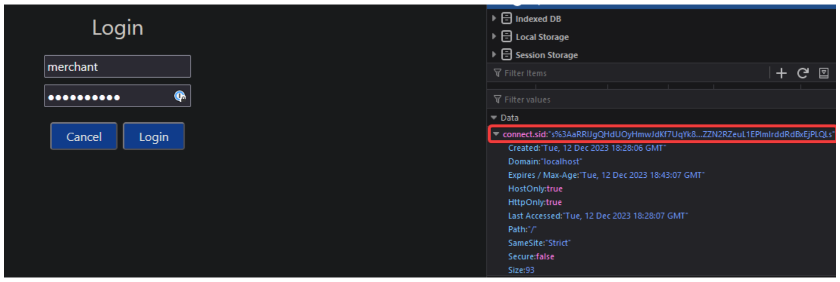
●●●●●●●●●●



Login

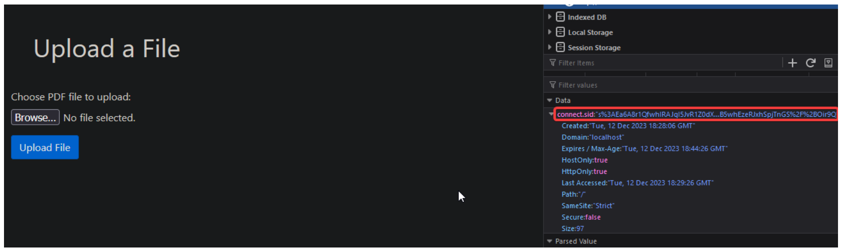
Invalid credentials. Please try again.

Session upgrade on auth



```
Data
connect.sid:"s%3AaRRIJgQHdUOyHmwJdKf7UqYk8...ZZN2RZeuL1EPlmIrddRdBxEjPLQLs"
  Created:"Tue, 12 Dec 2023 18:28:06 GMT"
  Domain:"localhost"
  Expires / Max-Age:"Tue, 12 Dec 2023 18:43:07 GMT"
  HostOnly:true
  HttpOnly:true
  Last Accessed:"Tue, 12 Dec 2023 18:28:07 GMT"
  Path:"/"
  SameSite:"Strict"
  Secure:false
  Size:93
```

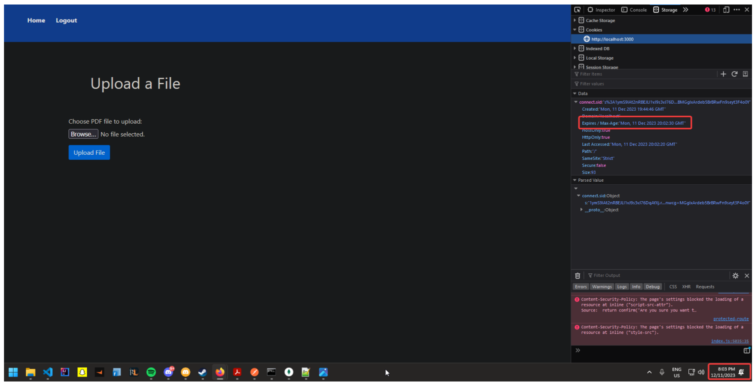
SID prior to login



```
Data
connect.sid:"s%3AEa6A8r1QfwhIRAJqI5JvR1Z0dX...B5whEzeRJxhSpjTnGS%2F%2BOir9Q
  Created:"Tue, 12 Dec 2023 18:28:06 GMT"
  Domain:"localhost"
  Expires / Max-Age:"Tue, 12 Dec 2023 18:44:26 GMT"
  HostOnly:true
  HttpOnly:true
  Last Accessed:"Tue, 12 Dec 2023 18:29:26 GMT"
  Path:"/"
  SameSite:"Strict"
  Secure:false
  Size:97
Parsed Value
```

New SID after successful login

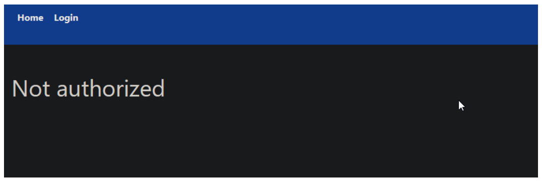
Session Timeout



```
Data
connect.sid:"s%3A1ymS9IAt2nRBEJLI1vJ9s3vJ76D...BMGglxArdeb5BrBRwFn9seyt3F4o0Y"
  Created:"Mon, 11 Dec 2023 19:44:46 GMT"
  Domain:"localhost"
  Expires / Max-Age:"Mon, 11 Dec 2023 20:02:30 GMT"
  HostOnly:true
  HttpOnly:true
  Last Accessed:"Mon, 11 Dec 2023 20:02:20 GMT"
  Path:"/"
  SameSite:"Strict"
  Secure:false
  Size:93
Parsed Value
  connect.sid:Object
    s:"1ymS9IAt2nRBEJLI1vJ9s3vJ76DqAYXj.r...nwcg+MGglxArdeb5BrBRwFn9seyt3F4o0Y"
  __proto__:Object
```

```
Content-Security-Policy: The page's settings blocked the loading of a
resource at inline ("script-src-attr").
Source:  return confirm('Are you sure you want t…
                                                   protected-route
Content-Security-Policy: The page's settings blocked the loading of a
resource at inline ("style-src").
                                                   index.js:5035:35
```

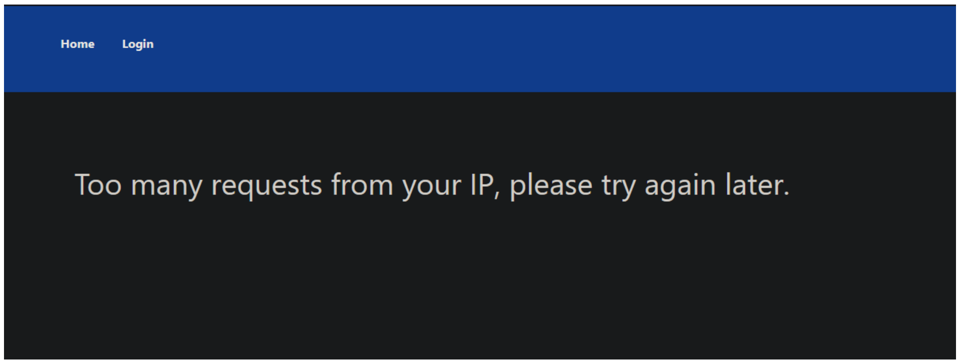
Waited for the session to timeout.



Not authorized

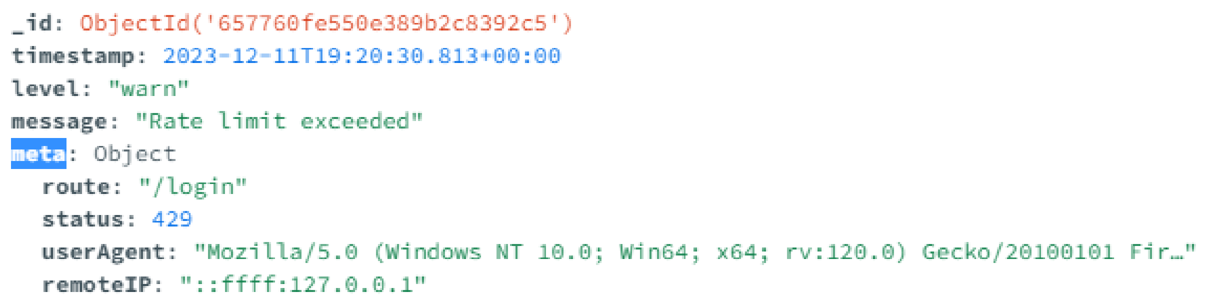
Make the request and am no longer auth'd.

Rate Limiting



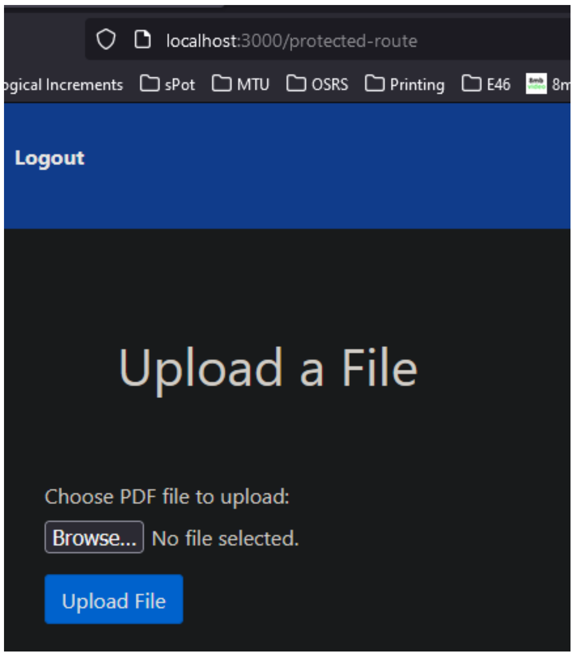
Too many requests from your IP, please try again later.

Attempting to make more requests than permitted (30) in 15 minutes.



```
_id: ObjectId('657760fe550e389b2c8392c5')
timestamp: 2023-12-11T19:20:30.813+00:00
level: "warn"
message: "Rate limit exceeded"
meta: Object
  route: "/login"
  status: 429
  userAgent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Fir…"
  remoteIP: "::ffff:127.0.0.1"
```

Testing RBAC as Merchant

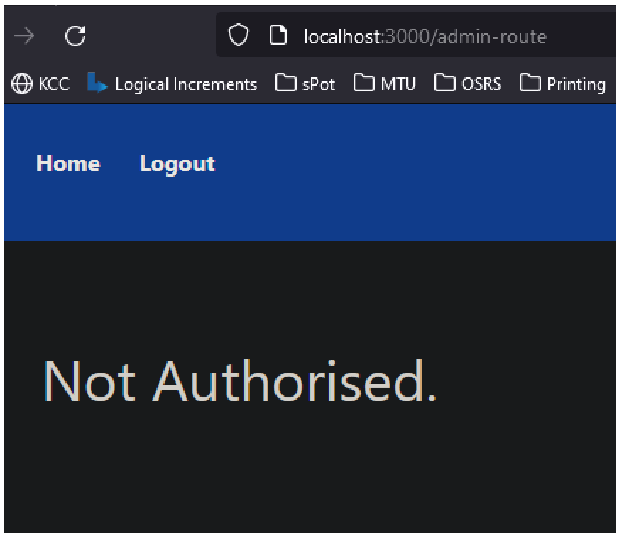


Upload a File

Choose PDF file to upload:

No file selected.

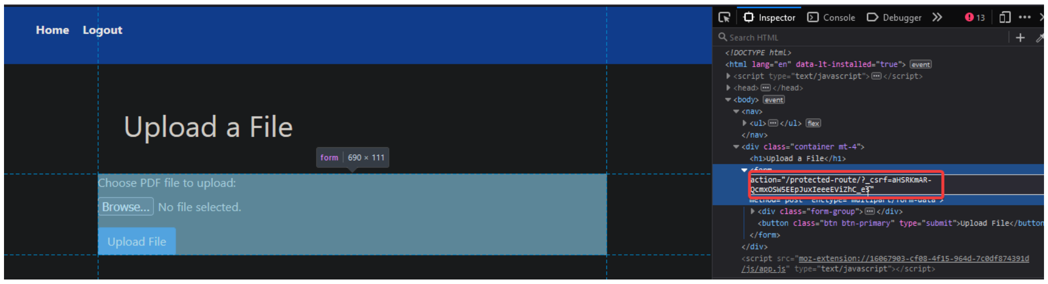
Can access merchant portal as merchant



Not Authorised.

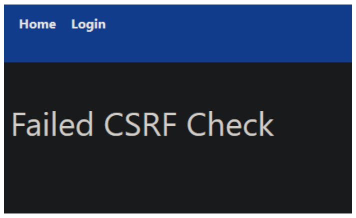
Cannot access admin portal as merchant

CSRF Validation



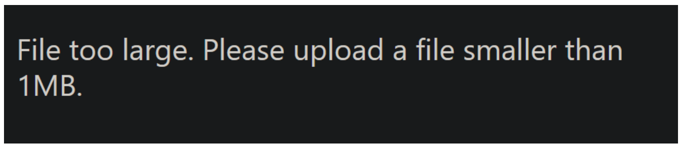
```
<!DOCTYPE html>
<html lang="en" data-lt-installed="true"> event
  <script type="text/javascript"> ··· </script>
  <head> ··· </head>
  <body> event
    <nav>
      <ul> ··· </ul> flex
    </nav>
    <div class="container mt-4">
      <h1>Upload a File</h1>
      <form
       action="/protected-route/?_csrf=aHSRKmAR-
       QcmxOSW5EEpJuxIeeeEViZhC_e$"
       method="post" enctype="multipart/form-data">
        <div class="form-group"> ··· </div>
        <button class="btn btn-primary" type="submit">Upload File</button
      </form>
    </div>
    <script src="moz-extension://16067903-cf08-4f15-964d-7c0df874391d
    /js/app.js" type="text/javascript"></script>
```

Manually invalidate CSRF token in source



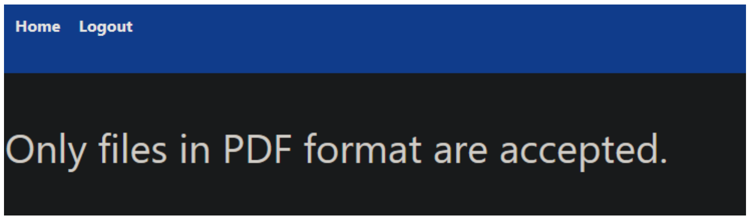
Failed CSRF Check

File validation



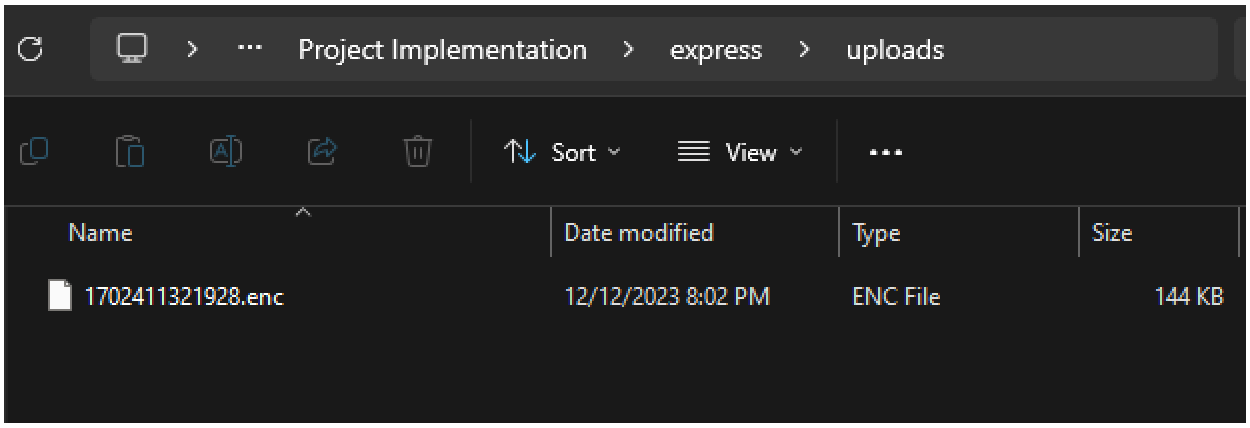
File too large. Please upload a file smaller than 1MB.

Uploading file larger than allowed



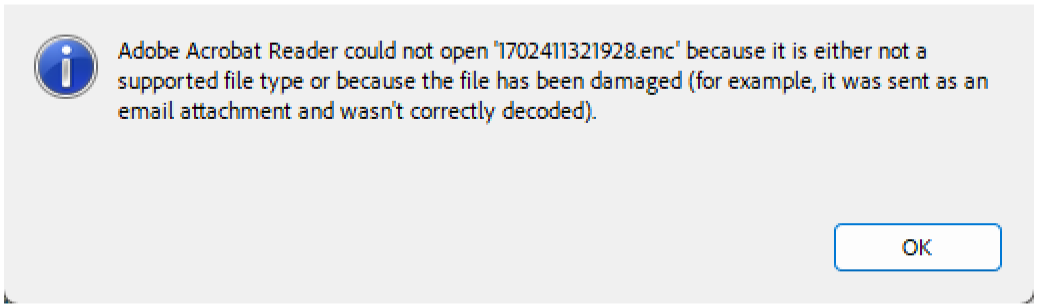
Only files in PDF format are accepted.

Testing file encryption



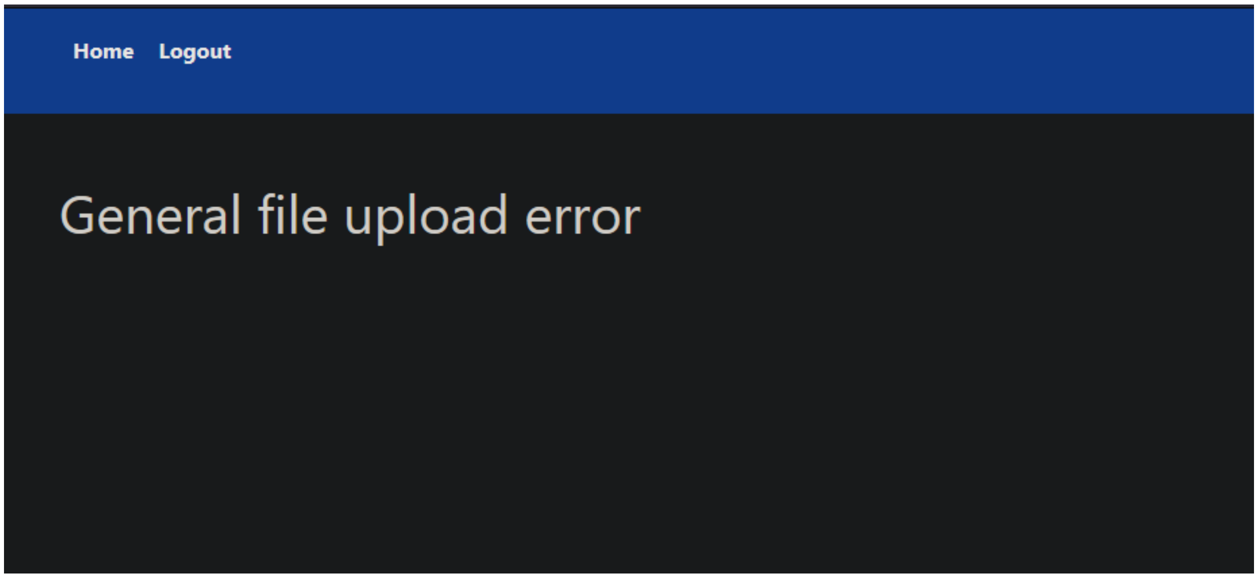
Name	Date modified	Type	Size
1702411321928.enc	12/12/2023 8:02 PM	ENC File	144 KB

Wrote file to /uploads temporarily to confirm the file is stored encrypted on the DB



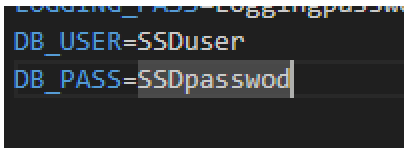
Adobe Acrobat Reader could not open '1702411321928.enc' because it is either not a supported file type or because the file has been damaged (for example, it was sent as an email attachment and wasn't correctly decoded).

Error messages have been obscured to protect application architecture



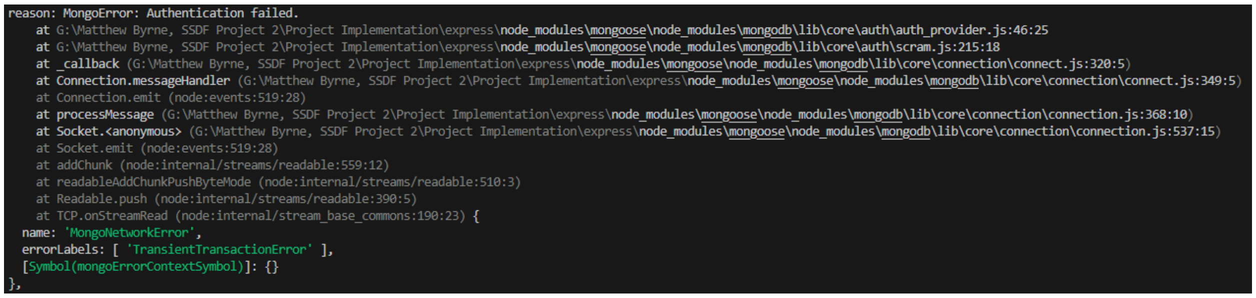
General file upload error

DB Auth



```
DB_USER=SSDuser
DB_PASS=SSDpasswod
```

Adjust password

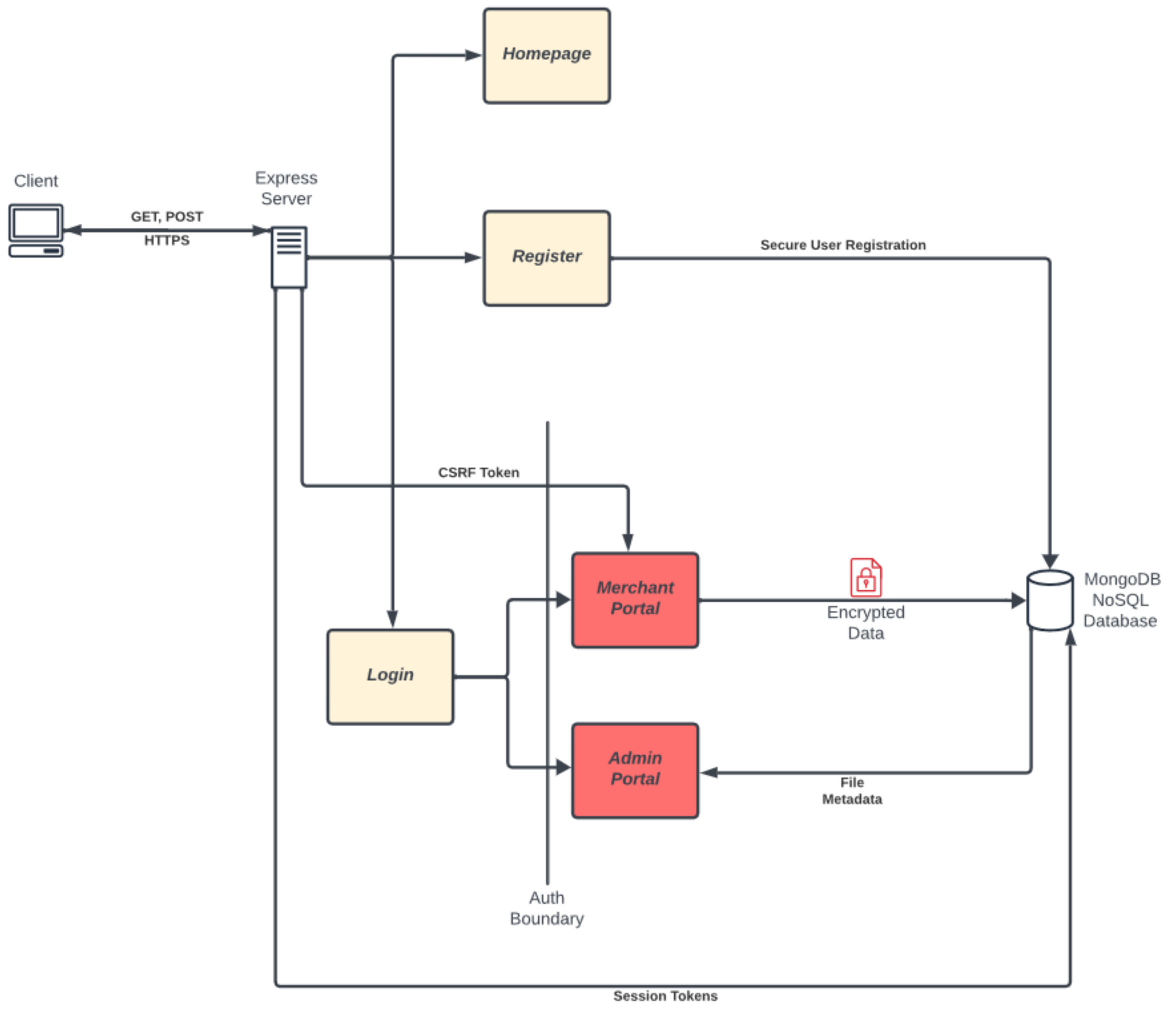


```
reason: MongoError: Authentication failed.
    at G:\Matthew Byrne, SSDF Project 2\Project Implementation\express\node_modules\mongoose\node_modules\mongodb\lib\core\auth\auth_provider.js:46:25
    at G:\Matthew Byrne, SSDF Project 2\Project Implementation\express\node_modules\mongoose\node_modules\mongodb\lib\core\auth\scram.js:215:18
    at _callback (G:\Matthew Byrne, SSDF Project 2\Project Implementation\express\node_modules\mongoose\node_modules\mongodb\lib\core\connection\connect.js:320:5)
    at Connection.messageHandler (G:\Matthew Byrne, SSDF Project 2\Project Implementation\express\node_modules\mongoose\node_modules\mongodb\lib\core\connection\connect.js:349:5)
    at Connection.emit (node:events:519:28)
    at processMessage (G:\Matthew Byrne, SSDF Project 2\Project Implementation\express\node_modules\mongoose\node_modules\mongodb\lib\core\connection\connection.js:368:10)
    at Socket.<anonymous> (G:\Matthew Byrne, SSDF Project 2\Project Implementation\express\node_modules\mongoose\node_modules\mongodb\lib\core\connection\connection.js:537:15)
    at Socket.emit (node:events:519:28)
    at addChunk (node:internal/streams/readable:559:12)
    at readableAddChunkPushByteMode (node:internal/streams/readable:510:3)
    at Readable.push (node:internal/streams/readable:390:5)
    at TCP.onStreamRead (node:internal/stream_base_commons:190:23) {
  name: 'MongoNetworkError',
  errorLabels: [ 'TransientTransactionError' ],
  [Symbol(mongoErrorContextSymbol)]: {}
},
```

Burp Suite header configuration

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:3000
Vary: Origin
Access-Control-Allow-Credentials: true
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 99
Date: Mon, 11 Dec 2023 23:29:01 GMT
X-RateLimit-Reset: 1702338242
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Content-Security-Policy: default-src 'self';script-src 'self';style-src 'self' https://stackpath.bootstrapcdn.com;img-src 'self';connect-src
'self';font-src 'self' https://stackpath.bootstrapcdn.com;object-src 'none';frame-ancestors 'none';upgrade-insecure-requests;base-uri
'self';form-action 'self';script-src-attr 'none'
Permissions-Policy: camera=(none), microphone=(none), geolocation=(none), notifications=(none), fullscreen=(self)
X-Frame-Options: DENY
Referrer-Policy: no-referrer
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Content-Type: text/html; charset=utf-8
Content-Length: 737
ETag: W/"2e1-VK3RNHLh7YBEvoKVcFPbp5m+Q8o"
Set-Cookie: connect.sid=s%3A5-MQkRj1uRIZnelKdgR3-taLlhI9E-5m.A4XFMX320ose6fyKnzE1ZlYPrVu33f2eQUkM3zJAv6o;
Domain=localhost; Path=/; Expires=Mon, 11 Dec 2023 23:44:01 GMT; HttpOnly; SameSite=Strict
Connection: close
```

Appendix A


Homepage
Client
GET, POST
HTTPS
Express Server
Register
Secure User Registration
CSRF Token
Merchant Portal
Encrypted Data
MongoDB NoSQL Database
Login
Admin Portal
File Metadata
Auth Boundary
Session Tokens

Appendix B

[Passport setup guide]
https://github.com/zachgoll/express-session-authentication-starter

[Multer CSRF Issue]
https://stackoverflow.com/questions/67453346/getting-forbiddenerror-invalid-csrf-token-with-multer-added-locally-to-image-up

[File Encryption code]
https://www.geeksforgeeks.org/node-js-crypto-createcipheriv-method/

[Helmet Config]
https://blog.logrocket.com/using-helmet-node-js-secure-application/#referrer-policy-header
https://www.npmjs.com/package/permissions-policy