Class 21: REST API Security

- Review: Query Parameters & Text Search
- API Security Best Practices
- Exam II Review
- Office Hours

Review

Review HTTP Query Parameters

A query parameter is a key-value pair that is appended to the end of a URI endpoint to provide additional information about the request.

The query parameters are separated from the URI endpoint by a `?` and multiple query parameters are separated by `&`.

Example: (2 parameters: `q` and `sort`)

```
GET /api/documents?q=cats&sort=asc
```

Review: Express.js: Query Parameters

Use the `req.query` object to access query parameters in an Express.js route handler.

```
GET /api/documents?q=cats&sort=asc
```

```
app.get('/api/documents', (req, res) => {
  const qParam = req.query["q"];
  const sortParam = req.query.sort;
})
```

Review: MongoDB Text Search

Text search in MongoDB requires a text index and the `$text` query operator.

Text Index

```
db.COLLECTION.createIndex({
  FIELD1: "text",
  FIELD2: "text",
  ...
});
```

Text Search Query

```
db.collection('documents').find({
  $text: { $search: "SEARCH TERMS" }
});
```

Review: REST API Sub-Resources

Using sub-resources ensures that the resource is being modified in a clear and atomic way.

```
{
  title: "Post Title",
  content: "Post Content",
  featured: false
}
```

To feature a post:

```
POST /api/posts/:postId/featured
```

To unfeature a post:

```
DELETE /api/posts/:postId/featured
```

Often we use `POST` (without request body) to affirmatively modify the sub-resource and `DELETE` to *undo* that change.

API Security

REST API Security Best Practices

- Keep all Node.js packages up to date (especially Express.js and MongoDB).
- Serve your API over HTTPS (not HTTP) using a TLS certificate.
- Never trust user input. Always validate and sanitize input on the server side.
- Employ HTTP security-related response headers.
- Reduce server software identifying information in HTTP response headers.
- Require authentication for privileged operations or private resources.
- Rate limit API requests to prevent abuse and denial-of-service attacks.

Node.js Package Security

Regularly update your Node.js packages to ensure you have the latest security patches and improvements.

Check for Updates

Check for **known** security vulnerabilities:

```
npm audit
```

Update vulnerable packages:

```
npm audit fix
```

Update All Packages

```
npm update
```

Force Update All Packages

```
npx npm-check-updates
```

Caution: This may introduce breaking changes.

Activity: Node.js Package Audit

1. Check to see if any of your project's dependencies have known security vulnerabilities using `npm audit`.
2. If there are any vulnerabilities, update the vulnerable packages using `npm audit fix` or `npm update`.

HTTPS & TLS Certificates

When deploying your API, serve the API over HTTPS (not HTTP) using a TLS certificate to encrypt data transmitted between the client and server.

Follow the instructions provided by your web host provider. (**Note:** Codespaces always serves over HTTPS; you need not worry about this when developing in Codespaces.)

Tip: Let's Encrypt is a free, automated, and open certificate authority that provides TLS certificates.

Never Trust User Input

Always validate and sanitize user input on the server side to prevent security vulnerabilities such as SQL injection, cross-site scripting (XSS), and other injection attacks.

`express-validator` is one possible tool to assist with input validation and sanitization in Express.js.

Example:

```
query('q').trim().notEmpty();
```

HTTP Security Headers

Employ HTTP security-related response headers to enhance the security.

Example Headers

- `Content-Security-Policy`
- `X-Content-Type-Options`
- `X-Frame-Options`
- `Strict-Transport-Security`
- `X-XSS-Protection`
- `Referrer-Policy`
- ...

Helmet.js

Helmet.js is Express.js middleware that sets HTTP security headers with sensible defaults.

```
import helmet from "helmet"

app.use(helmet())
```

Activity: Helmet.js

1. Install the `helmet` package in the API server project.

```
cd server
npm install helmet
```

2. Import and use the `helmet` middleware in the Express.js server.

```
import helmet from "helmet"
```

3. Enable the helmet middleware in the Express.js server.

```
app.use(helmet())
```

Reduce Server Fingerprinting

If an attacker is able to identify the Node.js packages (and other software) that your API server is using, they can easily exploit known vulnerabilities in that software.

Don't advertise that you're using Express.js by removing the `X-Powered-By` header:

```
app.disable('x-powered-by');
```

Activity: Reduce Server Fingerprinting

1. Observe the HTTP response headers from any endpoint in the API.

   You should see the `X-Powered-By` header advertising that you're using Express.js.

2. Disable the `X-Powered-By` header in the Express.js server.

   ```
   app.disable('x-powered-by');
   ```

3. Re-run the endpoint and observe the HTTP response headers again.

Require Authentication

Require authentication for privileged operations or private resources to ensure that only authorized users can access or modify sensitive data.

Principle of Least Privilege: Users should only have the minimum level of access necessary to perform their tasks.

Ask Yourself: Do all API endpoints need to be publicly accessible, or are there some that should be protected by authentication?

Activity: Principle of Least Privilege

Principle of Least Privilege: Users should only have the minimum level of access necessary to perform their tasks.

Working with your peers (2-4), update the design of the `produce` REST API on your **handout** employing the principle of least privilege.

Tip: Is the data private or public? What operations should be publicly accessible without authentication, and which operations should require authentication?

Express.js Authentication

This is out of scope for this course.

For further information, explore authentication in Express.js using JSON Web Tokens (JWTs) with the `express-jwt` package.

Note: There are security considerations when using MongoDB to store passwords.

Rate Limiting

Rate limit API requests to prevent abuse and denial-of-service attacks.

Denial-of-Service (DoS) Attack: An attack that aims to make a service unavailable by overwhelming it with a flood of requests.

Activity: Express.js Rate Limiting

Working with your peers (2-4), implement rate limiting in the Express.js server using the `express-rate-limit` package.

Tip: Follow the reference documentation, rather than trust generative AI (this is especially true for security-related concerns).

Exam II

Exam II

Exam 2 is this upcoming **Wednesday, April 15th, 2026**, during our **regularly scheduled class time (10:10am - 11:25am; 75 minutes)**.

Paper-based, closed-book, no computers or electronic devices allowed. About **10 questions**, same format as part I of the practice problem workshops.

Exam II: Study Guide

- **Class 8:** Lifting State Up
- **Class 10:** Handler Props & Controlled Inputs
- **Class 11:** Objects in State
- **Class 12:** Rendering Lists
- **Class 13:** Document Databases
- **Class 14:** Database CRUD Operations
- **Class 15:** Interactive Web Application Design Patterns
- **Class 16:** Utility First CSS Frameworks
- **Class 17:** HTTP Servers
- **Class 18:** Express.js & MongoDB

Q & A: Exam II

The exam will cover all material covered in:

- classes 8-18,
- project 1, milestones 3
- project 1, final milestone,
- project 2, milestone 1,
- project 2, milestone 2,
- homework 3,
- practice problem workshops 5-9.

Office Hours

What's Next

Today: Project 2, Milestone 3 Due

Wednesday: Exam II