**Final Exam Review**

This is an outline for the final exam. Please review slides as well because all lecture material is fair game for the final exam.

\*\*If you have an ADS form and intend on using your accommodation, you must schedule your exam with the [testing center](https://www.towson.edu/accessibility-disability-services/testing.html). Due to scheduling conflicts, you will not be able to get accommodations if the exam is taken in the classroom at the regular time. Please schedule this a week in advance to avoid any issues.

1. **NodeMailer:**

* **Nodemailer is a popular Node.js module for sending email from your server. It lets you configure transport and then send messages in a few lines.**

**Code Example:**

**// install: npm install nodemailer**

**const nodemailer = require('nodemailer');**

**async function sendTestEmail() {**

**// 1. Create a transporter. For real apps, secure your credentials!**

**let transporter = nodemailer.createTransport({**

**host: "smtp.example.com", // your SMTP server**

**port: 587, // usually 587 or 465**

**secure: false, // true for 465, false for other ports**

**auth: {**

**user: "your\_user", // SMTP user**

**pass: "your\_pass", // SMTP password**

**},**

**});**

**// 2. Define the email options**

**let mailOptions = {**

**from: '"Sender Name" <sender@example.com>',**

**to: "recipient@example.com",**

**subject: "Hello from NodeMailer",**

**text: "This is a plain-text body",**

**html: "<b>This is HTML body</b>",**

**};**

**// 3. Send it**

**let info = await transporter.sendMail(mailOptions);**

**console.log("Message sent: %s", info.messageId);**

**}**

**sendTestEmail().catch(console.error);**

* 1. Be able to send a simple email using NodeMailer
* **Check above**
  1. Know the three main protocols and their functionality

**1. POP3 (Post Office Protocol v3)**

**How it works**

**Connect to the mail server.**

**Download all new messages to your device (e.g. your desktop mail client).**

**By default, delete those messages from the server after download.**

**Disconnect.**

**Key characteristics**

**One-way transfer: server → client only.**

**Stateless on the server: once downloaded (and deleted), the server “forgets” about those messages.**

**Simple protocol, minimal server-side storage or syncing.**

**Example scenario**

**You check mail on your laptop at home. POP3 downloads 20 new messages and removes them from the server.**

**Later, on your phone, you try to check mail again. Since the server copy is gone, your phone sees zero new messages—you only have those 20 on your laptop.**

**Use POP3 when:**

**You have one primary device for mail.**

**You want to archive everything locally and don’t need to keep mail on the server forever.**

**Server storage is limited and you want to clear it out automatically.**

**2. IMAP (Internet Message Access Protocol)**

**How it works**

**Connect to the mail server.**

**List folders (Inbox, Sent, Drafts, etc.).**

**Fetch only the headers or bodies of new messages on demand.**

**Mark messages as read/unread, flagged, moved, or deleted—and these state changes live on the server.**

**Disconnect (or stay connected to receive updates).**

**Key characteristics**

**Two-way sync: any action you take (read, delete, move) is reflected on the server and thus on every device.**

**Stateful: server keeps track of which messages are read, flagged, in which folder, etc.**

**Selective download: you can fetch just headers first, then download full bodies or attachments as needed.**

**Example scenario**

**You read an email on your phone and mark it “read.”**

**Later on your laptop, you open your mail app and see that same message already marked “read.”**

**You move it into a “Project X” folder on your laptop—and on your tablet it’s also in “Project X.”**

**Delete one on your desktop, and it’s gone from all devices.**

**Use IMAP when:**

**You check mail from multiple devices (phone, laptop, tablet).**

**You need to keep your inbox/folders organized consistently everywhere.**

**Server storage is sufficient, and you want to offload local archiving.**

**2. IMAP (Internet Message Access Protocol)**

**How it works**

**Connect to the mail server.**

**List folders (Inbox, Sent, Drafts, etc.).**

**Fetch only the headers or bodies of new messages on demand.**

**Mark messages as read/unread, flagged, moved, or deleted—and these state changes live on the server.**

**Disconnect (or stay connected to receive updates).**

**Key characteristics**

**Two-way sync: any action you take (read, delete, move) is reflected on the server and thus on every device.**

**Stateful: server keeps track of which messages are read, flagged, in which folder, etc.**

**Selective download: you can fetch just headers first, then download full bodies or attachments as needed.**

**Example scenario**

**You read an email on your phone and mark it “read.”**

**Later on your laptop, you open your mail app and see that same message already marked “read.”**

**You move it into a “Project X” folder on your laptop—and on your tablet it’s also in “Project X.”**

**Delete one on your desktop, and it’s gone from all devices.**

**Use IMAP when:**

**You check mail from multiple devices (phone, laptop, tablet).**

**You need to keep your inbox/folders organized consistently everywhere.**

**Server storage is sufficient, and you want to offload local archiving.**

1. **Cheerio**
   1. If given an HTML page, be able to parse it and return some result

* **Cheerio is a jQuery like library for server-side HTML parsing and manupilating in Node.js. It loads HTML into a lightweight DOM you can querry with familiar CSS selectors just like jQuery but it runs in Node, without a browser.**
* **Example cheerio**

1. **Package.json**
   1. The advantages of using it
2. **React**
   1. Advantages of using React
      1. Virtual DOM
   2. Differences between JSX and JavaScript
      1. Be able to write an interpret JSX code
   3. Be able to write and interpret actual React code
      1. Components
         1. Why components are important
      2. Props
      3. State
      4. Conditional Rendering
   4. React Native
      1. How good mobile hybrid solutions work vs bad ones
      2. The Message Queue
         1. Its importance and functionality
3. **Docker**
   1. Why is Docker useful when dealing with full-stack applications?
      1. Team collaboration
   2. Explain what a Docker Registry is
      1. Be able to list two
   3. Best version of Linux for simple Node.js app containerization
   4. Very basic commands of a Dockerfile
4. **GraphQL**
   1. How is GraphQL different than a RESTful API
      1. The advantages of using GraphQL
   2. How can GraphQL and a pre-existing RESTful API work together?
   3. Setting up a simple Schema given a scenario
      1. Schemas with relations
   4. Queries
      1. Be able to make Queries given a Schema
         1. Nested Queries for specific data as well
   5. Mutations and how they work
   6. Subscriptions and how they work
   7. Setup a simple GraphQL endpoint