Class 17: HTTP Servers & Express.js

- Review: Last Class
- HTTP: Requests & Responses
- Express.js
- RESTful APIs

Review

Review: Design System

A design system is a complete set of standards to manage design at scale by using reusable components and patterns while creating a shared language and visual consistency across different pages and channels. Typically, this includes: a style guide, a component library, and a pattern library.

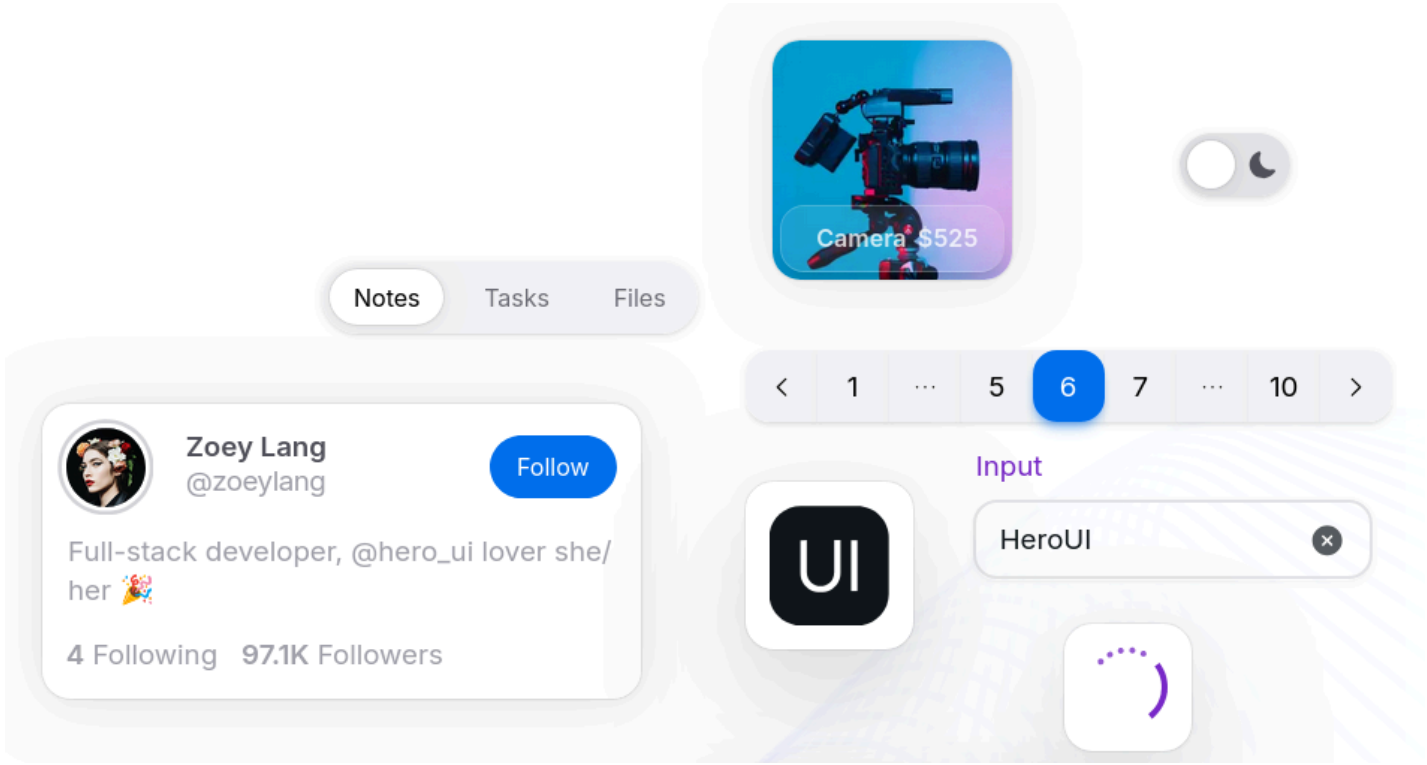
Review: Style Guide

A style guide is a document that provides specific implementation guidelines, visual reference, and design principles for the visual design of a product.

Typically, a style guide focuses on branding: colors, typography, etc. However, some style guides also offer guidance on content (i.e. tone of voice).

Review: Component Library

A component library is a collection of reusable components that are designed to be used in a design system.



Review: Pattern Library

A pattern library is a collection of reusable design patterns that are designed to be used in a design system.

Examples:

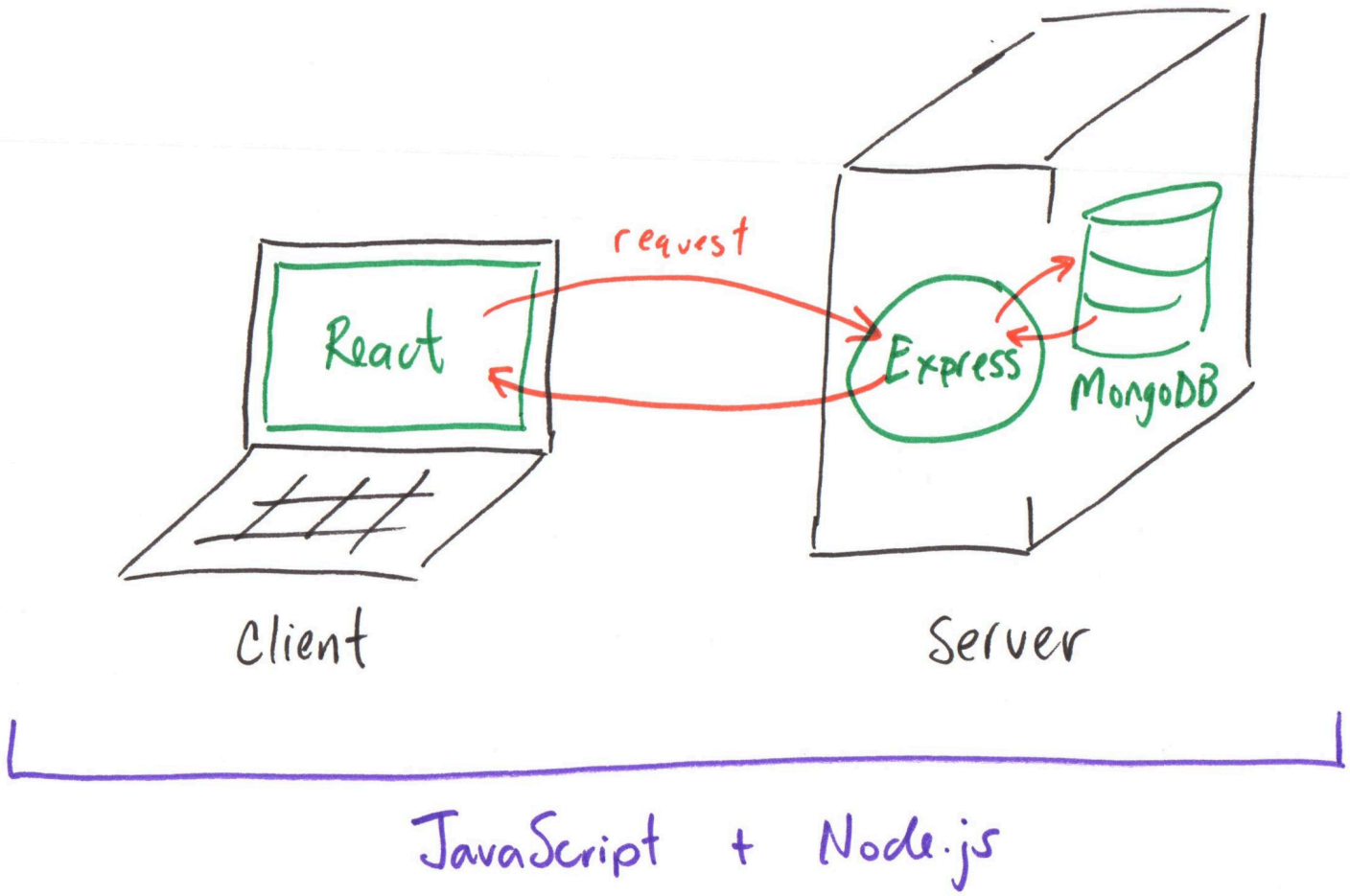
- Main content as cards
- Search bar visible in header
- Filter chips/pills displayed above content

Review: Utility-First CSS – Tailwind CSS

Tailwind CSS is a popular utility-first CSS framework that provides a large set of utility classes for styling web applications.

```
export default function Button({ color = 'default', children }) {
  const base = 'px-4 py-2 font-semibold rounded cursor-pointer';
  const styles = {
    default: 'bg-slate-100 text-black hover:bg-slate-500 hover:text-white',
    primary: 'bg-blue-400 text-white hover:bg-blue-600',
  };
  return <button className={`${base} ${styles[color]}`}>{children}</button>;
}
```

Review: MERN Stack



HTTP

HTTP

Hyper**t**ext **T**ransfer **P**rotocol

An application-layer protocol for transmitting resources between client and web servers.

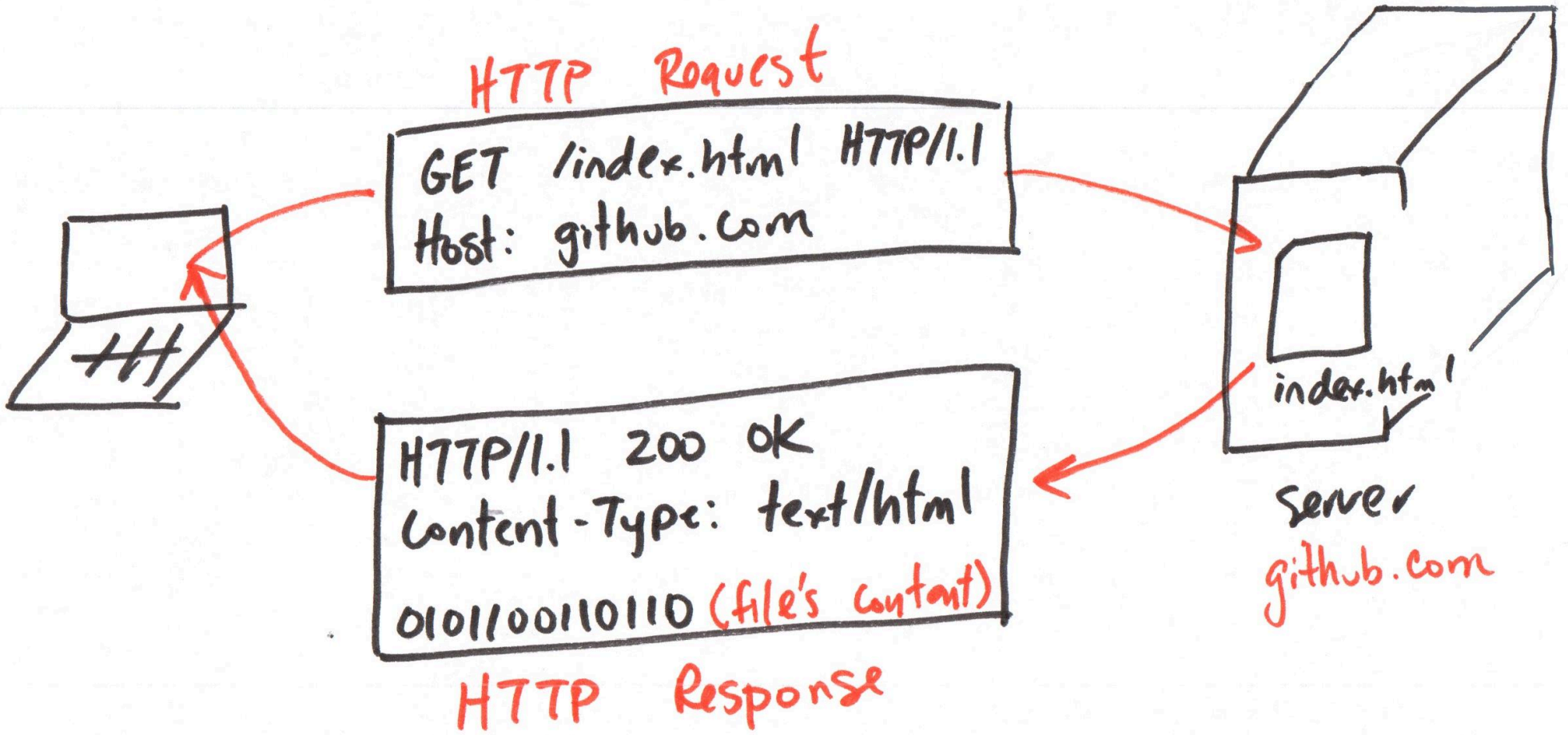
Activity: Dev Tools Network Tab

Together, open the **Network** tab in your browser's developer tools and then open any website (static website preferred).

Explore the **requests** and **responses**.

- How many requests are needed to load the web page?
- What resources are being requested?
- What information is included in the request and response headers?

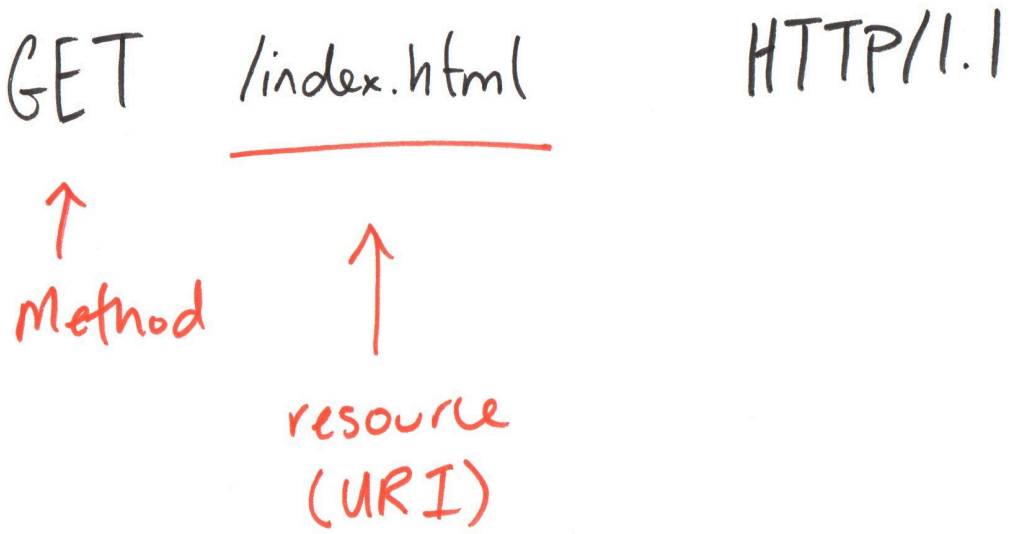
HTTP Request & Response



HTTP Request Format

```
METHOD /RESOURCE HTTP/1.1
```

Example:



HTTP Request Methods

GET: Retrieve a resource

POST: Create a new resource

(Other methods include PUT, PATCH, DELETE, etc.)

HTTP Response Format

```
HTTP/1.1 STATUS_CODE STATUS_MESSAGE
```

Example: Resource found

```
HTTP/1.1 200 OK

[contents of requested resource]
```

Example: Resource not found

```
HTTP/1.1 404 Not Found
```

Activity: Static Website HTTP Server

Working with your peers (2-4), write a pseudocode algorithm to **serve** the resources for a **static website** using HTTP.

You only need to serve two files: `/index.html` and `/styles/site.css`

Request:

```
GET /index.html HTTP/1.1
```

Response:

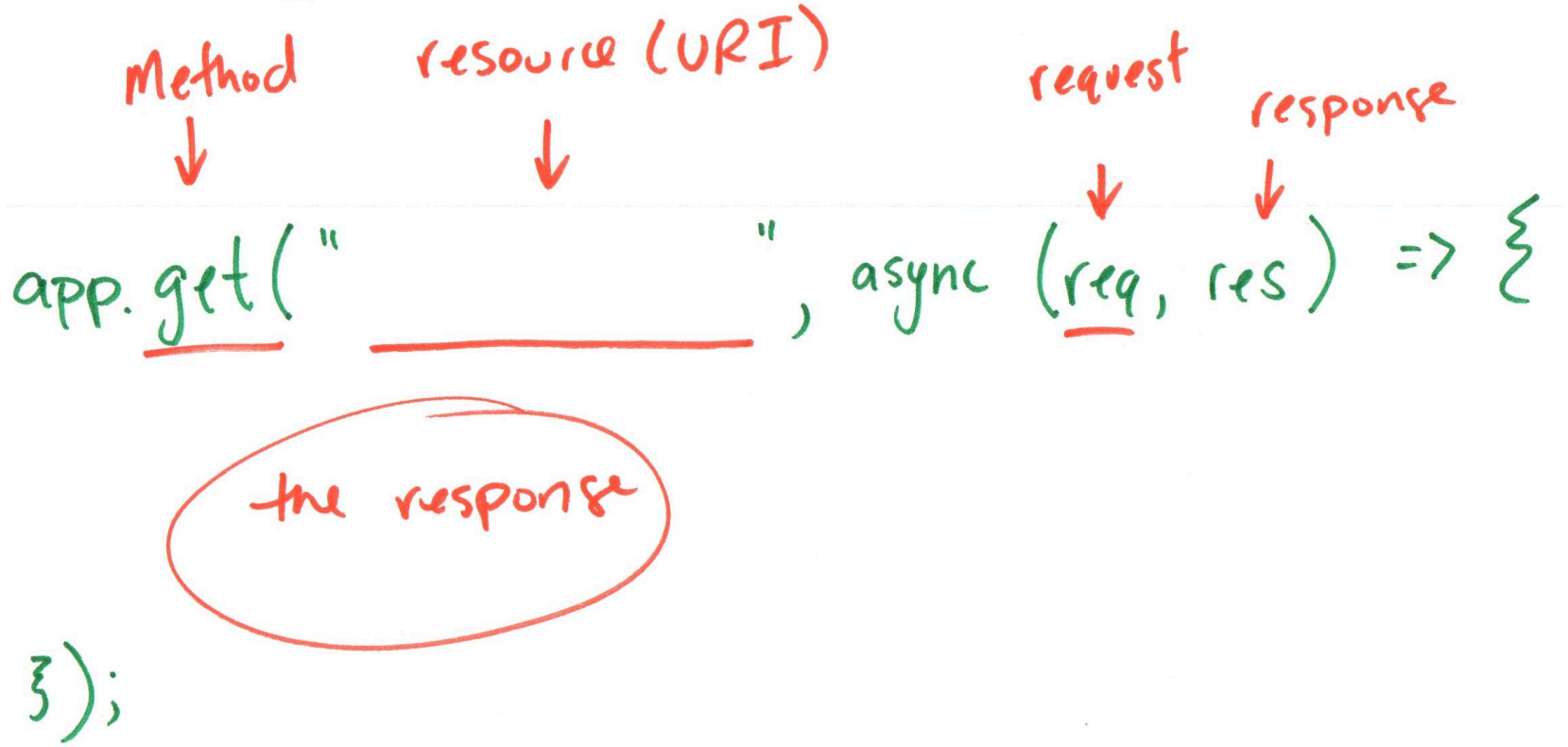
```
HTTP/1.1 200 OK

[contents of index.html]
```

Express.js

Express.js

A backend web application framework for building web servers and RESTful APIs in Node.js.

Express.js: Responding to HTTP Requests



Example: Respond to Request for `index.html`

```
app.get('/index.html', async (request, response) => {
  res.sendFile('index.html')
})
```

Activity: Static Website HTTP Server

Together, write the code for the Express.js server that serves the resources for a static website.

- `index.html`
- `styles/site.css`

RESTful APIs

REST (or RESTful) API

An **A**pplication **P**rogramming **I**nterface that adheres to the constraints of REST architecture to support communication between client and server to exchange resources, like data and media.

A **RESTful API** supports the **creation**, **retrieval**, **updating**, and **deletion** of resources (CRUD operations) using HTTP methods.

Example: RESTful API

Request:

```
GET /api/documents HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK

[
  { id: 1, title: 'Document 1', content: '...' },
  { id: 2, title: 'Document 2', content: '...' },
  ...
]
```

Activity: Express.js RESTful API

Together, write the code for an Express.js server that implements a RESTful API for **retrieving** a collection of documents.

```
// GET: /api/documents
app.get('/api/documents', async (req, res) => {
  const documents = [
    { name: "Apple", category: "fruit", ... },
    { name: "Banana", category: "fruit", ...}
  ]

  res.status(200).json(documents)
})
```

REST Client

A tool for exploring, testing, and debugging REST APIs.

A popular REST client is Postman. However, we'll be using `httpbook` because it works with with Codespaces.

Summary

Summary

- HTTP is the protocol for transmitting resources between client and web servers.
- An HTTP request includes a method (e.g. GET, POST) and a URI/path
- An HTTP response includes a status code (e.g. 200, 404) and the contents of the requested resource (if found)
- Express.js is a backend web application framework for building web servers and RESTful APIs in Node.js
- A RESTful API supports the creation, retrieval, updating, and deletion of resources (CRUD operations) using HTTP methods
- REST clients are tools for exploring, testing, and debugging REST APIs

What's Next

Today: Project 2, Milestone 2 Due

Wednesday: RESTful API Design

Friday: Practice Problem Workshop