Check-In Code: wooooohack

Hack School 4: APIs and Databases

ACM at UCSD

wooohack

Check-In
acmucsd.com

Check-In Code: wooohack

ACM at UCSD

acmurl.com/hackschool4-checkin

Check-In Code: woooohack

ACM at UCSD

today's agenda

1 HTTP & APIs

2 Creating API Routes

3 What are Databases?

4 Creating our Database with MongoDB

Check-In Code: wooooohack

Check-In Code: wooohack

How the Internet Works

- The internet is just a really long wire
- Computers interact with each other by sending data through this wire
- Web <u>clients</u> interact with web <u>servers</u> by requesting data through this wire
  - Client - The user (generally a web browser)
  - Server - Any computer that serves/provides data to a client
- Clients and servers communicate via <u>HTTP</u> through this wire
  - HTTP - Hypertext Transfer Protocol - a “language” that both clients and servers can speak

ACM at UCSD

Check-In Code: wooooohack

HTTP Requests and Responses

- Clients request for data from a server through an HTTP request
    - Can request different functionality through request methods
    - GET, POST, PUT, DELETE
        - CRUD – Read, Create, Update, Delete
- Servers respond to client with data through an HTTP response
    - Can respond with status codes to signify the response state
    - 200 - OK, 404 - Not Found, 500 - Internal Server Error
    - More information on status codes here

ACM at UCSD

HTTP Requests in Action

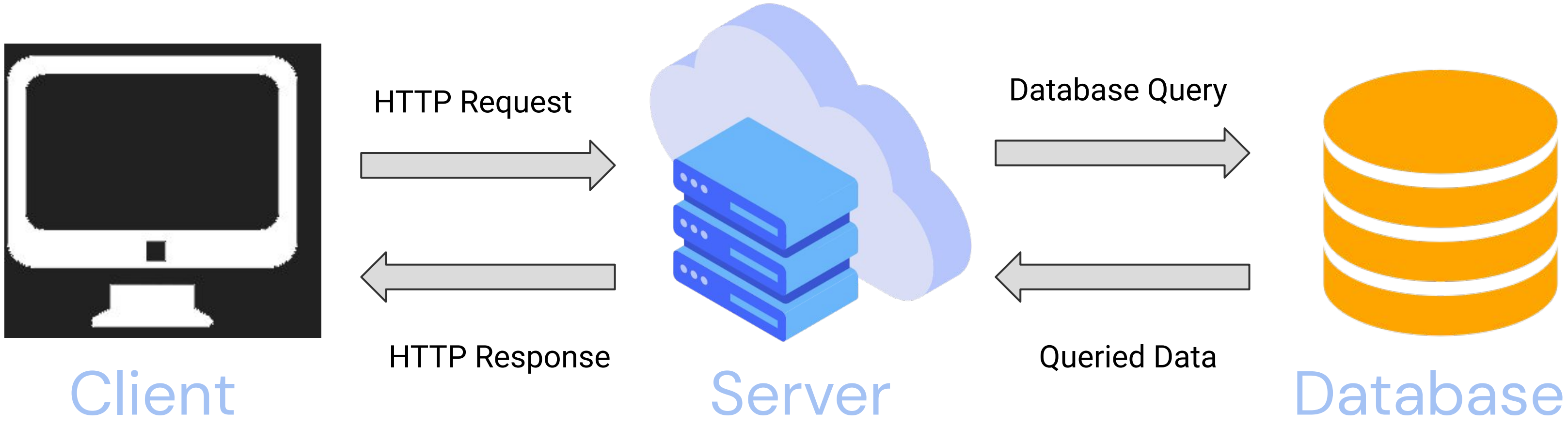
Let’s see HTTP requests in action on our website, acmucsd.com

Tips:

- Use Ctrl + Shift + J (Windows) or Cmd + Option + J (Mac) to open Chrome Dev Tools
- Hop to “Network” tab and refresh page to see all requests and responses

Check-In Code: wooooohack

A Standard Web App Architecture



ACM at UCSD

Check-In Code: wooooohack

APIs – The Middleman

- API – Application Programming Interface
  - A “toolkit” available for a client to interact with
  - Ex: Spotify API to integrate Spotify into your app
- Can be a toolkit of functions, or a toolkit of API routes
  - API route - an route on a server accessible via HTTP requests
- REST API – a collection of API routes

ACM at UCSD

API Route Syntax

```
http://localhost:5000/api/purchases
```

Web Server host and port

API Route

<u>Example:</u>

Sending a GET request to this route would get all purchases

Sending a POST request with a request body would create a purchase

Check-In Code: woooohack

Check-In Code: wooooohack

Example API + Documentation

- Bored API
- Spotify API

ACM at UCSD

Check-In Code: wooooohack

Express

- A web application framework for Node.js that lets us
    - Start a server locally
    - Create our own API routes
- Abstracts away all the complexity of serving a web application

```
const express = require('express');
const server = express();
server.listen(5000);
```

ACM at UCSD

Check-In Code: wooooohack

Let’s Create our Backend

Open up Terminal or Git Bash at your project directory

- We are going to:
    - Initialize a Node.js application with some NPM dependencies
        - express, nodemon, dotenv, cors
    - Import Express into our app
    - Create an Express server locally

ACM at UCSD

Creating an Express API Route

```
const router = express.Router();

router.get('/purchases, async (req, res) => {

    // fetch purchases from database

    res.status(200).json({ purchase });

});
```

This code:

- Creates a "/purchases" route that accepts GET requests
- Exposes a function with "req" and "res" parameters to use as request and response objects
- Returns purchase data with a 200 – OK status code

Check-In Code: woooohack

ACM at UCSD

Check-In Code: wooooohack

Let’s Create our own API Routes

- Let’s create an API route:
  - GET /purchases
    - Retrieves all purchases from our database
    - Purchases will be “hardcoded” for now since no database…yet!

```
router.get('/purchases, async (req, res) => {
    // fetch purchases from database
    res.status(200).json({ purchase });
});
```

ACM at UCSD

POST Requests

- Can send POST requests along with JSON data to a server to store/update it in the database.
- We will send our purchase name, description, …

```
router.post('/purchase, async (req, res) => {
    const purchase = req.body.purchase;
    const name = purchase.name;
    const { description } = purchase; // object destructuring
    // rest of code
});
```

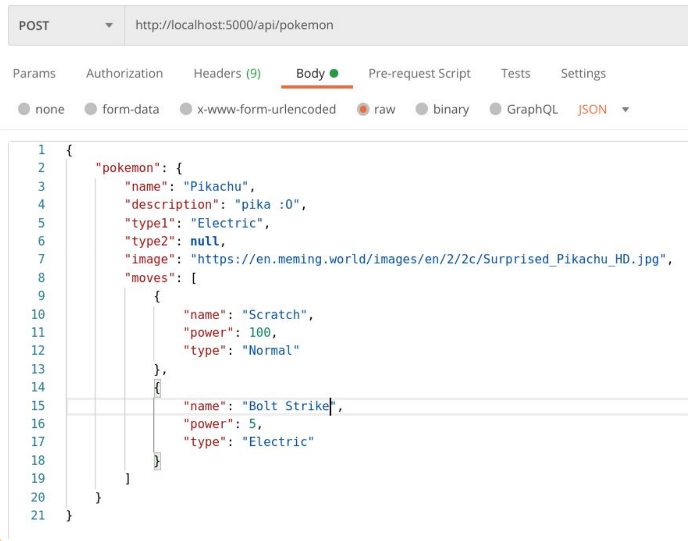
Check-In Code: wooohack

ACM at UCSD

Check-In Code: wooooohack

Postman

- Send requests to servers easily using Postman
  - https://www.postman.com/
- We can send requests to POST /purchase with a request body



http://localhost:5000/api/pokemon

```
{
    "pokemon": {
        "name": "Pikachu",
        "description": "pika :O",
        "type1": "Electric",
        "type2": null,
        "image": "https://en.meming.world/images/en/2/2c/Surprised_Pikachu_HD.jpg",
        "moves": [
            {
                "name": "Scratch",
                "power": 100,
                "type": "Normal"
            },
            {
                "name": "Bolt Strike",
                "power": 5,
                "type": "Electric"
            }
        ]
    }
}
```

ACM at UCSD

Check-In Code: wooooohack

Activity: Create POST /purchase

1. Create a route called POST /purchase
2. Return an error message response and proper status code if the purchase does not have a name, description, cost, or method.
3. If no errors occur:
    a. Return a JSON response with status OK!

That’s it for Express! Now time for our database o.O

What is a database?

Check-In Code: wooooohack

ACM at UCSD

Database –

An organized collection of data or information

Why do we need a database?

- **Store** and **persist** information between sessions/visits to website
- **Organize** our data in a logical way
- In the context of our Purchase tracker, this lets us **store** and **view** each Purchase we create!

Check-In Code: woooohack

ACM at UCSD

Organization of Data

Check-In Code: wooooohack

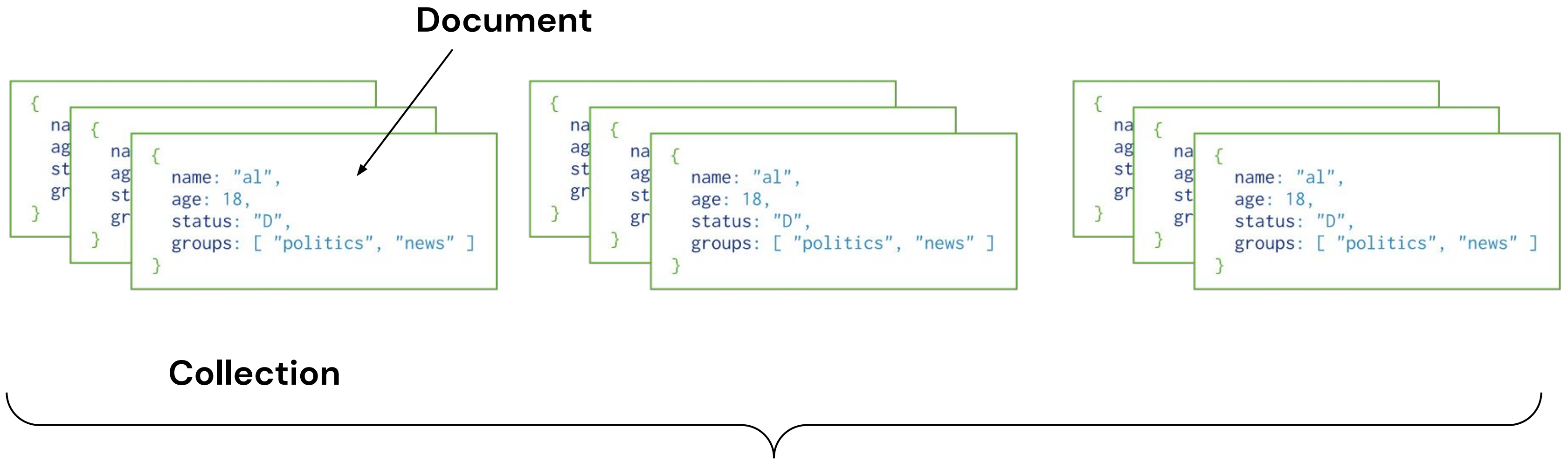
ACM at UCSD

SQL vs NoSQL Databases

- **SQL** (pronounced "sequel") **- S**tructured **Q**uery **L**anguage
  - **Relational database** - uses **tables** to organize information
- **NoSQL** - non-relational database
  - Typically, data is organized as **documents** instead

Check-In Code: woooohack

In MongoDB, data is stored in collections of documents, which contain key-value pairs



```
{
   name: "al",
   age: 18,
   status: "D",
   groups: [ "politics", "news" ]
}
```

Database

Check-In Code: wooookhack

ACM at UCSD

Database Setup

Check-In Code: wooooohack

ACM at UCSD

MongoDB Atlas

- MongoDB, but hosted on the cloud rather than locally
- Allows us to quickly set up database without having to install anything
- https://account.mongodb.com/account/login to create an account
- We need to create a new database and get the URL
  - Don't forget to whitelist your IP!

Mongoose setup

- Mongoose is a wrapper for MongoDB with additional features
  - Mongoose is used to define schemas - blueprints for the structure our data is going to take
  - We can use these schemas to generate models
    Each instance of a model = document
  - Install with npm install mongoose

Check-In Code: woooohack

Connecting to our database

- Copy the database URL from Atlas
  - Store it as a variable in your .env file! (Important because we don’t want to share this URL with the public to avoid reads/writes we don’t want)

```
const mongoose = require('mongoose');

mongoose.connect(config.databaseUrl, {
    useNewUrlParser: true,
    useUnifiedTopology: true }).then(() => {
  console.log('Connected to MongoDB database');
});
```

Defining Schemas

- Let's define what a **Purchase** is in our database
- The fields we need:
  - name: String
  - description: String
  - location: String
  - date: Date
  - cost: Number
  - method: String

Check-In Code: woooohack

ACM at UCSD

Main Operations

Check-In Code: wooooohack

ACM at UCSD

Read/Write Operations

- **C**reate, **R**ead, **U**pdate, **D**elete - known as **CRUD** Operations
- These are the main operations you will be using to interact with databases
- We'll only need to use the Create and Read operations for this project

Create

Add new document

- Create a new document from model with specified fields
- Example:

```
// Create a new purchase from the given
object
const newPurchase = {
    name: ‘iPhone’,
    description: '1TB iPhone 13',
    //...
};
Purchase.create(newPurchase);
```

Check-In Code: wooookack

ACM at UCSD

Read

Retrieve a document

- Allows us to get data from our database
- Filter documents based on a given query
- Examples:

```
// Find all purchases
Purchase.find().exec();

// Find a purchase by id
Purchase.findById(id).exec();
```

Check-In Code: wooohack

ACM at UCSD

Update

Modify an existing document

Check-In Code: woooohack

- Similar to create; we can update 1 or many
- Optional: create document if it doesn't exist
- Examples:

```
// update first document where 'a' is 1
// set 'a' to 2 instead
db.collection('example').updateOne({a: 1},
    {$set: {a: 2}});

// update ALL document where 'a' is 2
// add new field 'b'
db.collection('example').updateMany({a: 2},
    {$set: {b: 2}});

// create the document if it doesn't exist
db.collection('example').updateOne({a: 4},
    {$set: {b: 4}}, {upsert: true});
```

ACM at UCSD

Delete

Remove a document

- We can delete a single document, or delete multiple based on a query
- Examples:

```
// delete first iPhone in the database
Purchase.deleteOne({name: 'iPhone'});

// delete ALL electric types
Purchase.deleteMany({location: 'UTC'});
```

Check-In Code: woooohack

ACM at UCSD

Activity: Modify API routes to use database

TODO:

1. GET /purchases route
   a. Get all purchases currently in the database
2. POST /purchase route
   a. Create a new purchase based on the parameters passed in the body
3. Try testing your routes with Postman!
   a. POST to create a new document
   b. GET the document you just created
   c. View your changes from MongoDB Atlas as they happen

Check-In Code: woooohack

Check-In Code: wooooohack

That's it for Databases! Next time...we put it all together!

ACM at UCSD