

Lecture 19: Connecting to the Backend – Web Services and Databases



05-431/631 Software Structures for User
Interfaces (SSUI)

Fall, 2021

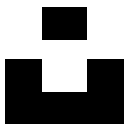


Logistics

- Homework 5 due today
- Start on Homework 6 – last homework!
 - Today will cover how to do hw6
- Clara will run review session on Sunday at 1:00pm in NSH 3001 and Zoom (same room)
 - See Canvas announcement
- Will be news about final projects next week

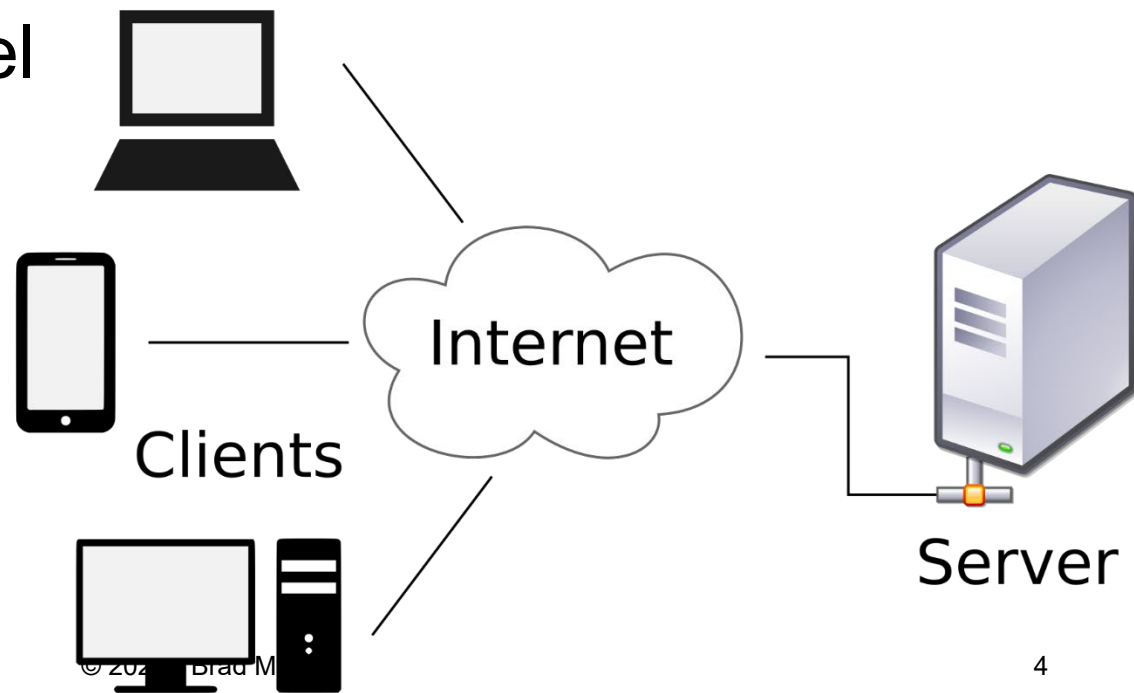
Background

- This is a front-end course
- Pretty much all Web and Smartphone apps connect to a backend server
 - As do many desktop applications
- Modern “Web Services” make creating integrated (“full stack”) apps quite easy
- Homework 6 asks you to use 2 different kinds:
 1. **Web service** for getting pictures
 - [Unsplash](#) – conventional REST interface; free for small tasks
 - No need for authentication, security, etc.
 2. **Networked database** for storing user-specific data
 - We selected Google’s **Firebase: NoSQL**, object-oriented so easier to learn
 - Also handles person authentication in an easy way



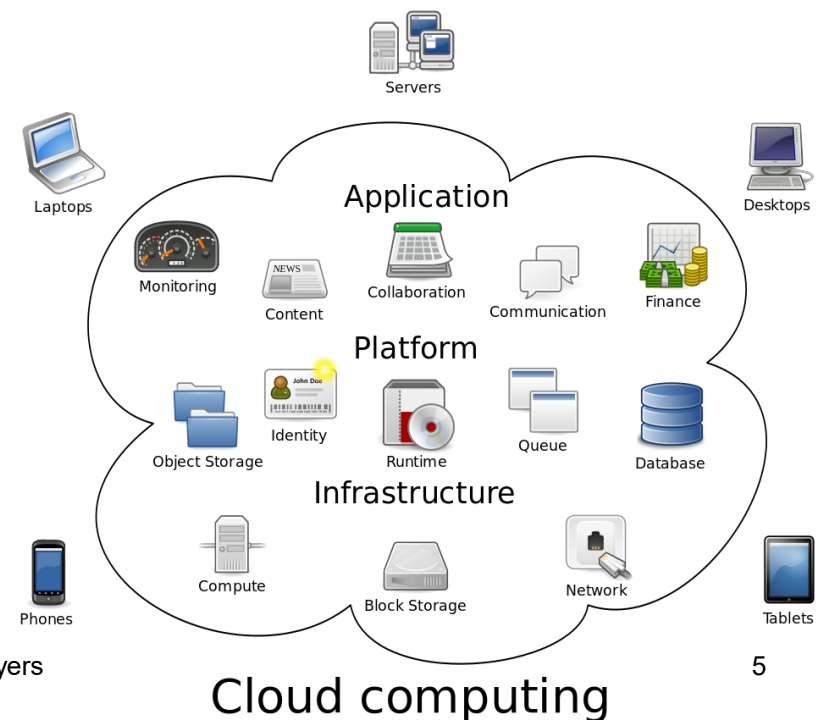
“Client Server Model”

- Client = smaller computers, phones, devices (IoT)
- Server = bigger computer, clusters
 - Does the bigger tasks, stores the bigger data
 - Manage sharing
- Client-server model dates from the 1960s
- Many protocols over the years



Cloud Computing

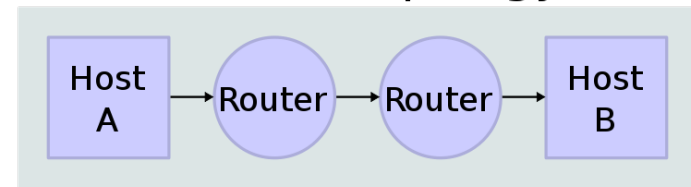
- Amorphous cloud of networked elements
- Don't necessarily address a specific server
- Not necessarily centrally managed
- “Cloud Computing” term started to be used around 2000
- (Reminder, WWW dates from 1990; “Internet” term from the 1980s, but ARPAnet from 1960s)



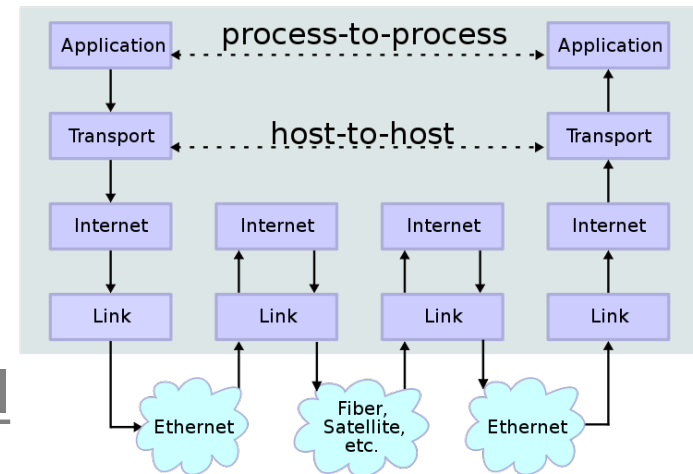
Important Protocols

- (Assuming have not taken a networking class!)
- “Protocol” – rules to allow communication
 - For Internet, are worldwide standards
 - Format of the messages
- Multiple levels of protocols – build higher ones using lower-level ones
- **TCP/IP** - Transmission Control Protocol and the Internet Protocol (IP) – how packets are sent around the internet
 - Handles naming of hosts (servers) – like cs.cmu.edu & IP numbers, like 128.2.42.95 (CMU)
 - Routing of packets with retry if one is lost (not for video) – may have many hops
- Headers say where each packet is going
- Examples: Simple Mail Transfer Protocol (SMTP), SSH, FTP, Telnet, **http**

Network Topology



Data Flow



Web protocols

- Hypertext Transfer Protocol (HTTP) & newer HTTPS (secure)
 - All transfers in plain text
- Others built on top of http
- SOAP - Simple Object Access Protocol
 - Started around 1999 by Microsoft
 - Data encoded in XML
 - Had to describe the format of all messages using the on Web Services Description Language (WSDL)
 - Specifies what specific fields and values are allowed
 - Very complex and hard to use



XML

- Extensible Markup Language (XML)
- Looks like html, but a little different
 - Yet another syntax
- Used as communication and storage format
- Arrays are implicit (like html)

```
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
</CATALOG>
```


JSON

- JavaScript Object Notation
- Alternative to XML for saving and exchanging inurns formation
 - Files and web services
- Yet another syntax!
- Similar, but not identical to the other ones we have been using
- **Note that names must be quoted strings**
- Like JavaScript objects
- Arrays using []
- Values can be JavaScript types or object or array
- See SSUI-hw6/tempResults.json

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```



REST

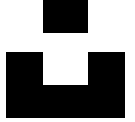
- Representational state transfer (REST)
- Created in 2000 by [Roy Fielding](#) in his PhD dissertation from UC Irvine
- Very simple protocol, in contrast to SOAP
 - Also more efficient
 - Everyone uses this today
- Encode commands and parameters in the URL
- Simple commands: Post, Get, Put, Patch, Delete
- Return values as HTML, XML, or JSON
- **Stateless** so don't have to worry about keeping track of thing – all supplied in each message
- No need to specify what will be in the messages
- RESTful APIs (web services) follow this format



Web Services

- *Lots* of available web services
 - Programmableweb.com lists over 24,000 public APIs
 - E.g., 476 APIs for “credit cards” (*down 3 from last year*)
 - Lots more available internally for companies
- Companies redoing their proprietary client-server or “mainframe” APIs to have web-services so easier to access on their own phone & web apps
- Many companies are monetizing their data assets as web services
 - Sometimes just to “trusted partners”
 - E.g., PNC bank + Insurance company adjusters

Using a RESTful Web Service



- Example: [Unsplash for Developers](#)
- Register as a developer
- Note: DEMO mode is “**50 requests per hour**”
- Accept the terms
- Make an application
- Scroll down to get your **API Keys**
 - We are using “Access Key” from [Unsplash](#)
 - Everyone should get their own – do not use mine!
 - Will need to put this into every message (since stateless)
 - How they make sure you are allowed to request pictures



Operations

- Look like html requests
- URL: <https://api.unsplash.com/>
- Command (looks like a path), e.g: [/search/photos](#)
- Then parameters and values for that command
 - First one separated by **?**, then by **&**
 - Parameter order usually doesn't matter
 - Always have **client_id** (Access Key) as the API key

- **Example query:**

```
https://api.unsplash.com/search/photos/?client_id=YOUR_ACCESS_KEY&page=1&query=lion&per_page=10
```

- Return is a JSON file (or error)
 - Pull out of it what you need
- Remember: stateless, so have to send the information each time

Keeping the API Key safe

- Your app uses the same access key for all users
- Could theoretically just have it as a string in your JavaScript file
 - But big security hole
 - People can get it from your downloaded build
 - Lets people write code that uses your app's access for free
- One alternative, keep in “environment”: see [stackoverflow](#)

- File called “.env” at top level, put in .env:

```
REACT_APP_UNSPASH_API_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

- Then, your code can say:

```
const API_KEY = process.env.REACT_APP_UNSPASH_API_KEY;
```

- Make sure .env is in .gitignore
- Doc = <https://stackoverflow.com/a/57103663>
- Can also keep it in your Firebase database:
<https://dev.to/remi/firebase-set-and-access-environment-variables-1gh8>

Warning from GitGuardian

- “a secret has been exposed in the git history”:

SECRET INCIDENTS > #31623

Generic High Entropy Secret

OCCURRENCE 1 TIMELINE

Occurrences

DATE	WHO	WHERE	TAGS
October 30th, 2020 13:49	lxieyang lxieyang@umich.edu	lxieyang/05631-hw6-refer... 4dbbe98 src/containers/CreateFromPicture/Search/	

1-1 of 1

Search.js
in commit: 4dbbe98

```

@@ -11,7 +11,8 @@ import Scotty5 from '../../assets/images/scotty-5.png';
11 11 import './Search.css';
12 12
13 13 const API_ENDPOINT = 'https://api.unsplash.com/search/photos/';
14 - const API_KEY = process.env.REACT_APP_UNSPASH_API_KEY;
14 + const API_KEY = 'sQM8NmczR9aKtYSeKu_3N8D1DRPYS6ySFctwA66ZALU'; //Brad's key
15 + // const API_KEY = process.env.REACT_APP_UNSPASH_API_KEY;
15 16
16 17 const Search = ({
17 18   setChosenImg,

```

WALL OF ❤️ ⓘ MX Michael Xieyang Liu
lxieyang@umich.edu

EXPLORE THE INCIDENT

A secret has been exposed in your git history.

STATUS **TRIGGERED**

DURATION **0h, 4min, 39s**

ASSIGNEE **None**

Resolve **Ignore**

HOW TO REMEDIATE

- 1 Revoke and rotate your **Generic High Entropy Secret**
- 2 If you have the rights to do so, remove secret completely from git history
`git commit --amend` and `git push --force`
- 3 Check access logs for potential breaches
- ✓ You can mark the incident as resolved

Environment variables

- To set it up for netlify, under "Site settings" -> "Build & deploy" => "Environment" -> "Environment variables"

Environment

Control the environment your site builds in and/or gets deployed to.

Environment variables

Set environment variables for your build script and add-ons.

REACT_APP_UNSPASH_API_KEY 

[Learn more about environment variables in the docs](#) ↗

Edit variables



Sending/receiving the request

- Sending the URL to a remote server and waiting for the results will take a noticeable time
 - So need to use asynchronous features of JavaScript
 - `async`, `await`, `then`
- Built-in call: `fetch(apiCall)` returns a *promise* (so can avoid needing explicit `async`) – see documentation
 - Call `.then` on the result to get the response and its data:

```
fetch('http://example.com/movies.json')  
  .then(response => response.json())  
  .then(data => console.log(data));
```
- Response is a “stream”; `.json` reads it all, returning another promise from which you can get the data as an object – see doc
 - `data` has fields for all of the json fields
 - E.g., for `tempResults.json`: `data.total = 2390`; `data.total_pages = 239`, `data.results` is an array with 10 elements
 - Use `.map` to create element for each item, can use `urls.small` as image
- Wrap everything in try-catch if network errors

Using a Library instead of Rest Calls



- Most web services provide an SDK that helps construct the requests correctly and handle the results
 - The unsplash JS library
 - In this case, it is pretty easy to concatenate up the required request, so maybe the library isn't needed?
 - But you are welcome to use it

Use Cloud Storage

- Database on a server (instead of on the client)
 - Also handles user-login and authentication
 - User can switch machines and still get to their data
- We will use Firebase database provided by Firebase (Google) <https://firebase.google.com/>



Cloud Firestore

Getting Started with Firebase

- Go to <https://firebase.google.com/>, hit “Get Started” (log in with your typical Google account). Then “Create a Project”. Give your project a name. I’ll use “ssui-hw6”. BTW, don’t use Google Analytics.
- Once a project is created, go ahead and add an app (web). I’ll name it “ssui-hw6”,
 - No need for “hosting”
 - hit register.
- Will use the code displayed in a minute





Getting Started, cont.

- Docs for web:
<https://firebase.google.com/docs/web/setup?authuser=0>
- Step 3,
 - 1. – use Node.js apps
 - Follow these instructions
 - No need to do npm init
 - `npm install --save firebase`
 - 2. – use exactly what there – authentication and firestore
 - Make a `src/firestore` folder, new file `index.js` there
 - Put this code there at the top
 - 3. – use the config information from other page in that file too

Finishing Setup

- Resulting file will look like the following:

```
import firebase from 'firebase/app';

import 'firebase/auth';
import 'firebase/firestore';

var firebaseConfig = {
  apiKey: '████████████████████████████████████████',
  authDomain: 'ssui-hw6.firebaseio.com',
  databaseURL: 'https://ssui-hw6.firebaseio.com',
  projectId: 'ssui-hw6',
  storageBucket: 'ssui-hw6.appspot.com',
  messagingSenderId: '██████████',
  appId: '████████████████████████████████████████',
};

firebase.initializeApp(firebaseConfig);

export default firebase;
```

Authentication

- “Continue to console”
- Click on authentication
 - Set up sign in method
 - Select Google - third one
 - Slide to “enable”
 - Support email – login one is fine
 - Save
- Need to add “Authorized Domains”
 - Need to add netlify as trusted domain
 - Need the full name of the domain you are using in netlify:
xxxx.netlify.app
- Should be configured!



In the code

- Create auth page route to get to from login button
 - See: <https://firebase.google.com/docs/auth/web/google-signin?authuser=0>
 - Skip 2,3,4
 - Step 5

```
const handleLogIn = () => {
  const provider = new firebase.auth.GoogleAuthProvider();
  firebase
    .auth()
    .signInWithPopup(provider);
};
```

- Now, set the user information:
 - See: https://firebase.google.com/docs/auth/web/start?authuser=0#set_an_authentication_state_observer_and_get_user_data
 - Put this code into app.js in componentDidMount ()
 - To set the user state variable: `this.setState({ user });`
- Also need the `this.unsubscribeAuthListener();` in componentWillUnmount ()
- Now, in auth.js
 - Display login or logout based on whether user variable is set
 - Use to logout: `firebase.auth().signOut();`
 - `user.displayName = user's name`
 - `user.photoURL = picture for login button – can do in an in navbar`

Database Functions

- Go to console; Click on app
- Cloud Firestore
 - Create Database
 - Start in Test mode
 - Location: default is fine
 - Enable
- Now it is ready to go
- Can come back to here to see what is in the database

Using Firestore

- Good documentation:
<https://firebase.google.com/docs/firestore>
- A Firestore database is consisted of a series of “collections” and “documents”.
 - On-line debugging interface

In the Code

- Should read the documentation first:
 - <https://firebase.google.com/docs/firestore?authuser=0>
 - Data model:
<https://firebase.google.com/docs/firestore/data-model?authuser=0>
 - Also: Add and Manage Data & Read data
- One collection: e.g., `ShoppingCartItems`
 - (Not supposed to have a bunch of collections)
 - Contains a set of items, each with the fields



Adding items

- Disable add to cart buttons based on user not logged in
- Add to cart function will use
 - Need user ID since per user: (assuming logged in) `this.state.user.uid`
 - Update to deal with new kind of image shirts
- For adding data: Add a document:
[https://firebase.google.com/docs/firestore/manage-data/add-data?authuser=0#add a document](https://firebase.google.com/docs/firestore/manage-data/add-data?authuser=0#add_a_document)
 - Use generated ID

```
let cartItemRef = firebase
    .firestore()
    .collection("ShoppingCartItems")
    .doc();
cartItemRef.add(cartItem);
```

Query list of cart items for this user

- Need a listener for cart change so will refresh automatically
- Read data, listen to multiple items:
 - [https://firebase.google.com/docs/firestore/query-data/listen?authuser=0#listen to multiple documents in a collection](https://firebase.google.com/docs/firestore/query-data/listen?authuser=0#listen_to_multiple_documents_in_a_collection)
- Foreach: collect the information you need from each item, including id
 - Sort by created time so newest at top
 - Can use array sort after query – lodash.com useful
 - Simpler than orderBy in database – first time run, will generate an error, use the url of the error to create a database index (slow)

Updating and Removing items

- **Update** doc:

<https://firebase.google.com/docs/firestore/manage-data/add-data?authuser=0#update-data>

- E.g., for quantity change

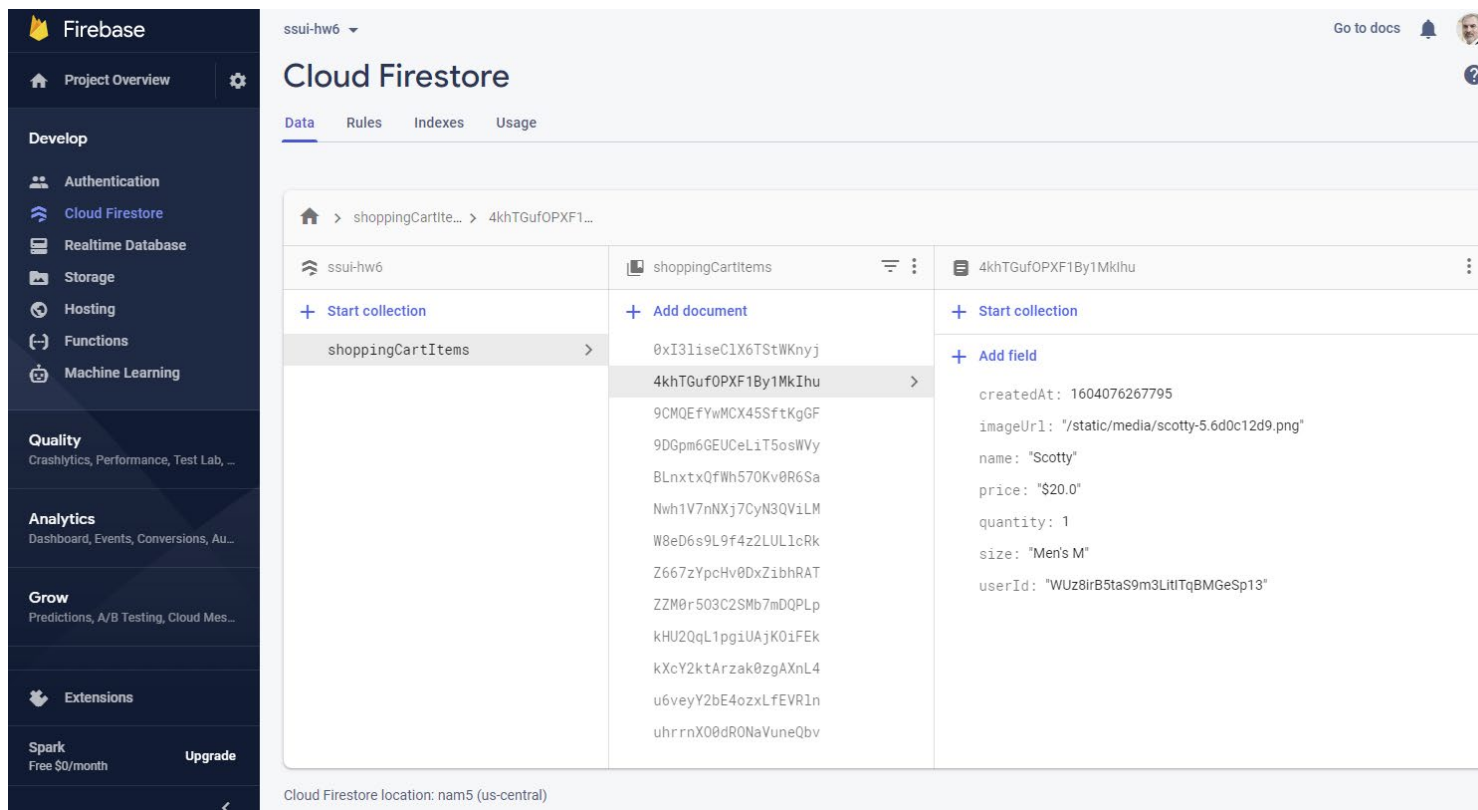
- **Delete** data in doc:

<https://firebase.google.com/docs/firestore/manage-data/delete-data?authuser=0>

- Need the id
- `.doc(id).delete()`

Debugging the Database

- Can see items come and go from database
 - Highlights changed items with orange
 - Can also add/edit/remove items interactively
- Available from Console / ssui-hw6 / “Cloud Firestore” (on left)



The screenshot shows the Firebase console interface for a project named 'ssui-hw6'. The main view is 'Cloud Firestore' with the 'Data' tab selected. The breadcrumb path is 'shoppingCartItems > 4khTGufOPXF1...'. The interface is divided into three panes:

- Left Pane:** Shows a collection named 'shoppingCartItems' with a '+ Start collection' button and a list of document IDs. The document '4khTGufOPXF1By1MkIhu' is highlighted in orange, indicating it has been recently changed.
- Middle Pane:** Shows the details of the selected document '4khTGufOPXF1By1MkIhu', including a '+ Add document' button and a list of document IDs.
- Right Pane:** Shows the details of the selected document, including a '+ Start collection' button, a '+ Add field' button, and the document's metadata and fields:
 - createdAt: 1604076267795
 - imageUrl: "/static/media/scotty-5.6d0c12d9.png"
 - name: "Scotty"
 - price: "\$20.0"
 - quantity: 1
 - size: "Men's M"
 - userId: "WUz8irB5taS9m3LitTqBMGeSp13"

At the bottom, the Cloud Firestore location is noted as 'nam5 (us-central)'.



Creating your own backend

- Doing Server-Side programming
- Building and supplying your own web services
- **Sorry, not covered in this course**
- Resources: `expressjs.com` using `node.js`
 - Popular library to support responding to requests
- Jeff Eppinger's fall course:
17-437/17-637: Web Application Development