Persistence

SWE 432, Fall 2017
Design and Implementation of Software for the Web
Today

• Demo: Promises and Timers
• What is “state” in a web application?
• How do we store it, and how do we choose where to store it?
Demo: Promises and Timers
What is “state” in a web app?
Application State

• All data in an application
• What kinds of data are we concerned about?
  • What user is logged in?
  • What interactions have they had with us before?
  • What data have they given us?
  • What data have others given us?
• Where do we store all of these things?
State: Example

Amazon.com...
HTTP is stateless

- Each request / response pair is independent of previous request / response pair
- Frontend cannot assume that it is making request to the same server.
  - Might be load balanced, crash, ...

HTTP Request
HTTP GET https://www.amazon.com

HTTP Request
HTTP GET https://www.amazon.com/Kakanuo-LED-Dimmable-Bi-pin-72X3014SMD/dp/B01FHKDQ58/ref=pd_sim_60_1?_encoding=UTF8&pd_rd_i=B01FHKDQ58&pd_rd_r=YWCC63ERTFEMK3E7F3H6&pd_rd_w=o3wIl&pd_rd_wg=dMfnr&psc=1&refRID=YWCC63ERTFEMK3E7F3H6
Where to persist application state?

• Many options
• Goals:
  • Cost
  • Efficiency
  • Stability
Where to persist application state?

• Should consider how often we need to show it to the user, and how permanently we need to store it

• Examples:
  • What user is logged in? (Transient, relevant to user and backend)
  • What’s in my shopping cart? (Semi-transient, relevant to user and backend)
  • What products am I looking at? (Transient, relevant to user)
  • What are all of the products (Long-term, parts are relevant to users)
Where to persist application state

- URL and query parameters
  - Really small amounts of data
  - Data that should be changed through forward/back buttons in browser
- Frontend
  - Data we might need to show again soon
  - Fairly small (KB’s or few MBs, not 100 MB’s or GB’s)
  - Data we don’t care about going away or being maliciously manipulated
- In memory on backend
  - Data that we are working with that will fit in memory (MB’s probably not GB’s)
  - Transient data that can disappear if the server crashes
  - Cache or index of data stored on backend disk, database, or storage provider
- On backend disk or database
  - Data we need persisted “permanently”
  - Data that only needs to be used by single server
- Storage provider
  - Data we need persisted "permanently"
  - Data that we need to share across multiple servers
Frontend State: Cookies

- String associated with a name/domain/path, stored at the browser
- Series of name-value pairs, interpreted by the web application
- Create in HTTP response with “Set-Cookie: ”
- In all subsequent requests to this site, until cookie’s expiration, the client sends the HTTP header “Cookie: ”
- Often have an expiration (otherwise expire when browser closed)
- Various technical, privacy and security issues
  - Inconsistent state after using “back” button, third-party cookies, cross-site scripting, …
Cookies and Requests

Web “Front End”

HTTP Request

HTTP Response

HTTP/1.1 200 OK
...
Set-Cookie: class=swe432
...

HTTP Request

GET / HTTP/1.1
...
Cookie: class=swe432
...

HTTP Response

HTTP Request

GET / HTTP/1.1
...
Cookie: class=swe432
...

HTTP Response

Server “Back End”
Cookies & NodeJS

• cookie-parser enables reading and writing cookies
• npm install cookie-parser
• let cookieParser = require('cookie-parser');

• Stateful Hello World

```javascript
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();

app.use(cookieParser());

app.get('/', (req, res) => {
    if(req.cookies.helloSent == "true")
        res.send("I already said hello to you!");
    else
        res.cookie("helloSent","true").send('Hello World!');
});

app.listen(3000);
```
Persisting more complex state

• The most cookies you can have: 4KB (TOTAL per DOMAIN)
• Old solution
  • Cookie is a key to some data stored on server
  • When client makes a request, server always includes this “extra data” being stored on server
• What’s wrong with this old solution?
  • Really slow
  • For every request
    • Client passes key to server using cookie
    • Server loads data corresponding to key
    • Client downloads data as part of HTTP response
Frontend State with LocalStorage

- HTML5 added support for persisting larger data on the frontend

  ```javascript
  localStorage (Persists forever)
  sessionStorage (Persists until tab is closed)
  ```

- To use localStorage and sessionStorage

  ```javascript
  localStorage.setItem("key", "value");
  localStorage.getItem("key");
  ```

  ```javascript
  var id = localStorage.getItem("userId");
  ```

- Can store any string
- All pages in the same domain see the same localStorage and sessionStorage
- Alternatively: SQLite (SQL DB) that you can use in JS
Persisting state on the backend
Storing state in a global variable

- Global variables

```javascript
var express = require('express');
var app = express();
var port = process.env.port || 3000;

var counter = 0;
app.get('/', function (req, res) {
    res.send('Hello World has been said ' + counter + ' times!');
    counter++;
});

app.listen(port, function () {
    console.log('Example app listening on port ' + port);
});
```

- Pros/cons?
  - Keep data between requests
  - **Goes away** when your server stops
    - Should use for transient state or as cache
What forms of data might you have

- Key / value pairs
- JSON objects
- Tabular arrays of data
- Files
Options for backend persistence

• Where it is stored
  • On your server or another server you own
    • SQL databases, NoSQL databases
    • File system
  • Storage provider (not on a server you own)
    • BLOB store
    • NoSQL databases: Next time
Blobs: Storing uploaded files

• Example: User uploads picture
  • ... and then?
  • ... somehow process the file?
How do we store our files?

• Dealing with text is easy - we already figured out firebase
  • Could use other databases too… but that’s another class!

• But
  • What about pictures?
  • What about movies?
  • What about big huge text files?

• Aka…Binary Large OBject (BLOB)
  • Collection of binary data stored as a single entity
  • Generic terms for an entity that is array of bytes
Working with Blobs

• Module: express-fileupload

• Simplest case: take a file, save it on the server

```javascript
app.post('/upload', function(req, res) {
  var sampleFile;
  sampleFile = req.files.sampleFile;
  sampleFile.mv('/somewhere/on/your/server/filename.jpg', function(err) {
    if (err) {
      res.status(500).send(err);
    } else {
      res.send('File uploaded!');
    }
  });
});
```

• Long story... can't app.use(body-parser) when you are handling file uploads. Instead:

```javascript
app.use(express.json())
  .use(express.urlencoded());
```
Where to store blobs

- Saving them on our server is fine, but…
  - What if we don't want to deal with making sure we have enough storage
  - What if we don't want to deal with backing up those files
  - What if our app has too many requests for one server and state needs to be shared between load-balanced servers
  - What if we want someone else to deal with administering a server
Blob stores

- Amazon, Google, and others want to let you use their platform to solve this!

Diagram:
- Client
  - Node Backend
  - Uploads file
  - Distributed file

Google Cloud

- Server
- Server
- Server
- Server
- Server
- Server
- Server
- Server
Blob Stores

Typical workflow:
Client uploads file to your backend
Backend persists file to blob store
Backend saves link to file, e.g. in Firebase
Google Cloud Storage

• You get to store 5GB for free

• Setup

```
npm install --save @google-cloud/storage
```

```
var storage = require('@google-cloud/storage');

var fs = require('fs');

// Authenticating on a per-API-basis. You don’t need to do this if you auth on a
// global basis (see Authentication section above).

var gcs = storage({
    projectId: 'grape-spaceship-123',
    keyFilename: '/path/to/keyfile.json'
});

// Create a new bucket.
gcs.createBucket('my-new-bucket', function(err, bucket) {
    if (!err) {
        // "my-new-bucket" was successfully created.
    }
});
```

• https://www.npmjs.com/package/google-cloud
Google Cloud Storage

```javascript
// Reference an existing bucket.
var bucket = gcs.bucket('my-existing-bucket');

// Upload a local file to a new file to be created in your bucket.
bucket.upload('/photos/zoo/zebra.jpg', function(err, file) {
  if (!err) {
    // "zebra.jpg" is now in your bucket.
  }
});

// Download a file from your bucket.
bucket.file('giraffe.jpg').download({
  destination: '/photos/zoo/giraffe.jpg',
}, function(err) { });
```
Exercise: Where to Persist

• You are building a news aggregator site and want to recommend articles based on past articles the user has clicked on. Where should you persist this?
Exercise: Where to Persist

- You are building a shopping app and need to track a shopping card. How might you persist this?
Where to persist application state

- Frontend
  - Data we might need to show again soon
  - Fairly small (KB’s or few MBs, not 100 MB’s or GB’s)
  - Data we don’t care about going away or being maliciously manipulated
- In memory on backend
  - Data that we are working with that will fit in memory (MB’s probably not GB’s)
  - Transient data that can disappear if the server crashes
  - Cache or index of data stored on backend disk, database, or storage provider
- On backend disk or database
  - Data we need persisted “permanently”
  - Data that only needs to be used by single server
- On storage provider
  - Data we need persisted "permanently"
  - Data that we need to share across multiple servers
Readings for next time

- Firebase Get Started
  - https://firebase.google.com/docs/database/web/start

- Firebase Structure Data
  - https://firebase.google.com/docs/database/web/structure-data

- Firebase Read and Write Data
  - https://firebase.google.com/docs/database/web/read-and-write