

Sending Data

You can do A LOT with just linked HTML documents

- Wikipedia

Big Fun: send dynamic data to webserver

- generate dynamic responses
- How Wikipedia gets new articles/updates

Most common method: HTML Forms

The <form> element

```
<form action="ACTION" method="METHOD">
  <button type="submit">TEXT</button>
</form>
```

- ACTION is **url to submit to**
 - defaults to current url!
 - can be fully-qualified, path, etc
 - e.g. `/login`, `/logout`, `/add/more/stuff`
- METHOD is **HTTP Method**
 - `GET` or `POST` for html forms
- "submit" **navigates**
 - sends request
 - renders response

More about method and forms

- HTTP Method is sent in **web request**
 - **GET** or **POST** for html forms
 - **GET** should cause no changes on server
 - Request to read ("get") data from server
 - Sends form fields (if any) in url
 - **query params** in url
 - **POST** can change server data
 - Anything not a GET
 - Sends form data in **request body**
 - Can still have query params in url

Elements inside a form

Form can have most HTML inside

- Only certain elements matters to the "form"
- `<input>` is the most common
- `<input type="TYPE">`
 - TYPE defaults to **"text"**
 - see MDN for more

Sending Text with <input>

```
<form action="/lookup" method="GET">
  <input name="cat" placeholder="Cat Name">
  <button type="submit">Register Cat</button>
</form>
```

Input **type** defaulted to "text"

```
<input type="text" name="cat" placeholder="Cat Name">
```

On submit:

- Navigates to `/lookup?name=`
 - Followed by url-encoded cat name value
- Will get 404 if server not ready for `/lookup`

Query Parameters

In URL, a `?` separates **path** from **query parameters**

- Query parameters can be any string value
- Almost always html form style
 - `key=value` pairs (no spaces)
 - Separated by `&` (when multiple params)
 - `key` and `value` are **url encoded**

URL Encoding

- Any "special" characters are converted
 - `:/? $[] @ ! $ & ' () * + , ; = %`
- `%` followed by their **hexadecimal ascii value**
 - Other characters MIGHT be converted
 - Spaces often converted to `+`
 - Or may be `%20`
- Allows us to give these characters meaning
 - Like `?`, `&`, `=`, `%`
 - But still allow the characters in text

Server gets data in request object

Route-handling **callback**

- Is passed **request** and **response objects**
- By convention `req` and `res` (yuck)

```
app.get('/lookup', (req, res) => {  
  // handling code here  
});
```

Notice the `.get()`!

- We match method AND path
- We don't match query params

Common Conventions

Verbose:

```
app.get('/lookup', function( request, response ) {  
  // handling code here  
});
```

Conventional:

```
app.get('/lookup', (req, res) => {  
  // handling code here  
});
```

Reading the query params

```
app.get('/lookup', (req, res) => {  
  console.log(req.query);  
  res.send('received');  
});
```

- If you don't `.send()` a response, browser waits until timeout
- `console.log()` goes to server
 - This JS does not run on browser!
- `req.query` is an **object** of name:value pairs
 - One (common) way of parsing **query string**
- All values are text (strings)
 - Url can only send text

Dynamic response

```
const catStatus = {  
  Jorts: 'in trash bin',  
  Jean: 'opening closet',  
  Nyan: 'flying high',  
};  
  
app.get('/lookup', (req, res) => {  
  const { cat } = req.query;  
  // Same as:  
  // const cat = req.query.cat;  
  const activity = catStatus[cat] || 'cat not found';  
  res.send(activity);  
});
```

Notice you can change the URL in browser

- Don't need to go through web page
- Web is stateless, only request matters

Other elements

More than `<input type="text">`

- other `input` types
- other elements entirely
- won't cover everything

<input type="checkbox">

A clickable checkbox

- Tends to be small
- Good to surround with a `<label>` element
 - Clicking label counts as clicking input
- Only sends field when checked(!)
- Defaults to value `"on"`
- A `checked` attribute sets default checked status

<input type="radio">

- Multiple inputs can have same `name`
- Only one can be selected at a time
 - Defaults to one with `checked` attribute
 - Or none if no `checked` attribute
 - Can never unselect a choice
- You can have multiple radio groups
 - Same `name` will impact each other

<input type="password">

- Just like "text"
- Doesn't show the typed characters
- NOT ENCRYPTION
 - Only about being visible on screen
 - with a GET method, still shows in url

<input type="hidden">

- Like `type="text"`
- Except not shown to user
- Still sent to server

Why?

- Send info in a different way than shown to user
- Persist info across a series of requests

<textarea>

- Allows for multiple lines of text
- Some formatting/resizing options
- Otherwise like `<input type="text">`

Dropdowns

Two elements

- `<select>` wraps `<option>`s
- `<select>` takes `name`
- Each `<option>` has a `value`
 - Defaults to contents (DON'T DO THAT)
- One `<option>` can have `selected`
 - Defaults to first option if no `selected`
- `<select>` has options for selecting multiple
 - Confuses users, avoid :(

<input type="file">

All other elements input/send some form of text string

- `type="file"` is special
- upload a file (image, document, whatever)
- requires extra effort server-side
 - To read the file
 - Attackers can send huge/hostile data
 - Do we need to save it as visible to users?
- Can be complicated to make the form look good
- Varies a lot by browser

Body Data

A **GET** request

- Sends all data in the URL
- Should not cause the server to change data
 - Searches, reads

A **POST** request (from HTML form)

- Sends data in the **body of the request**
- Can cause the server to change data
 - Updates, data entry, etc

Making a POST form

The HTML is identical

- except `method="POST"` in the `<form>`
- Browser will url-encode and place in request body

You can see the sent data in the Browser Dev Tools

HTML form is not the only option

HTML Forms are limited

- We will cover other forms of sending later
- HTML Forms still can send basically everything
 - Differences are in UI and navigation
 - More later

Reading data from a request body

Server doesn't know if you are using HTML Forms

- It can handle countless options
 - with the right libraries
- We translate the request before our handler
 - in the "middle" of incoming and next step
 - Known as `middleware`
 - Generic term, this is how express does it

Express allows a "chain" of handlers

All requests regardless of method

- Converts the body of all requests
 - Using a `url-encoded` approach

```
app.use( express.urlencoded() );
```

Or a specific route

```
app.post( '/cats', express.urlencoded(), (req, res) => {  
  // code here  
});
```


Accessing the parsed body

`express.urlencoded()` gives warning

- Unless you pass `{ extended: false }`

Body fields are mapped to the `req.body` object

- If `req.body` is `undefined`
 - You probably forgot to parse the body w/middleware!

Example POST Form

```
<form action="/cats" method="POST">
  <label>
    Name:
    <input name="name" placeholder="Cat Name">
  </label>
  <label>
    Is Tabby?
    <input type="checkbox" name="isTabby">
  </label>
  <button type="submit">Register Cat</button>
</form>
```

Making the form look good

Appearance is the job of CSS!

- Build a **semantic** form
 - Don't use elements for appearance
- Good class names helpful
 - I skipped in demo for space
- `<label>` are important for a11y
 - No `id/for` needed if wrapping target

Styling `<form>` well a very common need!

- Be cautious of fighting browser standards
- CSS `appearance: none;` can help style

Server POST example

```
const cats = {
  Jorts: {
    isTabby: false,
  },
  Jean: {
    isTabby: true,
  },
};

app.post(
  '/cats',
  express.urlencoded({ extended: false }),
  ( req, res ) => {
    const { name, isTabby } = req.body;
    cats[name] = {
      isTabby,
    };
    res.redirect('/cats');
  }
);
```

Redirects

A redirect is a special response

- Sets the status code of the response
- Includes a `Location` header with destination
- Browser will get response with 3xx status
 - And auto request that Location url
 - Request for Location will be a GET request
- Non-browser clients make decision
 - Whether to auto-follow redirect response

Redirect after a successful POST for pages

Why?

- If they "reload" page
 - Browser RE-sends POST to change server
- After a redirect
 - "reload" just loads the Location again

Prevents unintended double-entry (or worse)

This is for *pages*

- Requests that return HTML pages
- Service calls follow different rules

Server-side Data

This class doesn't cover databases

- But the web doesn't change databases
- Your server program talks to database
 - Storing data from requests
 - Reading data to make responses
- We will do the same thing
 - Only using data in variables
 - Skip the "read/write" from/to Database step
- Our data will "reset" when server is restarted

Dynamic responses

How do we generate interesting HTML in responses?

- Based off of data?

Response Methods

- `res.redirect()` we've seen
- `res.status()` sends a status code
 - Defaults to 200 if you don't use
 - Does NOT complete response
- Remember response structure and order
 - one status line (first)
 - headers
 - body
- `res.send()` sends body content
 - Content is a string
 - Not a rendered page, just text

Dynamic Response Example

```
const cats = {
  Jorts: { isTabby: false },
  Jean: { isTabby: true },
};

function catList() {
  return Object.keys(cats).map( cat => `
    <li>
      ${cat}
      ${ cats[cat].isTabby ? 'is' : 'is NOT' } a Tabby
    </li>
  `).join('\n');
}

app.get('/cats', (req, res) => {
  res.send(` <html>
    <head> <title>Cat Results</title> </head>
    <body>
      ${ catList() }
    </body>
  </html> `);
});
```

Generating HTML

Using template literals to build HTML

- Tedious
- Minimal functionality

BUT it shows how web servers serve dynamic pages

- Using stored and passed data
- Generate a string of HTML
- Return it in response

All templating libraries and frameworks

- Still do this same process

Have Patience

This is what we'll use for now

- Including upcoming project(!)

To ensure you understand

- server vs client
- request vs response
- stateless web
- browser rendering

Later we'll write React and web services

Dynamic Routes

One more way to pass data to server

- in the path itself
 - not query params AFTER path

Example: `/students/12345657`

- Might show student records with NEUID `1234567`
- Can handle ANY NEUID
- Uses what is in the path

Server Route can have "params"

- Not "query params"
- Stored in `req.params` object
- Defined in path with `:`

```
app.get('/student/:neuid', (req, res) => {  
  console.log(req.params);  
  res.send(`I see neuid of ${req.params.neuid}`);  
});
```

- `:` not part of variable name
- `/` can separate multiple params

We will make more use of this when we get to services

Summary - HTML Forms

`<form>` element

- various data elements
 - `<input>` with many `type`
 - `<select>` with `<option>`
 - `<textarea>`
- `<button type='submit'>` to submit
- `action` for target
 - will NAVIGATE
- `method` of GET or POST
 - GET if not changing server
 - POST if changing server

Summary - Query Params

Sends form data in URL

- after `?`
- `key=value` pairs
- separated by `&`
- `key` and `value` are **URL-encoded**

A GET method form sends query params

Summary - URL Encoding

- Converts "special" characters to `%NN`
 - Where `NN` is hex ascii code for character
 - spaces often converted to `+` instead

Summary - reading query in Express

Request and Response objects in Express

- conventionally `(req, res)`

`req.query` object holds the key/value pairs from URL

Summary - reading body in Express

POST method forms send data in body

- still URL-encoded
- need to tell express to use middleware to translate
 - can be done for all requests, or per matching route
 - `express.urlencoded({ extended: false })`
- will populate `req.body` object
 - if undefined, forgot to call middleware
- Servers reading POST forms should `redirect(...)`

Summary - Reading path params in Express

- Define route with colon(:) + variable name in path
- Will populate `req.params`
- Colon isn't part of variable name in `req.params`
- Can have multiple variables separated by /