Comp 324/424 - Client-side Web Design

Spring Semester 2024 Week 13

Dr Nick Hayward

HTML5, CSS, & JS - example - part 1

working with Flickr API - update travel notes

- add option to Travel Notes app to allow a user to view images from Flickr
- need to update app's HTML, CSS, and JS
- modify how our notes, and associated options, are rendered to our users
- add a search option for photos on Flickr
- render our images to match the notes
- app's structure still reflects three primary content categories
  - `header` , `main` , and `footer` with slight modifications to the `main` category
- `main` content category updated to create two distinct rows for initial content
  - contain defined semantic containers
- row containing `.note-input` and Flickr search option `.image-search`
- many options to consider for adding image results to DOM
  - hide note output and show image search output
    - split current output to include both notes and images
  - …

HTML5, CSS, & JS - example - part 2

working with Flickr API - update travel notes HTML

- update the HTML for rendering the images
  - add alongside our notes
- create another row for these containers
  - add two section containers for `.note-output` and `.image-output`
  - optional usage includes hiding note output, fading out, split view…

```
<!-- note output -->
<section id="note-output" class="note-output">

</section>
<!-- note output -->
<section id="image-output" class="image-output">

</section>
```

HTML5, CSS, & JS - example - part 3

working with Flickr API - update travel notes JS

- add further functionality to **Travel Notes** app
- underlying logic for the notes will remain the same
- updates for searching, returning, and rendering images from Flickr
  - respond to user clicking on the search button
  - submit our query to Flickr
  - process the returned JSON for the images
  - render them for viewing

```
// define handler for image search btn click event
imgSearchBtn.addEventListener('click', () => {
    // search images
    // reset page no for queries after using pager buttons...
    pageNo = 1;
    // add extra args for search input, image output, note output
    const imgQuery = searchFlickr(pageNo, sort, searchInput, imageOutput, noteOutput);
    imgQuery.then(pages => totalPages = pages);

    searchInput.value = "";
});
```

- DEMO - Travel Notes - Flickr
- DEMO - Travel Notes - basic Flickr API search

Image - HTML5, CSS, & JS - Travel Notes & Flickr

Image - Working with APIs - Flickr API

HTML5, CSS, & JS - example - part 4

working with Flickr API - update travel notes CSS

- need to update and modify existing CSS
  - helps with correct rendering of the thumbnail images
- CSS additions are initially modest
  - reflects integration with existing app, grid, and flex layouts
- use existing ruleset for card design
  - same usage for `image-output` and `note-output`

```
/* note & image container - flex */
.note-output, .image-output {
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    row-gap: 20px;/*applies to rows of items - not above first row... */
    column-gap: 10px;
    padding-top: 20px;
}
```

- DEMO - Travel Notes - Flickr API search with cards

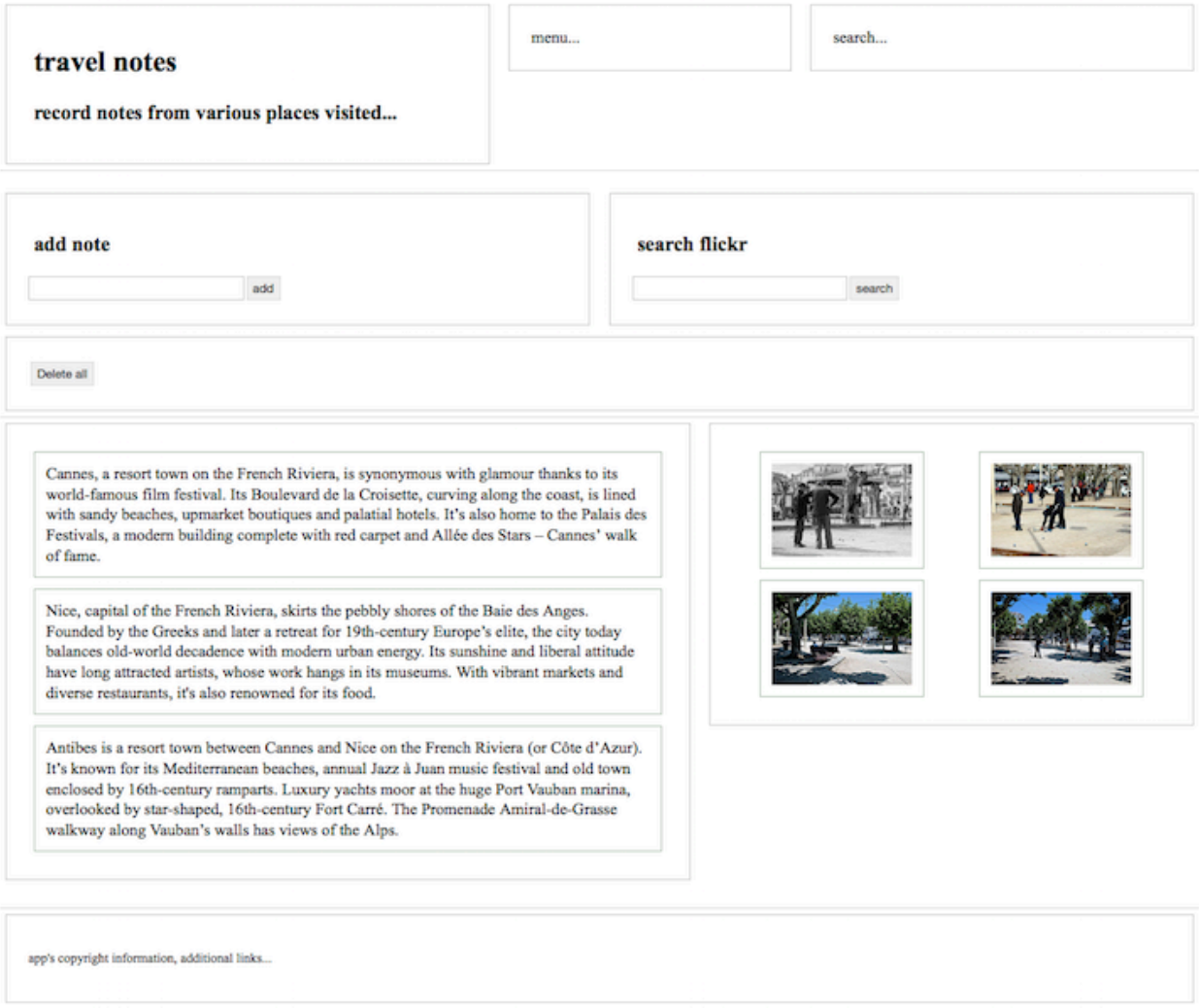

Figure 1: Travel Notes - Flickr - test loading images

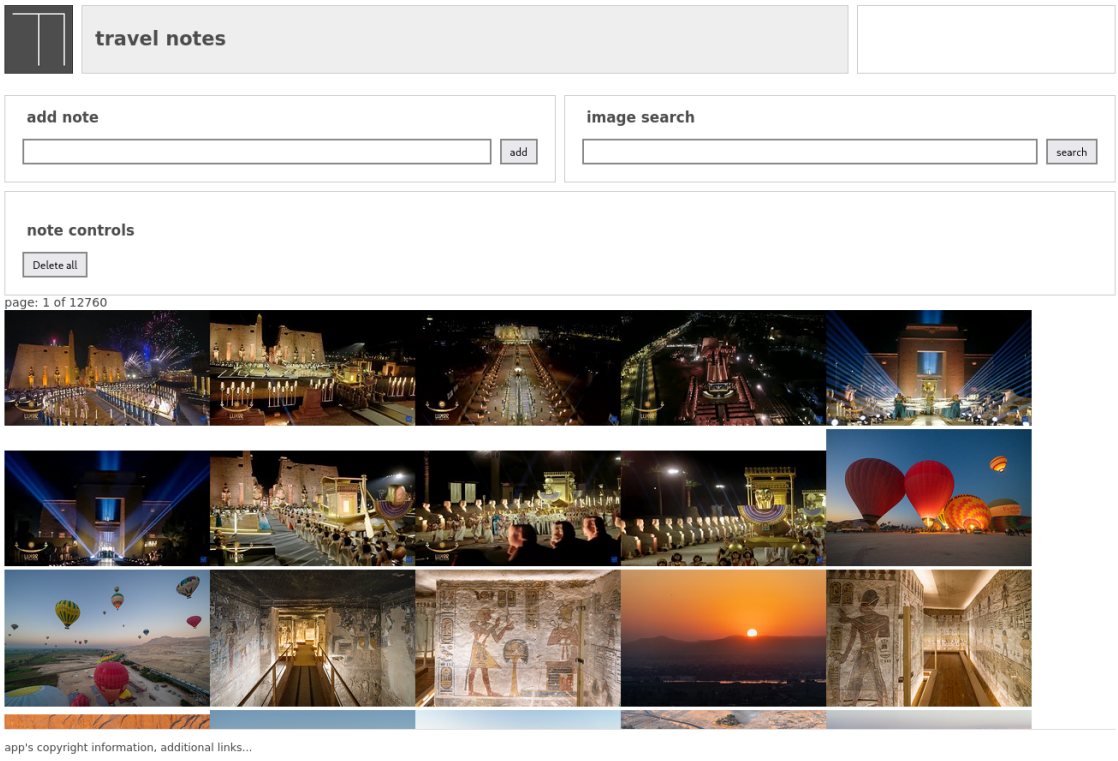

Figure 2: Travel Notes - basic Flickr API search

Image - HTML5, CSS, & JS - Travel Notes & Flickr

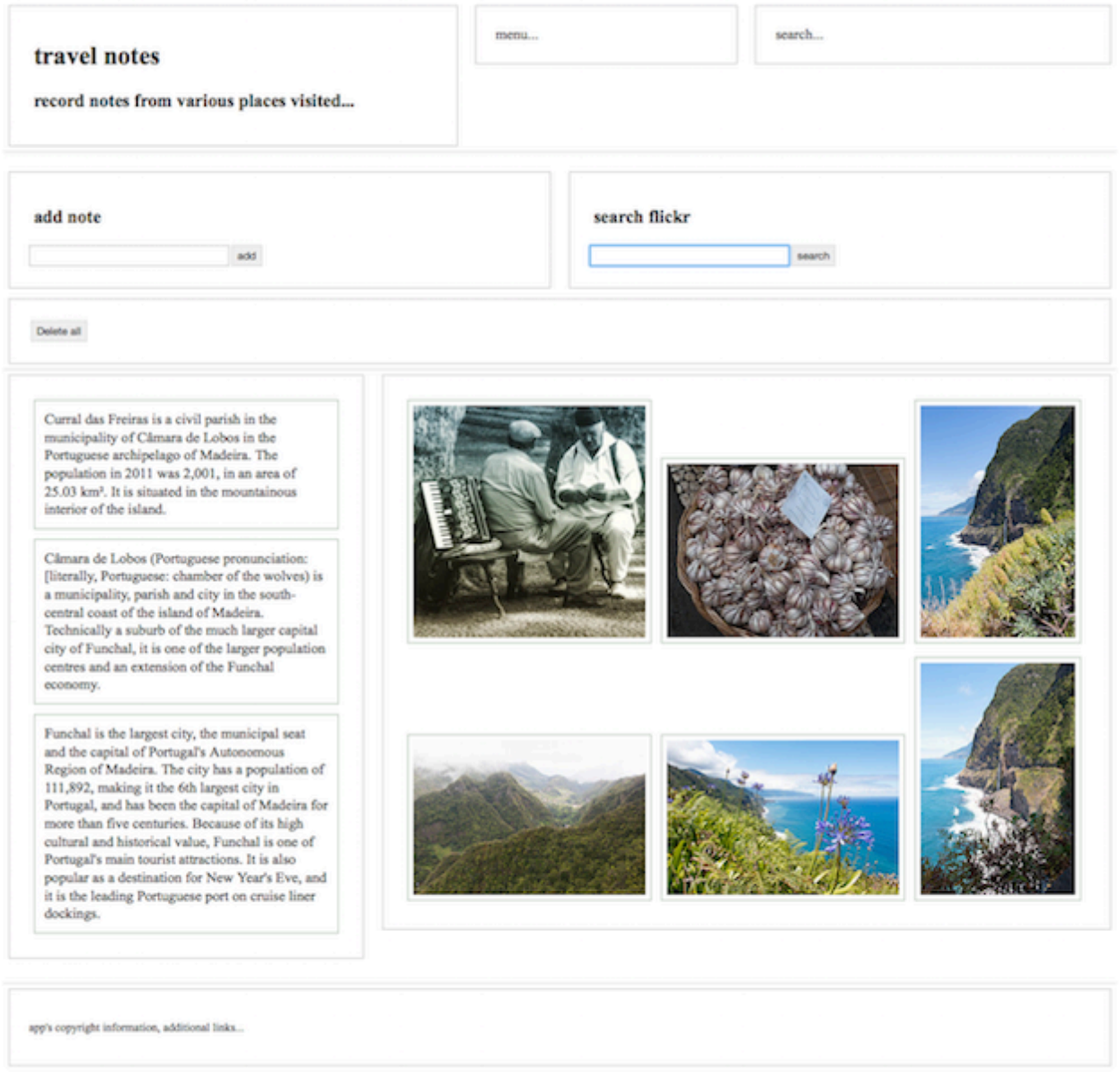

Figure 3: Travel Notes & Flickr - different layout

HTML5, CSS, & JS - example - part 5

working with Flickr API - modify travel notes JS

- common usage between note creation and image search, render &c.
  - create content for card based design
  - create node to nth depth - elements, attributes, and node content
  - hierarchical node structure and recursive usage

  - metadata handling
  - ...

Image - HTML5, CSS, & JS - Travel Notes & Flickr

HTML5, CSS, & JS - example - part 6

working with Flickr API - modify travel notes JS

- define metadata for each image card
  - get properties from each photo in data
  - pass custom `photoMetadata` object to `createImage()`

```
...
const photos = jsonData.photos.photo.map((val) => {
    const photoMetadata = {
        title: val.title,
        id: val.id,
        owner: val.owner,
    };
    const photoURL = buildPhoto(val);
    const photo = createImage(photoURL, photoMetadata);
    return photo;
});
...
```

HTML5, CSS, & JS - example - part 7

working with Flickr API - modify travel notes JS

- update node structure for photo card
- render metadata in card footer
  - or add to card header &c.

```
function createImage(inputVal, metadata) {
    const imgTitle = metadata.title;

    const imageStructure = [
        {elem: 'div', attr: 'class', attrVal: 'card-view'},
        [
            {elem: 'div', attr: 'class', attrVal: 'card-content'},
            {elem: 'img', attr: 'src', attrVal: inputVal},
        ],
        [
            {elem: 'footer'},
            {elem: 'p', attr: 'class', attrVal: 'img-metadata', txtNode: `${imgTitle}`},
        ],
    ];

...

}
```

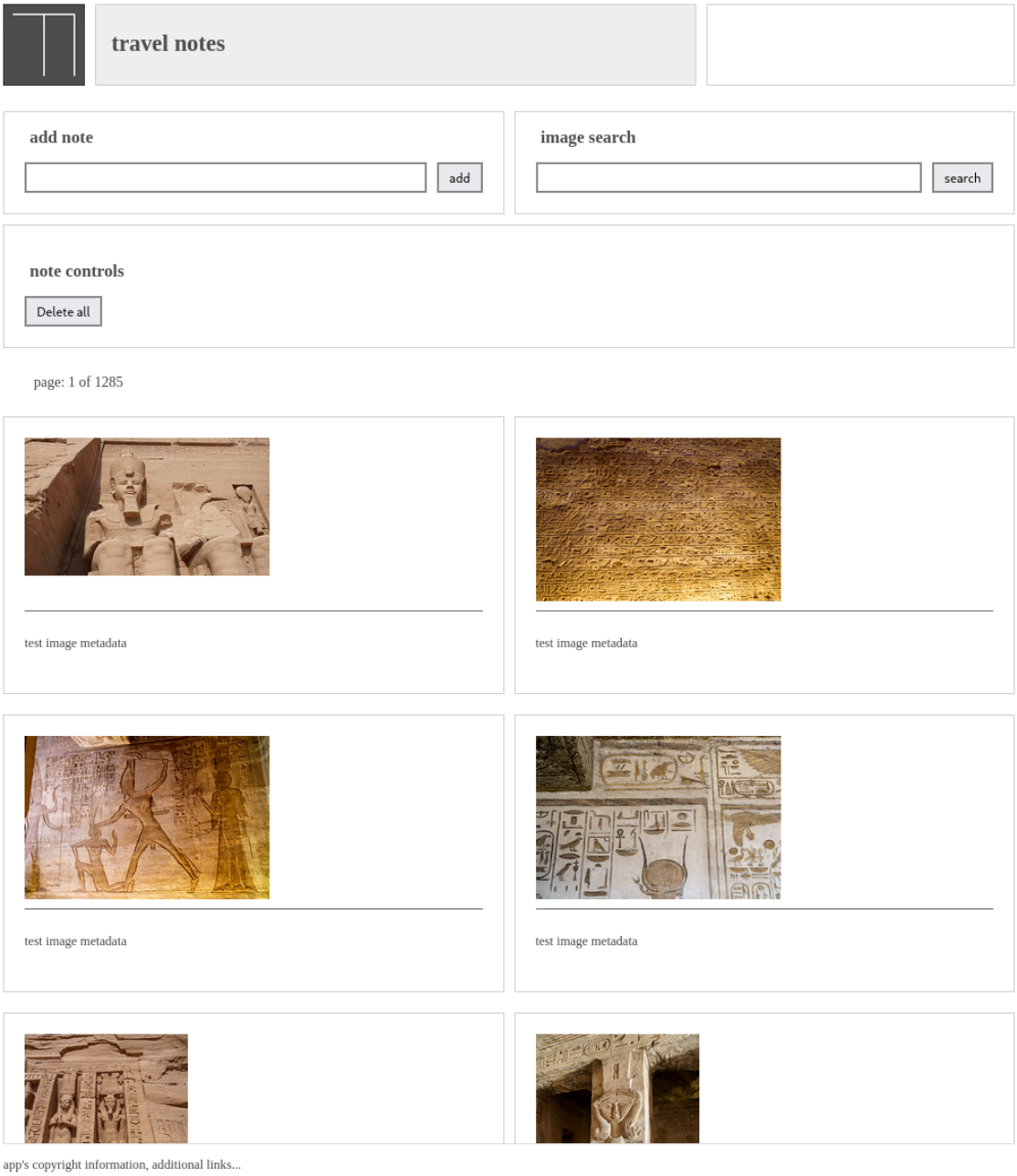

Figure 4: Travel Notes & Flickr - card layout

- DEMO - Travel Notes - Flickr API search with cards & metadata

---

Image - HTML5, CSS, & JS - Travel Notes & Flickr

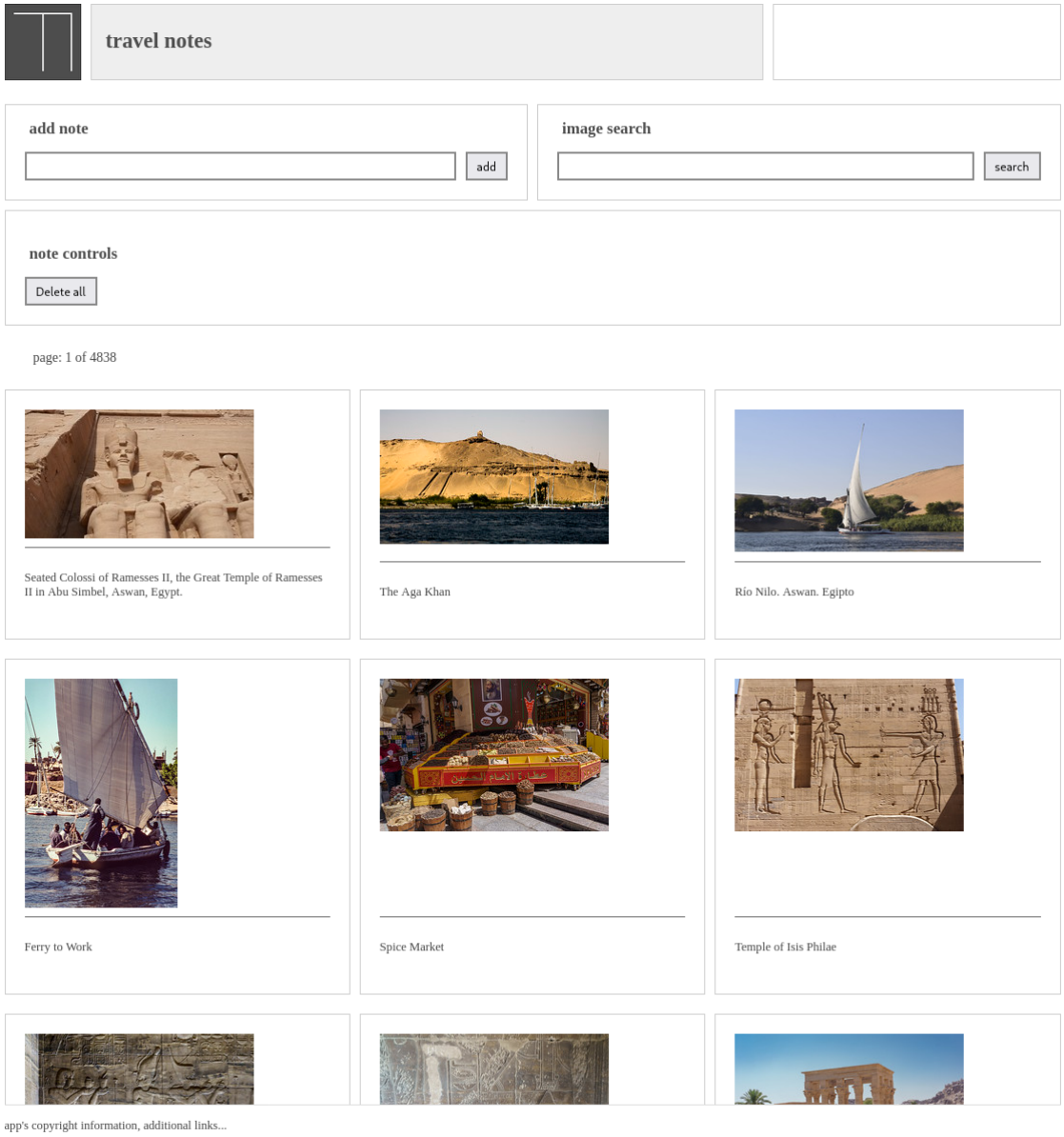

Figure 5: Travel Notes & Flickr - card layout with metadata

HTML5, CSS, & JS - example - part 8

working with Flickr API - modify travel notes JS

- continue to modify and build our Travel Notes app

- add some metadata for the returned image collection
  - e.g. using title and link from search query response
- add initial metadata output
  - metadata from Flickr JSON response in Promise object
- DEMO - Travel Notes & Flickr - initial metadata

Image - HTML5, CSS, & JS - Travel Notes & Flickr

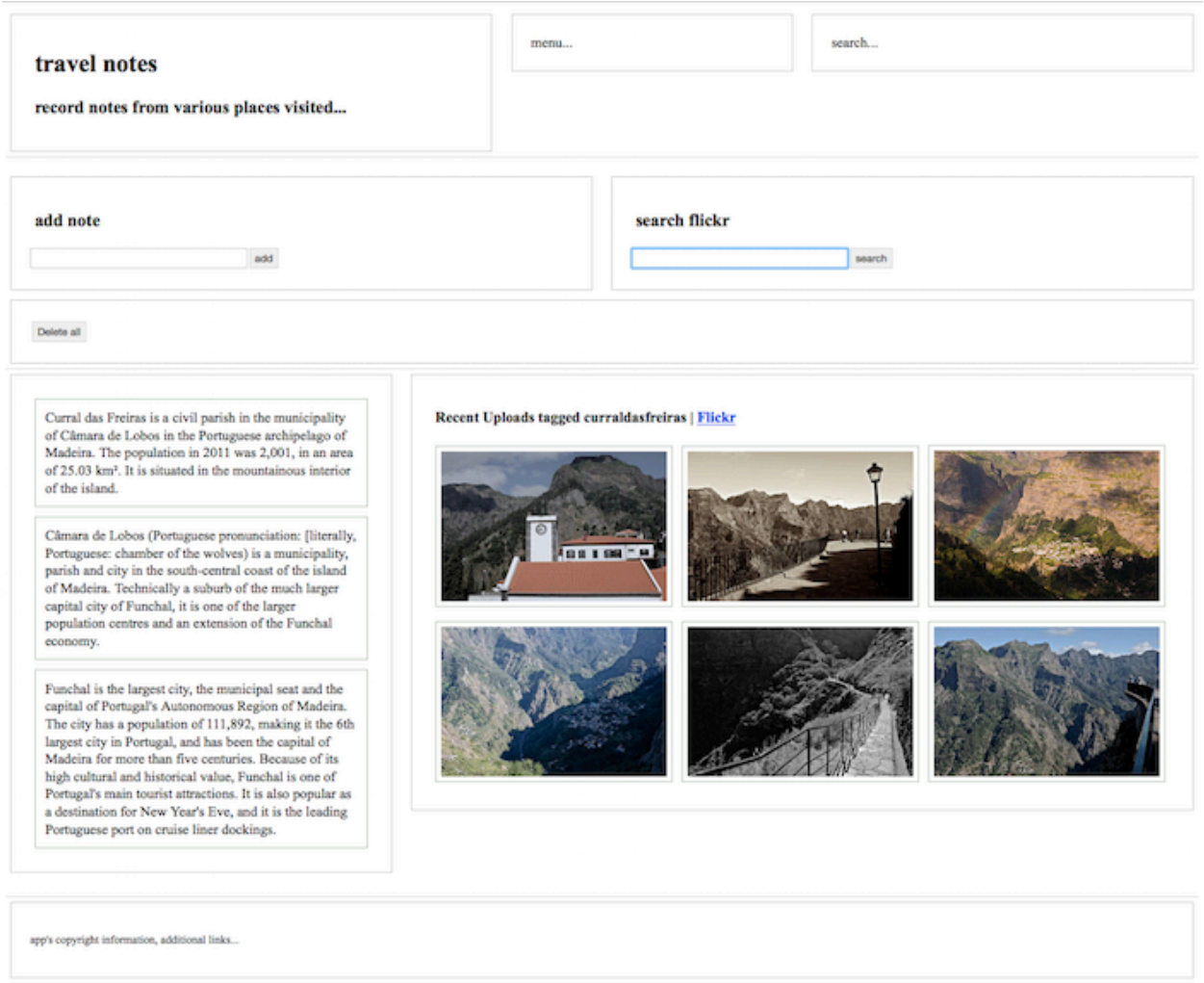

Figure 6: Travel Notes & Flickr - initial metadata

HTML5, CSS, & JS - example - part 9

working with Flickr API - modify travel notes JS

- consider alternative renderings for notes and images
- e.g. add modification to resize the `.notes-output`
  - create more space for the images
  - gently shift focus to the new images

- add check to ensure images are not visible in the DOM
  - create split view with images and notes
  - focus on current active zone in app
- DEMO - Travel Notes & Flickr

Image - HTML5, CSS, & JS - Travel Notes & Flickr

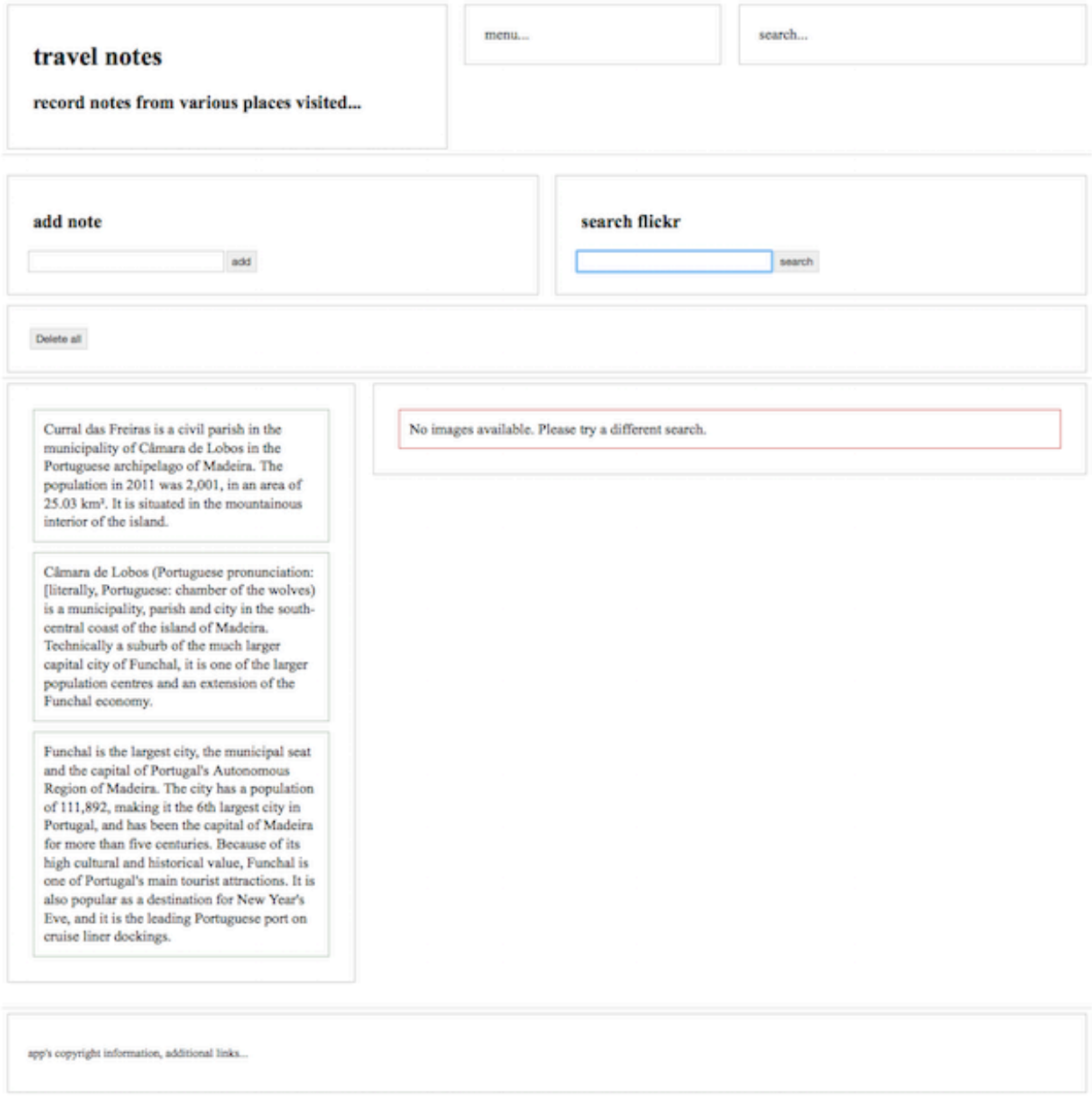

Figure 7: Travel Notes & Flickr - error checking

HTML5, CSS, & JS - example - part 10

working with Flickr API - modify travel notes JS

- add a modification to check for the image loading and the notes
  - offer status feedback to the user
- remove it when the promise object has returned
  - e.g. request data from image API
  - show loading animation, spinner &c.
  - get data, process images…
  - remove, hide loading animation, spinner &c.
  - render photo cards with images
- DEMO - travel notes & Flickr - spinner

Image - HTML5, CSS, & JS - Travel Notes & Flickr

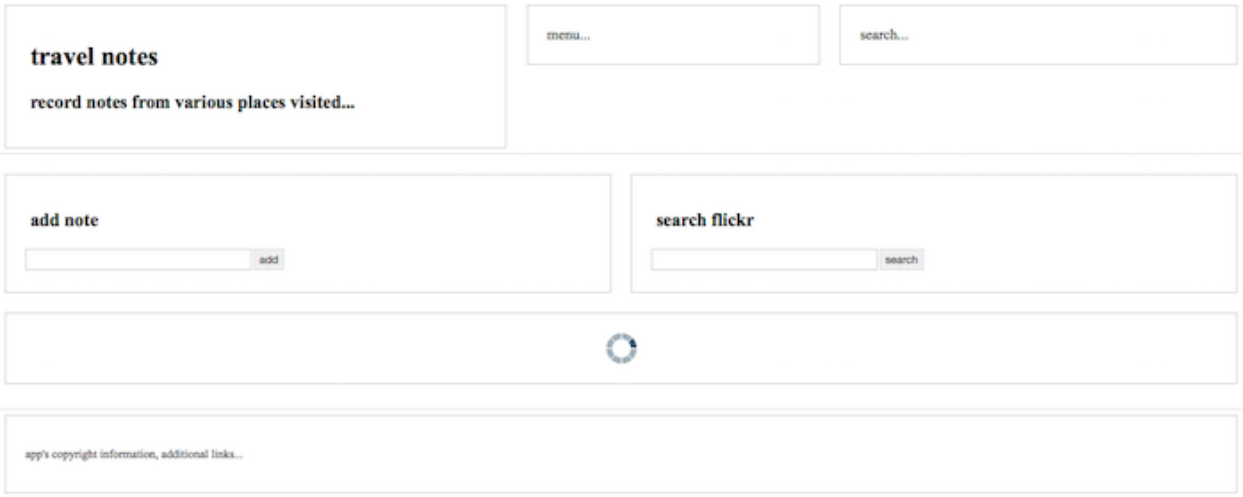

Figure 8: Travel Notes & Flickr - spinner

JavaScript extras - API examples

further API examples

- Google Maps & Places
- Google Distance Matrix
- Google Maps markers & resizing
- Twitter
  - user queries &c.
  - OAuth based login and authentication
- Yelp
  - custom server and remote API query
  - sample handling of local API for queries
- YouTube
  - custom query
  - custom with account authentication
  - search and playback

- …

ES2017 Async & Await

- in ES2017, JavaScript gained native syntax to describe asynchronous operations
- now use *async/await* to work with asynchronous operations
- `Async` functions allow developers to take a promise-based implementation
  - then use synchronous-like patterns of a generator
  - e.g. async implementation with sync usage patterns…
- `await` may only be used inside `async` functions
  - denoted with the `async` keyword
- `async` function works in a similar manner to standard generators
  - e.g. suspending execution in *local* context until a promise settles
- if awaited expression is not originally a promise object
  - it will be cast to a promise in this context…

ES2017 Async & Await - example 1

- example usage with try/catch

```
async function read() {
  // use try/catch to handle errors in awaited promises within async function
  try {
    const model = await getRandomBook();
  } catch (err) {
    console.log(err);
  }
}
// call function as usual
read();
```

- use return Promise object

```
async function read() {
  const model = await getRandomBook();
}
// call function as usual - work with return promise object...
read()
  .then()
```

ES2017 Async & Await - example 2

Node.js and command line

- example usage with command line arguments
  - custom Promise object
  - async/await with try/catch block
  - initial error handling

```
/*
* basic-error.js
* - error handling for async...
```

```
*/

function getArgs() {
  // Node Process command line arguments
  const args = process.argv;
  // custom Promise object with resolve and reject
  return new Promise((resolve, reject) => {
      if (args[2] === 'test') {
        resolve(args);
      } else {
        reject('no args');
      }
  });
}

async function main() {
  try {
    let data = await getArgs();
    return data;
  } catch(e) {
    throw new Error(`main failed...${e}`);
  }
}

main()
.then(console.log)
.catch(console.log);
```

ES2017 Async & Await - example 3

```
// FN: 'fetch' from JSON
function getNotes() {
  return fetch('./assets/files/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json());
}
```

initial `fetch`

ES2017 Async & Await - example 4

- example fetch usage

```
/*
* basic-async1.js
* async called with sync-like try/catch block
* 'awaits' return from fetch to local JSON file
*/
```

```
// FN: 'fetch' from JSON
function getNotes() {
  return fetch('./assets/files/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json());
}

// FN: async/await
async function read() {
  try {
    const notes = await getNotes();
    console.log(`notes FETCH successful`);
  } catch (err) {
    console.log(err);
  }
}

read();
```

- DEMO - Async & Await - Fetch example

ES2017 Async & Await - example 5 - part 1

```
/*
* FNs: iterable computed data
* functions support all major ES6 data structures
* - arrays, typed arrays, maps, sets...
*/

// FN: iterable entries() - default iterator for data structure entries
function dataEntryIterator(data) {
  for (const pair of data.entries()) {
    console.log(pair);
  }
}

// FN: iterable keys() - default iterator for data structure keys
function dataKeysIterator(data) {
  for (const key of data.keys()) {
    console.log(key);
  }
}

// FN: iterable values() - default iterator for data structure values
function dataValuesIterator(data) {
  for (const value of data.values()) {
    console.log(value);
  }
```

```
}
```

sample iterable functions

ES2017 Async & Await - example 5 - part 2

```
// FN: async/await
async function read() {
  try {
    // await return from FETCH for notes.json file
    const data = await getNotes();
    const notes = data['notes'];
    // wrap return notes array in iterator
    const iter = notes[Symbol.iterator]();
    // test iterator with next for each result...
    console.log(iter.next());
    console.log(iter.next());
    console.log(iter.next());
    console.log(iter.next());
    console.log(`notes FETCH successful`);
    dataEntryIterator(notes);
    dataKeysIterator(notes);
    dataValuesIterator(notes);
  } catch (err) {
    console.log(err);
  }
}

read();
```

async and await usage - a bit of fun...

- DEMO - Async & Await - example with iterables

HTML5, CSS, & JS - example - part 1

abstract logic and components - module usage

- app logic and components may be further abstracted and organised
- opportunity to define overlap in logic
  - e.g. between notes and images
- define logic in helper functions
  - e.g. parsers, logging tools, rendering options &c.
- add ES Modules to structure app

Cross-platform - modular design

ES Module pattern - intro

- React Native modules use ES6 module system
- simpler and easier to work with than CommonJS (NodeJS modules…)

- enhance JS syntax to use *static dependency definitions*
  - helps with code sharing across apps
  - improves efficiency of tools for *static code analysis*, *dead-code removal*, and *tree-shaking*
  - unifies module usage for client and server-side
  - optimises compiler analysis of code
- JavaScript `strict` mode is enabled by default
- `strict` mode helps with language usage - check for poor usage
  - stops hoisting of variables
  - variables must be declared
  - function parameters must have unique name
  - assignment to read-only properties throws errors
  - …

Cross-platform - modular design

ES Module pattern - static module

- modules are exported with `export` statements
- modules are imported with `import` statements
- static definitions have various advantages over dynamic
  - allow JS runtime to prefetch / preload modules
  - allow tools to optimise package size for modules
  - e.g. removing code that will never execute…
- `import` statement abstracts process of traversing filesystem
- `import` statements must be declared at top of module
- static helps compilers and build tools
  - helps map app structure ahead of execution
  - helps with tools such as static code analysis &c.

Cross-platform - modular design

ES Module pattern - live bindings

- ESM provides synchronous live binding statements and asynchronous APIs
- ESM uses *immutable*, *live code bindings*
  - CJS uses a regular mutable copy of an object
- exported props bound to defined props in respective module
- binding is bi-directional
  - i.e. a change to exported binding within module affects other bindings
  - i.e. ripple effect to same prop in various places in app…
- every module has its own scope
  - usually one module per file

HTML5, CSS, & JS - example - part 2

abstract logic and components - module usage

- define initial directory structure for module usage

```
.
|-- assets
|-- config
```

```
|-- docs
|-- src
|__ index.html
|__ index.js
```

- app JavaScript and logic called from `index.js`

HTML5, CSS, & JS - example - part 3

abstract logic and components - module usage

- `src` directory for app logic and modules

```
.
|-- src
|   |__ app.js
|   |__ config.js
|   |__ data.js
|   |__ dom.js
|   |__ flickr.js
|   |__ handlers.js
|   |__ notes.js
```

- app is bootstrap using the `app.js` module
  - loads from `index.js`

HTML5, CSS, & JS - example - part 4

abstract logic and components - module usage

- `config` directory for app metadata and config settings &c.

```
.
|-- config
|   |__ app.json
|   |__ config.json
```

- fetch metadata to bootstrap app
  - e.g. default settings, app name &c.
- use `config.json` to test local API settings

HTML5, CSS, & JS - example - part 5

abstract logic and components - module

- `config.json` for app metadata and settings
- also use for local testing of API settings
- production app needs to secure API settings, e.g. keys, secrets &c.
- options include
  - remote server with hashed properties
  - cloud based service for authentication data
  - …

```
{
    "apis": {
        "flickr": {
            "key": "123456",
            "user_id": ""
        }
    },
    "localData": {
        "notes": {
            "dir": "./docs/",
            "file": "notes.json"
        }
    }
}
```

Cross-platform - modular design

ES Module pattern - basic usage

- modules set to `strict` by default
  - no explicit declaration required
- path specifiers must be valid URIs
  - including file endings for `import`

```
// relative paths require explicit context, i.e. ./ ../ &c.
import Validation from './Validation.js';
import Validation from '../Validation.js';
import ( HasValidation ) from '/utilities/Validation.js';
// valid path for browser based module usage
import ( HasHash ) from 'https://myserver.com/utilities/mixins.js';
```

- browser usage now includes extra `type` attribute to specify module

```
<script type="module">
    src="https://myserver.com/utilities/mixins.js"
</script>
```

- ESM does not support directory imports
  - unlike CJS option…

Cross-platform - modular design

ES Module pattern - `export` statements

- ES6 modules are individual files
  - expose an API using `export` statements
- declarations are scoped to the local module
- e.g. variables declared inside a module
  - not available to other modules
  - need to be explicitly exported in module API
  - need to be imported for usage in another module
- export statements may only be added to *top-level* of a module
  - e.g. not in `return` from function expression &c.

- cannot dynamically define and expose API using methods
  - unlike CommonJS module system - Node.js &c.

Cross-platform - modular design

ES Module pattern - `export default`

- `default` used to export single piece of data from module
- common option is to export a default binding, e.g.

```
export default `hello world`
```

```
export default {
    name: 'Alice',
    place: 'Wonderland'
}
```

```
export default [
    'Alice', 'Wonderland'
]
```

```
export default function name() {
    ...
}
```

Cross-platform - modular design

ES Module pattern - bindings

- ES modules export **bindings**
  - not values or references
- e.g. an export of `count` variable from a module
  - `count` is exported as a binding
  - export is bound to `count` variable in the module
  - value is subject to changes of `count` in module
- offers flexibility to exported API
  - e.g. `count` might originally be bound to an object
  - then changed to an array…
- other modules consuming this export
  - they would see change as `count` is modified
  - modified in module and exported…
- **n.b.** take care with this usage pattern
  - useful for counters, logs &c.
  - can cause issues with API usage for a module

HTML5, CSS, & JS - example - part 6

abstract logic and components - module

- define initial `app.js` logic to load app
- helps aggregate abstracted modules to export app

```
...
// main fn - bootstrap app
const app = () => {
...
};

export default app;
```

- then define imports for initial config settings

```
import loadConfig from './config.js';

// main fn - bootstrap app
const app = () => {

    // get config, then load initial app
    const config = loadConfig('../config/config.json').then(data => {
    ...
    })
    .catch(err => {
        console.error(`error = ${err}`);
    });

};
```

- check `config` is available and loads
- then use `data` to call handlers for APIs, notes, datastores &c.

Cross-platform - modular design

ES Module pattern - export lists

- lists provide a useful solution to previous refactor issue
- syntax for list export easy to parse
- export lists of named *top-level* declarations
  - variables &c.
- e.g.

```
let counter = 0
const count = () => counter++
export { counter, count }
```

- also rename binding for export, e.g.

```
let counter = 0
const count = () => counter++
export { counter, count as increment }
```

- define `default` with export list, e.g.

```
let counter = 0
const count = () => counter++
export { counter as default, count as increment }
```

HTML5, CSS, & JS - example - part 7

abstract logic and components - module

- define module for DOM objects, `dom.js`
- useful separation of concern within app structure

```
// get note-output DOM object
const noteOutput = document.getElementById('note-output');
// get object for add note button
const addNoteBtn = document.getElementById('add-note');
// get input field for note
const noteInput = document.getElementById('input-note');
// get input field for image search
const imgInput = document.getElementById('input-image');
//get image-output DOM object
const imgOutput = document.getElementById('image-output');
// get object for image search
const imgSearchBtn = document.getElementById('search-images');

export {
    noteOutput,
    addNoteBtn,
    noteInput,
    imgInput,
    imgOutput,
    imgSearchBtn,
};
```

- single module to define and access DOM object for app usage

---

Cross-platform - modular design

ES Module pattern - proxy `export from ...`

- expose another module's API using `export from...`
  - i.e. proxy, a kind of pass through…
- e.g.

```
export { increment } from './myCounter.js'
```

- bindings are not imported into module's local scope
- current module acts as conduit, passing bindings along export/import chain…
- module does not gain direct access to `export from ...` bindings
  - e.g. if we call `increment` it will throw a `ReferenceError`
- aliases are also possible for bindings with `export from...`
  - e.g.

```
export { increment as addition } from './myCounter.js'
```

Cross-platform - modular design

ES Module pattern - `import` **statements**

- use `import` to load another module
- `import` statement are only allowed in top level of module definition
  - same as `export` statements
  - helps compilers simplify module loading &c.
- import default exports
  - give default export a name as it is imported
  - e.g.

```
import counter from './myCounter.js'
```

- importing binding to `counter`
- syntax different from declaring a JS variable

Cross-platform - modular design

ES Module pattern - `import` named exports

- also imported any named exports
  - import more than just default exports
- named import is wrapped in braces
  - e.g.

```
import { increment } from './myCounter.js'
```

- also import multiple named exports
  - e.g.

```
import { increment, decrement } from './myCounter.js'
```

- import aliases are also supported
  - e.g.

```
import { increment as addition } from './myCounter.js'
```

- combine default with named
  - e.g.

```
import counter, { increment } from './myCounter.js'
```

Cross-platform - modular design

ES Module pattern - `import` with wildcard

- we may also import using the *wildcard* operator
- name for wildcard import acts like object for module
- call module exports on wildcard

```
import * as counter from './myCounter.js'
counter.increment()
```

- common pattern for working with libraries &c.
- namespace used to access library

```
counter.increment();
```

Cross-platform - modular design

ES Module pattern - benefits & practical usage

- offers ability to explicitly publish an API
  - keeps module content local unless explicitly exported
- similar function to *getters* and *setters*
  - explicit way in and out of modules
  - explicit options for reading and updating values…
- code becomes simpler to write and manage
  - module offers encapsulation of code
- import binding to variable, function &c.
  - then use it as normal…
- removes need for encapsulation in main JS code
  - e.g. with patterns such as IIFE…
  - *immediately invoked function expression*
- *n.b.* need to be careful how we use modules
  - e.g. priority for access, security, testing &c.
  - all now moved to individual modules…

Cross-platform - modular design

ES Module pattern - benefits for tooling

- static, declarative structure offers benefits for development, testing &c.
- ESM helps with IDE support
  - e.g. static code checking
- other common benefits include
  - dead-code elimination
  - tree-shaking
  - faster property lookups
  - type friendliness
  - …

HTML5, CSS, & JS - example - part 8

abstract logic and components - module

- define `handlers.js` for logic to handle various app requests
- e.g. handlers include
  - image search
  - load local JSON
  - create a new note
  - add note button
  - add note keypress
  - …

```
// notes handler - load local JSON data
const loadLocalNotes = (data, noteOutput, imgOutput) => {
    console.log(data);

    loadLocalJSON(data.dir, data.file).then(response => response.json())
        .then(jsonData => {
            console.log(jsonData);
```

```
            // build notes from JSON data
            jsonData['notes'].map((val) => {
                const metadata = val.metadata;
                // create note, output to dom
                createNote(val.note, noteOutput, metadata, imgOutput);
            });
        });
};
```

- and example handler for a new note

```
const newNote = (noteInput, noteOutput, imgOutput) => {
    const date = new Date().toUTCString();
    const metadata = {
        "created": date,
        "author": "daisy",
        /*"tags": "note"*/
    };
    createNote(noteInput.value, noteOutput, metadata, imgOutput);
    noteInput.value = "";
};
```

- DEMO - Travel Notes - modules - load local notes

Image - HTML5, CSS, & JS - Travel Notes

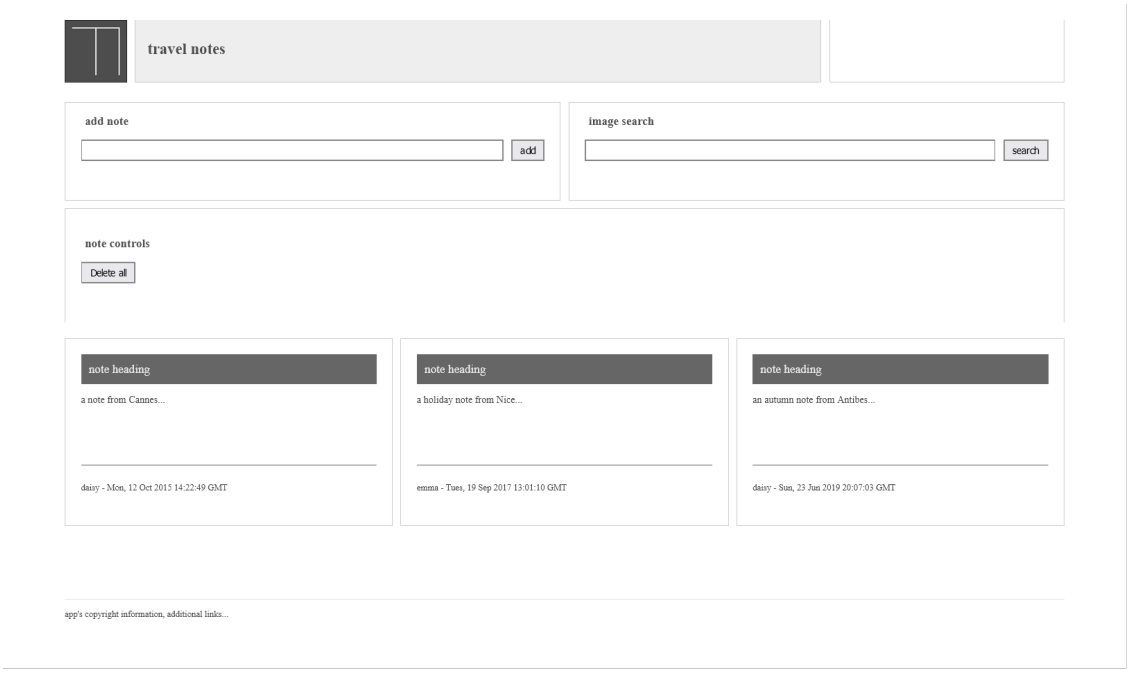

Figure 9: Travel Notes - modules - load local notes

HTML5, CSS, & JS - example - part 9

abstract logic and components - module

- add handler for common events
- then setup and call as part of bootstrap in `app.js`

```
// notes handler - add button
const addNoteBtnHandler = (noteInput, noteOutput, imgOutput) => addNoteBtn.addEventListener('click', ()
    newNote(noteInput, noteOutput, imgOutput);
});
// notes handler - return keypress
const addNoteKeyHandler = (noteInput, noteOutput, imgOutput) => inputNote.addEventListener('keypress',
    if (e.keyCode === 13) {
        newNote(noteInput, noteOutput, imgOutput);
    }
});
```

- and then `export` for use in other modules

```
...
export {
    addNoteBtnHandler,
    addNoteKeyHandler,
    imgSearchHandler,
    imgSearchKeyHandler,
    loadLocalNotes
};
```

- DEMO - Travel Notes - modules - load local notes

Image - HTML5, CSS, & JS - Travel Notes

HTML5, CSS, & JS - example - part 10

abstract logic and components - module

- add module for required methods for notes
  - e.g. create note, build note &c.
- add tested logic from previous versions

```
...
/* FN: createNote
*  - input = value for note
*  - return = DOM node for note output
*/
function createNote(inputVal, noteOutput, metadata, imgOutput) {

....

}
...
```

- and `export` for use in other modules such as `./src/handlers.js`

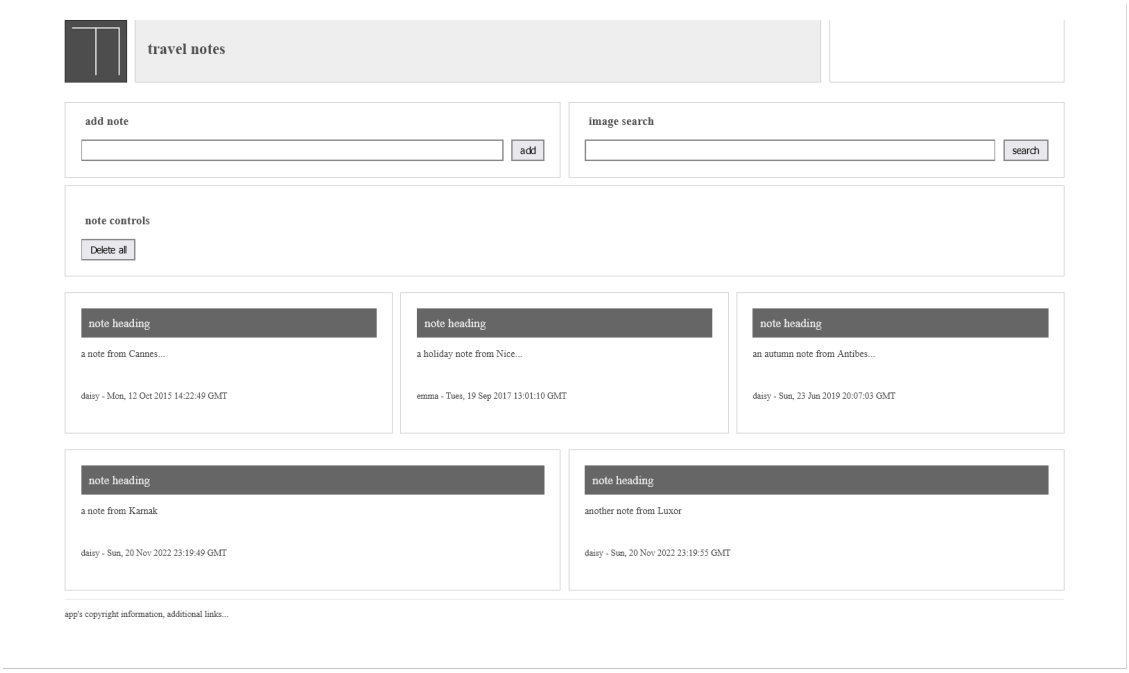

Figure 10: Travel Notes - modules - add notes

```
export {
    createNote
};
```

HTML5, CSS, & JS - example - part 11

abstract logic and components - module

- add module `data.js` for loading data
- includes fetching local and remote data sources

```
const loadLocalJSON = (dir, file) => {
    // add validation, checks &c...
    ...
    // return data for JSON file
    return fetch(dir + file);
};

export {
    loadLocalJSON
};
```

- use to query and load any local data, e.g. JSON file

Resources

JavaScript

- MDN - Fetch API
- MDN - Generators
- MDN - Iterators and Generators
- MDN - Promises
- MDN - JS

Various

- Flickr
  - Flickr API - Public feeds
  - Flickr API - Public feed - public photos & video