INFS3202/7202 – Web Information Systems

Lecture Week 8: RESTful API's and JavaScript

Dr Aneesha Bakharia (Senior Lecturer, EECS)
Email: a.bakharia1@uq.edu.au

Contents

Course Updates

Issue/Feedback	Change
Grading for Design Document	• Underway • Will release Grade on Friday Week 8
Code Review	• In the Week 9 Lab you can work on your project • During the Week 10 Lab you will need to complete your code review with a demonstrator

Code Review Assessment Item Rubric

INFS3202 & INFS7202

Rubric

Web Project Code Review (10%)

No Marks		Half Marks		Full Marks	
A version of the student's *Project* was **not demonstrated** during the Week 10 Lab, or was **substantially incomplete**.	**0.00 %**	A version of the student's *Project* was **demonstrated** during the Week 10 Lab, but was **not near half-complete** & partially functional.	**2.00 %**	A **near half-complete** & partially functional version of the student's *Web Project* was **demonstrated** during the Week 10 Lab.	**4.00 %**
The student **did not demonstrate** an understanding of Web Technologies during the Code Review.	**0.00 %**	The student **demonstrated** some understanding of Web Technologies during the Code Review.	**2.00 %**	The student **demonstrated** a strong understanding of Web Technologies during the Code Review.	**4.00 %**
The student **did not articulate** issues they experienced during the production of there Web Project, with **poor oral** communication evident.	**0.00 %**	The student **articulated** some issues they experienced during the production of their Web Application.	**1.00 %**	The student **articulated** issues they experienced during the production of their Web Application, **reflecting** on how they might make improvements.	**2.00 %**

Week 8 RiPPLE Task

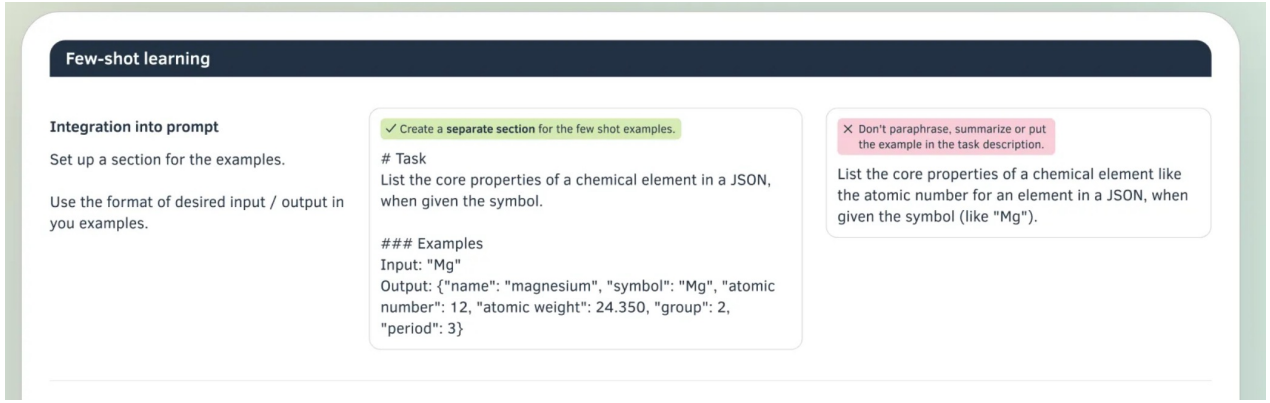
RiPPLE Weekly Activity - Create Resource - Cycle 3

In Lecture 3 and 7 you learnt about writing prompts to help you code and also to incorporate GenAI functionality into your Web application!

For this RiPPLE Create task:

1. **Share a Prompt:** Share 1 prompt using the Note resource type in RiPPLE. The prompt could use a Prompt Engineering technique (e.g. Chain of Thought) or include examples (e.g. Few shot) or be persona based (e.g. Act as a Web Developer). The prompt must be related to Web development (e.g. acting as a tutor to help you learn HTML, CodeIgniter or Databases; assisting with making HTML and CSS or generating Code). You need to include a title for the prompt, the prompt text and reflect on the quality of the output produced (eg did it have errors or work?; would you be able to use it in a project?, what was the quality like?). Use any **free** GenAI chatbot to try the prompt (i.e. OpenAI ChatGPT 3.5, Google Gemini or Microsoft Copilot).
2. **Only 1 prompt is required.**

Due Date: Friday 19 April 2024 at 3pm



Example Prompt Cheatsheet
https://medium.com/the-generator/the-perfect-prompt-prompt-engineering-cheat-sheet-d0b9c62a2bba

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON)

```
{
    "student_id": "123456",
    "student_firstname": "John",
    "student_surname": "Doe"
}
```

- JSON is a lightweight data interchange format that's easy for humans to read and write and easy for machines to parse and generate.
- The structure of JSON includes two types of structures:
  - a collection of key/value pairs and an ordered list of values (array)
- The JSON format is text only and can be easily sent to and from a server (post to an API and get from an API)

JavaScript Object Notation (JSON)

```
{
    "student_id": "123456",
    "student_firstname": "John",
    "student_surname": "Doe"
}
```

- All keys and string values are written in double quotes
- A colon separates each key from its value.
- The entire set of key/value pairs is enclosed in curly braces {}.

JavaScript Object Notation (JSON)

```
{
    "students": [
        {
            "student_id": "001",
            "student_firstname": "John",
            "student_surname": "Doe"
        },
        {
            "student_id": "002",
            "student_firstname": "Jane",
            "student_surname": "Smith"
        },
        {
            "student_id": "003",
            "student_firstname": "Bob",
            "student_surname": "Johnson"
        }
    ]
}
```

```
const students = [
    {
        "student_id": "001",
        "student_firstname": "John",
        "student_surname": "Doe"
    },
    {
        "student_id": "002",
        "student_firstname": "Jane",
        "student_surname": "Smith"
    },
    {
        "student_id": "003",
        "student_firstname": "Bob",
        "student_surname": "Johnson"
    }
];
```

Storing Arrays of Objects

JSON can be Nested

```
const students = [
    {
        "student_id": "001",
        "student_firstname": "John",
        "student_surname": "Doe",
        "quiz_scores": [
            {"quiz_name": "Math Quiz", "score": 85},
            {"quiz_name": "History Quiz", "score": 90}
        ]
    },
    {
        "student_id": "002",
        "student_firstname": "Jane",
        "student_surname": "Smith",
        "quiz_scores": [
            {"quiz_name": "Math Quiz", "score": 92},
            {"quiz_name": "History Quiz", "score": 88}
        ]
    },
    {
        "student_id": "003",
        "student_firstname": "Bob",
        "student_surname": "Johnson",
        "quiz_scores": [
            {"quiz_name": "Math Quiz", "score": 78},
            {"quiz_name": "History Quiz", "score": 80}
        ]
    }
];
```

RESTFul API’s

What is an API?

- **API (Application Programming Interface):**
  A set of rules and protocols for building and interacting with software applications.
- **Bridge for Communication:**
  APIs allow different software systems to communicate with each other without needing to know how they're implemented.
- **Automation and Efficiency:**
  APIs allow routine tasks to be handled automatically without human intervention, increasing efficiency.

How are API’s used in Web Development?

- **Backend Connectivity:**
  APIs serve as the backend framework for the web app, connecting the user interface to server-side databases and processes. This enables the application to retrieve and store data dynamically.
- **Data Services:**
  APIs provide web applications with dynamic access to external data services, such as weather updates, stock prices, or geographic information.
- **Dynamic Content Loading:**
  APIs allow web applications to load content dynamically without the need to reload the entire page. This is often seen in social media feeds, where new content is seamlessly loaded as the user scrolls.
- **Expose your Apps Functionality to Other Sites:**
  Just as you use API’s from other sites, you too can make available services for other to use.

What is a RESTful API?

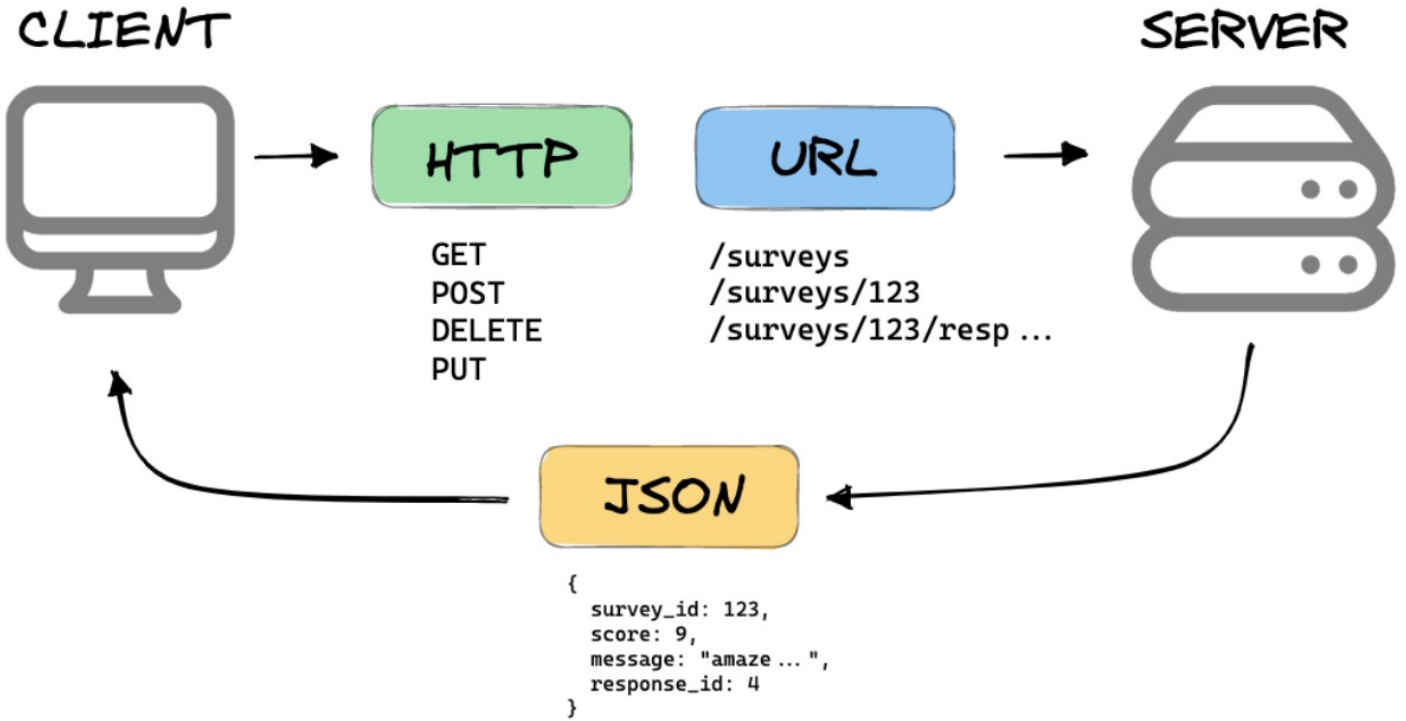
- Representational State Transfer
- Architectural style for distributed systems

- **Stateless Communication**
  - Each request from a client contains all the information needed for processing
- **Resource-Based**
  - Resources are identified by URLs
  - Once you know the resource name the process of listing, creating, editing and deleting is the same
- **HTTP Methods**
  - GET, POST, PUT, DELETE

CRUD with RESTful

- You can perform CRUD (Create, Read, Update, Delete) operations on a resource (eg Item) using various HTTP methods.
- Listing Items (GET)
  https://url/api/item
- Reading an Item by id (GET)
  https://url/api/item/4
- Create an Item (POST) to https://url/api/item
- Edit an Item (PUT) to https://url/api/item/4
- Delete an Item (DELETE) to https://url/api/item/4

WHAT IS A REST API?



mannhowie.com

https://mannhowie.com/rest-api

RESTful API with CodeIgniter (pt 1)

- CodeIgniter makes it easy to make Routes for a RESTFul API
- Add this in your Config/Routes.php file

Resource Routes

You can quickly create a handful of RESTful routes for a single resource with the `resource()` method. This creates the five most common routes needed for full CRUD of a resource: create a new resource, update an existing one, list all of that resource, show a single resource, and delete a single resource. The first parameter is the resource name:

```php
<?php

$routes->resource('photos');

// Equivalent to the following:
$routes->get('photos/new', 'Photos::new');
$routes->post('photos', 'Photos::create');
$routes->get('photos', 'Photos::index');
$routes->get('photos/(:segment)', 'Photos::show/$1');
$routes->get('photos/(:segment)/edit', 'Photos::edit/$1');
$routes->put('photos/(:segment)', 'Photos::update/$1');
$routes->patch('photos/(:segment)', 'Photos::update/$1');
$routes->delete('photos/(:segment)', 'Photos::delete/$1');
```

https://codeigniter4.github.io/CodeIgniter4/incoming/restful.html

RESTful API with CodeIgniter (pt 2)

- CodeIgniter includes common status codes for API's

API Response Trait

Much of modern PHP development requires building APIs, whether simply to provide data for a javascript-heavy single page application, or as a standalone product. CodeIgniter provides an API Response trait that can be used with any controller to make common response types simple, with no need to remember which HTTP status code should be returned for which response types.

- Example Usage
- Handling Response Types
- Class Reference

Example Usage

The following example shows a common usage pattern within your controllers.

```php
<?php

namespace App\Controllers;

use CodeIgniter\API\ResponseTrait;
use CodeIgniter\Controller;

class Users extends Controller
{
    use ResponseTrait;

    public function createUser()
    {
        $model = new UserModel();
        $user  = $model->save($this->request->getPost());

        // Respond with 201 status code
        return $this->respondCreated();
    }
}
```

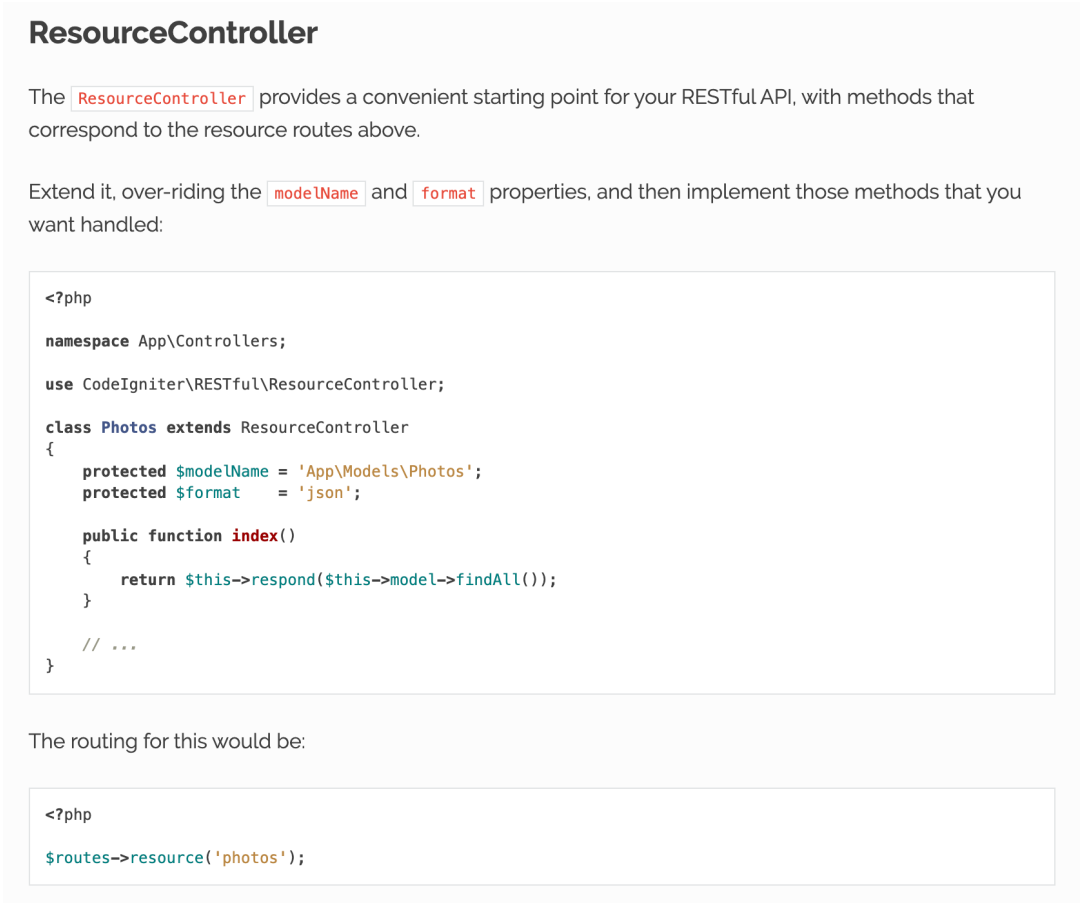
In this example, an HTTP status code of 201 is returned, with the generic status message, 'Created'. Methods exist for the most common use cases:

```php
<?php

// Generic response method
$this->respond($data, 200);

// Generic failure response
$this->fail($errors, 400);

// Item created response
$this->respondCreated($data);

// Item successfully deleted
$this->respondDeleted($data);

// Command executed by no response required
$this->respondNoContent($message);

// Client isn't authorized
$this->failUnauthorized($description);

// Forbidden action
$this->failForbidden($description);

// Resource Not Found
$this->failNotFound($description);

// Data did not validate
$this->failValidationError($description);

// Resource already exists
$this->failResourceExists($description);

// Resource previously deleted
$this->failResourceGone($description);

// Client made too many requests
$this->failTooManyRequests($description);
```

https://codeigniter4.github.io/CodeIgniter4/outgoing/api_responses.html

RESTful API with CodeIgniter (pt 3)

- CodeIgniter includes a ResourceController that we can extend



ResourceController

The `ResourceController` provides a convenient starting point for your RESTful API, with methods that correspond to the resource routes above.

Extend it, over-riding the `modelName` and `format` properties, and then implement those methods that you want handled:

```php
<?php

namespace App\Controllers;

use CodeIgniter\RESTful\ResourceController;

class Photos extends ResourceController
{
    protected $modelName = 'App\Models\Photos';
    protected $format    = 'json';

    public function index()
    {
        return $this->respond($this->model->findAll());
    }

    // ...
}
```

The routing for this would be:

```php
<?php

$routes->resource('photos');
```

https://codeigniter.com/user_guide/incoming/restful.html#resourcecontroller

Let’s Build a RESTful API for an Item

- We will make a TodoList application
- A TodoList has Items
- The RESTful Resource will be for an Item
- We first need an Item table with `item` and `completed` fields
- We make a Migration and a Model
  - `php spark make:migration create_items_table`
  - `php spark migrate`
  - `php spark make:model Items`

The Migration file

- You can also make a table using SQL Create
  or
  Use the GUI in PHPMyAdmin
- Insert some records

cidemo > app > Database > Migrations > 2024-04-13-094755_CreateItemsTable.php

```php
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class CreateItemsTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => [
                'type'           => 'INT',
                'constraint'     => 5,
                'unsigned'       => true,
                'auto_increment' => true,
            ],
            'item' => [
                'type'       => 'TEXT',
                'null'       => false,
            ],
            'completed' => [
                'type'       => 'BOOLEAN',
                'default'    => false,
            ]
        ]);

        $this->forge->addKey('id', true);
        $this->forge->createTable('items');
    }

    public function down()
    {
        $this->forge->dropTable('items');
    }
}
```

The Items Model

cidemo > app > Models > Items.php

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class Items extends Model
{
    protected $table            = 'items';
    protected $primaryKey       = 'id';
    protected $useAutoIncrement = true;
    protected $returnType       = 'array';
    protected $allowedFields = ['id', 'item', 'completed'];

}

```

The Item RESTFul Route/s

- CodeIgniter makes it easy to make Routes for a RESTFul API
- Add this in your Config/Routes.php file

- This makes all the RESTful Routes

The Item Controller (pt 1)

cidemo > app > Controllers > Item.php

```php
<?php namespace App\Controllers;

use CodeIgniter\RESTful\ResourceController;
use CodeIgniter\API\ResponseTrait;
use App\Models\EducationModel;

class Item extends ResourceController
{
    use ResponseTrait;

    protected $modelName = 'App\Models\Items';

    /**
     * Handle GET requests to list education entries or filter by user_id.
     */
    public function index()
    {
        // Retrieve all entries.
        $data = $this->model->findAll();

        // Use HTTP 200 to return data.
        return $this->respond($data);
    }

    /**
     * Handle GET requests to retrieve a single education entry by its ID.
     */
    public function show($id = null)
    {
        // Attempt to retrieve the specific education entry by ID.
        $data = $this->model->find($id);

        // Check if data was found.
        if ($data) {
            return $this->respond($data);
        } else {
            // Return a 404 error if no data is found.
            return $this->failNotFound("No Item entry found with ID: {$id}");
        }
    }
```

The Item Controller (pt 2)

cidemo > app > Controllers > Item.php

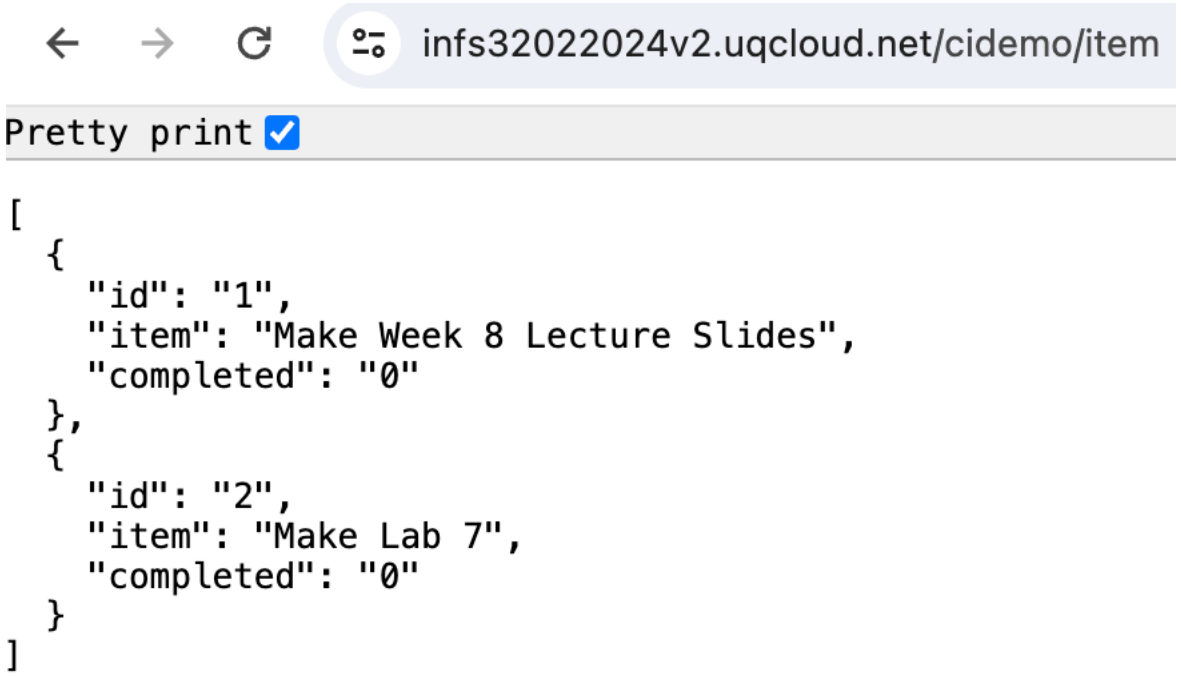
```php

    /**
     * Handle POST requests to create a new education entry.
     */
    public function create()
    {
        $data = $this->request->getJSON(true);  // Ensure the received data is an array.

        // Validate input data before insertion.
        if (empty($data)) {
            return $this->failValidationErrors('No data provided.');
        }

        // Insert data and check for success.
        $inserted = $this->model->insert($data);
        if ($inserted) {
            return $this->respondCreated($data, 'Item data created successfully.');
        } else {
            return $this->failServerError('Failed to create education data.');
        }
    }

    /**
     * Handle PUT requests to update an existing education entry by its ID.
     */
    public function update($id = null)
    {
        $data = $this->request->getJSON(true);  // Ensure the received data is an array.

        // Check if the record exists before attempting update.
        if (!$this->model->find($id)) {
            return $this->failNotFound("No Item entry found with ID: {$id}");
        }

        // Update the record and handle the response.
        if ($this->model->update($id, $data)) {
            return $this->respondUpdated($data, 'Item data updated successfully.');
        } else {
            return $this->failServerError('Failed to update Item data.');
        }
    }

```

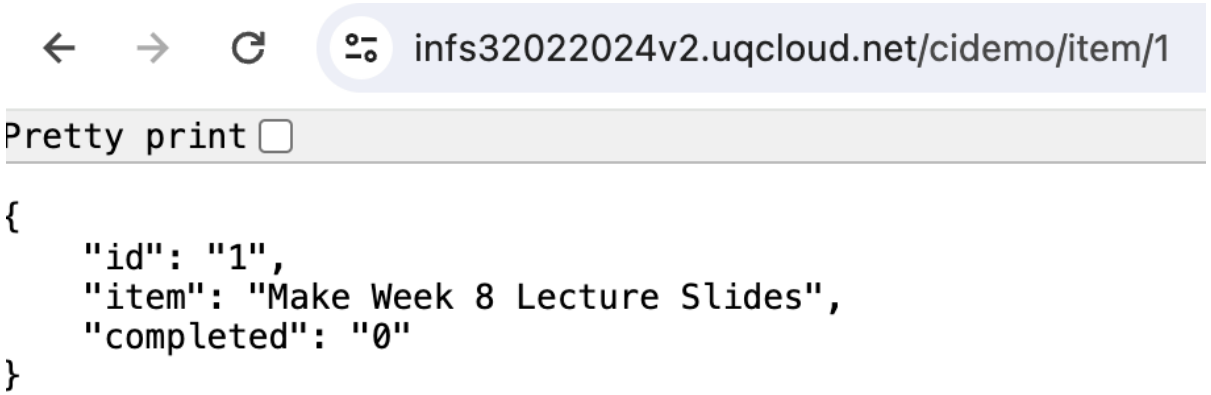
The Item Controller (pt 3)

cidemo > app > Controllers > Item.php

```php
    /**
     * Handle DELETE requests to remove an existing education entry by its ID.
     */
    public function delete($id = null)
    {
        // Check if the record exists before attempting deletion.
        if (!$this->model->find($id)) {
            return $this->failNotFound("No Item entry found with ID: {$id}");
        }

        // Attempt to delete the record.
        if ($this->model->delete($id)) {
            return $this->respondDeleted(['id' => $id, 'message' => 'Item data deleted successfully.']);
        } else {
            return $this->failServerError('Failed to delete Item data.');
        }
    }
}
```

Test the Item RESTful API





JavaScript Recap

Javascript – Basic Syntax - Variables

Building Block	JavaScript Example

Variable Assignment	const y = 2; //Declare a constact
	let x = 5; // Block scoped variable that can be changed

String	let myString = "Hello, world!";
Integer	let myInt = 10;
Float	let myFloat = 20.5;
Boolean	let myBool = true;
Array/List	let myList = [1, 2, 3, 4, 5];
Object	let myObject = {'key1': 'value1', 'key2': 'value2'};

Javascript – Basic Syntax – Conditionals

Building Block	**JavaScript Example**
Conditional (if)	if (x > 5) { console.log("x is greater than 5"); }
Conditional (if-else)	if (x > 5) { console.log("x is greater than 5"); } else { console.log("x is less than or equal to 5"); }
Conditional (if-else if-else)	if (x > 5) { console.log("x is greater than 5"); } else if (x == 5) { console.log("x is equal to 5"); } else { console.log("x is less than 5"); }

Javascript – Basic Syntax - Loops

Building Block	JavaScript Example
Loop (for)	for (let i = 0; i < myList.length; i++) { console.log(myList[i]); }
Loop (while)	while (x < 10) { console.log(x); x++; }

Javascript – Functions

Building Block	JavaScript Example

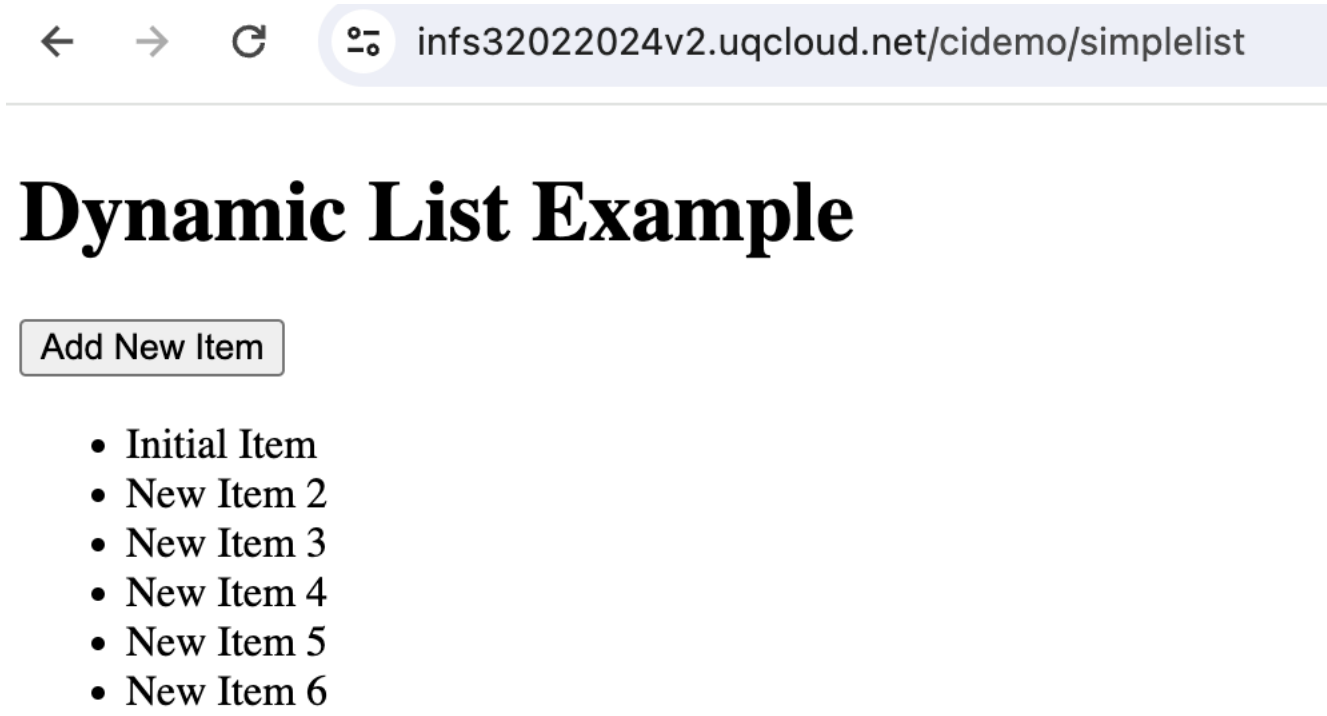
Functions	function myFunction() { console.log("This is a function"); }
	
	let square = x => x * x;
	
	let add = (a, b) => a + b;
	
	let getObject = () => ({ name: "John", age: 30 });
	
	console.log(getObject()); // Output: { name: "John", age: 30 }

	function createGreeting(name) {
	  return `Hello, my name is ${name}.`;
	}
	
	const greeting = createGreeting("Alice");
	console.log(greeting);

Adding List Items (pt 1)

cidemo > app > Views > simplelist.php

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add List Item</title>
</head>
<body>
    <h1>Dynamic List Example</h1>
    <button id="addItem">Add New Item</button>
    <ul id="itemList">
        <li>Initial Item</li>
    </ul>

    <script>
        // Access the button and the list by their IDs
        const addButton = document.getElementById('addItem');
        const itemList = document.getElementById('itemList');

        // Add an event listener to the button for click events
        addButton.addEventListener('click', function() {
            // Create a new list item element
            const newItem = document.createElement('li');

            // Add some text to the new item
            newItem.textContent = 'New Item ' + (itemList.children.length + 1);

            // Append the new item to the list
            itemList.appendChild(newItem);
        });
    </script>
</body>
</html>

```



Adding List Items (pt 2)

cidemo > app > Views > simplelist.php

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add List Item</title>
</head>
<body>
    <h1>Dynamic List Example</h1>
    <button id="addItem">Add New Item</button>
    <ul id="itemList">
        <li>Initial Item</li>
    </ul>

    <script>
        // Access the button and the list by their IDs
        const addButton = document.getElementById('addItem');
        const itemList = document.getElementById('itemList');

        // Add an event listener to the button for click events
        addButton.addEventListener('click', function() {
            // Create a new list item element
            const newItem = document.createElement('li');

            // Add some text to the new item
            newItem.textContent = 'New Item ' + (itemList.children.length + 1);

            // Append the new item to the list
            itemList.appendChild(newItem);
        });
    </script>
</body>
</html>

```

How it works:

- **HTML Structure**:

  The page includes a button labeled "Add New Item" and an unordered list (<ul>) with an initial list item.

- **JavaScript Logic**:

  When the button is clicked, the JavaScript code executes:

  - It creates a new <li> element.
  - It sets the text of the new list item to "New Item" followed by the number, which is determined by the number of existing children in the list plus one.
  - It then appends this new list item to the existing list (<ul>).

HTML Template Tag & Javascript

Adding List Items (pt 1)

cidemo > app > Views > htmltemplatetag.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Template Example</title>
</head>
<body>
    <h1>Dynamic Table Example</h1>
    <button onclick="addRow()">Add New Row</button>
    <table id="dataTable" border="1">
        <tr>
            <th>Column 1</th>
            <th>Column 2</th>
        </tr>
    </table>

    <!-- Template for table rows -->
    <template id="rowTemplate">
        <tr>
            <td>Column 1, New Data</td>
            <td>Column 2, New Data</td>
        </tr>
    </template>

    <script>
        function addRow() {
            // Get the template element
            const template = document.getElementById('rowTemplate');

            // Get the contents of the template
            const templateContent = template.content.cloneNode(true);

            // Insert new data values into the template clone if necessary
            templateContent.querySelector('td').textContent = 'New Entry ' + (document.getElementById('dataTable').rows.length);
            templateContent.querySelectorAll('td')[1].textContent = 'More Data ' + (document.getElementById('dataTable').rows.length);

            // Append the cloned template to the table
            document.getElementById('dataTable').appendChild(templateContent);
        }
    </script>
</body>
</html>
```

Dynamic Table Example

Add New Row

Column 1	Column 2
New Entry 1	More Data 1
New Entry 2	More Data 2
New Entry 3	More Data 3
New Entry 4	More Data 4
New Entry 5	More Data 5
New Entry 6	More Data 6

Adding List Items (pt 1)

cidemo > app > Views > htmltemplatetag.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Template Example</title>
</head>
<body>
    <h1>Dynamic Table Example</h1>
    <button onclick="addRow()">Add New Row</button>
    <table id="dataTable" border="1">
        <tr>
            <th>Column 1</th>
            <th>Column 2</th>
        </tr>
    </table>

    <!-- Template for table rows -->
    <template id="rowTemplate">
        <tr>
            <td>Column 1, New Data</td>
            <td>Column 2, New Data</td>
        </tr>
    </template>

    <script>
        function addRow() {
            // Get the template element
            const template = document.getElementById('rowTemplate');

            // Get the contents of the template
            const templateContent = template.content.cloneNode(true);

            // Insert new data values into the template clone if necessary
            templateContent.querySelector('td').textContent = 'New Entry ' + (document.getElementById('dataTable').rows.length);
            templateContent.querySelectorAll('td')[1].textContent = 'More Data ' + (document.getElementById('dataTable').rows.length);

            // Append the cloned template to the table
            document.getElementById('dataTable').appendChild(templateContent);
        }
    </script>
</body>
</html>
```

How This Works:

- **HTML Structure**: The page includes a button and a table with headers. Below the table, there's a <template> tag containing a template for the table rows.
- **Template Tag**: Inside the <template> tag, there's a predefined table row (<tr>) with two cells (<td>). This row will not be displayed on the page until it is instantiated by JavaScript.
- **JavaScript Functionality**:
  The function addRow() is triggered when the user clicks the "Add New Row" button.
  This function retrieves the template's content and clones it.
  It modifies the text content of the table cells to include dynamic data based on how many rows are currently in the table.
  Finally, the cloned and updated template content is appended to the existing table.

JavaScript Fetch API

Cats or Dogs

Cats or Dogs

https://api.thecatapi.com/v1/breeds

https://api.thedogapi.com/v1/breeds

Free API with facts about different dog/cat breeds

https://api.thedogapi.com/v1/breeds

Fetch API

```
fetch('https://api.thedogapi.com/v1/breeds')
    .then( (response) => {
        console.log("resolved", response);
    })
    .catch( (err) => {
        console.log("rejected", err);
    })
```

Browser Console

```
resolved ▼ Response i
            body: (...)
            bodyUsed: false
          ▶ headers: Headers {}
            ok: true
            redirected: false
            status: 200
            statusText: ""
            type: "cors"
            url: "https://api.thedogapi.com/v1/breeds"
          ▶ [[Prototype]]: Response
```

- Returns a Promise (which you can think of like a real promise) – at some point it will resolve for a success or reject if there is an error

Fetch API

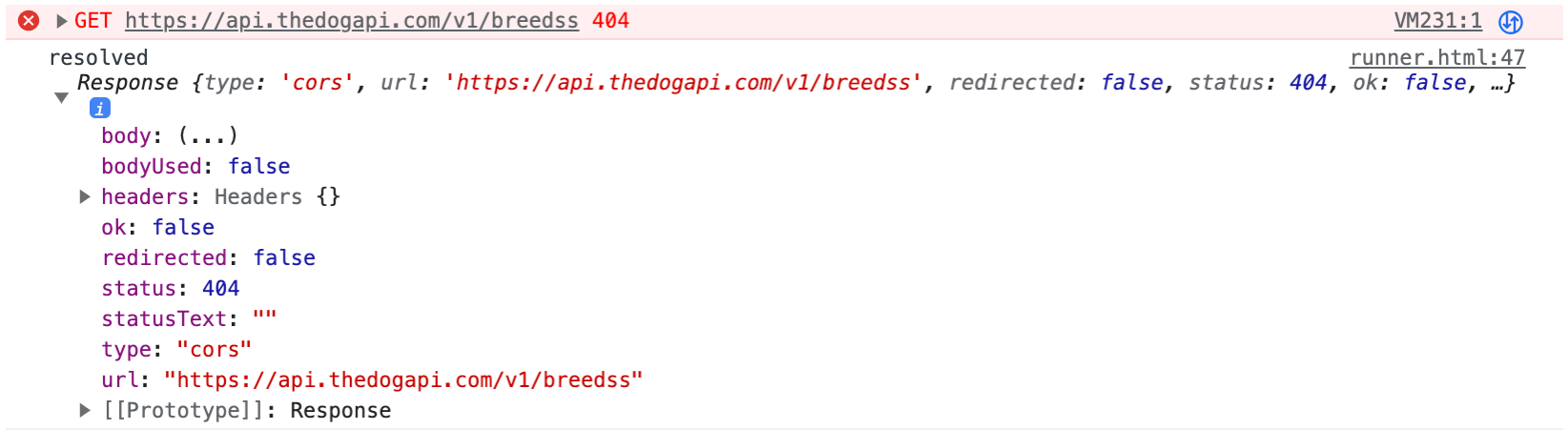
```
fetch('https://api.thedogapi.com/v1/breedss')
    .then( (response) => {
        console.log("resolved", response);
    })
    .catch( (err) => {
        console.log("rejected", err);
    })
```

- What if we have an error in the API endpoint URL?

Fetch API

```
fetch('https://api.thedogapi.com/v1/breedss')
    .then( (response) => {
        console.log("resolved", response);
    })
    .catch( (err) => {
        console.log("rejected", err);
    })
```

- What if we have an error in the API endpoint URL?

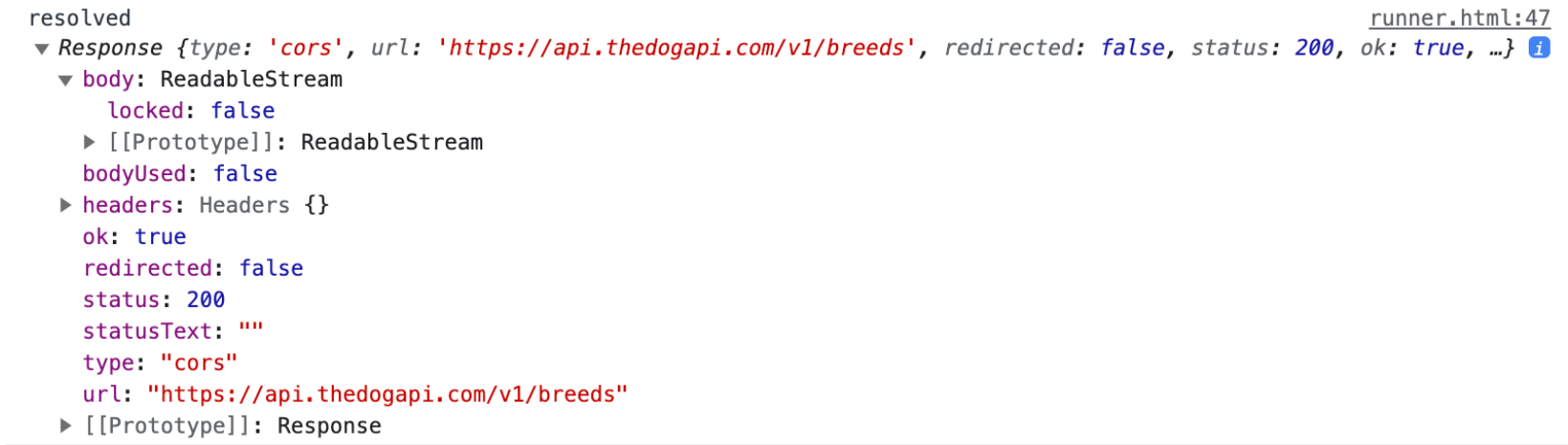


- We get a 404 error but get resolved! We will only get an error if the endpoint can't be reached or there is no internet network connection.

Fetch API

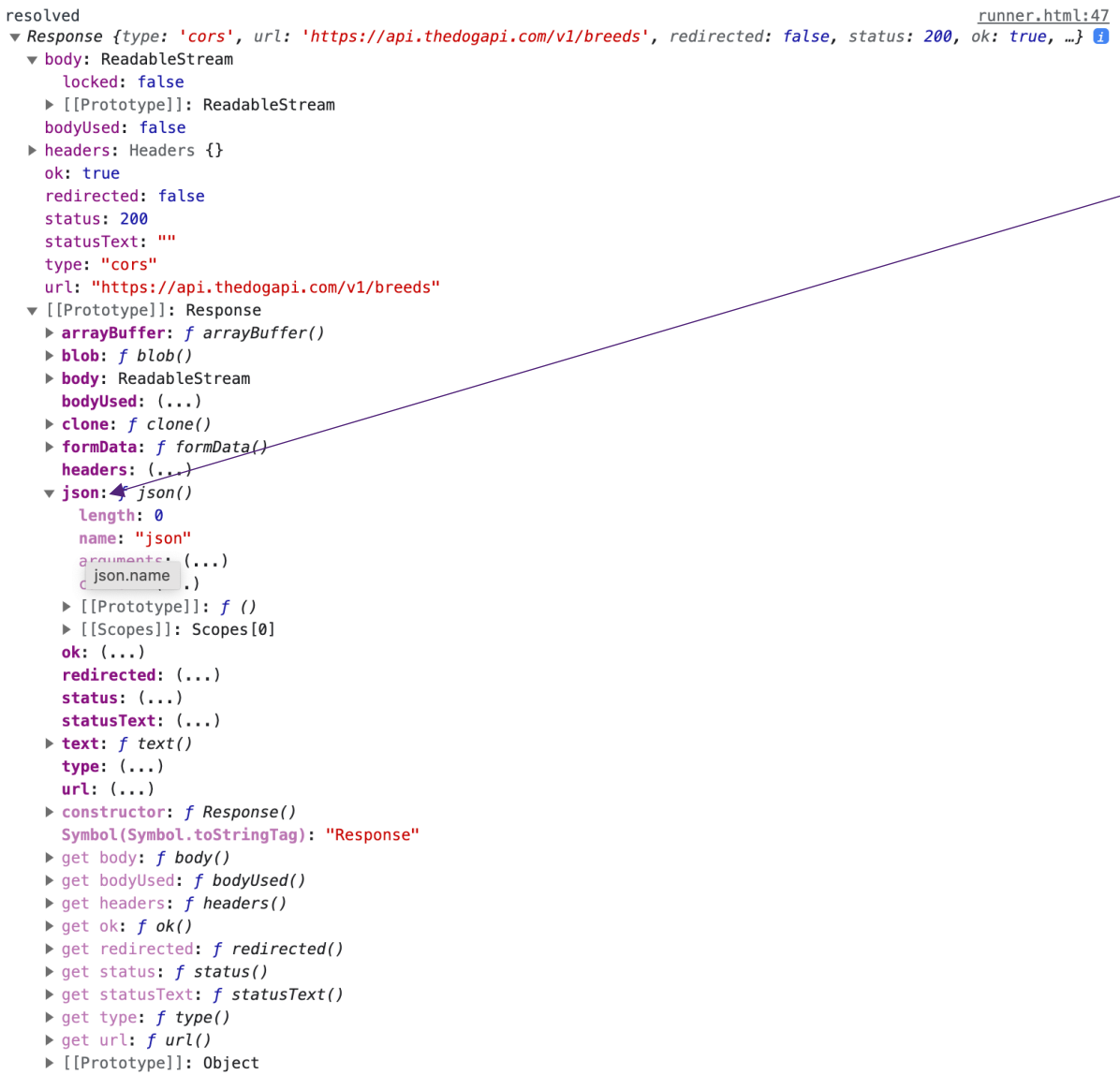
```
fetch('https://api.thedogapi.com/v1/breedss')
    .then( (response) => {
        console.log("resolved", response);
    })
    .catch( (err) => {
        console.log("rejected", err);
    })
```

- How do we get the returned JSON?



Fetch API

- How do we get the returned JSON?



Response.json will return the json

Fetch API

- How do we get the returned JSON?

```
fetch('https://api.thedogapi.com/v1/breeds')
    .then( (response) => {
        console.log("resolved", response);
        return response.json(); // returns a promise
    })
    .then( data => {
        console.log(data)
    })
    .catch( (err) => {
        console.log("rejected", err);
    })
```

Fetch API

- How do we get the returned JSON?

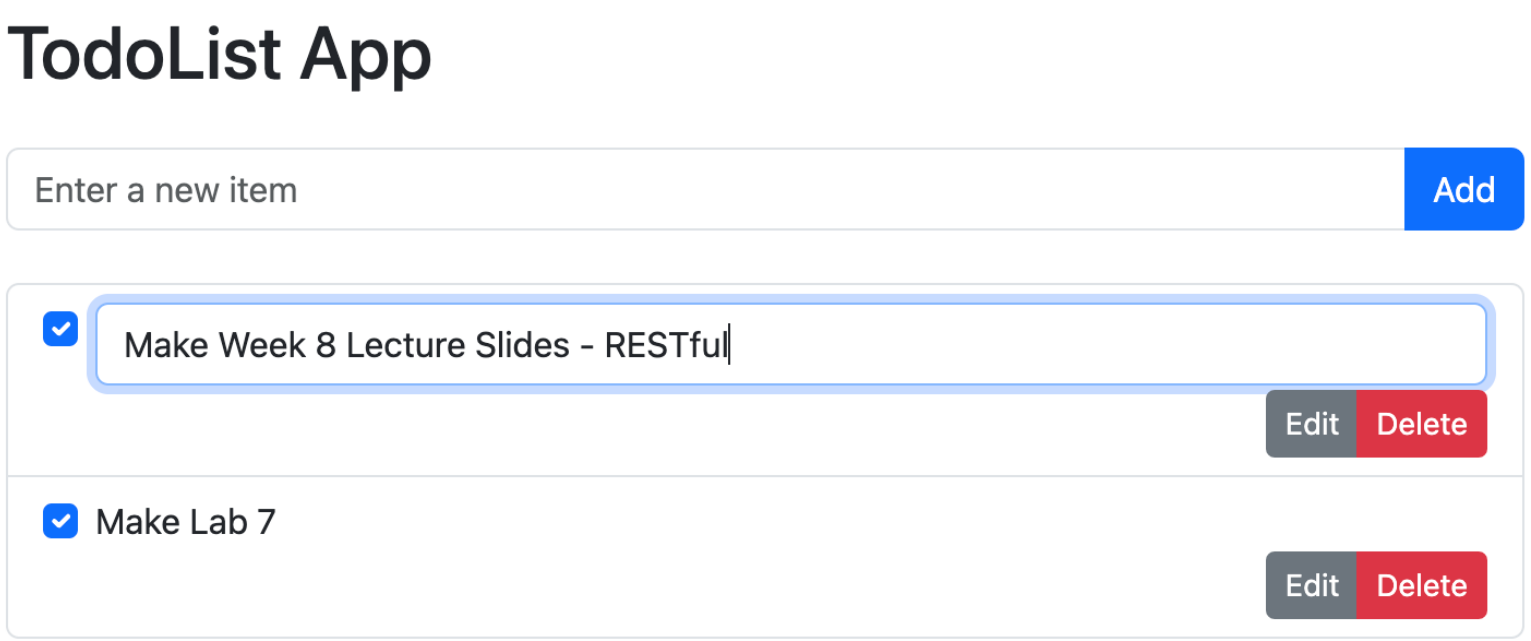
```
resolved                                                                    runner.html:47
▶ Response {type: 'cors', url: 'https://api.thedogapi.com/v1/breeds', redirected: false, status: 200, ok: true, …}

                                                                            runner.html:47
(172) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
▼ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, …] i
  ▼ [0 … 99]
    ▶ 0: {weight: {…}, height: {…}, id: 1, name: 'Affenpinscher', bred_for: 'Small rodent hunting, lapdog', …}
    ▶ 1: {weight: {…}, height: {…}, id: 2, name: 'Afghan Hound', country_code: 'AG', …}
    ▶ 2: {weight: {…}, height: {…}, id: 3, name: 'African Hunting Dog', bred_for: 'A wild pack animal', …}
    ▶ 3: {weight: {…}, height: {…}, id: 4, name: 'Airedale Terrier', bred_for: 'Badger, otter hunting', …}
    ▶ 4: {weight Object height: {…}, id: 5, name: 'Akbash Dog', bred_for: 'Sheep guarding', …}
    ▶ 5: {weight: {…}, height: {…}, id: 6, name: 'Akita', bred_for: 'Hunting bears', …}
    ▶ 6: {weight: {…}, height: {…}, id: 7, name: 'Alapaha Blue Blood Bulldog', description: 'The Alapaha Blue Blood Bu
    ▶ 7: {weight: {…}, height: {…}, id: 8, name: 'Alaskan Husky', bred_for: 'Sled pulling', …}
    ▶ 8: {weight: {…}, height: {…}, id: 9, name: 'Alaskan Malamute', bred_for: 'Hauling heavy freight, Sled pulling',
    ▶ 9: {weight: {…}, height: {…}, id: 10, name: 'American Bulldog', breed_group: 'Working', …}
    ▶ 10: {weight: {…}, height: {…}, id: 11, name: 'American Bully', country_code: 'US', …}
    ▶ 11: {weight: {…}, height: {…}, id: 12, name: 'American Eskimo Dog', country_code: 'US', …}
    ▶ 12: {weight: {…}, height: {…}, id: 13, name: 'American Eskimo Dog (Miniature)', country_code: 'US', …}
    ▶ 13: {weight: {…}, height: {…}, id: 14, name: 'American Foxhound', country_code: 'US', …}
    ▶ 14: {weight: {…}, height: {…}, id: 15, name: 'American Pit Bull Terrier', country_code: 'US', …}
    ▶ 15: {weight: {…}, height: {…}, id: 16, name: 'American Staffordshire Terrier', country_code: 'US', …}
    ▶ 16: {weight: {…}, height: {…}, id: 17, name: 'American Water Spaniel', country_code: 'US', …}
    ▶ 17: {weight: {…}, height: {…}, id: 18, name: 'Anatolian Shepherd Dog', bred_for: 'Livestock herding', …}
    ▶ 18: {weight: {…}, height: {…}, id: 19, name: 'Appenzeller Sennenhund', bred_for: 'Herding livestock, pulling car
    ▶ 19: {weight: {…}, height: {…}, id: 21, name: 'Australian Cattle Dog', country_code: 'AU', …}
    ▶ 20: {weight: {…}, height: {…}, id: 22, name: 'Australian Kelpie', country_code: 'AU', …}
    ▶ 21: {weight: {…}, height: {…}, id: 23, name: 'Australian Shepherd', country_code: 'AU', …}
    ▶ 22: {weight: {…}, height: {…}, id: 24, name: 'Australian Terrier', country_code: 'AU', …}
    ▶ 23: {weight: {…}, height: {…}, id: 25, name: 'Azawakh', bred_for: 'Livestock guardian, hunting', …}
```

Fetch API

- Also used to send (post) data back to an API endpoint
- An example would be to post data from a form and use an API to save the data

```
const url = 'https://jsonplaceholder.typicode.com/posts';

const data = {
  title: 'foo',
  body: 'bar',
  userId: 1
};

fetch(url, {
  method: 'POST',
  body: JSON.stringify(data),
  headers: {
    'Content-Type': 'application/json'
  }
}).then(response => response.json())
  .then(json => console.log(json))
  .catch((error) => console.error('Error:', error));
```

Todo List
Demo Walkthrough
(Uses FetchAPI with Items RESTful API)



Lab8_code/todolist

Week 8: Todo

- Weekly RiPPLE task is due in Week 8 on Friday at 3pm
- Complete Week 8 Lab 7
  - Covers creating a RESTful API, Fetch API and Google Social Login
- Continue to work on your Project
  - Code Review in Week 10

Thank you