REST

This section is not specific to Vue.

Overview

- Stands for REpresentational State Transfer
- An architectural style, not a standard or API
- Described in Roy Fielding's dissertation in 2000 (chapter 5)
 - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Main ideas
 - a software component requests a "resource" from a service
 - by supplying a resource identifier and a desired media type
 - resource identifier can be a URL and request can be made using Ajax
 - a "representation" of the resource is returned
 - a sequence of bytes and metadata to describe it (can be JSON; name/value pairs in HTTP headers, ...)

2

- can contain identifiers of other resources
- obtaining this representation causes the software component to "transfer" to a new "state"

Typical REST

- Relies on HTTP
- Uses carefully selected URLs to identify resources
- Uses HTTP methods for specific operations | a.k.a. HTTP verbs

primarily for Create-Retrieve-Update-Delete (CRUD) operations

- **GET** retrieves representation of existing resource identified by URL
- **POST** <u>creates</u> a new resource using data in body
 - URL of new resource is not yet known
 - often the URL of the request identifies an existing parent resource
- **PUT** <u>updates</u> existing resource identified by URL using data in body

return URL of new resource in Location header and status 201 Created

- URL of resource is already known
- like POST, body contains entire description of resource
- can also create a new resource if its URL is already known
- **PATCH** like PUT, but body only describes changes
 - other properties retain their current value
- **DELETE** deletes existing resource identified by URL
- **POST** for everything else (non-CRUD operations)

ex. POST to URL for a music artist, supplying data describing a new album; URL of new album resource is returned

Idempotent



- Means doing something multiple times is no different than doing it once
 - in REST this applies to server-side state and responses
- GET, PUT, and DELETE requests should be idempotent
 - it's possible for data returned by repeated GET requests to differ,
 but the differences shouldn't be caused by processing the GET request
- POST requests are not necessarily idempotent
 - each request can cause different changes to server state based on the state at the time the request is processed
 - POST examples
 - can create a new resource and each can be assigned a different identifier
 - can order an item; remaining inventory changes, so subsequent requests can fail
- These rules aren't enforced,
 but violating them goes against HTTP specification

Sending HTTP Requests

- Can use low-level XMLHttpRequest approach
- Typically a library is used instead
- Many choices
 - Fetch, axios, superagent, jQuery, ...
- We'll focus on Fetch

Fetch API

- "The Fetch Standard defines the fetch() JavaScript API, which exposes ... networking functionality at a fairly low level of abstraction."
- A Web Hypertext Application Technology Working Group (WHATWG) standard
 - https://fetch.spec.whatwg.org/
- Browser support
 - Chrome, Firefox, Edge, and Safari, but not IE

Fetch Polyfill

7



- https://github.com/github/fetch
- Works in IE9+
- To install
 - npm install --save whatwg-fetch
- To make **fetch** function available globally
 - import 'whatwg-fetch';
- HTTP method names can be lowercase
 - except for PATCH, so just make all uppercase to avoid confusion

GET

To send a GET request to obtain text

```
try {
  const res = await fetch(restUrl);
  if (!res.ok) throw new Error(res.statusText);
  const text = await res.text();
  // Do something with text.
} catch (e) {
  // Handle error.
}
```

assumes in an async function

To send a GET request to obtain JSON

```
try {
  const res = await fetch(restUrl);
  if (!res.ok) throw new Error(res.statusText);
  const obj = await res.json();
  // Do something with obj.
} catch (e) {
  // Handle error.
}
```

assumes in an async function

DELETE

To send a DELETE request

```
try {
  const res = await fetch(restUrl, {method: 'DELETE'});
  if (!res.ok) throw new Error(res.statusText);
  // Do something after successful delete.
} catch (e) {
  // Handle error.
}
```

assumes in an async function

POST/PUT/PATCH Text

To send a POST (or PUT or PATCH) request with a text body

```
try {
  const res = await fetch(restUrl, {
    method: 'POST',
    headers:{'Content-Type': 'text/plain'},
    body: text
  });
  if (!res.ok) throw new Error(res.statusText);
  // Do something with success response.
} catch (e) {
  // Handle error.
}
```

assumes in an async function

POST/PUT/PATCH JSON

To send a POST (or PUT or PATCH) request with a JSON body

```
try {
  const res = await fetch(restUrl, {
    method: 'POST',
    headers:{'Content-Type': 'application/json'},
    body: JSON.stringify(obj)
  });
  if (!res.ok) throw new Error(res.statusText);
  // Do something with success response.
} catch (e) {
  // Handle error.
}
```

assumes in an async function

can also send a POST with no body

can also add Accept header to describe acceptable response MIME types

Fetch Caveats

- From polyfill documentation
 - "The Promise returned from fetch() won't reject on HTTP error status even if the response is a HTTP 404 or 500. Instead, it will resolve normally, and it will only reject on network failure, or if anything prevented the request from completing."

check res.ok Or res.status in success handler

- "By default, fetch won't send any cookies to the server, resulting in unauthenticated requests if the site relies on maintaining a user session."
 - "To automatically send cookies for the current domain, the credentials option must be provided"

Making Fetch Easier ...



- Using some utility functions helps
- Recommend placing these in fetch-util.js
 and importing where needed

```
// Change this to match the URL prefix of your REST services.
// If your project uses REST services with more than one URL prefix,
// drop use URL_PREFIX and just pass full URLs into the functions.
const URL_PREFIX = 'http://localhost:1234/';

// If there are any common options that are
// desired in all HTTP requests, place them here.
const options = {};

// Can't name this "delete" because that is a JavaScript keyword.
export async function deleteResource(urlSuffix) {
   const url = URL_PREFIX + urlSuffix;
   return fetch(url, {...options, method: 'DELETE'});
}
```

... Making Fetch Easier ...



```
export async function getJson(urlSuffix) {
  const url = URL_PREFIX + urlSuffix;
  const res = await fetch(url, options);
  if (!res.ok) {
    const text = await res.text();
    throw new Error(text || res.statusText);
  handles case where response body contains error message text
}
  return res.json();
}

export async function getText(urlSuffix) {
  const url = URL_PREFIX + urlSuffix;
  const res = await fetch(url, options);
  const text = await res.text();
  if (!res.ok) throw new Error(text || res.statusText);
  return text;
}
```

... Making Fetch Easier

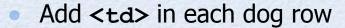


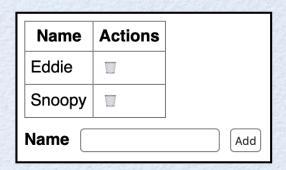
```
export function postJson(urlSuffix, obj) {
  return postPutJson('post', urlSuffix, obj);
export function putJson(urlSuffix, obj) {
  return postPutJson('PUT', urlSuffix, obj);
export function postPutJson(method, urlSuffix, obj) {
  const url = URL PREFIX + urlSuffix;
  const headers = {'Content-Type': 'application/json'};
  const body = JSON.stringify(obj);
  const res = fetch(url, {...options, method, headers, body});
  if (!res.ok) {
    const text = await res.text();
    throw new Error(text || res.statusText);
  return res.json();
```

Exercise ...

- Modify dog app to use supplied REST services
- Add <+h> for Actions column

```
Actions
```





- Get dogs in mounted method
- Modify addDog method to send POST request to add dog
- Add deleteDog method that sends DELETE request to delete dog
- Solution on last two slides
- It's helpful if you know about JavaScript async/await

.. Exercise

- Supplied REST services
 - implemented with Node.js and Express
 - GET /dog retrieves all dogs
 - response body contains JSON array of dog objects
 - DELETE /dog/{id} deletes a dog
 - POST /dog creates a dog
 - request body contains dog name
 - response body contains JSON dog object
- Supplied code is on next slide
- To start server
 - cd server
 - npm install (first time only)
 - npm start

Node Express Server



```
server.is
const bodyParser = require('body-parser');
const cors = require('cors');
const express = require('express');
const morgan = require('morgan');
const app = express();
app.use(morgan('short'));
app.use(bodyParser.text());
app.use(cors());
const dogMap = {};
let lastId = 0;
function addDog(name) {
  const id = ++lastId;
  const dog = {id, name};
  dogMap[id] = dog;
  return dog;
addDog('Eddie');
addDog('Snoopy');
```

```
app.get('/dog', (req, res) => {
    res.send(Object.values(dogMap));
});

app.delete('/dog/:id', (req, res) => {
    const {id} = req.params;
    const found = Boolean(dogMap[id]);
    if (found) delete dogMap[id];
    res.sendStatus(found ? 200 : 404);
});

app.post('/dog', (req, res) => {
    const name = req.body;
    res.send(addDog(name));
});

app.listen(1919, () => console.log('ready'));
```

Exercise Solutions ...

```
const REST_URL = 'http://localhost:1919/dog';
```

```
async mounted() {
  try {
    const res = await fetch(REST_URL);
    if (!res.ok) throw new Error(res.statusText);
    this.dogs = sortDogs(await res.json());
  } catch (e) {
    console.error('error getting dogs:', e.message);
  }
},
```

... Exercise Solutions

```
async addDog() {
    // If a dog with that name is already present, do nothing.
    const exists = this.dogs.some(dog => dog.name === this.name);
    if (!exists) {
        try {
            const res = await fetch(REST_URL, {method: 'POST', body: this.name});
        if (!res.ok) throw new Error(res.statusText);
        const dog = await res.json();
        this.dogs = sortDogs(this.dogs.concat(dog));
    } catch (e) {
        console.error('error adding dog:', e.message);
    }
}

this.name = '';
},
```

```
async deleteDog(id) {
  const url = REST_URL + '/' + id;
  try {
    const res = await fetch(url, {method: 'DELETE'});
    if (!res.ok) throw new Error(res.statusText);
    this.dogs = this.dogs.filter(dog => dog.id !== id);
  } catch (e) {
    console.error('error deleting dog:', e.message);
  }
}
```