R.E.S.T. API's

R.E.presentational
S.tate
T.ransfer

REST is the architectural style on which the web is built.

What is it? A set of principles built around the HTTP and URI standards

1. **Everything is a resource**

Every piece of data on the internet is a resource, and it can be thought of in terms of its format. For example, JPEG images, MPEG videos, HTML, XML, text documents, and binary data. All are resources with the following content-types: image/jpeg, video/mpeg, text/html, text/xml, and application/octet-stream.

A set of principles built around the HTTP and URI standards: #2

1. Everything is a resource
2. **Each resource is identifiable by a unique identifier**

Because the Internet contains so many different resources, they should all be accessible via URI's and should be identified uniquely.

A set of principles built around the HTTP and URI standards: #3

1. Everything is a resource
2. Each resource is identifiable by a unique identifier
3. **use the standard HTTP methods**

    - GET: request a resource
    - POST: create a new resource
    - PUT: update an existing resource
    - PATCH: update part of an existing resource
    - DELETE: delete a resource

A set of principles built around the HTTP and URI standards: #3 & #4

1. Everything is a resource
2. Each resource is identifiable by a unique identifier
3. use the standard HTTP methods
4. **resources can have multiple representations**
5. **communicate statelessly**

some videos explaining REST:

REST API concepts and examples

Intro to REST: Google Developers

we care because we're building a RESTful API using Express

express.js

express is a web application framework for Node.js

- it is minimal and flexible - the current version, 4.13, has removed much of the middle-ware that used to be built into it prior to version 4
- it's the foundation for other tools and frameworks such as Kraken and Sails
- it's great for building web API's, specifically REST API's, which is what we'll use it for

Installing express inside your project directory:

```
// express should be installed locally on a per-project basis
mkdir my_project
cd my_project

// do an npm init to create your initial package.json
npm init

// use the --save switch to add express as one of your project's dependencies
npm install --save express
```

a simple "Hello, students!" application:

```
// requiring the express module returns a function
var express = require('express');

// calling express creates an application instance
var app = express();

// the get function creates a route that accepts HTTP GET requests
app.get('/', function (request, response) {
    response.send('Hello, students!');
});
app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

Like the callback function we created for the Node HTTP server, the `get` function takes a callback that defines request and response parameters. In our simple Express server, the callback writes to the response stream using the `send` method to send text to the browser.

calling Node functions from within Express:

```
var express = require('express');
var app = express();

app.get('/', function (request, response) {

    // because Express extends Node, we can use Node functions here
    response.write('Hello, students!');
    response.end();
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

responding with JSON:

```
var express = require('express');
var app = express();

app.get('/trucks', function (request, response) {

    // here we want to send the trucks array as JSON date to the response
    var trucks = ['Crisp Creperie', 'Ezell\'s Express', 'Marination'];

    // the send function will convert this to JSON and set response headers
    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

the send function will convert markup strings and set the response headers accordingly:

```
var express = require('express');
var app = express();

app.get('/trucks', function (request, response) {
    var trucks = '<ul><li>Crisp Creperie</li><li>Ezell\'s Express</li>' +
            '<li>Marination</li></ul>';

    // the send function here will set response headers to 'text/html'
    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

Express's response object can redirect:

```
var express = require('express');
var app = express();

app.get('/trucks', function (request, response) {

    // redirect to /food-trucks instead
    response.redirect('/food-trucks');
});

app.get('/food-trucks', function (request, response) {
    var trucks = '<ul><li>Crisp Creperie</li><li>Ezell\'s Express</li>' +
            '<li>Marination</li></ul>';

    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

express middle-ware

middle-ware are the building blocks of express

what is middle-ware?

"Middle-ware is a function with access to the request object (request), the response object (response), and the next middle-ware in line in the request-response cycle of an Express application, commonly denoted by a variable named next. " - express documentation

express's "static" middle-ware - the only middle-ware to still ship with express:

```
var express = require('express');
var app = express();

// to use the "static" middle-ware, we call it from our express object
var serveStatic = express.static('public');

// this will allow us to serve up static files from "public"
app.use(serveStatic);

app.get('/trucks', function (request, response) {

    var trucks = '<ul><li>Crisp Creperie</li><li>Ezell\'s Express</li>' +
            '<li>Marination</li></ul>';

    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

Using middle-ware we can do things like validation, authentication, data-parsing. When a request comes in, it passes through each of these middle-ware functions before reaching the routes.

express middle-ware functions typically get bound to an express application by a call to app.use():

```
var express = require('express');
var app = express();

// a middle-ware function without a "mount" path gets executed on every reques
app.use(function (request, response, next) {
    console.log('Time: ', Date.now());
    next();
});

app.get('/trucks', function (request, response) {

    var trucks = ['Crisp Creperie', 'Ezell\'s Express', 'Marination'];

    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

middle-ware can be "mounted" on a path, meaning they will only be called when a request is made to that URL:

```
var express = require('express');
var app = express();

// a middle-ware with a path mounted, only fires on '/food-trucks' path
app.use('/food-trucks', function (request, response, next) {
    console.log('Time: ', Date.now());
    console.log('will only be called when /food-trucks path is hit');
    next();
});

app.get('/trucks', function (request, response) {

    var trucks = ['Crisp Creperie', 'Ezell\'s Express', 'Marination'];

    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

a series of middle-ware functions can be passed to a single use call:

```
var express = require('express');
var app = express();

// passing two middle-ware functions in series
app.use(function (request, response, next) {
    console.log('Time: ', Date.now());
    next();
}, function (request, response, next) {
    console.log('Request URL: ', request.originalUrl);
    next();
});

app.get('/trucks', function (request, response) {

    var trucks = ['Crisp Creperie', 'Ezell\'s Express', 'Marination'];

    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

middle-ware functions can be passed in sequence as arguments to the route handler:

```
var express = require('express');
var app = express();

// passing middle-ware functions in to route
app.get('/trucks', function (request, response, next) {
    console.log('Time: ', Date.now());
}, function (request, response) {
    var trucks = ['Crisp Creperie', 'Ezell\'s Express', 'Marination'];
    response.send(trucks);
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

sequence matters - once the response stream has been closed, subsequent middle-ware won't be used:

```
var express = require('express');
var app = express();

app.use(function (request, response, next) {
    console.log('Request URL: ', request.originalUrl);
    next();
});

app.get('/trucks', function (request, response) {
    var trucks = ['Crisp Creperie', 'Ezell\'s Express', 'Marination'];
    response.send(trucks);
});

// this middle-ware won't get called: response stream was closed
app.use(function (request, response, next) {
    console.log('Time: ', Date.now());
    next();
});

app.listen(3000, function () {
    console.log('server started on port 3000');
});
```

User Params - reading from the URL

reading from the querystring - given a url, we can limit the number of results returned:

`/trucks?limit=3`

```
var express = require('express');
var app = express();

app.use(express.static('public'));

app.get('/trucks', function (request, response) {
    var trucks = ['314 Pie', 'Crisp Creperie', 'Ezell\'s Express',
            'Marination', 'Maximus Minimus'];

    // if the "limit" value on the querystring is > 0, return that # of result
    if (request.query.limit > 0) {
        response.json(trucks.slice(0, request.query.limit));
    // otherwise return all of the results
    } else {
        response.json(trucks);
    }
});

app.listen(3000, function () {
    console.log('listening on port 3000');
});
```

dynamic routes - using url parameters: a dynamic route is one that can be anything - it will be assigned to the variable name that follows a ':' in the route

```
var express = require('express');
var app = express();

var trucks = {
    '314 Pie': 'pie',
    'Crisp Creperie': 'French',
    'Ezell\'s Express': 'Southern',
    'Marination': 'Hawaaian-Korean',
    'Maximus/Minimus': 'BBQ'
};
app.use(express.static('public'));

// the 'name' parameter will be available on the request.params object
app.get('/trucks/:name', function (request, response) {
    var truck = request.params.name;
    var foodType = trucks[truck];
    response.send(foodType);
});
app.listen(3000, function () {
    console.log('listening on port 3000');
});
```

because the value in a dynamic route can be anything, we have to explicitly check if the resource requested was found and set an appropriate status code:

```
var express = require('express');
var app = express();
var trucks = {
    '314 Pie': 'pie',
    'Crisp Creperie': 'French',
    'Ezell\'s Express': 'Southern',
    'Marination': 'Hawaaian-Korean'
};
app.use(express.static('public'));

// the 'name' parameter will be available on the request.params object
app.get('/trucks/:name', function (request, response) {
    var truck = request.params.name;
    var foodType = trucks[truck];

    // foodType will be undefined if no food truck matches 'name'
    if (!foodType) {
        response.status(404).json('No food type found for ' +
            request.param.name);
    } else {
        response.send(foodType);
    }
});

app.listen(3000, function () { console.log('listening on port 3000'); });
```

the parameter for our dynamic route is case sensitive - we can make it less brittle:

```
var express = require('express');
var app = express();
var trucks = {
    'Athenas': 'Meditteranean',
    'Beanfish': 'Asian',
    'Marination': 'Hawaaian-Korean'
};
app.use(express.static('public'));

app.get('/trucks/:name', function (request, response) {
    var truck = request.params.name;
    var foodType = trucks[truck];

    // capitalize first letter, make everything else lowercase
    var foodTruck = truck[0].toUpperCase + truck.slice(1).toLowerCase();

    if (!foodType) {
        response.status(404).json('No food type found for ' + truck);
    } else {
        response.send(foodType);
    }
});

app.listen(3000, function () { console.log('listening on port 3000'); });
```

we can refactor the name variable parsing into a special middle-ware function called app.param() - this method maps placeholders to callback functions

```
var express = require('express');
var app = express();
var trucks = {
    'Athenas': 'Meditteranean', 'Beanfish': 'Asian', 'Marination': 'Hawaaian-Korea
};
app.use(express.static('public'));

// the param method will map a placeholder variable to a callback
app.param('name', function (request, response, next) {
    var truck = request.params.name;
    var foodTruck = truck[0].toUpperCase() + truck.slice(1).toLowerCase();
    request.foodTruck = foodTruck;
    next().
});

app.get('/trucks/:name', function (request, response) {
    var foodType = trucks[request.foodTruck];
    if (!foodType) {
        response.status(404).json('No food type found for ' + truck);
    } else {
        response.send(foodType);
    }
});
app.listen(3000, function () { console.log('listening on port 3000'); });
```