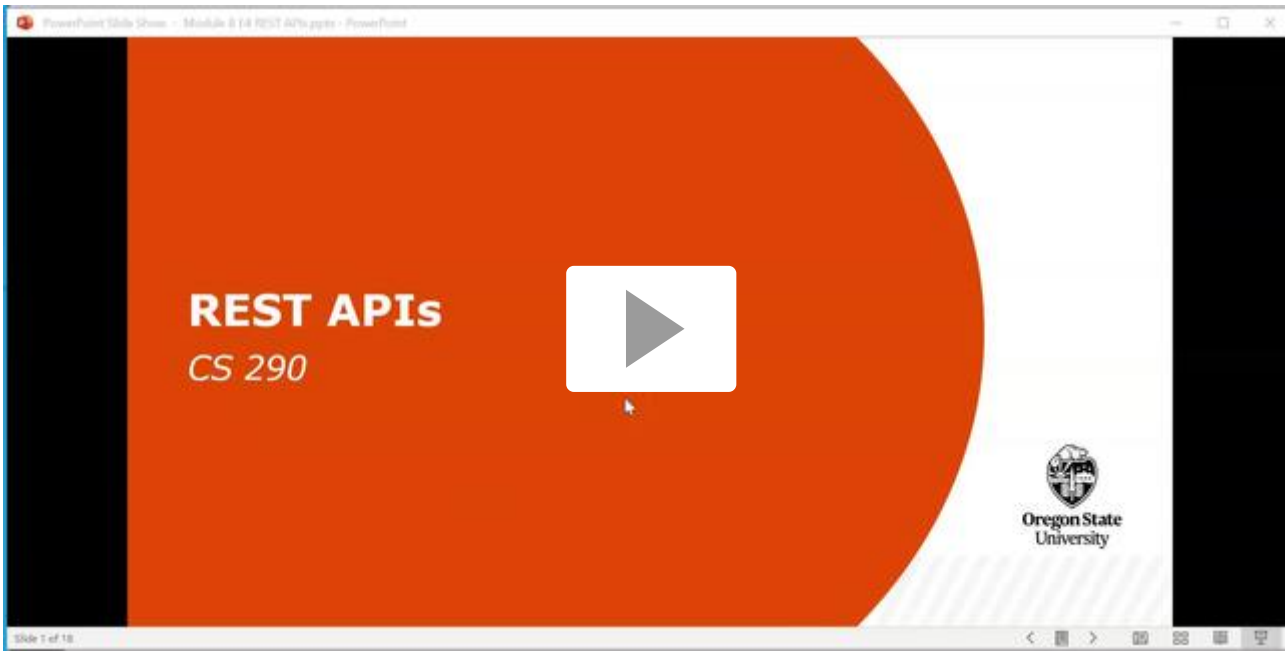
Exploration — REST API

Introduction

As discussed earlier, React applications are typically Single Page Applications (SPAs). SPAs follow this sequence of steps: 1) It downloads the HTML, CSS, and JavaScript for a web application from the web server to the browser exactly once. 2) Once the web application is loaded into the browser, then JavaScript code makes changes to the DOM so that the user feels that they are navigating to a different page. 3) After the initial bundle of files has been downloaded, the web app communicates with the server for various data operations, e.g., to fetch new data, or to update, create and delete data. For this communication, the server program does not respond with HTML documents. 4) Instead, the response is in a format such as JSON.

Such a server program exposes its functionality as an API which is accessible over HTTP and the SPA makes calls on this API. The term **web service** is used for an API that can be accessed using the HTTP protocol.

In this exploration, we study REST APIs which are a very widely used architectural style of implementing web services that follows certain guidelines.



0:00 / 23:24

Characteristics of a REST API

REST stands for Representational State Transfer. A web service that follows the REST guidelines is called a RESTful service. Certain constraints have been defined formally for REST APIs. But we will identify certain important characteristics of such APIs and follow those in this course. To illustrate these characteristics, we will use the example of managing movie data that we have used in previous explorations in the current module. We will describe how can we can define an API that conforms to REST guidelines and performs CRUD operations on the movie collection. For the purposes of examples, we assume that the API is running at the URL `http://localhost:3000`. If the API were running at a different domain and port number, simply change the values in the URL examples.

Collections and Resources

A server manages collections of resources.

- In the movie app, each movie is a resource and all the movies form a collection.
- In an app to manage photos, each photo is a resource and the collection is all the photos.

A server should expose collections and resources to the clients using unique URLs.

- In the movie app, the collection of movies will have the URL `http://localhost:3000/movies/`.
- In the movie app, a movie with ID `xyz123` will have the URL `http://localhost:3000/movies/xyz123`. Since each movie has a unique ID, the URL for each movie is unique.

CRUD Operations and HTTP Methods

A server should support CRUD operations by providing a set of HTTP methods as follows:

- Create resources using the POST method.
- Read resources using the GET method.
- Update resources using the PUT method.
- Delete resources using the DELETE method.

Keep the Server Stateless

The server should be stateless. Specifically, each HTTP request happens in isolation. The server cannot remember information from the previous request by a client to service a subsequent request from that client. The client must send all the information needed to process a request within the request itself. The following example of a non-RESTful API illustrates the idea:

- A client makes two requests, one to get movies for the year 2018 and the next request to get movies for the year 2019.
- Consider we had implemented the movie API as follows:
  - When a request is made by a client to get movies for a particular year, e.g., 2018, the server stores this year in an HTTP session tied to this client.

- In the next request, the client can ask for movies in the “following” year, i.e., without specifying which year they are referring to.
- On seeing that the request is for the “following” year, the server looks up the HTTP session for this client and finds out that the previous request was for 2018. The server then responds by sending back the movies for 2019.

Such an API is **not RESTful** because the server used information from the previous request to service the next request from the client. For an API to be RESTful, the server must never rely on information from previous requests by the client. Any information needed from a previous request must be sent by the client in subsequent requests.

Route Parameters

We already have almost all the knowledge needed to implement RESTful services using Express. The only additional concept we need to know is **route parameters**. We have mentioned that a resource is identified by a URL that contains the unique ID for that resource. For example, a movie with ID `xyz123` will have the URL `http://localhost:3000/movies/xyz123`. To implement a REST API, our route handler needs to get the value of this ID from the URL. Express supports this as follows:

- In the `path` argument of a route, we can specify which parts of the URL we want to be made available to the route handler.
- These parts of the URL are called route parameters.
- Express populates an object `req.params` in which
  - The names of the properties are the names used in the `path` argument.
  - The values of the properties are set to the corresponding part of the URL.

For example:

- We define a route with the `path` argument `/movies/:id`.
- Express will add a property `id` to the object `req.params`
- If the request URL is `http://localhost:3000/movies/xyz123`.
- `req.params` object will be `{ "id" : "xyz123" }`
- This means we can define a route handler that accesses the movie ID as follows:

```
app.get('/movies/:id', function (req, res) {
  const movieId = req.params.id;
  ...
```

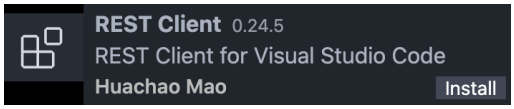
Note that Express can populate multiple route parameters from a request. For an example, see Express docs on **routing (https://expressjs.com/en/guide/routing.html)**.

Tools for Sending Requests with Different HTTP Methods

We cannot use the browser to send requests for certain HTTP methods, such as `DELETE` and `PUT`. Fortunately, a large number of tools are available to send HTTP requests corresponding to the

different methods used in REST APIs. We discuss a few of these now.

REST Client for Visual Studio Code



If you are using Visual Studio Code, then you should install the extension REST Client. It is a very easy-to-use tool for sending HTTP requests to a REST API.

Examples: You can see example HTTP requests, sent with methods POST, GET, PUT and DELETE, using the VS Code extension **REST Client (https://marketplace.visualstudio.com/items?itemName=humao.rest-client)** in this **m8-test-requests.http (https://canvas.oregonstate.edu/courses/1879154/files/93831942?wrap=1)** (https://canvas.oregonstate.edu/courses/1879154/files/93831942/download?download_frd=1) file.

Invoke-RestMethod with Windows PowerShell

Windows PowerShell provides a utility called `Invoke-RestMethod` that can be used to send requests with different HTTP methods to a REST API.

Examples: You can see example HTTP requests, sent with methods POST, GET, PUT and DELETE, using `Invoke-RestMethod` in **this file (https://canvas.oregonstate.edu/courses/1879154/files/93831930?wrap=1)** (https://canvas.oregonstate.edu/courses/1879154/files/93831930/download?download_frd=1) .

curl

The `curl` command can be used to send HTTP requests to test a REST API. Using `curl` on Windows can be somewhat tedious because of the need to escape quotes in JSON objects.

Examples:

- You can see example HTTP requests, sent with methods POST, GET, PUT and DELETE, using `curl` from a MacOS or Linux shell in **this file (https://canvas.oregonstate.edu/courses/1879154/files/93831941?wrap=1)** (https://canvas.oregonstate.edu/courses/1879154/files/93831941/download?download_frd=1) .
- You can see example HTTP requests, sent with methods POST, GET, PUT and DELETE, using `curl` on WSL (Windows Subsystem for Linux) in **this file (https://canvas.oregonstate.edu/courses/1879154/files/93831956?wrap=1)** (https://canvas.oregonstate.edu/courses/1879154/files/93831956/download?download_frd=1) .

Summary

REST APIs are a widely used architectural style for implementing web services. Implementing RESTful services requires following certain guidelines. We think of our app as managing collections and resources within these collections. REST specifies a certain pattern of URLs to identify each

collection and each resource within a collection. CRUD operations are carried out using specific HTTP methods and the server implementing a REST API should not maintain state about client requests. With this knowledge, we will now proceed to implement a REST API in the next exploration.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- **The Wikipedia article on REST** (https://en.wikipedia.org/wiki/Representational_state_transfer) gives a good overview of REST services.
- Route parameters are discussed in the Express **doc on routing** (https://expressjs.com/en/guide/routing.html).
- The GitHub repository for the **VS Code REST Client** (https://github.com/Huachao/vscode-restclient) gives a detailed overview of its usage.
- See the Windows docs for an overview of **Invoke-RestMethod** (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-restmethod).
- Sending HTTP requests using the `curl` command is discussed **here** (https://www.baeldung.com/curl-rest).