# RESTful Web APIs

Introduction to RESTful Web APIs

## What is *REST*?

- *Representational state transfer*:
  - Originally for accessing and manipulating textual representations of Web resources using a set of stateless operations
  - Today: More generic, encompassing every entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web
- Architectural pattern, not a standard
  - Request-response pattern
- Today, HTTP-based RESTful APIs dominate
  - URLs for addressing
  - JSON, sometimes XML for representing data elements
  - Standard HTTP methods aka *verbs* (e.g. *GET*, *PUT*, *POST*, and *DELETE*)
  - Standard HTTP status codes for representing results
  - HTTP header fields (standard or custom) for sending parameters
  - TLS for encrypting data in-transit

## Important Tools (Examples)

- API Clients
  - *Postman*
  - *Insomnia*
  - *REST Client* in *Visual Studio Code*
- HTTP Request and Response Service
  - *httpbin.org*

## Sample Requests

```
GET https://pokeapi.co/api/v2/pokemon HTTP/1.1
Accept: application/json

###

GET https://pokeapi.co/api/v2/pokemon/1/ HTTP/1.1
Accept: application/json

GET http://services.odata.org/V4/Northwind/Northwind.svc/Customers HTTP/1.1
Accept: application/json

###

GET http://services.odata.org/V4/Northwind/Northwind.svc/Customers HTTP/1.1
Accept: application/atom+xml

###

GET http://services.odata.org/V4/Northwind/Northwind.svc/Customers?$filter=Count
Accept: application/json
```

---

## Sample Requests (cont.)

```
POST https://httpbin.org/post HTTP/1.1
Content-Type: application/json

{ "Foo": "Bar", "Answer": 42 }

###

DELETE https://httpbin.org/delete HTTP/1.1
```

## Important REST principles

- Stateless
  - No client context stored on the server
  - Each request is complete
- Cacheable
  - Responses explicitly indicate their cacheability
- Layered System
  - Client cannot tell if connected directly to the server (e.g. reverse proxies)
- URIs
  - Resources are identified using *Uniform Resource Identifiers* (URIs)
- Resource representation
- *XML*, *JSON*, *Atom* - today mostly JSON

---

## RESTful Web APIs in the Browser

- Old but still relevant: *XMLHttpRequest*
- Newer, but only in modern browsers: *fetch*
  - Detailed MDN documentation about *fetch*...

```
const pokemonList = document.getElementById('pokemons');

(function() {
fetch('https://pokeapi.co/api/v2/pokemon/').then(response => {
  response.json().then(pokelist => {
    let html = '';
    for (const pokemon of pokelist.results) {
      html += `<li>${pokemon.name}</li>`
    }

    pokemonList.innerHTML = html;
  });
});
})();
```

---

## RESTful Web APIs in the Browser (cont.)

With `async/await`:

```
const pokemonList = document.getElementById('pokemons');

(async function() {
    const response = await fetch('https://pokeapi.co/api/v2/pokemon/');
    const pokelist = await response.json();

    let html = '';
    for(const pokemon of pokelist.results) {
        html += `<li>${pokemon.name}</li>`
    }

    pokemonList.innerHTML = html;
})();
```

---

## Building RESTful Web APIs with Node.js

- In practice, frameworks are used for that
- Here: *Express.js*
  - Larger framework, not just for RESTful Web APIs
  - Very commonly used
  - Lots of plugins
- Installation
  - `npm install express`
  - For TypeScript: `npm install @types/express --save-dev`

---

## RESTful Web API with *Express.js*

```
// If you have problems with the following line, try:
// import express = require('express');
import * as express from 'express';
```

```
var server = express();
server.get('/api/echo/:word', (request, response) => {
    response.send({youSent: request.params.word});
});

const port = 8080;
server.listen(port, function() {
  console.log(`API is listening on port ${port}`);
});
```

```
GET http://localhost:8080/api/echo/Foo-Bar HTTP/1.1
Accept: application/json
```

---

### RESTful Web API with *Express.js*

- `express()` function
  - Creates an Express application
  - Documentation
- Application
  - Represents the Express application
  - Created with `express()`
  - Documentation
- `request` object
  - Represents the HTTP request
  - Use it to get headers, parameters, body, etc.
  - Documentation
- `response` object
  - Represents the HTTP response
  - Use it to build response (e.g. status, headers, body, etc.)
  - Documentation

---

## *Express.js* Examples

### app.ts

```
// If you have problems with the following line, try:
// import express = require('express');
import * as express from 'express';

import {deleteSingle} from './delete-single';
import {getAll} from './get-all';
import {getSingle} from './get-single';
import {post} from './post';

const app = express();
app.use(express.json());

// Add routes
app.get('/api/customers', getAll);
app.post('/api/customers', post);
app.get('/api/customers/:id', getSingle);
app.delete('/api/customers/:id', deleteSingle);

app.listen(8080, () => console.log('API is listening on port 8080'));
```

---

## *Express.js* Examples (cont.)

### data.ts

```
export interface ICustomer {
  id: number;
  firstName: string;
  lastName: string;
}

export const customers: ICustomer[] = [
  {id: 1, firstName: 'Donald', lastName: 'Duck'},
  {id: 2, firstName: 'Mickey', lastName: 'Mouse'},
  {id: 3, firstName: 'Minnie', lastName: 'Mouse'},
  {id: 4, firstName: 'Scrooge', lastName: 'McDuck'}
];
```

**get-all.ts**

```
import {Request, Response} from 'express';
import {customers} from './data';

export function getAll(req: Request, res: Response): void {
    res.send(customers);
}
```

---

## *Express.js* Examples (cont.)

**get.single.ts**

```
import {Request, Response} from 'express';
import {NOT_FOUND, BAD_REQUEST} from 'http-status-codes';
import {customers} from './data';

export function getSingle(req: Request, res: Response): void {
  const id = parseInt(req.params.id);
  if (id) {
    const customer = customers.find(c => c.id === id);
    if (customer) {
      res.send(customer);
    } else {
      res.status(NOT_FOUND).send();
    }
  } else {
    res.status(BAD_REQUEST).send('Parameter id must be a number');
  }
}
```

---

## *Express.js* Examples (cont.)

**post.ts**

```
import {CREATED, BAD_REQUEST} from 'http-status-codes';
import {Request, Response} from 'express';
import {customers, ICustomer} from './data';
```

```
export function post(req: Request, res: Response): void {
  if (!req.body.id || !req.body.firstName || !req.body.lastName) {
    res.status(BAD_REQUEST).send('Missing mandatory member(s)');
  } else {
    const newCustomerId = parseInt(req.body.id);
    if (!newCustomerId) {
      res.status(BAD_REQUEST).send('ID has to be a numeric value');
    } else {
      const newCustomer: ICustomer = { id: newCustomerId,
        firstName: req.body.firstName, lastName: req.body.lastName };
      customers.push(newCustomer);
      res.status(CREATED).header({Location:
↪  `${req.path}/${req.body.id}`}).send(newCustomer);
    }
  }
}
```

---

### *Express.js* Examples (cont.)

**delete-single.ts**

```
import {NO_CONTENT, NOT_FOUND, BAD_REQUEST} from 'http-status-codes';
import {Request, Response} from 'express';
import {customers} from './data';

export function deleteSingle(req: Request, res: Response): void {
  const id = parseInt(req.params.id);
  if (id) {
    const customerIndex = customers.findIndex(c => c.id === id);
    if (customerIndex !== (-1)) {
      customers.splice(customerIndex, 1);
      res.status(NO_CONTENT).send();
    } else {
      res.status(NOT_FOUND).send();
    }
  } else {
    res.status(BAD_REQUEST).send('Parameter id must be a number');
  }
}
```

## *Lokijs*

- Lightweight in-memory key-value store
- Fast and easy to use
- Works in…
  - …browser
  - …apps
  - …Node.js on the server or in the command line
- Persistence adapter can write data to disk/indexeddb

## *Lokijs*

**db.ts**

```
// If you have problems with the following line, try:
// import loki = require('lokijs');
import * as loki from 'lokijs';

export class Datastore {
    constructor(public db: loki, public customers: loki.Collection) { }
}

export function init(): Datastore {
    const db = new loki('./data.db', { autosave: true });

    let customers = db.getCollection('customers');
    if (!customers) {
        customers = db.addCollection('customers');
    }

    return new Datastore(db, customers);
}
```

## *Express.js* Examples with *Lokijs*

### app.ts

```
// If you have problems with the following line, try:
// import express = require('express');
import * as express from 'express';

import {deleteSingle} from './delete-single';
import {getAll} from './get-all';
import {getSingle} from './get-single';
import {post} from './post';
import {init} from './db';

const app = express();
app.use(express.json());
app.locals = init();

// Add routes
app.get('/api/customers', getAll);
app.post('/api/customers', post);
app.get('/api/customers/:id', getSingle);
app.delete('/api/customers/:id', deleteSingle);

app.listen(8080, () => console.log('API is listening on port 8080'));
```

---

## *Express.js* Examples with *Lokijs* (cont.)

### get-all.ts

```
import {Request, Response} from 'express';
import {Datastore} from './db';

export function getAll(req: Request, res: Response): void {
    res.send((<Datastore>req.app.locals).customers.find());
}
```

---

## *Express.js* Examples with *Lokijs* (cont.)

### get-single.ts

```
import {Request, Response} from 'express';
import {NOT_FOUND, BAD_REQUEST} from 'http-status-codes';
import {Datastore} from './db';

export function getSingle(req: Request, res: Response): void {
  const id = parseInt(req.params.id);
  if (id) {
    const store = <Datastore>req.app.locals;
    const customer = store.customers.get(id);
    if (customer) {
      res.send(customer);
    } else {
      res.status(NOT_FOUND).send();
    }
  } else {
    res.status(BAD_REQUEST).send('Parameter id must be a number');
  }
}
```

## *Express.js* Examples with *Lokijs* (cont.)

### post.ts

```
import {CREATED, BAD_REQUEST} from 'http-status-codes';
import {Request, Response} from 'express';
import {Datastore} from './db';
import {ICustomer} from './model';

export function post(req: Request, res: Response): void {
  if (!req.body.id || !req.body.firstName || !req.body.lastName) {
    res.status(BAD_REQUEST).send('Missing mandatory member(s)');
  } else {
    const newCustomerId = parseInt(req.body.id);
    if (!newCustomerId) {
      res.status(BAD_REQUEST).send('ID has to be a numeric value');
    } else {
      const store = <Datastore>req.app.locals;
      const newCustomer: ICustomer = { id: newCustomerId,
        firstName: req.body.firstName, lastName: req.body.lastName };
      store.customers.insert(newCustomer);
      res.status(CREATED).header({Location:
↪    `${req.path}/${req.body.id}`}).send(newCustomer);
    }
  }
}
```

---

## *Express.js* Examples with *Lokijs* (cont.)

### delete-single.ts

```
import {NO_CONTENT, NOT_FOUND, BAD_REQUEST} from 'http-status-codes';
import {Request, Response} from 'express';
import {Datastore} from './db';

export function deleteSingle(req: Request, res: Response): void {
  const id = parseInt(req.params.id);
  if (id) {
    const store = <Datastore>req.app.locals;
    const customerToDelete = store.customers.get(id);
    if (customerToDelete) {
      store.customers.remove(customerToDelete);
      res.status(NO_CONTENT).send();
    } else {
      res.status(NOT_FOUND).send();
    }
  } else {
    res.status(BAD_REQUEST).send('Parameter id must be a number');
  }
}
```

---

## Web APIs + Single Page Apps (SPA)

---

## Web APIs + Single Page Apps (SPA)

- Client can be a browser
  - Anything that can speak HTTP, JSON, etc. (e.g. mobile app, CLI, server, desktop app, IoT device)
- Static HTML/CSS/JS for SPA
- Logic
  - HTTP Web API requests for running server-side business logic
  - View logic (e.g. manipulating DOM) runs on client
  - JSON for transmitting data

---

## Consuming Web APIs in Angular

- Use module `HttpClientModule`
- Read more in Angular docs...

```
...
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [ ..., HttpClientModule, ... ],
  declarations: ...,
  bootstrap: ...
})
export class AppModule {}
```

---

## Consuming Web APIs in Angular

Get instance of `HttpClient` in constructor (*Dependency Injection*)

```
...
import { HttpClient } from '@angular/common/http';

@Component(...)
export class MyComponent {
  constructor(private http: HttpClient) { ... }
  ...
}
```

---

## Consuming Web APIs in Angular

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';

interface IPerson { name: string; }

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent implements OnInit {
  people: Observable<IPerson[]>;
  persons: IPerson[] = [];

  constructor(private httpClient: HttpClient) {
    this.people =
↪   httpClient.get<IPerson[]>('http://localhost:8080/api/people');
  }

  ngOnInit() {
    this.httpClient.get<IPerson[]>('http://localhost:8080/api/people')
       .subscribe(result => this.persons = result);
  }
}
```

---

## Consuming Web APIs in Angular

```
<h1>People</h1>

<ul>
```

```
  <li *ngFor="let person of people | async">{{ person.name }}</li>
</ul>

<select>
  <option *ngFor="let person of persons">{{ person.name }}</option>
</select>
```

---

### Consuming Web APIs in Angular

| Method | Docs link |
|---|---|
| `get` | Read more… |
| `post` | Read more… |
| `patch` | Read more… |
| `put` | Read more… |
| `delete` | Read more… |

---

### Further Readings

- Want to know more? Read/watch…
  - Microsoft's REST API Guidelines
  - *Express.js* documentation