SD540 Server-Side Programming

Maharishi University of Management

Masters of Software Development

Associate Professor Asaad Saad

Maharishi International University - Fairfield, Iowa

Stateless vs Stateful Web Apps

Stateful

Keeps data (Authorization servers are often stateful services, they store issued tokens in database/memory for future checking)

Stateless

Does not keep any data (issuing JWT tokens as access tokens)

REST = Representational State Transfer

Architectural style for distributed systems

Exposes resources (state) to clients

Resources identified with a URI

Uses HTTP, URI, MIME types (JSON)

HTTP Verbs and CRUD Consistency

The following are the most commonly used server architecture HTTP methods:

- **GET** Retrieves an entity or a list of entities
- **HEAD** Same as GET, only without the body
- **POST** Submits a new entity
- **PUT** Updates an entity by complete replacement
- **PATCH** Updates an entity partially
- **DELETE** Delete an existing entity
- **OPTIONS** Retrieves the capabilities of the server

REST Example

A RESTful (stateless) API (application program interface) uses HTTP verbs/requests (GET, PUT, PATCH, POST, and DELETE) to control an entity. The entity name must be one plural noun, ideally in small letters.

URL	HTTP Verb	POST Body	Result
http://yourdomain.com/api/entries	GET	empty	Returns all entries
http://yourdomain.com/api/entries	POST	JSON String	New entry Created
http://yourdomain.com/api/entries/:id	GET	empty	Returns single entry
http://yourdomain.com/api/entries/:id	PUT	JSON string	Updates an existing entry
http://yourdomain.com/api/entries/:id	DELETE	empty	Deletes existing entry

A Sub-Entity extends the **`/root_entity/:root_entity_id`**

express.Router()

The Router class is a mini Express application that has only middleware and routes. This is useful for **abstracting modules** based on the business logic that they perform.

myRoute.js

```
import express from 'express';
const router = express.Router();

router.get('/', get_all_handler);
router.post('/', express.json(), post_handler);
router.get('/:id', get_one_handler);
router.put('/:id', express.json(), put_handler);
router.delete('/:id', delete_handler);

export default router; // export the Router middleware
```

Pass `{ mergeParams: true }` to the sub-entity Router instance, to extend the main route params.

Extend the `Request` Object

To extend the **`Request`** object, you may create a custom definition file **`./types/express/index.d.ts`** to merge the namespace declaration.

```
declare namespace Express {
    interface Request {
        secret?: string;
    }
}
```

Update your **`tsconfig.json`** file to consider the new type definition:

```
{ "compilerOptions": {
      "typeRoots": [
          "./types"
      ],
  }
}
```

Standard Response

A good REST practice is to standardize your response, which makes it easy for the client to predict the structure of the response:

```
export interface StandardResponse<T = unknown> {
    success: boolean,
    data: T;
}
```

Standard Error

```
export class ErrorWithStatus extends Error {
    status?: number;

    constructor(message: string, statusCode: number) {
        super(message);
        this.status = statusCode;
    }
}
```

Same-Origin Policy

In browsers, only certain **XHR requests** are permitted on **different domains**, unless the server permits such requests. There is no restriction on XHR request s on the same domain.

Example: If you send an **XHR request** from a **browser** app `http://www.app.com`, the following types of requests fail:

- `https://www.app.com` – Different protocol
- `http://www.app.com:8080/myUrl` – Different port
- `http://www.backendapp.com/` – Different domain
- `http://app.com/` – Different domain

Cross-Origin Resource Sharing (CORS)

CORS has been adopted by API providers as the primary way to share resources, it allows cross-origin requests without relying on JavaScript but rather HTTP.

CORS headers allow servers to specify a set of origins that are allowed to access its resources. If the request referrer header is on that list, It will be able to inspect the answer and use the data.

Accepting CORS in Express

```
$ npm i cors
$ npm i @types/cors -D

import express from 'express';
import cors from 'cors';

const app = express();

app.use(cors());
```