PROGRESSIVE.NET TUTORIALS 2017

"HELLO, GALAXY: HELLO WORLD AT SCALE"

EXERCISE 1: IMPLEMENT THE GREETING API

For these exercises, we're going to implement a simple API based on the classic 'Hello, World' example.

Here's the API specification. You can implement it using whatever variant of .NET and HTTP you like – WebAPI, NancyFX, OpenRasta, ServiceStack – and whatever persistence you want to use – local filesystem, local database, cloud database, in-memory dictionary.

GET	/	(unused for now)
GET	/greetings	Return a list of all the greetings available in the system
POST	/greetings	Add a new greeting to our system.
GET	/greetings/{id}	Return a "Hello, World!" style greeting in the specified format
GET	/greetings/{id}?name={name}	Return a "Hello, {name}" style greeting
PUT	/greetings/{id}	Replace the greeting at {id} with the supplied greeting
DELETE	/greetings/{id}	Delete the specified greeting

A greeting must be supplied as a string template that we can pass to .NET's String.Format method.

For example:

```
POST /greetings
Content-Type:  application/json
{
    "id" : "informal",
    "template" : "Hey, {0}"
}
```

Should return

```
HTTP 201 Created
Location: /greetings/informal
```

You should then be able to invoke your new greeting:

```
GET    /greetings/informal?name=Bill+Gates
```

and get a response like this:

```
HTTP 200 OK
Content-type: application/json

{ “greeting” : “Hey, Bill Gates!” }
```

If you’re fluent and happy writing unit tests, write them. If you’re not, don’t worry – we’re not here to learn about unit testing and the important thing is to get a running codebase to use for the rest of the exercises.

For testing your API, I’d recommend using Postman (https://www.getpostman.com/) – it’s an HTTP client that’s more powerful and flexible than a regular web browser but easier to play around with than something like curl.

BONUS POINTS: use content negotiation to return the same greeting in multiple languages, so we can greet people formally, informally or casually in more than one language by supplying an ISO language code in our request headers.

EXERCISE 2: API SECURITY

Add security to the API, so that it’s not wide open. For this implementation, use HTTP Basic authentication (don’t worry, we’d be running over HTTPS), with the following constraints:

- Authentication must specific a username and password in an HTTP Basic authentication header.
- Usernames and passwords must be alphanumeric characters
- If the username is shorter than 8 characters, return an HTTP 403 Forbidden
- If the password is ‘prognet2017’, the request will succeed
- If the password is anything else, the request will return HTTP 401 Unauthorized

EXERCISE 3: LOGGING

Add logging to the API, that will let us see which users are using it and what they’re doing.

For this exercise, we’re all going to send our logs to the same place – a hosted logging provider called **logit.io**.

Depending on how you’ve implemented your solution, you can use one of the standard .NET logging providers (NLog, Serilog, log4net), either directly or via a filesystem monitor – or you can log directly to logit.io using their HTTP interface.

Check out their documentation at https://logit.io/sources for details on the various formats, sources and providers that work with logit.io

```
Logit.io API key:      a1d6a48d-21c3-48a5-9989-c060798e2f1a
```

Here’s the key things that we want to capture in our logging

For every request:

- The URL and HTTP method requested
- The username associated with the request (so we can do things like per-request billing)
- The HTTP response code – 200, 201, 401, 403, etc
- The total time taken to handle the request and response.
- The hostname of the machine where your API is running (`Environment.MachineName` on most .NET runtime platforms)

EXERCISE 4: DISCOVERABILITY AND MONITORING

Add a discovery endpoint to your API. HTTP requests to / should return a JSON document describing the API, and include hypermedia links to the various endpoints and methods supported by the API.

The idea here is that developers don't need to find your API documentation or your support forums – you can share a link directly to your API endpoint that they can open in a browser, and find their way around from there.

Add a new endpoint to your application at /status.

Doing an HTTP GET to /status should return the following details:

- Application name, codebase and assembly version
- The current local date/time on the server that's running this instance of your application
- Database connectivity (if you're using a database) – time taken to connect and verify availability
- The number of greetings that have been added to this instance of the application

EXERCISE 5: GOING SERVERLESS

For this exercise, we're going to try and implement the same endpoint all over again, but using Azure functions, a serverless hosting technology that's part of the Microsoft Azure stack.

If you don't already have an Azure account available, you can get a one-hour free trial of Azure Functions here: https://functions.azure.com/try