Create RESTful Web Services with PHP

Course by Michael Spinks

Contents

These are the topics we are going to cover throughout this course.

This course is a fundamentals course for developing RESTful APIs using PHP.

1) **Brief overview of the given scenario**
2) **What is a RESTful (REST) API and how is it implemented**
3) **Run through what software we will be using and setting it up**
4) **Our API Requirements**
5) **Create the necessary database and tables for the backend using MySQL**
6) **Develop the set of APIs required, including testing**
7) **Token Based Authentication**
8) **Secure the APIs**
9) **Completed API overview and conclusion**

Scenario

Build a task list system that will allow users to log in and create, update and delete tasks.

Each user's tasks will be private to them and other users will not be able to view them.

We are responsible for the **database** back end, the **web services** and the **Authentication Module**.

We are not responsible for the **front end** or the **server** set up.

What is a RESTful (REST) API

REST (Representational state transfer) API is an interface that is stateless (so for each request the receiving system doesn't know about any last requests). It uses a Client <-> Server model via a Request -> Response architecture and utilizes the standard HTTP verbs and status codes, such as; GET, POST, PATCH and DELETE to create, read, update and delete data (CRUD).

It is important to note that REST is not a standard, it is a set of principles that if an API was to follow would make it RESTful.

REST is generally preferred over SOAP due to its simpler implementation and mostly uses JSON for the response output, JSON is simpler and less verbose than XML which is what SOAP uses. We will be using JSON exclusively in this course.

Examples of what an API can be used for:

1) **Load and update data using a client application (mobile or desktop) to a back-end system.**
2) **Upload an image to a social network**

Example usage of a RESTful API to get user details of a user with an ID of 1:

GET /users/1

How is it implemented

A RESTful API is implemented using the HTTP verbs in a Request:

POST – **Create**

GET – **Retrieve**

PATCH – **Update**

PUT – **Replace**

DELETE – **Delete**

The server takes the Request and processes it and then replies with a Response, this includes any output as well as a HTTP response code:

GET /users/1

```
200 OK
{
        "id": 1,
        "name": "User 1",
        "age": 20
}
```

Common HTTP Response Status Codes:

200 – OK
201 – Created

---

400 – Bad Request
401 – Unauthorised
403 – Forbidden
404 – Not Found
405 – Method Not Allowed
409 – Conflict

---

500 – Internal Server Error

How is it implemented – cont'd

To call the APIs we use URL endpoints – these are known as routes:

GET https://api.mysite.com/**products** - This would return a list of ALL products

However if we specify a product ID number at the end of the URL this would be a different route:

GET https://api.mysite.com/**products/{:id}** - This would return one product that has ID of what ever you pass in, e.g. /products/1

A route can carry out different actions depending on what HTTP verb is used, for example if we use the same route as above that we used to retrieve all products but swap the GET with POST:

POST https://api.mysite.com/**products** - Along with a JSON body of the product details in the request would CREATE a new product and return it as a response to the request.

For routes it is advisable to use nouns, e.g. **/users, /products**, **/jobs**

It is best practice to have endpoints as **plurals**, since it doesn't matter if you are retrieving one user or many users, the endpoints would be consistent, but if you just wanted 1 user then you would add the user ID at the end of the route, e.g. /users/1

Never use routes like; /getUser, /createUser, /deleteUser – these don't follow the REST principles as you are mixing verbs with nouns (getUser), keep the verbs for use in the HTTP request (GET, POST, DELETE, PATCH) and the noun as part of the route (/users).

Installing and setting up the software

What software will we be using

The free software we will be using throughout this course:

MAMP – https://www.mamp.info/en/

Development Web Server Application that has PHP, Apache and MySQL built in

PHPMyAdmin – Built into MAMP ^^

A Graphic User Interface application used to implement the MySQL database

ATOM – https://atom.io

An open source text editor with syntax highlighting and code folding features

POSTMAN – https://www.getpostman.com

An application to test the RESTful APIs during development and testing

API Requirements

The following requirements are what the API needs to deliver:

1) **Return a JSON response for all APIs and allow caching where appropriate.**
2) **A task has an ID, title, description, deadline date, completion status**
3) **Return a list of details for all tasks for a user using a URL of:** */tasks*
4) **Return a list of details for all tasks for a user with pagination using a URL of:** */tasks/page/{:page}*
5) **Return a list of details for a single task for a user using a URL of:** */tasks/{:taskid}*
6) **Return a list of details for all incomplete tasks for a user using a URL of:** */tasks/incomplete*
7) **Return a list of details for all complete tasks for a user using a URL of:** */tasks/complete*
8) **Delete a task for a user using a URL of:** */tasks/{:taskid}*
9) **Update title, description, deadline date or completion status and return updated task using a URL of:** */tasks/{:taskid}*
10) **Create a new task and return the details for the new task using a URL of:** */tasks*

API Requirements - Authentication

The following requirements are what the Authentication API needs to deliver:

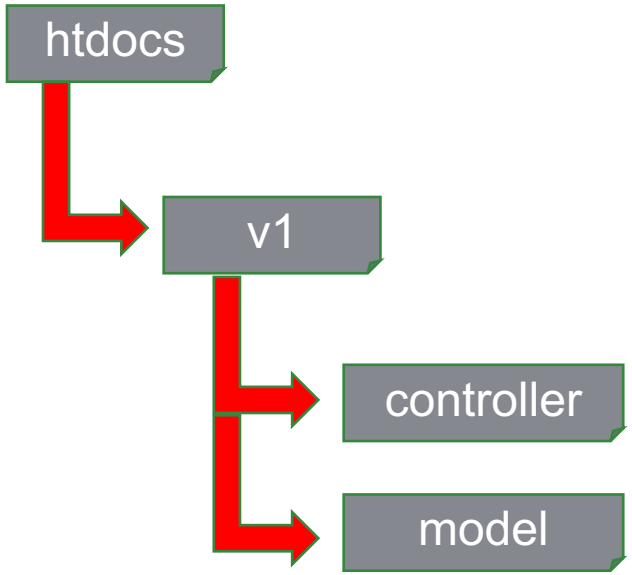
1) **Return a JSON response for all APIs**
2) **A user has an ID, full name, unique username, hashed password, user active status and login attempts**
3) **A user can log in on more than one device and should not log out a previous device (sessions)**
4) **Create a new user using a URL of:** */users*
5) **Log in a user using a URL of:** */sessions*
6) **Log out a user using a URL of:** */sessions/{:sessionid}*
7) **Limited lifetime of a session access token, refreshed using a URL of:** */sessions/{:sessionid}*

Demo of API

Implement the MySQL database

Implement the folder structure

Folder structure



Implement the Response Object Model

Implement the Database Connection

Implement the Task Model

Implement GET – single task

Implement DELETE – single task

Implement GET – all incomplete/complete

Implement GET – all tasks

Implement GET – all tasks with pagination

Implement POST – create a task

Implement PATCH – update single task

Review - What have we done so far….

Token Based Authentication

What is Token Based Authentication?

A token is like a password with a limited lifespan, when a user authenticates with a user name and password they are given two tokens, an access token and a refresh token.

An access token has a really short lifespan (usually minutes or hours) and a refresh token is valid for a lot longer (usually weeks or months). Both tokens are usually just a random set of base64 encoded characters, e.g:

KUPDfxgdsWef2kJVjtGss1X6ra5UA==

This random base64 encoded string is sent in the HTTP header and this is used as a ‘password’ to authenticate you for every request.

When an access token expires, you then use the refresh token to get a new access token (and accompanying new refresh token).

The reason why a refresh token has a longer lifespan is because it is only ever sent with a request to get a new access token so it is less likely to be leaked or exposed to a potential hacker.

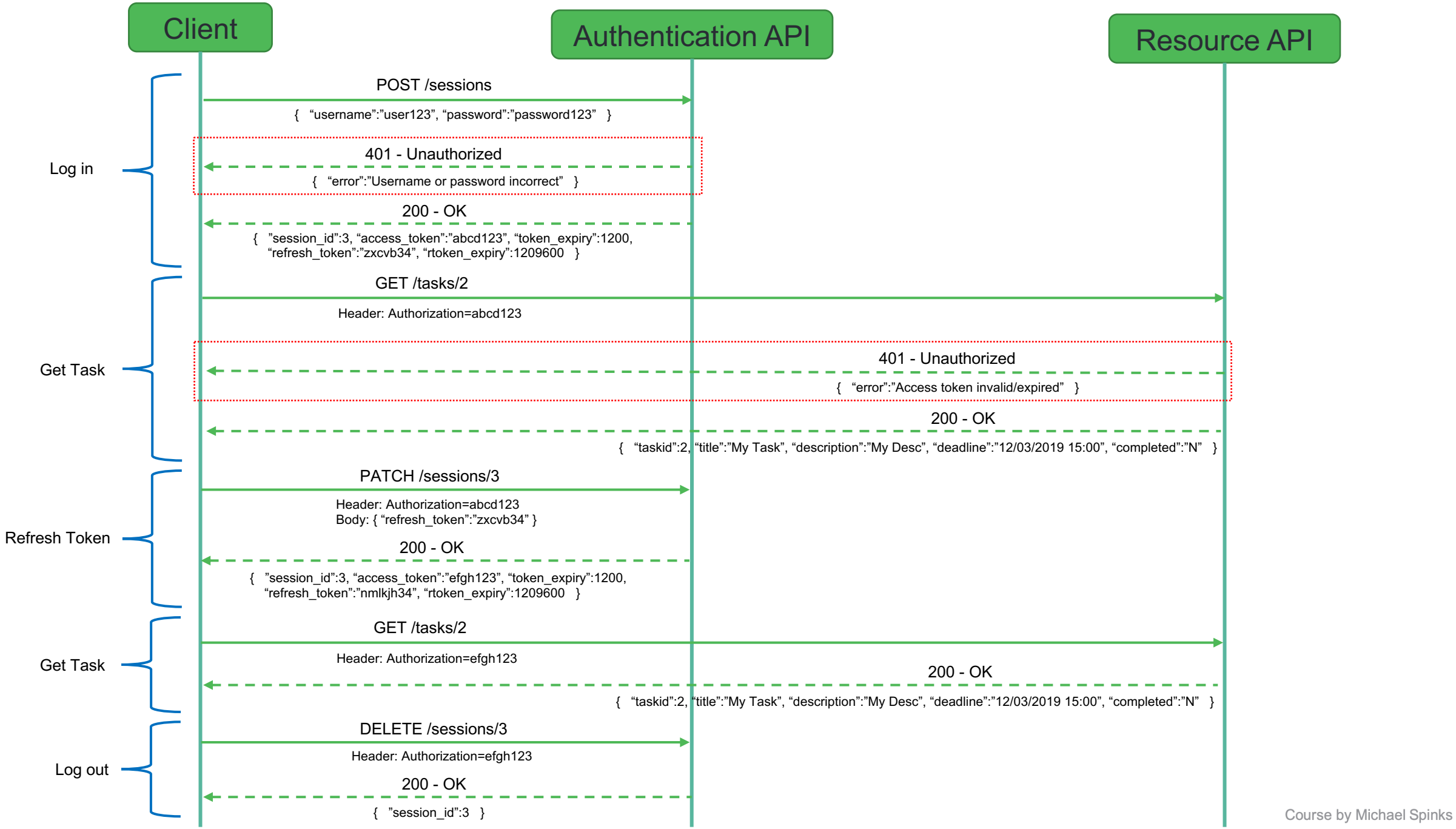
Token Based Authentication - Sessions

We use sessions so we can use the system from multiple devices at the same time, if you have ever used Facebook or another social networking service you will know that you can be logged in on both your computer as well as your smartphone at the same time.

If we didn't have sessions then if you move from using Facebook on your computer to your mobile phone it would ask you to log in again, this then would log you out of Facebook on your computer.

This would not good from a user experience perspective and can increase the likelihood of a password being exposed as you would be sending a password every time you switched devices.

Token Based Authentication – Flow Diagram



Implement the Users table in the MySQL Database

Implement POST – create user account

Implement the Sessions table in the MySQL Database

Implement POST – create user session

Implement DELETE – log out user session

Implement PATCH – refresh access token

Modify Tasks Table in the MySQL Database for Users

Integrate User Authentication into our API

Conclusion and Example App Use

Thank You