

Authorizing Requests

<https://github.com/learn-co-curriculum/python-p4-authorization><https://github.com/learn-co-curriculum/python-p4-authorization/issues/new>

Learning Goals

- Understand the difference between *authentication* and *authorization*.
 - Restrict access to routes to authorized users only.
-

Key Vocab

- **Identity and Access Management (IAM)**: a subfield of software engineering that focuses on users, their attributes, their login information, and the resources that they are allowed to access.
 - **Authentication**: proving one's identity to an application in order to access protected information; logging in.
 - **Authorization**: allowing or disallowing access to resources based on a user's attributes.
 - **Session**: the time between a user logging in and logging out of a web application.
 - **Cookie**: data from a web application that is stored by the browser. The application can retrieve this data during subsequent sessions.
-

Introduction

So far, we've been talking about how to **authenticate** users, i.e., how to confirm that a user is who they say they are. We've been using their username as our means of authentication; in the future, we'll also add a password to our authentication process.

In addition to **authentication**, most applications also need to implement **authorization**: giving certain users permission to access specific resources. For example, we might want **all** users to be able to browse blog posts, but only **authenticated** users to have access to premium features, like creating their own blog posts. In this lesson, we'll learn how we can use the session object to authenticate users' requests, and give them explicit permission to access certain routes in our application.

First Pass: Manual Checks

Let's say we have a **Document** resource. Its `get()` method looks like this:

```
class Document(Resource):
    def get(self, id):
        document = Document.query.filter(Document.id == id).first()
        return document.to_dict()
```

Now let's add a new requirement: documents should only be shown to users when they're logged in. From a technical perspective, what does it actually mean for a user to *log in*? When a user logs in, all we are doing is using cookies to add their `user_id` to the `session` object.

The first thing you might do is to add a **guard clause** as the first line of `Document.get()` :

```
class Document(Resource):
    def get(self, id):

        if not session['user_id']:
            return {'error': 'Unauthorized'}, 401

        document = Document.query.filter(Document.id == id).first()
        return document.to_dict()
```

Unless the session includes `user_id` , we return an error. In this case, if a user isn't logged in, we return `401 Unauthorized` .

Refactor

This code works fine, so you use it in a few places. Now your `Document` resource looks like this:

```
class Document(Resource):
    def get(self, id):

        if not session['user_id']:
            return {'error': 'Unauthorized'}, 401

        document = Document.query.filter(Document.id == id).first()
        return document.to_dict()

    def patch(self, id):

        if not session['user_id']:
            return {'error': 'Unauthorized'}, 401
```


```
# patch code

def delete(self, id):

    if not session['user_id']:
        return {'error': 'Unauthorized'}, 401

# delete code
```

That doesn't look so DRY. Wouldn't it be great if there were a way to ask Flask to run some code **before** any **action**?

Fortunately, Flask gives us a solution: **before_request** 

(https://flask.palletsprojects.com/en/2.2.x/api/?highlight=before_request#flask.Flask.before_request).

We can refactor our code like so:

```
@app.before_request
def check_if_logged_in:
    if not session['user_id']:
        return {'error': 'Unauthorized'}, 401

class Document(Resource):
    def get(self, id):

        document = Document.query.filter(Document.id == id).first()
        return document.to_dict()

    def patch(self, id):

        # patch code

    def delete(self, id):

        # delete code
```

We've moved our guard clause into its own function and that's it! Request hooks in Flask act upon objects that manipulate the **request** context *automatically*. This means that if an object is configured to do anything to a request, our **before_request** hook will be executed first.

Skipping Filters for Certain Endpoints

What if we wanted to let anyone see a list of documents, but keep the **before_request** hook for the **Document** methods? We could do this:

```
@app.before_request
def check_if_logged_in:
    if not session['user_id'] \
        and request.endpoint != 'document_list' :
        return {'error': 'Unauthorized'}, 401

class Document(Resource):
    def get(self, id):

        document = Document.query.filter(Document.id == id).first()
        return document.to_dict()

    def patch(self, id):

        # patch code

    def delete(self, id):

        # delete code

class DocumentList(Resource):
    def get(self):

        documents = Document.query.all()
        return [document.to_dict() for document in documents]

api.add_resource(Document, '/documents/<int:id>', endpoint='document')
api.add_resource(DocumentList, '/documents', endpoint='document_list')
```

This added if/else logic tells Flask to ignore certain resources, defined by a string `endpoint` . This is set automatically to the lowercase version of the class or function name, but it's often best to be explicit and name it when we add our resources.

Conclusion

To **authorize** a user for specific actions, we can take advantage of the fact that all logged in users in our application will have a `user_id` saved in the session object. We can use a `before_request` hook to run some code that will check the `user_id` in the session and only authorize users to run those actions if they are logged in.



Check For Understanding

Before you move on, make sure you can answer the following questions:

- ▶ 1. *What is the difference between authentication and authorization?*

- ▶ 2. *Which Flask decorator can we use to add an authorization step before each of the actions in our app?*

Resources

- [API - Flask: `before_request\(f\)`](https://flask.palletsprojects.com/en/2.2.x/api/?highlight=before_request#flask.Flask.before_request)  [_\(https://flask.palletsprojects.com/en/2.2.x/api/?highlight=before_request#flask.Flask.before_request\)](https://flask.palletsprojects.com/en/2.2.x/api/?highlight=before_request#flask.Flask.before_request)
- [What is Authorization? - auth0](https://auth0.com/intro-to-iam/what-is-authorization)  [_\(https://auth0.com/intro-to-iam/what-is-authorization\)](https://auth0.com/intro-to-iam/what-is-authorization)