miguelgrinberg.com

The Flask Mega-Tutorial, Part XII: Dates and Times

Posted by Miguel Grinberg *on* December 3, 2023 *under* Python Programming Flask

This is the twelfth installment of the Flask Mega-Tutorial series, in which I'm going to tell you how to work with dates and times in a way that works for all your users, regardless of where they reside.

You are reading the 2024 edition of the Flask Mega-Tutorial. The complete course is also available to order in e-book and paperback formats from Amazon. Thank you for your support!

If you are looking for the 2018 edition of this course, you can find it here.

For your reference, here is the complete list of articles in this series:

One of the aspects of my Microblog application that I have ignored for a long time is the display of dates and times. Until now, I've just let Python render the `datetime` object in the `User` model, and have not even bothered displaying the one in the `Post` model. In this chapter you will learn how to work with these timestamps.

The GitHub links for this chapter are: [Browse](#), [Zip](#), [Diff](#).

Timezone Hell

Using Python on the server to render dates and times that are presented to users on their web browsers is really not a good idea, because what the server considers to be its local time is not going to make sense for users that live in a different timezone.

It is pretty clear that the server must manage times that are consistent and independent of its own location and the location of users. If this application grows to the point of needing several production servers in different regions around the world, I would not want each server to write timestamps to the database in a different timezone, because that would make working with these times impossible. Since UTC is the most used uniform timezone and is supported in the `datetime` class, that is what I'm going to use.

In [Chapter 4](#) you have seen how to generate UTC timestamps for blog posts. As a reminder, here is a short example that shows how this was done:

```
>>> from datetime import datetime, timezone
>>> str(datetime.now(timezone.utc))
'2023-11-19 19:05:51.288261+00:00'
```

But there is an important problem with this approach. For users in different locations, it will be awfully difficult to figure out when a post was made if they see times in the UTC timezone. They would need to know in advance that the times are in UTC so that they can mentally adjust the times to their own timezone. Imagine a user in, say, the PDT timezone in the US West Coast that posts something at 3:00pm, and immediately sees that the post appears with a 10:00pm UTC time, or to be more exact 22:00. That is going to be very confusing.

While standardizing the timestamps to UTC makes a lot of sense from the server's perspective, this creates a usability problem for users. The goal of this chapter is to present a solution that keeps all the timestamps that are managed by the server in the UTC timezone, without alienating users.

Timezone Conversions

The obvious solution to the problem is to convert all timestamps from the stored UTC units to the local time of each user when they are rendered. This allows the server to continue using UTC for consistency, while an on-the-fly conversion tailored to each user solves the usability problem. The tricky part of this solution is to know the location of each user.

Many websites have a configuration page where users can specify their timezone. This would require me to add a new page with a form in which I present users with a dropdown with the list of timezones. Users can be asked to enter their timezone when they access the site for the first time, as part of their registration.

While this is a decent solution that solves the problem, it is a bit odd to ask users to enter a piece of information that they have already configured in their operating system. It seems it would be more efficient if I could just grab the timezone setting from their computers.

As it turns out, the web browser knows the user's timezone, and exposes it through the standard date and time JavaScript APIs. There are actually two ways to take advantage of the timezone information available via JavaScript:

- The "old school" approach would be to have the web browser somehow send the timezone information to the server when the user first logs in to the application. This could be done with an Ajax call, or much more simply with a meta refresh tag. Once the server knows the timezone it can keep it in the user's session or write it to the users table in the database, and from then on adjust all timestamps with it at the time templates are rendered.
- The "new school" approach would be to not change a thing in the server, and let the conversion from UTC to local timezone happen in the browser, using JavaScript.

Both options are valid, but the second one has a big advantage. Knowing the timezone of the user isn't always enough to present dates and times in the format expected by the user. The browser has also access to the system locale configuration, which specifies things like AM/PM vs. 24-hour clock, DD/MM/YYYY vs. MM/DD/YYYY date rendering format and many other cultural or regional styles.

And if that isn't enough, there is yet one more advantage for the new school approach. There is an open-source library that does all this work!

Introducing Moment.js and Flask-Moment

Moment.js is a small open-source JavaScript library that takes date and time rendering to another level, as it provides every imaginable formatting option, and then some. And a while ago I created Flask-Moment, a small Flask extension that makes it very easy to incorporate moment.js into your application.

So let's start by installing Flask-Moment:

```
(venv) $ pip install flask-moment
```

This extension is added to a Flask application in the usual way:

app/__init__.py: Flask-Moment instance.

```
# ...
from flask_moment import Moment

app = Flask(__name__)
# ...
moment = Moment(app)
# ...
```

Unlike other extensions, Flask-Moment works together with *moment.js*, so all templates of the application must include this library. To ensure that this library is always available, I'm going to add it in the base template. This can be done in two ways. The most direct way is to explicitly add a `<script>` tag that imports the library, but Flask-Moment makes it easier, by exposing a `moment.include_moment()` function that generates the `<script>` tag:

app/templates/base.html: Include moment.js in the base template.

```
    ...

        {{ moment.include_moment() }}
      </body>
</html>
```

In most cases JavaScript libraries used by the application are included at the end of the `<body>` contents, where the Bootstrap JavaScript code is.

Using Moment.js

Moment.js makes a `moment` class available to the browser. The first step to render a timestamp is to create an object of this class, passing the desired timestamp in ISO 8601 format. Here is an example running in the browser's JavaScript console:

```
t = moment('2021-06-28T21:45:23+00:00')
```

If you are not familiar with the ISO 8601 standard format for dates and times, the format is as follows:

```
{yyyy}-{mm}-{dd}T{hh}:{mm}:{ss}{tz}
```

I have already decided that I was only going to work with UTC timezones, so the last part is always going to be `+00:00` or in some cases the equivalent `Z`, which represents UTC in the ISO 8601 standard.

The `moment` object provides several methods for different rendering options. Below are some of the most common options:

```
moment('2021-06-28T21:45:23+00:00').format('L')
'06/28/2021'
moment('2021-06-28T21:45:23+00:00').format('LL')
'June 28, 2021'
moment('2021-06-28T21:45:23+00:00').format('LLL')
'June 28, 2021 10:45 PM'
moment('2021-06-28T21:45:23+00:00').format('LLLL')
'Monday, June 28, 2021 10:45 PM'
moment('2021-06-28T21:45:23+00:00').format('dddd')
'Monday'
moment('2021-06-28T21:45:23+00:00').fromNow()
'2 years ago'
```

This example creates a moment object initialized to June 28th 2021 at 9:45pm UTC. You can see that all the options I tried above are rendered in UTC+1, which is the timezone configured on my computer. You can enter the above commands in your browser's console, making sure the page on which you open the console has moment.js included. You can do it in microblog, as long as you made the changes above to include moment.js, or also on *https://momentjs.com/*.

Note how the different methods create different representations. With `format()` you control the format of the output with a format string. The `fromNow()` method is interesting because it renders the timestamp in relation to the current time, so you get output such as "a minute ago" or "in two hours", etc.

If you were working directly in JavaScript, the above calls return a string that has the rendered timestamp. Then it is up to you to insert this text in the proper place on the page, which unfortunately requires working with the DOM. The Flask-Moment extension greatly simplifies the use of moment.js by enabling a `moment` object similar to the JavaScript one in your templates.

Let's look at the timestamp that appears in the profile page. The current *user.html* template lets Python generate a string representation of the time. I can now render this timestamp using Flask-Moment as follows:

app/templates/user.html: Render timestamp with moment.js.

```
{% if user.last_seen %}
<p>Last seen on: {{ moment(user.last_seen).format('LLL
{% endif %}
```

So as you can see, Flask-Moment uses a syntax that is similar to that of
the JavaScript library, with one difference being that the argument to
`moment()` is now a Python `datetime` object and not an ISO 8601 string.
The `moment()` call issued from a template automatically generates the
required JavaScript code to insert the rendered timestamp in the proper
place of the DOM.

The second place where I can take advantage of Flask-Moment is in the
_post.html sub-template, which is invoked from the index and user
pages. In the current version of the template, each post preceded with a
"username says:" line. Now I can add a timestamp rendered with
`fromNow()`:

app/templates/_post.html: Render timestamp in post sub-template.

```
<a href="{{ url_for('user', username=post.author.userr
    {{ post.author.username }}
</a>
said {{ moment(post.timestamp).fromNow() }}:
<br>
{{ post.body }}
```

Below you can see how both these timestamps look when rendered with
Flask-Moment and moment.js:
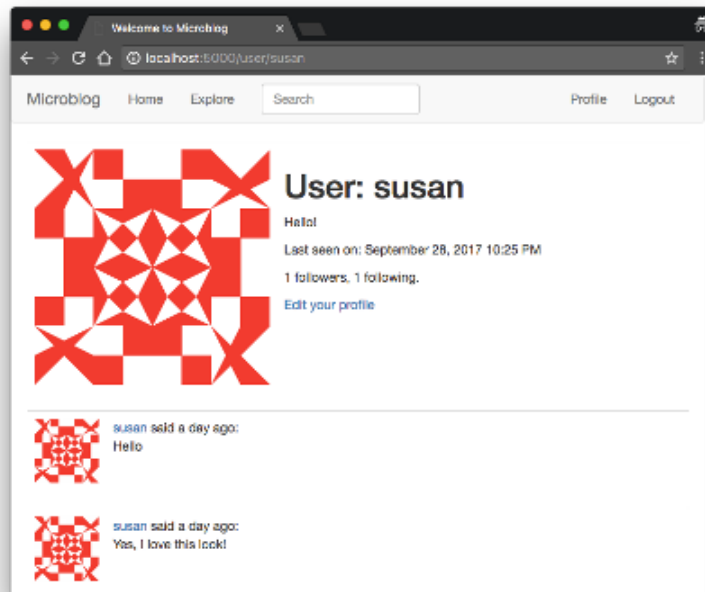
Continue on to the next chapter.

Become a Patron!

Hello, and thank you for visiting my blog! If you enjoyed this article, please consider supporting my work on this blog on Patreon!

Share this post:

Hacker News   Reddit   Twitter   LinkedIn   Facebook

E-Mail

8 comments

#1   **Finlay**   said 9 months ago

Hi, firstly thank you for putting this tutorial together, I have really benefitted from it so far and still have a lot to digest.
I am creating my own project which displays matplotlib graphs. Now that the app has grown to be a certain size it is taking quite a long time to load some pages. I would like to know which part of my script is taking a long time so that I can trim those bits or make them more efficient. I have read that there are things that can be added to code to measure the execution time. However I'm not sure how to implement that, especially across the flask application as it switches between different .py and .html files in

the execution. And I'm not sure in which file the 'slow code' is located. Do you have any suggestions?

#2   **Miguel Grinberg**   said 9 months ago

@Finlay: You can use the Profiler Middleware: https://werkzeug.palletsprojects.com/en/3.0.x/middleware/profiler/. This provides a detail of time spent in every function called during the course of a request. The current version of the Flask Mega-Tutorial does not use this profiler, but long ago I had a short section that showed how to use it, which you can find by searching for the "Profiling for Performance" section in this page: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xvi-debugging-testing-and-profiling

#3   **Finlay**   said 9 months ago

@Miguel Thank you. I have added the script to routes.py and it is working. There are many functions that are printed. I would like to possibly have the printed functions narrowed down, but this is a good start.

#4   **Miguel Grinberg**   said 9 months ago

@Finlay: you are going to need to learn a bit how to work with the profiler. See the `restrictions` argument for how to trim down the list of shown functions.

#5   **Vab**   said 4 months ago

Converting time to UTC has been really useful for many of my projects, thank you for sharing this Miguel.

However, I want to know what could be best way to convert datetime that client enters, for example, `in_time` and `out_time` for a class. Eg, if user enters in_time as 6:20pm, should I also get the timezone of user using the old school JavaScript API?

#6   **Miguel Grinberg**   said 4 months ago

@Vab: if you can get the browser to do the conversion and send UTC to the server that would be the most convenient, because the browser knows the timezone of the user, as you clearly know already. You can also send the ISO8601 representation of the

time entered by the user, which includes the timezone, and then the server would convert that to UTC.

#7 **Momoko Price** said 4 months ago

I noticed in the Moment.js documentation they recommend using Luxon and other alternatives these days: https://momentjs.com/docs/#/-project-status/

I haven't found any flask extension for these newer alternatives, and haven't dug into the details enough to fully understand the pros vs. cons of *not* using Moment.js quite yet.

Would it be substantially more difficult to try using something like Luxon, etc. in a flask app without an extension to simplify the implementation as you've demonstrated above with Flask-Moment?

#8 *Miguel Grinberg* said 4 months ago

@Momoko: it really depends on how comfortable you are writing JavaScript. If you can do it, then you can use any library that you like without a Flask integration.

| «« | « | » | »» |

Leave a Comment

Name

Email

Comment

Captcha

I'm not a robot

reCAPTCHA
Privacy - Terms

Submit

Flask Web Development, 2nd Edition

If you want to learn modern web development techniques with Python and Flask, you may find the second edition of my O'Reilly book useful.

Click here to get this Book!

About Miguel

Welcome to my blog!

I'm a software engineer and technical writer, currently living in Drogheda, Ireland.

You can also find me on Twitter, Mastodon, Github, LinkedIn, YouTube, Facebook and Patreon.

Thank you for visiting!

Categories

AI   *2*
AWS   *1*
Arduino   *7*
Authentication   *10*

Blog *1*

C++ *5*

CSS *1*

Cloud *10*

Database *22*

Docker *5*

Filmmaking *6*

Flask *126*

Games *1*

Heroku *1*

IoT *8*

JavaScript *35*

MicroPython *9*

Microdot *1*

Microservices *2*

Movie Reviews *5*

OpenStack *1*

Personal *3*

Photography *7*

Product Reviews *2*

Programming *187*

Project Management *1*

Python *169*

REST *7*

Rackspace *1*

Raspberry Pi *8*

React *18*

Robotics *6*

Security *12*

Video *22*

WebSocket *2*

Webcast *3*

Windows *1*