Saving time in UTC doesn't work and offsets aren't enough

Sometimes you learn a lesson so painfully deep, you'll have wisdom to share for the rest of your days. This is one of those.

We built a system for recurring appointments. Supports features like *"I want to see my provider every Wednesday at 3pm"*

Our system saves the ops team many hours per week of manual work and unlocks scaling the business. You can't scale if overhead grows faster than users. Great success for engineering 🥳

Besides, computers are better than humans at making weekly events. Obviously. Why we haven't done this sooner, nobody knows.

And then March 13th happened. Daylight savings. 💥

Your first instinct is pretty okay

How do you handle timezones?

Conventional wisdom says that dealing with timezones is tricky to think about and trivial to solve. You save and communicate in UTC and use the client's current timezone for display.

Wednesday at 3pm looks like this in your database and your API payload:

```
2022-03-09T23:00:00Z
```

That `Z` at the end means Zulu Time. *The* universal UTC timezone.

Pass that into a date constructor and you get the correct time for your client's timezone. Part of every modern programming language.

JavaScript running in San Francisco gives you:

```
new Date('2022-03-09T23:00:00Z')
> Wed Mar 09 2022 15:00:00 GMT-0800 (PST)
```

Yay Wednesday at 3pm.

If you never need more than that, you're good. Save in UTC. That's what we did and thought we were done. 👍

You might need UTC offset on the server

What happens if you need to send users a reminder?

"You have an appointment in 2 hours" is easy. Delta time works great with UTC. Run a process, take current server time, fetch appointments at `current_time + 2 hours`, send notifications. Great.

"You have an appointment today at 3pm" is tricky. Your server runs in UTC (usually), the user is who knows where. Their 3pm is not your server's 3pm 💩

One solution is to save time with a UTC offset.

Like this:

```
2022-03-09T15:00:00-08:00
```

Instead of `Z` for zulu time, we have an offset that says this time is 8 hours behind UTC. At 3pm.

Now your server understands the *user's intention* of 3pm *and* the exact point in time for UTC. Fantastic.

You can run a process, fetch all appointments for today, and send notifications rendered using the timestamp without offset. It's 3pm. For the user.

Great! Problem solved. Except ...

Your UTC offset is wrong

What if your user scheduled an appointment in San Francisco then traveled to New York. 3pm turns into 12pm.

At best your notification will be wrong, at worst the user wanted 3pm in NYC because they were planning to travel.

But at least `new Date('2022-03-09T15:00:00-08:00')` does the right thing. Run that on the client and users get `Wednesday 12pm`. As long as that's what they meant, all good.

You can solve this by asking users what timezone they're scheduling for. Explicitly. You'll need an up-to-date current location for server-side rendering like notifications.

A nice trick for physical appointments is to schedule in the location's timezone and ignore your user's current time. They'll be there in person when it matters.

We decided not to worry about this case for now.

UTC offsets and recurring events 💩

Swizec Teller published ServerlessHandbook.dev
@Swizec

You haven't seen timezone hell until you've tried scheduled and recurring events.

Here's where it gets cooky. Your UTC offset may be wrong *even if the user never moves*. Because of DST.

The March sequence for "Wednesday at 3pm" looks like this:

```
2022-03-02T15:00:00-08:00
2022-03-09T15:00:00-08:00
2022-03-16T15:00:00-07:00
2022-03-23T15:00:00-07:00
2022-03-30T15:00:00-07:00
```

Notice the offset change from the 9th to 16th. That's because USA springs forward by an hour on March 13th.

That same sequence for a European user looks like this:

```
2022-03-02T15:00:00+01:00
2022-03-09T15:00:00+01:00
2022-03-16T15:00:00+01:00
2022-03-23T15:00:00+01:00
2022-03-30T15:00:00+02:00
```

Because Europe springs forward on March 27th. 🙃

Users in Arizona and many countries around the world don't use DST at all. As a US company focusing on the US market, Arizona is the only oddity we have to handle. Phew.

As a side note, the situation used to be way worse! Before the law standardized American DST in 1966, each state, even city, had different rules. Europe standardized in 1996.

You can try to avoid this issue by saving in UTC zulu time, but that's worse. The time changes:

```
2022-03-02T23:00:00Z
2022-03-09T23:00:00Z
2022-03-16T22:00:00Z
2022-03-23T22:00:00Z
2022-03-30T22:00:00Z
```

At least, the time is *supposed to* change. But *you* have to make that happen. We didn't. This meant all schedules were by 1 hour after March 13th 💩

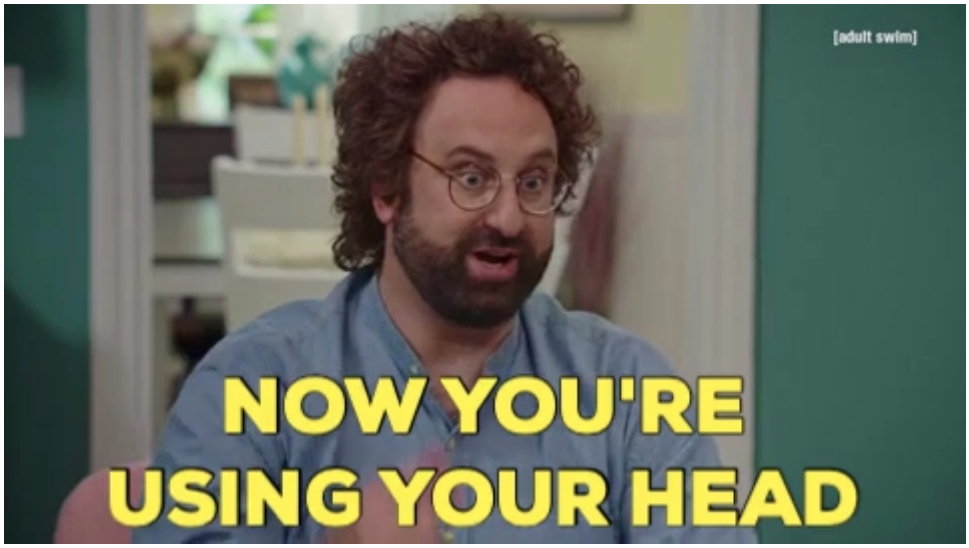
And because we didn't store the user's intent, this data was hard to fix. The time shouldn't change for people in Arizona.

How to correctly handle scheduled events

The problem with scheduled events is that timezones change.

"US West Coast Time" shifts by 1 hour twice a year. Egypt canceled DST with 3 days of warning in 2016. Samoa changed which side of the date line they're on in 2011. USA is thinking about ending DST soon. The 1582 Gregorian calendar change skipped 10 days of that year.

When a user says *"Gimme appointment next Wednesday at 3pm"* and all you get is a UTC timestamp (with or without offset), you can't quite know what that means. 3pm in their current timezone? 3pm in the timezone that Wednesday? What if that Wednesday's timezone changes between now and when the event happens?



Use the IANA Timezone Database my friend! That's what it's for. A whole group of people that keeps track of timezones for you 🥳

> We are responsible for coordinating some of the key elements that keep the Internet running smoothly. Whilst the Internet is renowned for being a worldwide network free from central coordination, there is a technical need for some key parts of the Internet to be globally coordinated, and this coordination role is undertaken by us.

Time + timezone is the way

Here's what you do:

1. Timestamp without offset on the API
2. Extra param with desired timezone in IANA format
3. Store as timezone aware in your database (`timestamptz` for postgres)
4. *Store the intended timezone*

You'll need the intended timezone for date math and rendering on the

timezone, which by convention is UTC on the server. Means you need the user's timezone to translate back.

Fun fact: Different database clients behave differently. A query that runs fine for one engineer may bork the database for another. We learned that gotcha when production shifted by 7 hours 💩

The correct way to send "Wednesday at 3pm in San Francisco" looks like this:

```
{ ...
    timestamp: {
        time: '2022-03-16T15:00:00`
        tz: 'America/Los_Angeles'
    }
}
```

ISO8601 format without timezone for the time, IANA Timezone label for the timezone. Notice that IANA is based on nearest major city *not* current offset. That's important.

You save that in the database as:

```
time                                  | tz
2022-03-16T15:00:00 America/Los_Angeles | America/Los_Angeles
```

Postgres allows saving time with IANA timezones, your database may differ. Keep the timezone in a separate column so you can translate.

Doing timezone aware date math

Date math is where all this gets even more fun

A day is not 60*60*24 seconds long and a week may not be exactly 7 days either. That *will* break your recurring event logic. Ask me how I know 😈

We started this article with *"Every Wednesday at 3pm"* and figured out how to handle "Wednesday at 3pm". What about the "every" part?

Your first instinct is likely the same as mine – date + 1 week. Done.

But the result of `2022-03-09T15:00:00 + 1 week` depends on which timezone you're talking about. For Arizona, that's a normal week. For the rest of USA, it's 1 hour shorter.

Naively add `1 week` and you get `2022-03-16T14:00:00`, which is wrong. Then you have to fix everyone's data and wow that was an un-fun weekend I'll tell you that.

Here's what you do, using date-fns-tz or similar:

```
import { zonedTimeToUtc, utcToZonedTime } from 'date-fns-tz'
import { addWeeks } from 'date-fns'

// returns Date
const lastTime = zonedTimeToUtc('2022-03-09T15:00:00', 'America/Los_Angeles')

const nextTimestamp = addWeeks(lastTime, 1)

// prints 2022-03-16T22:00:00.000Z in UTC
const timeToSave = utcToZonedTime(nextTimestamp, 'America/Los_Angeles')
```

Take time without offset from your database, interpret in the target timezone. Becomes a UTC point in time. Do your date math. Convert back to a timezoned Date object, which you can send straight to your timezone-aware database column or client.

For me, printing the zoned time in node.js renders as UTC. In a browser, it would render as local time. This makes timezone math confusing to debug 🙃

Key takeaway

All that to say: Use UTC for specific points in time, time + timezone for scheduled and recurring events.

Good luck!

Cheers,
~Swizec

Did you enjoy this article? 👎 👍

Published on March 25th, 2022 in Time, UTC, Daylight Saving, Lessons

Join Swizec's Newsletter

And get thoughtful letters 💌 on **mindsets, tactics, and technical skills** for your career. Real lessons from building production software. No bullshit.

> "Man, love your simple writing! Yours is the only newsletter I open and only blog that I give a fuck to read & scroll till the end. And wow always take away lessons with me. Inspiring! And very relatable. 👌"
> ~ Ashish Kumar

Your Name

John Doe

Your Email

email@example.com

Subscribe & Become an expert 💌

Join 15,161+ engineers just like you already growing their careers with my emails, workshops, books, and courses.

4.5 stars average rating

Have a burning question that you think I can answer? Hit me up on twitter and I'll do my best.

Who am I and who do I help? I'm Swizec Teller and I turn coders into engineers with *"Raw and honest from the heart!"* writing. No bullshit. Real insights into the career and skills of a modern software

Curious about Serverless and the modern backend? Check out Serverless Handbook, for frontend engineers 👉 ServerlessHandbook.dev

Want to Stop copy pasting D3 examples and create data visualizations of your own? Learn how to build scalable dataviz React components your whole team can understand with React for Data Visualization

Want to become a *true* senior engineer? Getting the title's easy. Just stick around. Being a *true* senior takes a new way of thinking. The Senior Mindset Series can help 👉 SeniorMindset.com. It's my most loved series of essays.

Want to get my best emails on JavaScript, React, Serverless, Fullstack Web, or Indie Hacking? Check out swizec.com/collections

Want to brush up on modern JavaScript syntax? Check out my interactive cheatsheet: es6cheatsheet.com

Did someone amazing share this letter with you? Wonderful! You can sign up for my weekly letters for software engineers on their path to greatness, here: swizec.com/blog

Want to brush up on your modern JavaScript syntax? Check out my interactive cheatsheet: es6cheatsheet.com

By the way, just in case no one has told you it yet today: I love and appreciate you for who you are ❤️

Created by Swizec with ❤️

Articles
Interviews
Collections
Books
Courses
Workshops
About