Why Am I Always Late?
Or Is It Early?
Why Are Dates So Hard?

Dan Mallott

28 FEBRUARY 2025

AGENDA

01 Date and Time Fallacies

02 Storing Dates and Times

03 What About Code?

04 Display to the User

05 Remediation

Please fill an evaluation!

Dan Mallott

@danielmallott.bsky.social
github.com/danielmallott
linkedin.com/in/danielmallott
dmallott@westmonroe.com

Senior Principal in Technology & Experience at West Monroe

Developing software (and an occasional DBA) since 2011

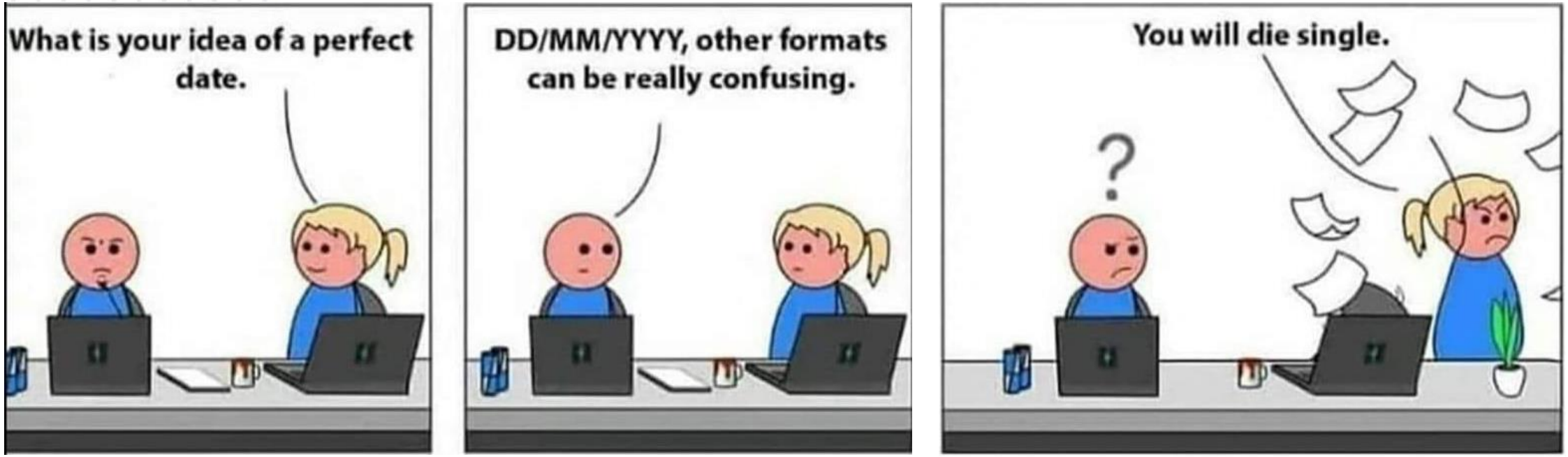
Most experience is with Microsoft technologies, specifically .NET and SQL Server

Ice hockey referee (USA Hockey and DIU) – ask me about how much fun playoffs are!

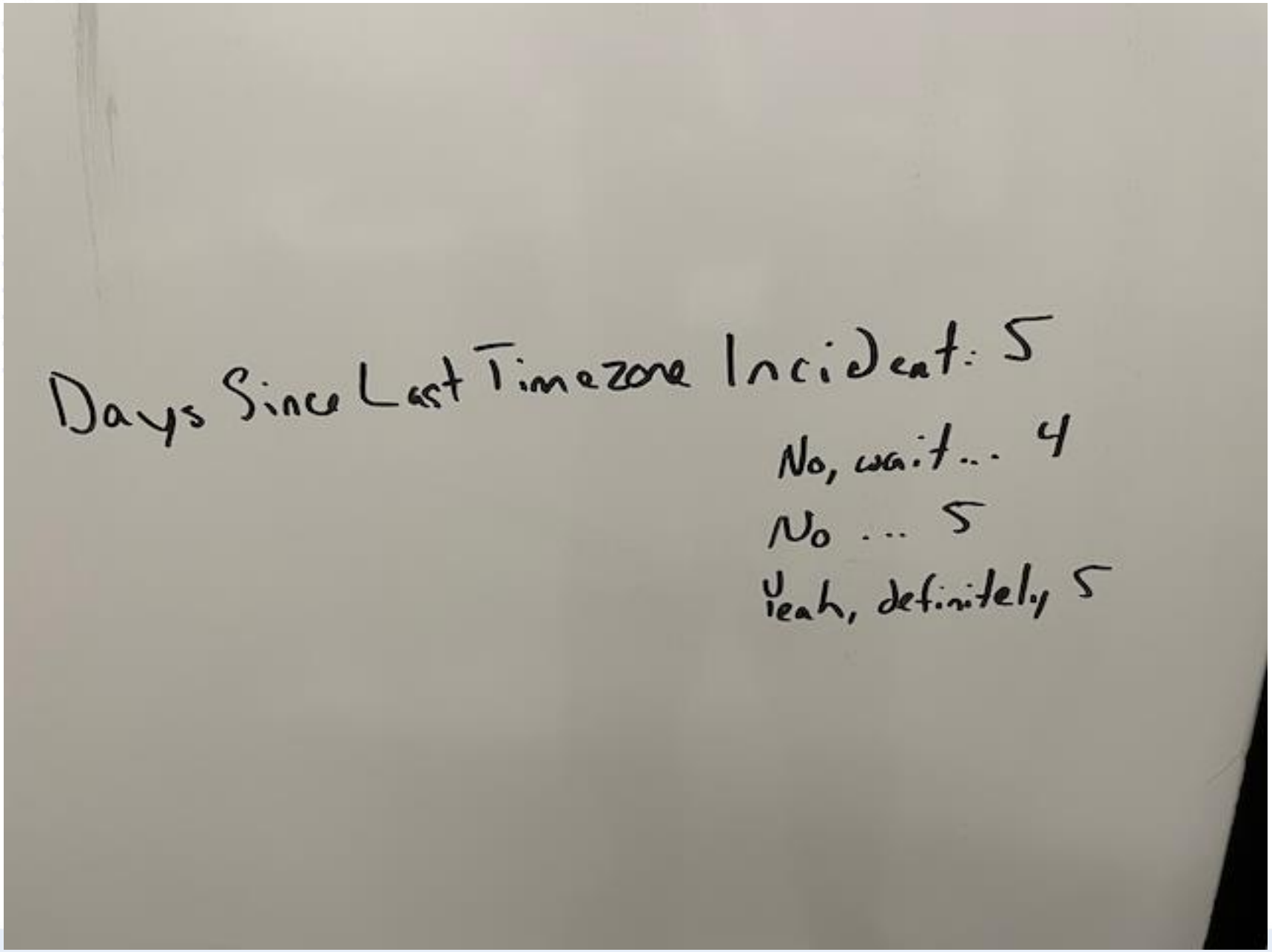
01

Dates and Time Zones

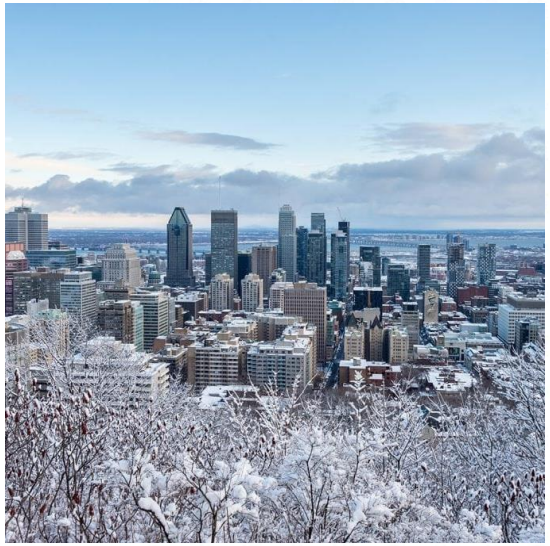
Or…how to drive people who think in 1’s and 0’s nuts



Source: Unknown



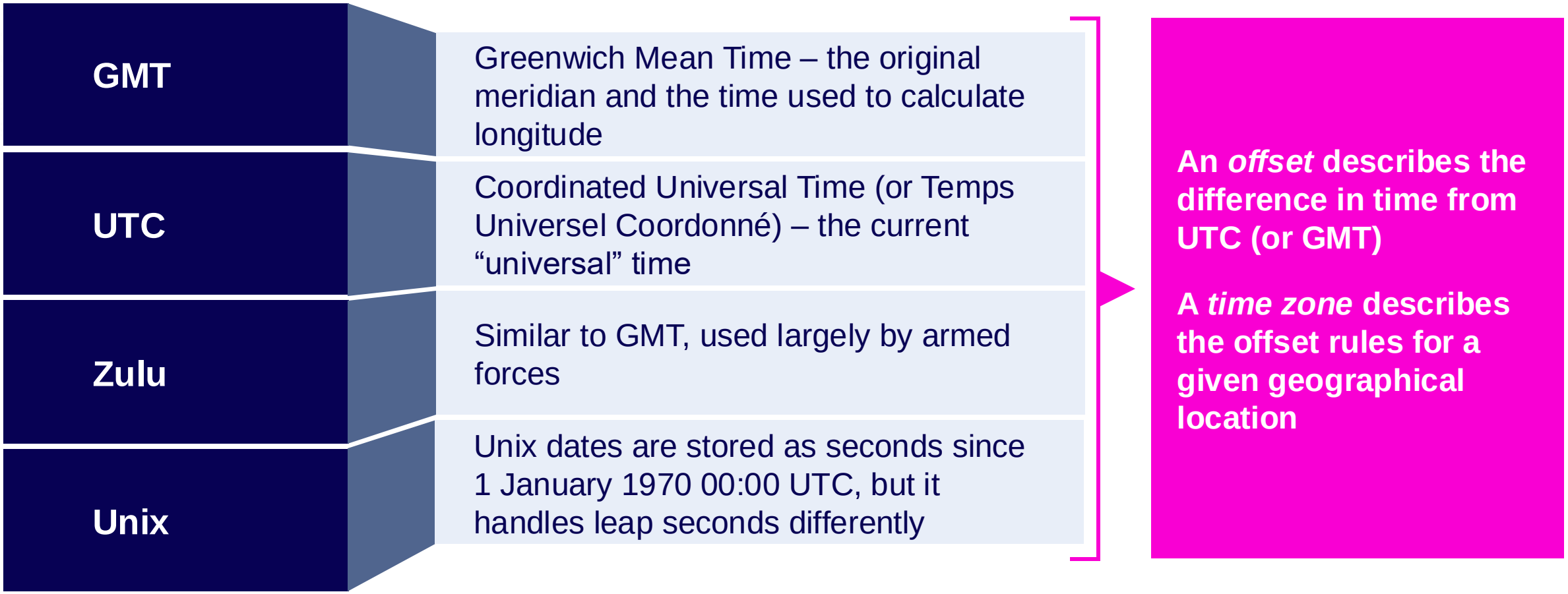
First sanity check:
Are CDT and EST the same?



Time Zone Fallacies

01	**UTC offsets range from -12 to 12 in increments of ~~1~~ 0.5**	UTC offsets range from -12 to +14 and can increment by as little as 0.25 (Nepal, the Chatham Islands, Eucla)
02	**Every offset corresponds to one and only one time zone name**	UTC+5 corresponds to ***10*** named time zones, and most offsets correspond to at least 2
03	**A country or area stays at the same offset for the entire year**	The Americas and Europe ”enjoy” Daylight Savings Time (or Summer Time), changing the offset for those areas for part of the year
04	**All offsets change at the same time every year**	Europe and the Americas switch from Standard Time to Daylight Savings Time at different times (usually a week apart)
05	**UTC offsets always change by 1 hour for DST**	Lord Howe Island shifts by 30 minutes(!)

What Are Offsets and Time Zones?



At Least Dates Are Simple, Right?

We are very used to using the Gregorian calendar

- Developed by the Catholic church and propagated by Pope Gregory XIII in 1582
- Corrected calendar drift in the Julian calendar by using a more precise number for year length
- Initially adopted by European Catholic countries in 1582, most of the rest of the world by 1919

But what about other calendars?

Gregorian	28 Feburary 2025
Hebrew	30 Shevat 5785
Ethiopic	21 Yakatit 2017
Chinese	Cycle 78, year 42, month 2, day 1
Julian Day	2460735

Julian	15 Feburary 2025
Islamic	29 Sha’ban 1446
Coptic	21 Amshir 1741
Mayan	Long count = 13.0.12.6.12; tzolkin = 10 Eb; haab = 15 Kayab
Unix Date	1740700800 - 1740787199

02

Storing Dates and Times

Or…how to try not to mess it up

So if I just stored everything in UTC, my problem is solved, right?

Have You Asked These Questions?

01	DOES THE DATE RELATE TO SPECIFIC PLACE?
02	WHERE ARE THE USERS?
03	WHERE & WHAT ARE THE SYSTEMS INVOLVED?
04	WHAT IS THE DATE RANGE?
05	WHAT IS THE NECESSARY PRECISION?
06	ARE THERE ANY LEGAL CONCERNS OR IMPLICATIONS?

General Guidance

Past Dates

- Storing historical dates as UTC is generally accepted to be the best practice.
- If you want to (reliably) display the time, consider storing the offset (or maybe the entire local time)
- NOTE: The time zone is less useful, as these change over time

Future Dates

- Storing future dates as UTC might create problems
- Even storing the offset might not help – what happens when the dates for Daylight Savings Time change again?
- Store user-entered or controlled future times in the time zone they were entered in – this is the intended time

Some (More) Specific Guidance

I only need to store a date	**DATE (or DATETIME or DATETIME2)**
I only need to store a time, and I don't care about time zone	**TIME**
I need to store a date and time, but I don't care about time zone	**DATETIME or DATETIME2**
I need to store a date and time, and I care about time zone	**DATETIMEOFFSET**
I need to store a date and time, and I care about time zone, and I care about storing the original offset	**DATETIMEOFFSET + DECIMAL**

We are assuming SQL Server 2008 or newer here. Other databases will have other options

03

What About Code?

Or…what you really care about in this talk

Let's Talk About Libraries

Language	Library	Minimum Value	Maximum Value	Time Zone Aware?
C#	DateTime	1 January 0001, 00:00	31 December 9999 23:59	Nope
C#	DateTimeOffset	1 January 0001, 00:00 +00:00	31 December 9999 23:59 +00:00	Offset only
C#	DateOnly	1 January 0001	31 December 9999	Nope
C#	TimeOnly	00:00:00	23:59:59	Nope
Java	LocalDateTime	-999999999-01-01T00:00:00	+999999999-12-31T23:59:59.99	Nope
Java	OffsetDateTime	-999999999-01-01T00:00:00+18:00	999999999-12-31T23:59:59.99-18:00	Offset only
Java	ZonedDateTime	-999999999-01-01T00:00:00+18:00	999999999-12-31T23:59:59.99-18:00	Yes
JavaScript	Date	-271821-04-20T00:00:00+00:00	275760-09-13T00:00:00-00:00	Sort of…

Python, PHP, and other languages have their own conceptions of how to represent dates and times in code

It is worth your time to read the documentation!

And Testing!

ALWAYS	Write tests for date- and time-related functions
NEVER	Use non-deterministic built-in functions to generate dates and times in tests
ALWAYS	Deliberately construct date and time objects with known values
REMEMBER	CI/CD servers (and application servers) have time zones that are (probably) different from your laptop‘s
ALWAYS	Test edge cases, like minimums, maximums, daylight savings, elapsed time calculations

04

Display to the User

Or…why do browsers have a mind of their own?

Some General Guidance

DON'T HAVE TO DISPLAY IN A TIME ZONE?

- JavaScript (and its many frameworks) already handle this well
- Particularly useful when storing as UTC or in the user's time zone

USERS CAN CONFIGURE THE TIME ZONE THEY WANT TO SEE?

- Pass offset information on every API call
- Be careful with using JS's `Date()` object
- Test your code for displaying dates

LIBRARIES ARE EASIER?

- Luxon (the modern version of Moment.js) has native time zone and locale support
- Date-fns is a good alternative with native time zone support

BEWARE OF SHARP EDGES

- JSON does not have a built-in date format, so make sure you are passing ISO 8601 strings
- Watch out for localization issues (not everyone starts the week on Monday)

05

Remediation

Or…the thing no one wants to do

You Can’t Fix the Past…Or Can You?

1: ADMIT YOU HAVE A PROBLEM

Determining you currently have a problem is the first step – your product owner may already know this!

2: START DOING IT RIGHT

Fix the code and update data storage to store the correct granularity

Write tests to verify functionality

3: FIX HISTORY

Update dates to include time zone information (if known)

Discard time information if it doesn’t matter

Document everything

06

Questions?

Thank You!

@danielmallott.bsky.social
github.com/danielmallott
linkedin.com/in/danielmallott
dmallott@westmonroe.com

Please fill an evaluation!

west monroe