Scenario A -- Complex Logic

> **TARGET EXPERTISE**: Expert/Senior Engineer
> **EXPECTED TIME**: 3.0 - 4.0h
>
> **The focus and goal of this assessment is to understand your ability to parse dense information and subsequently build understandable, maintainable code that implements complex processing logic over equally rich data structures.**
>
> Based on prior candidates to date, expert and senior engineers with English fluency typically complete this exercise within 2.50 - 3.50 hours. Intermediate or non-fluent engineers typically take up to 6.00 - 8.00 hours.
>
> The speed record, set in 2021, currently stands at approximately 1.50 hours! However, it is noteworthy that particular candidate did not make the cut. We favored and hired others who in fact took twice as long. Their ability to communicate, an attention to detail, and code quality factors played a far more important role.
>
> Do not rush, take your time, and *communicate!*
>
> Good luck!

Background

A key element of the Omedym Platform is presenting content to the members of personalized microsites, which are called portals. But unfortunately, answering exactly what content can be shown at a particular point in time is not simple.

Content is comprised of different media, such as videos and PDF documents. But each of these digital assets, in turn, can have a subset of questions and answers. Content and conversely these questions and answers, can also be separately grouped.

We frequently need to determine what content someone can see at a given point in time. However, it is subjective based on differing criteria and even the available state of the underlying content itself.

So much like security in a file system, the content that one portal has access to can be different from another.

However, unlike a file system, which traditionally relies on ACLs, content in Omedym relies on a series of rules that determine if content is in a state of availability, and then a series of overlapping filters and settings at upstream and downstream levels that indicate if it is selected and at that point-in-time should be available.

Domain Concepts

Content: Video or documents that can be shown in a *portal*. Often an individual piece of content, i.e. a *content item* is known as a *digital asset*. Although in fact, while all *digital assets* are content items, not all content items are *digital assets*. At any point in time, a particular piece of content in the system may not always be available, either because it's newly ingested and is still being processed, or it may not presently be in a published state due to point-in-time or other conditional rules.

Content Set: A collection of content. This has sometimes been referred to as *content scope* and was historically limited to Marketing, Sales, and Success.

Shared Content: *Content items* that are used across one or more *templates* and *portals*, as opposed to *content items* that belong to and are only available within a specific *portal*.

Organization: A logical *tenant* on the platform, used to separate different customer instances.

Portal: A microsite, often private and authenticated, that members visit and browse and consume content within. Portals are based on a template which helps to govern what content is available in its instance. However, a portal can further be configured to hide or highlight particular content originating from its template. A portal can also be configured to include other shared, ad-hoc content that is not in the template. What's more, portals can also have their own content, only available within the portal.

Template: Portal's are based on a template. Among other things, the template defines what content may be available and how a portal can further amend or override the content a template provides.

Member: A known (or sometimes anonymous) individual who accesses a *portal* and its content.

Tenant User An application user who based on permissions can curates *shared content*, *content sets*, and *templates*, and/or create and manage *portals* and their *members*.

Level(s): A way of describing upstream and downstream hierarchies of data, entities, or other concepts that overlap in a way similar to object inheritance. For example, a configuration option, like a color's hexcode, may be initially set at the *level* of the *organization* -> then at a *template* -> then for an individual *portal*. At any level, the setting or construct (if allowed) may override the setting from the level upstream of it. And if not set, this downstream level will inherit from the upstream level.

The Problem

How do you determine what *content* should (or should not) be available and visible according to its settings, along with applying additional optional filter criteria?

Solution Approach

1. Build a logic component that calculates what content is available based on a union of an input set of multiple levels of content items and their state, with certain filtering rules and criteria applied along the way.

> **HINT**: Do not overthink. Do not attempt to build a REST/GraphQL or otherwise full fledge operational microservice. For the purpose of this assessment we are *only* looking at the types, functions, and/or classes that comprise a self-contained logic component that supports the relative inputs and which returns a result set as output.

Inputs

1. A data structure comprised of an ordered collection of [1..n] sub-data structures, i.e. "levels", each of which contains:
    - A *level type*: `contentset` -> `template` -> `portal`
    - An unordered collection of [0..n] content items wherein each record consists of identifier and attributes comprised of metadata and configuration settings:
        - A unique identifier for the content item
        - A unique identifier for the content set the item belongs to
        - A *content type*: `digitalasset`, `question`
        - A *format type*: `video`, `document`
        - An *availability* setting: `never` > `hide` > - < `show` < `always`
        - A *locked* flag to prevent/allow overrides by downstream level
        - A *publish start* timestamp with an optional value
        - A *publish end* timestamp with an optional value
2. An optional, options data structure, which specifies additional content and format *filters* to apply.
    - For example, to only return video digital assets, as opposed to the default of returning all digital asset and questions of all formats: `{ contentType: 'digitalasset', formatType: 'video' }`

Outputs

1. Upon evaluation of the inputs and rules, return a data structure with a collection of the union of all the content item's unique identifiers, and along with each identifier, include a flag indicating item whether or not the content item *is available*.

Rules

1. If a content item's *availability* is `always` or `show` it is available, otherwise it is not available.
2. When building up the "union" of content, a downstream level's settings for a given content item overrides the upstream level -- except as stipulated by other rules
3. If a content item's *availability* is `never`, it is never available, regardless of any downstream level's setting
4. If a content item's *availability* is `always`, it is always available, regardless of any downstream level's setting
5. If a content item is *locked*, it cannot be overridden by any downstream *locked*, *availability*, *publish start*, or *publish end* setting
6. If a content item's *publish start* is specified and it is less than "now" it is not available
7. If a content item's *publish end* is specified and it is greater than "now" it is not available

Futures

As in real-life, things are never static, and systems evolve. The people in product management also indicate a need to add one or more of these enhancements in the future:

- Add a time machine filter by changing when "now" is
- Add additional content types, e.g. `correspondence`
- Change the content item `format type` to use IANA MIME types, i.e. `application/pdf`, `video/mp4`, `text/markdown`
- Support additional filtering using a tags attribute
- Sort results based on a new attribute, like sequence
- Introduce an additional level type, `member` to be able to further personalize content available to an individual portal member

> **HINT**: These are not in scope, but knowing where you're headed, maybe there is at least one thing you should *definitely* be mindful of as you design and build your solution. It might even help you test...

Technical Guidelines & Concerns

Your solution should address these guidelines and ensure concerns are accounted for:

Language & Frameworks

1. TypeScript >= 4.0 (ECMAScript >= 2020)
2. NodeJs >= 14.0

Code

3. Readable, maintainable code wins the day -- practice SOLID/DRY/SRP
4. Favor immutable, pure functions where possible
5. Strong typing is always your friend
6. Comment the why; when complex overview the how

Data Structures

7. The platform is global and time zone sensitive -- You must not lose time zone information when processing date time data structures

Security

8. The platform is multi-tenant -- You must always guard against and ensure data for one tenant is never seen by another

Other Resources

These resources and libraries may be beneficial:

1. Jest: https://jestjs.io/
2. Luxon: https://moment.github.io/luxon/index.html