Technical Issues and Debt

Technical Challenges

- Saving of stale records and optimistic concurrency (https://issues.folio.org/browse/MODINV-109 as an example)
  - When a record is saved in the UI, it is based upon the representation when it was fetched, with changes reflecting the edits. This means that if an edit page is open for an extended period of time, changes by other clients might be lost
- Scheduling
  - How to do in non-storage modules?
  - How to grant permissions to other endpoints?
  - How to segregate between tenants?
  - How to avoid collisions between scheduled tasks when running multiple instances of a module?
- Publish / Subscribe
  - Is there work planned that is intended to use this?
  - There are existing features that have significant compromises (e.g. patron notice sending)
- Record migration during module upgrades
  - When to perform this e.g. during the invocation to the Tenant API?
  - How should the system behave whilst it is happening (if not done during activation)?
- Upgrading modules and schemas (json or database)
  - Currently there is no regard to existing records' forward compatibility w.r.t. the schema. It would be great if a script is provided at the time the schema is updated to transform the current data in the database. When a system is running in production, it will be extremely difficult to blow away all the data and reconstruct them to be compatible with the new schema without significant down time. This would have to be done via scripting while the system is running.
- Standardise API behaviour, for example:
  - should PUT allow for creation of records
  - what status code should PUT return when record replaced vs. created
  - what status code should delete return record does not exist
- Lack of Unit Testability in back-end modules
  - mod-circulation-storage tests cannot be tested for error conditions because of inability to inject dependencies into storage module classes without lots of effort in abstraction and vertx.
  - Other code are not unit testable due to highly coupled classes, impossible to inject a mock object for testing.
    - In general it would be better if the code that handles vertx requests are separated from the code that implements the logic to make the implementation unit testable.
    - In general it would be better if objects are injected in instead of inline instantiations for testability.
- Currently error messages (or any messages) from the back end are consumed by the front end verbatim. This makes it brittle because a) messages on the back end could change at any time due to refactoring - no one would be aware of how or if the messages are being used. b) difficult for translations, and could break translations as well if the wordings get changed.
  - It would be better to supply enough information about a condition for the front end to construct a message to display on the UI.
- Some requests to mod-circulation fans out to a bunch of calls (performing a checkout, for example). Could the fan-out calls be done concurrently? These calls typically wait for the result of the previous call to proceed, but some calls are not related at all until at some point, perhaps series of concurrent streams of calls can be made to improve performance.
- Performance monitoring: no custom metrics available (or the ability to create custom metrics) to expose performance at module level.

BE:

1. In order for clients to make decisions (e.g. localisation) based upon validation errors provided by the API it is necessary to uniquely identify them in some way

   ✓ **FOLIO-1716** - Uniquely identify backend API validation errors DRAFT

FE:

1.