MTD API Architecture to Accommodate New Entity Types

[TOC]

17 September 2020

Contributors

- Toby Porter, MTD Front Facing API Architect
- Dave Beeston, CDIO Architect

1. Problem Statement

The current ITSA individual APIs are not suitable to reuse for other entity types because they are tied to use of a nino. This makes delivery of additional return types expensive and time consuming. We want to investigate the possibility of making a breaking change to the individual APIs to enable them to be reused for Trusts/Partnerships etc.

1.1 High Level Assumptions

1. There are 3 additional entities that need to be supported: Partnerships, Trusts & Pension Schemes
2. Return requirements are broadly the same as Individuals but may differ by 10%, 30% & 50% respectively
3. More entities may need to be supported in future
4. Calculations API would need minimal modifications
5. For estimating purposes it is assumed that each microservice development requires approximately one (2 week) sprint and the same amount of support activities i.e. business analysis, content creation, publishing, E2E testing, transaction monitoring etc.

2. As Is/Planned MTD APIs

The Majority of current/planned APIs are tied to an individual and published under an "*individuals*" context.

3. Tactical Option

Copy existing individuals APIs and modify for each new entity type.

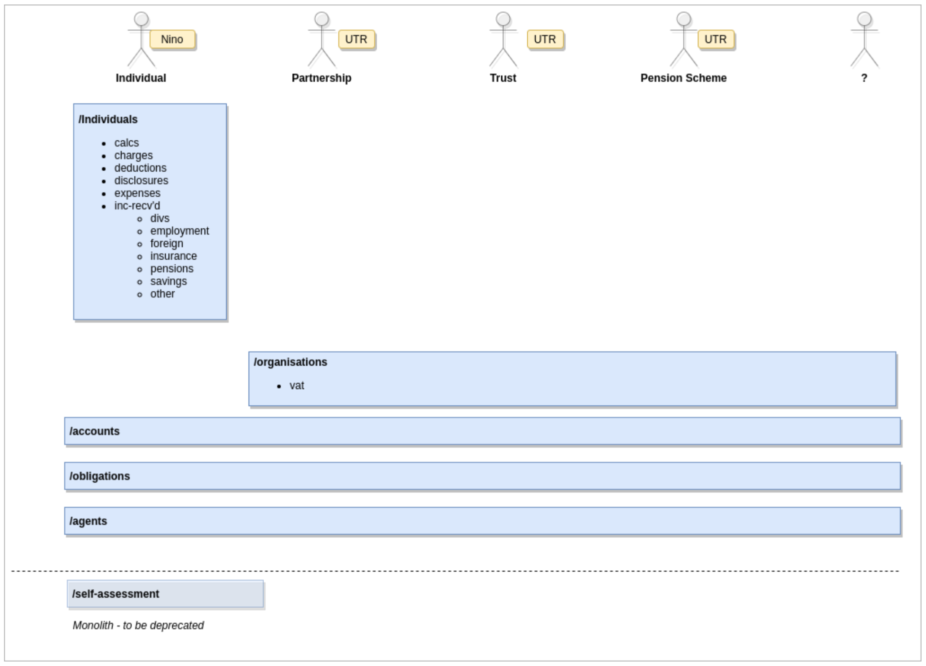

Figure 1: MTD-Logical-Data-Model-Add'l API Actors - As Is

Pros

- No breaking changes to existing APIs
- Code re-use for faster delivery - low upfront effort
- Future breaking change scope limited by entity

Cons

- Exponential increase in endpoints
- High maintenance overhead
- need a new API per entity
- cross cutting changes need to be made in multiple APIs

Estimate

"Large" to "XX Large"

- Partnerships APIs: ~20 sprints
- Trusts APIs; ~16 sprints

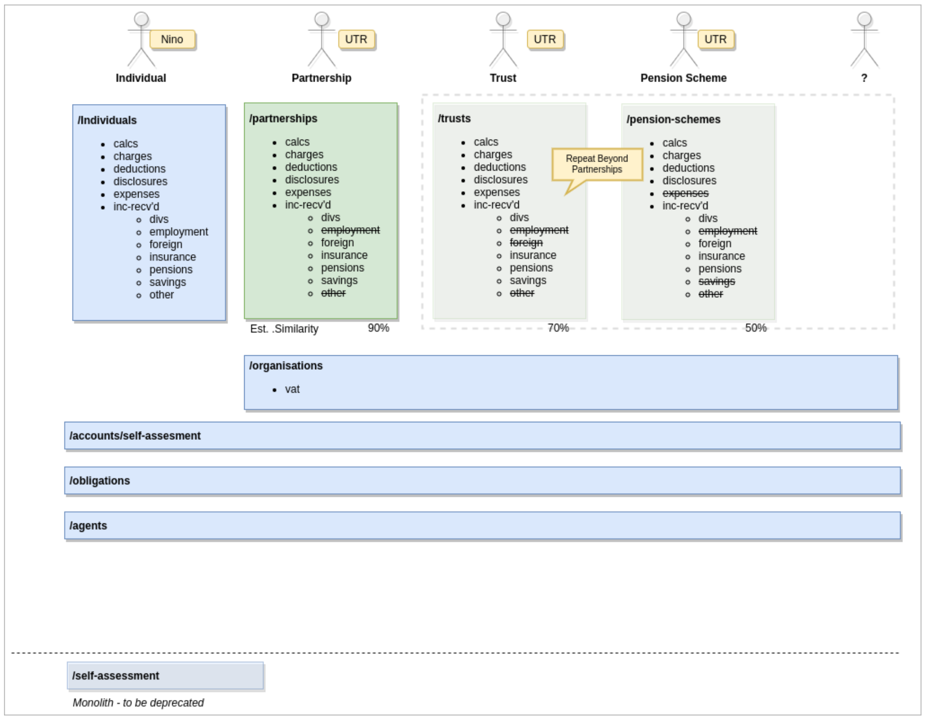

Figure 2: MTD-Logical-Data-Model-Add'l API ActorsTactical

- Pension Schemes: ~11 sprints
- Plus x sprints for every new entity

4. Strategic Option

Republish *individuals* APIs under entity agnostic contexts and add polymorphic behaviour to handle Partnerships etc.

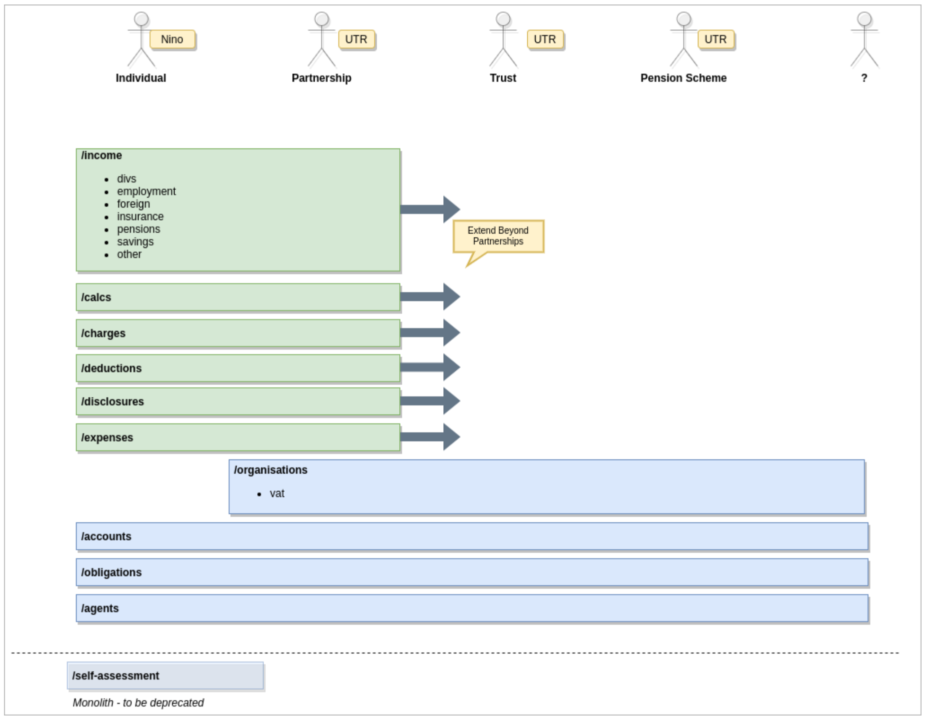

Figure 3: MTD-Logical-Data-Model-Add'l API ActorsStrategic

Pros

- Minimises endpoint proliferation
- Entity agnostic
- respects domain driven design
- reusable for future entities
- Lower maintenance overhead

Cons

- Breaks all current *individuals* endpoints
- Increased complexity
- Risk of creating new “monoliths”
- Breaking changes potentially affect all entities
- Significant refactoring - higher upfront effort

Estimate

“Large”

- ~35 Sprints

5. Integration Layer Impact

{Dave Beeston to comment}

6. ITSD Impact

{Dave Beeston to comment}

7. Recommendations

{Dave Beeston to comment}