**Main questions/concerns:**

Need to decide if we are building in house or using a vendor product, do we have a list of mandatory requirements and nice to haves?. With the former we should focus on defining our vision, features relative priority while with the later we need to spend time playing with the vendor tools and be at peace with limiting our feature set to what the vendor provides.

* Is this a table stake product or a differentiator that could be leveraged to build other products and/or gain new clients (or deepen the relationship with existing ones)
* What is the business appetite to deal with inherent limitations with any of the shelf products, would they be comfortable with a situation were the product would be close but not quite there?
* Are categories strong/stable groups (they would be our main models and each field belongs to its parent category) or just labels to organize the UX (fields can be moved from one category to another)
* How flexible we want to be for inter-fields validation

**Business scope discussions (B):**

* Is this an Omni channel product? I.e. we need to support phones, pads, desktops? Or can we limit to desktops? Can we limit which browsers are officially supported?
* Who is the consumer of this tool? Would they be using the site multiple times a day (so we give higher priority to productivity) or most would be occasional users (so we focus on discoverability)?
* Who is the consumer of the admin tool? Technical staff at the banks? Ex. can we assume they can provide regular expressions for custom validation? How much hand holding they would need to indicate relative order?
* Do we really need to provide single page and wizard? Why would clients want to have hundreds of fields in a single page? Could we leave single page to after v1? I suspect once clients experience a well-built wizard they would forget about single pages
* Is the validation per field, per category or can we have a Product field constrained by the value in another field in the same category? Another field in another category?
* Can you provide some examples of how a field's value could affect other fields? Can they all be grouped by these rules
  + A field value changes the visibility of other fields
  + A field value changes if other fields are required or optional
  + A field value changes the definition of accepted values in other fields (its validation reg. expression)
* Does a field always belong to the same category, just the order in which it is displayed changes or can the banks move fields from one category to another. Can the same field be seen through multiple categories (i.e. duplicated fields, I assume the value would be the same, otherwise they are two different fields that happen to have the same label)

**Architectural discussions (A):**

* Is the platform that would contain this module a SPA? (since this might impose some technical restrictions that we would need to consider)
* Is building the backend for this solution in scope or are we relying on existing services for persisting this data?
* Are we hosting this service (multitenant platform) or it would be hosted by clients in their internal networks?
* Any known requirements regarding authentication/authorization? Should we integrate with Single Sign On providers or can we rely on windows authentication (i.e. active directory creds)?
* Is FIS a relational DB shop or do we have infrastructure in place with NoSQL databases (i.e. MongoDB, Redis, CouchDB). Can we go schema less during the first rounds until we have a clearer picture of the model?
* Can we leverage existing in house components to capture certain categories or fields?
* Do we need to track each interaction with the form; is Event Sourcing pattern applicable here?
* What is the standard pattern to implement the service layer in FIS: WebServices, messaging middleware, REST services, Microservices?
* Microservices that we might need:
  + SyntaxAndGrammarCheck (given a text and a language provides a response with misspelled words)
  + CategoryValidation (category id and set of values returns if there are validation errors)
  + ModelPersistency (given a model updates the DB, given a session retrieves a list of categories and its fields)
* Are we allowed to go PROD with Docker containers, does FIS have prod applications using that technology or would we be pioneers? Have friends trying to push a containerized solution to PROD at RBC and they have run into quite a few unexpected surprises with their infrastructure folks.

**UX considerations (U):**

* First level of defense is in place per field validation
* In real time indicate if a category is in a valid state (subtle indicator to avoid annoying users). Do not trigger validation until a user touches a field
* Are we using popups for error message or show them inline at the bottom of each category?
* One field per row or for wide resolutions put multiple fields for a more compact view with less vertical scrolling?
* Do we have designers in house for this project or should we rely 100% on Bootstrap/Materialize?
* Are the user keyboard/accelerators centric or accustomed to use the mouse as well?
* If we need to implement a single page view we could provide a search/filter component (similar to Chrome's settings dialog) to help locate individual fields
* We could follow 3 different approach to organize the UI, each one has merit depending on the use cases we would like to emphasise:
  + Single Page:
    - Pros: Easier to navigate for not technically sophisticated users, better suited if users need to quickly consult different categories
    - Cons: Easier to lose changes, for hundreds of fields users would spend a lot of time scrolling to locate certain fields
  + Wizard:
    - Pros: Well suited for linear workflows, can adjust the content of future sections based on the values in the previous one
    - Cons: Harder to skip sections and navigate across different categories, could get confusing if values in one step invalidate values entered in a previous one
  + Tab layout (my preference):
    - Pros: Maximum flexibility, users fill it in any order they choose, easier to move across sections, could still be used as a wizard, could have lightweight error indicators so user can act on them right away or leave it for later, since users would get accustomed to follow a non-linear flow it would be easier for clients to introduce new categories or shuffle around sections,
    - Cons: more effort to implement on the server side since each save operation stands by itself, changes in one category might affect another and users might not notice it if there are no validation errors (can be mitigated with a report at the end were users get to see the filled form in one page)

**Recommendations (R):**

* Based on my market research and your requirements, this doesn't have to be a "basic form entry app", there is nothing in the market to cover your needs and if architected properly we could use this opportunity to build an extensible and generic enough product to cover many other use cases that could be integrated in other FIS products down the pipe
* B-Let's avoid the single page approach, having hundreds of fields in a single page is asking for trouble at the UX level, potential problems: user goes for a coffee and the session expires, browser crashes, multiple validation errors in different categories (no clean way to indicate which categories have errors)
* B-Let's start prototyping with a vendor product (forms.io seems to be the best candidate) while we build our product to support our scenarios
* B-Limit fields to its category, provide a default order for the categories but let the users override
* A-Each category should be self-contained (i.e. a component) so we can arrange them in a wizard or as a single page if we end providing support for this
* A-For each field have metadata to indicate: visibility, relative position, dependant fields (if validation fails which other fields should also fail validation), regular expression for validation
* A-Restrict fields to categories at the UI level but make the model generic to future proof the persistency layer
* A-Save each category using asynchronous micro services calls as soon as they go out of view (vertical scroll, changing tab, next step in wizard mode)
* *Value of going agile: iterative approach well suited if requirements are not set in stone and even more for complex problems, something to show after the first couple of sprints, close alignment with PO to limit the chance of broken telephones, define the scope of the: prototype, MVP, groom the backlog*

**Market research:**

1- form.io

* Features
  + B-form builder can be embedded in our apps
  + b-drag and drop of form components
  + b-simple fields data masks (phone numbers)
  + b-simple conditional rules: when field A contains X field B should hide/be visible
  + b-support single page and wizard forms
  + b-support multiple columns layouts
  + b-Seem to provide Multilanguage support
  + b-Free tier to play with
  + b-Sample apps available at https://github.com/formio(formio-apps), walkthroughs https://help.form.io/tutorials/videos/walkthrough/ and their form at https://codepen.io/travist/full/xVyMjo
  + a-their form builder provides json schema (instead of an HTML form), this json is consumed by their visualizer (flexible and powerful architecture, even if built in house this approach is a must)
  + a-available on premise, private cloud and hosted at forms.io
  + a-provide a js api so we can build our custom UI and use them as form builder and service layer
  + a-modern and best of breed tooling (node, apis for Ag and React, Docker for deployment, bower for building/packaging)
  + a-seem to allow custom JS code to do client side validation
  + a-certain components are open source
  + a-provide js hooks so we can inject our own services
  + a-UI for Ag2 and React
  + a-Automatic API generation, they host or self-hosted using Docker
  + a-not clear from their documentation how well structured and flexible is their data store, how easy would be to integrate and correlate with other data stores (seem to be using MongoDB). We could store the forms in our own DBs by using the their SQL Action to store in our own DB or their Webhooks action to call our own service
    - a-support JWT SSO for single sign on
* Limitations
  + -To design the forms users still have to go to Forms.IO as disclosed in the fine print of their pricing page , when self-hosting only the API is on our side (they open source the client side and let self-hosting of the API but the designer is closed and only available through them)
  + -Seems to have basic validation: required, unique (primary key), max length
  + -No indication of their data storage policies (ex. at rest encryption), this could be a concern with international privacy laws
  + -missing captcha support
  + -couldn't find in their doc reference to how to inject our own controls to their forms designer or add our own actions
  + -feature set seems rich but team size looks to be light

2- activeforms.com

* Features
  + -Going after the enterprise market
  + -Richer validation
  + -Server side validation
  + -Custom fields
* Limitations
  + -Don't share too many details on their site, a bit cryptic
  + -Seems like they host the designer and the forms
  + -No server side hooks and integrations

3-typeform.io

* -Features
  + -modern UI
  + -might be a better fit for retail uses cases(surveys, basic questionnaires)
* -Limitations
  + -gimmicky trying to much to be cute/friendly, don't think it is a good match for enterprise customers
  + -VERY limited editor

Other alternatives quickly discarded as bad fits: formstack.com (second best after forms.io but not extensible enough, designed to be used by non-developers, lots of integrations), connexica.com (going after the enterprise market but they are limited in capabilities and their site is just an online brochure), google.ca/forms (simplistic, survey oriented), wufoo.com (very limited),

NinjaForms.com (WordPress oriented, retail), formbakery.com (simplistic, php code output), planso.net (simplistic, old), jotform.com (simplistic, retail)