# Bluesquare Analysis

# Introduction

The purpose of this document is to feed the discussion as teams prepare to undertake their responses to the Request for Application #2019-016 openIMIS Modularization, Nepali Functionality, and Community Development concept note phase. It should be considered a snapshot of our current understanding and vision to guide the efficient allocation of the RFA funding for the community.

> We acknowledge that, thanks to the additional visibility provided by this RFA, new opportunities and/or actors may be revealed that could change the priorities. We remain open and in fact welcome these insights and feedback. There may be areas where we have misperceived the scope and/or the importance of some points.
>
> In other words, all statements made in this document should be challenged, tempered or even contradicted: the most important thing for us is to bring these topics forward to support the openIMIS community and help move the process forward.

As the current lead on the openIMIS software architecture work, in this document Bluesquare would like to provide:

- A summary of the current situation to clearly depict the starting point of this work assignment.
- A high-level decomposition of the work to reach the objectives setup in the RFA and, for each block, a rough estimate of the complexity along two dimensions: business complexity and technical complexity.
- A first look at the dependencies between the identified topics and propose a possible roadmap to realize them, that is aligned to the current product roadmap.

The RFA is split into three work streams A, B and C. The work streams A & B are related to openIMIS as a software (platform), while workstream C concerns the user community, its governance and the initiatives required to improve/extend it.

Work items in Streams A and B are rather interrelated. So this document will discuss these points in one section without distinguishing between their associated stream. Choices made for Streams A and B will highly impact the work to be done in Stream C, however some generic aspects related to Stream C are highlighted in a dedicated section.

Thank you in advance to the community for your thoughtful input on the proposed elements herein.

# Current Situation

## IT Point of View

As described in the current product roadmap, openIMIS is being re-written to reach a modular architecture. The new modular architecture means that the openIMIS platform is in fact pursuing two antagonistic objectives:

- openIMIS is implemented in Nepal and Tanzania, as a standalone (or close to it) “all in” platform, and the re-written openIMIS will at least offer the same capabilities.
  In other words, while the policies and claim processing are clearly identified as the core feature of openIMIS, the software also offers, in a minimalist approach, the necessary features to manage locations and health facilities, insurees, insurance products, medical catalogs,... (cfr. https://openimis.readthedocs.io/en/latest/)
  Having a standalone ready-to-deploy platform is recognized as very important to ensure openIMIS can be installed in any context.
- openIMIS added value primarily lies in policies and claims management. For all other areas, relying on existing solutions is seen as a more comprehensive approach.
  Integrating openIMIS into the OpenHIE ecosystem and delegating features (instead of rewriting them into openIMIS) is thus identified as a major acceleration factor to reach a fully functional management of UHC in various locations.

Until now, the priority has been given to the first objective (openIMIS “as is”, standalone). The main concern has indeed been to minimise risk on current openIMIS implementations (Nepali and Tanzania) and, therefore, put the priority on having the current set of features “as is” in the new platform. The first allocated budget for the re-writing (Notice C.) is dedicated to the setup the new platform and migrating two plugins (or modules). There was also a firm will to engage in an iterative and incremental approach: at the cost of a lengthier and more complex IT project, the rewriting had to be fully functional after each step. In order to prevent any side effect on the legacy application, one major acknowledged constraint is to avoid any modification to the current database structure. Until the .NET code is fully decommissioned, it will indeed be very hard to guarantee (maintain) existing application stability while changing the database structure. Following this, three sequential phases have been undertaken:

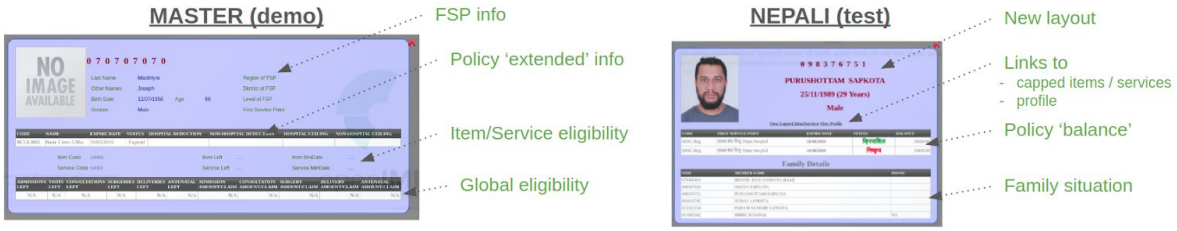
1. Software rewriting, with database ‘as is’ and legacy application proxied until completion.
2. Database switch: once all application code ported to the new platform (and notably all stored procedures rewritten in python services). The roadmap was to move from Microsoft MS-SQL Server to Postgres, and this for 2 reasons:
   - better embrace the open source community
   - enable JSONB technology for future developments and refactorings
3. Once on the new database, review priorities and make modules more flexible, implementing new features, integrate with other solutions…

Note: one possible refactoring considered was to adopt the FHIR model as core (internal) model in openIMIS (using fhirbase/Postgres)

We are currently in phase 1 (code rewrite), 2 of the 9 identified plugins/modules will be delivered by the end of ongoing work (before the start of this RFA work assignment). One of the complexities we are facing is the difference between the “Master” (normally aligned with Tanzanian installation) and Nepali. A clear example of this complexity is the ‘enquiry’ feature.

Aim of the screen:

- The enquiry screen is dedicated to **give a quick overview** an insuree, knowing his/her CHFID (identifier in openIMIS)
- **TODAY**, openIMIS master and nepali versions don’t display the same information



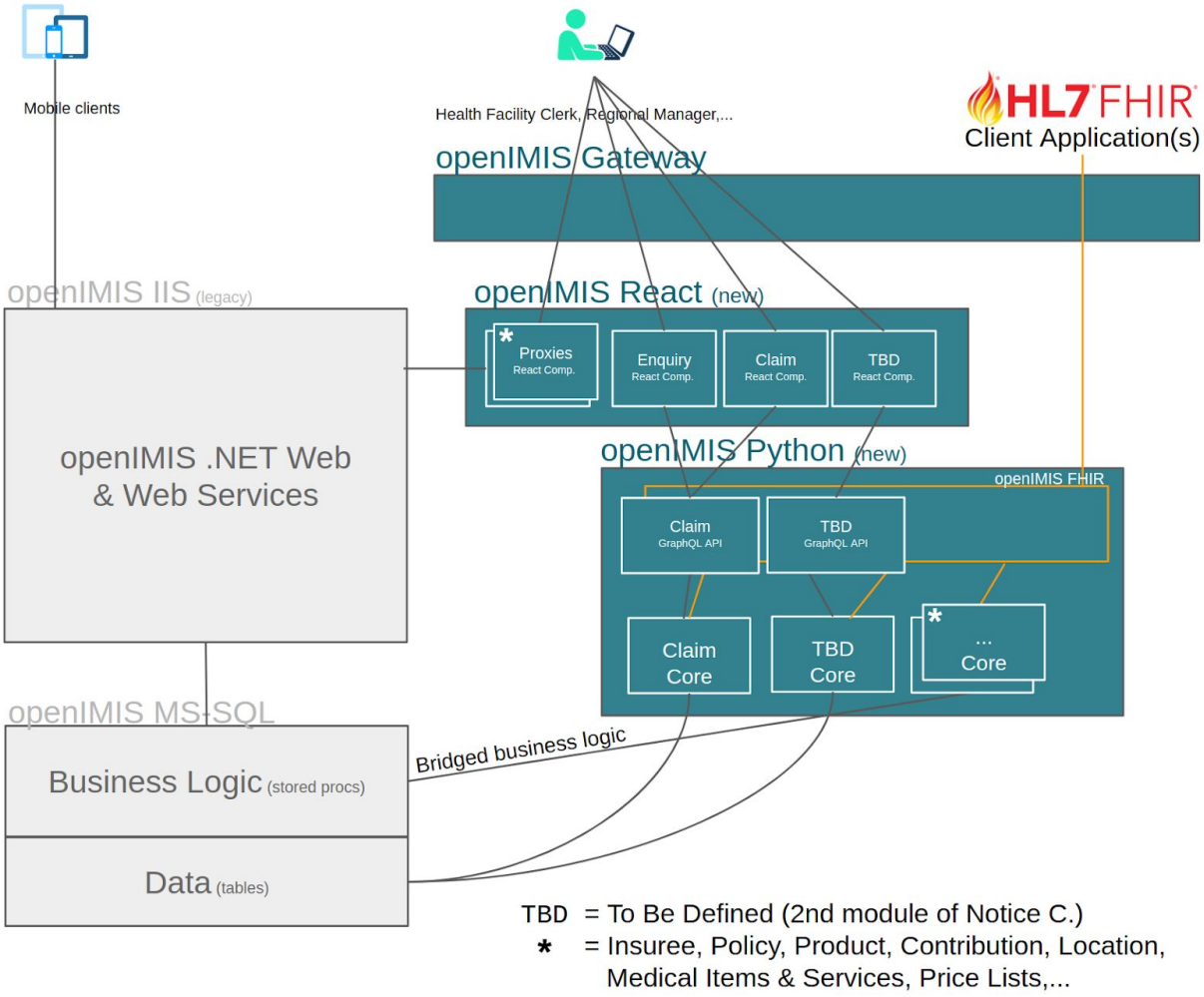

- **Questions** (not to be solved today):
  - what about Tanzania and Cameroon?
  - what would/could it look like for other countries ? (is the foreseen flexibility already outdated)

The solution to address this in the architecture has been presented during the 05/08 Gumzo ya Mwezi and clearly highlights the added value of the contributions mechanism put in place. Yet we expect to have more of these discrepancies and as a community we need to insist on using the plugin/module composition approach instead of “branching” the code bases.

The Notice C. had also expressed the need for an HL7 FHIR API (initially only the claims) and, during the kickoff workshop for that work assignment, we assessed the possibility to implement it directly on the new platform.
This was made possible (at a reasonable cost) for several reasons:

- We were able to rapidly expose the business logic contained in the database stored procedures for all necessary entities (claims, patients,...).
- The FHIR API is a backend-only component. FHIR API didn’t require the complete platform setup before being developed: the sole online backend component was needed.
- No database modifications were necessary.

To summarize, once the work assignment of Notice C. will be accomplished, openIMIS will be:




**In other words:**

- most of the features are still in legacy application (.NET or Stored Proc)
- it will still **not be safe** to change the database scheme (adding tables should not be a problem)

**About modularity:**

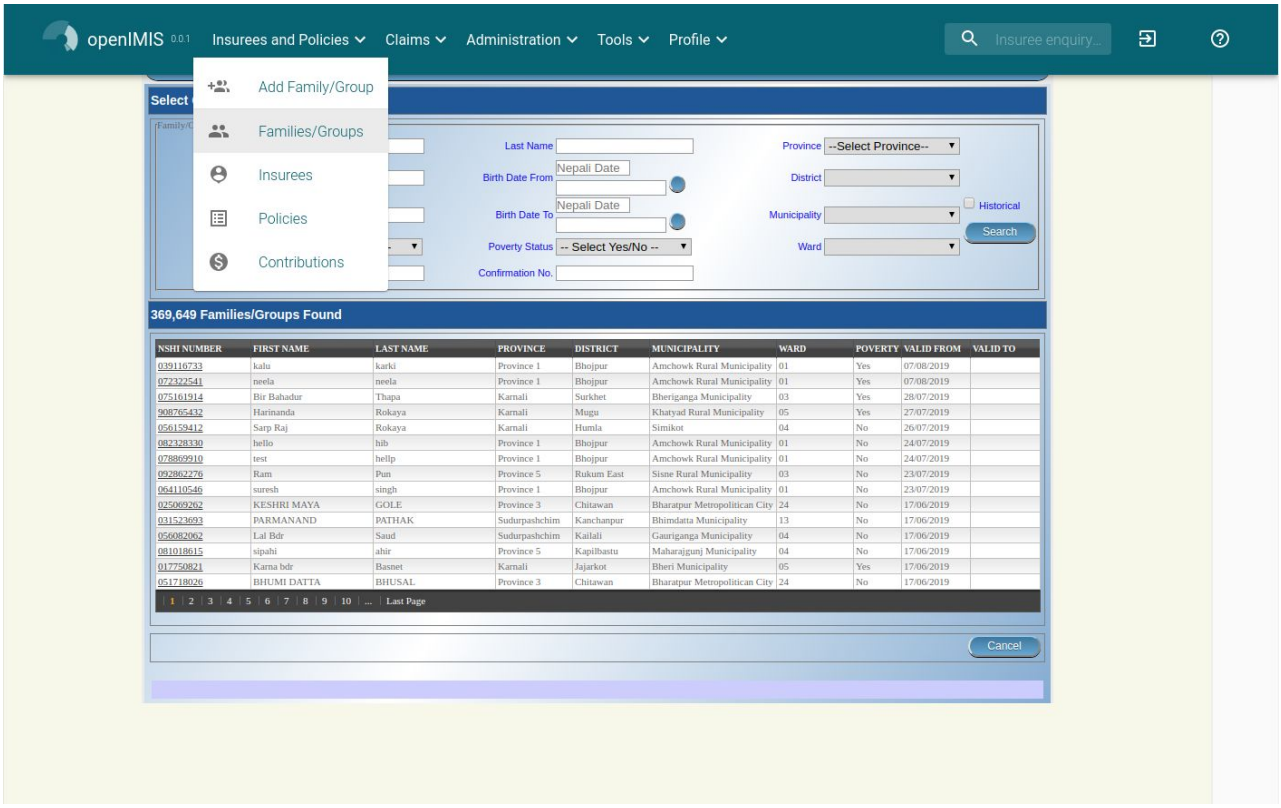
In the current situation, the FHIR API is a monolith on top of modularized core modules.
This design is the result of a **community fully aware decision (05/2019)** where time to availability and necessary workload have been favored over clean modularized implementation.
Within that agreed design, **the quality of the work done for the FHIR API is not questioned at all**… but it also highlights how easy and tempting it is to 'forget' about modularity: going modular is not natural, it is even counter-intuitive and requires a strong commitment from all community members.
Since FHIR API is not a "mandatory" module for openIMIS to operate, the consequence is "just" that the FHIR API (as a whole) may not be available for some openIMIS implementations: the

core modules assembly may not be compatible with that (monolithic) reference FHIR API module.
However, this kind of decision should **never** be allowed for core modules (insuree, policy,...) as it would jeopardize the benefits of the architecture in place.

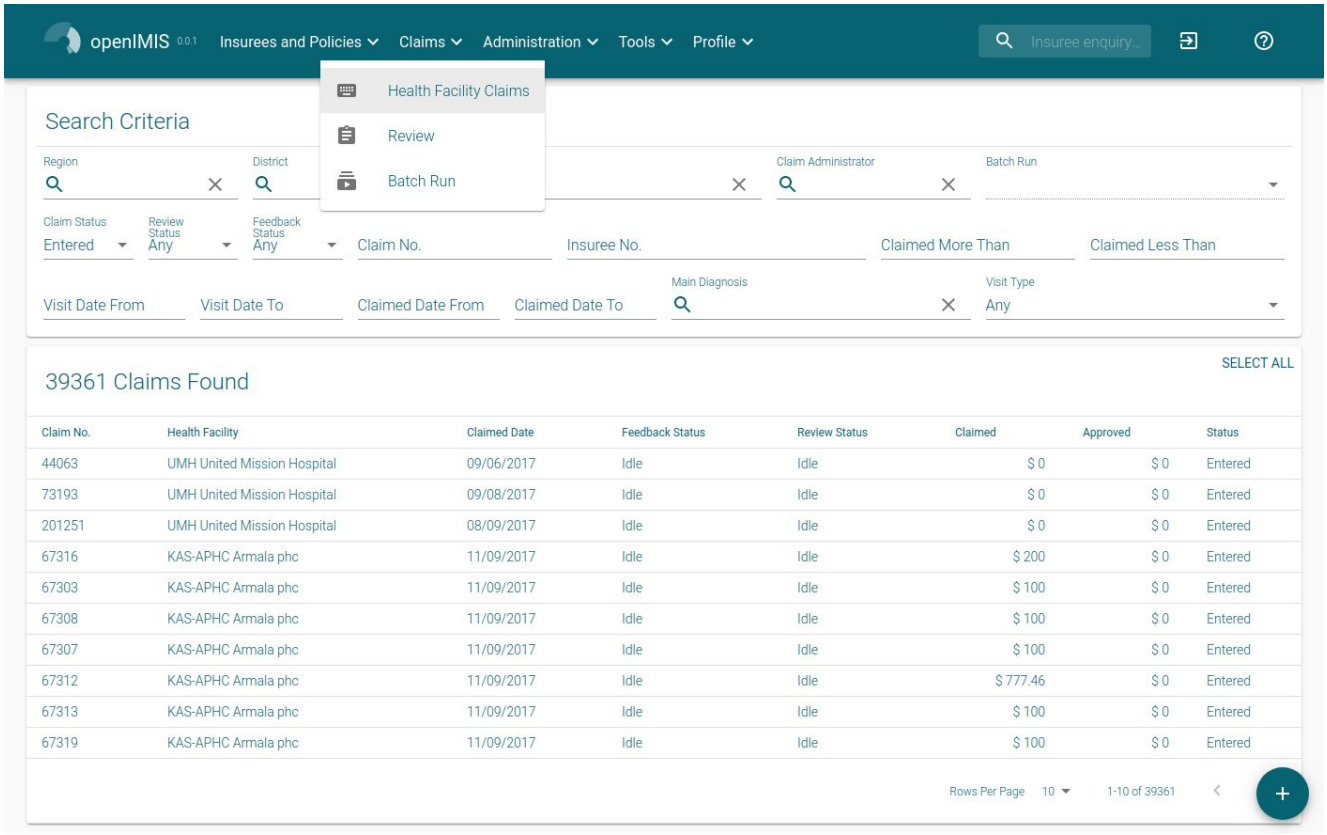
## User Point of View

The new platform is dedicated to progressively replace the legacy one. In order to do so, the new platform is providing the main application menu and either displays the new module pages when available… or the legacy one when not available (we call them ‘proxied’ page).

For example, the Person & Families module is not yet migrated, so the Families/Groups menu entry displays the legacy page:



The Claim module on the other side is being migrated. Thus the Health Facility Claims menu entry displays the new UI:




## Conclusion

The current setup has tremendous advantages for existing openIMIS installations… as a transition phase:

- risk is minimized
- the fallback plan is “by design”
- ... and new features via the new platform (like the FHIR API) are immediately available.

However there are also drawbacks:

- hybrid old/new UX (User Experience)
- hybrid (.NET | React/Python on Docker) servers

We believe that the benefits largely outweigh the drawbacks for the existing implementation. The Nepali team has given clear signs that new platform is the future: the FHIR API will be in production very soon… and their high level of involvement as demonstrated by workstream B scope in this current RFA is extremely positive.

**Provided that we setup a carefully and jointly planned migration with Tanzania**, we are very certain that they will also see the new platform as a real opportunity... and embrace it eagerly.

Furthermore, as long as two technologies (.NET and React/Python) are used, **it will continue to split the developers community in two very distinct teams, with the risk that communication between the two remains too low**. This lack of communication is **not** the result of a lack of willingness : as things are set up today each development team has its own “reality” (objectives, deadlines…) and can’t do much for the other. If we were using a single development platform, the situation would be very different: thanks to the new modular architecture, we could directly profit from each other’s progress and thus be more involved/empowered/able to help each other.

**For all these reasons, we believe a high priority should be given to work items that include the migration of the legacy modules to the new platform in their scope.**

# Streams A. and B.

The RFA states that openIMIS core modules should be migrated to the new platform and also identifies several extensions as high priority. Migrating a module from .NET to the new platform isn't something new and we believe that the approach undertaken until now is effective and can be continued:

- Ensure the scope of the module is identified and clear to all;
- Design screens, taking the country specificities into account via community contributions;
- Develop/deploy in an iterative way, allowing the community to see the progress and react when necessary.

In this section, we will assess the 'new' items from the RFA and propose either one or several approaches to manage them. The assessment will be completed on a scale from 1 (small workload/budget) to 7 (high workload/budget). It will also be performed along two axes: technology (pure developer work) and business (complexity to set-up/operate in a specific openIMIS implementation which links then directly to the workstream C).

In addition, before addressing the requested points of the RFA, we would like to formulate some general remarks about assessment confidence, custom vs. delegated development and architecture guidelines.

## General Remarks

### Assessment confidence

As the ToR notes, the different topics *" are at varying levels of finalization or point to current discussions and it is envisaged that the successful applicant will actively engage in the refinement of the functionality"*.

Furthermore, while the master user guide is a good source of information on how things are currently working, experience has shown that there are country variations and so all module development has to integrate the necessary flexibility to cope with the existing implementations from the very start.

Finally our own diverse backgrounds and experience may widely impact how the complexity of the topic is evaluated by any community actor: the complexity of business workflows to support can be very different from one country (implementation) to the other. Having a tool that tries to 'solve it all' from the start is probably not the most realistic approach were it even feasible. It requires program team to be able to effectively anticipate the program realities and correctly identify the future needs and thus determine appropriate tool flexibility for each piece of the puzzle.

For these three reasons, while we do provide a general assessment of the complexity and workload, these should not be considered the final or only say. We would recommend that each work item should start with an analysis of the situation and the best pragmatic “next step” to be identified as a community to determine the appropriate direction.

## Custom vs. Delegated development

There are several ways to develop a desired module (set of features). These range from “all custom code in the openIMIS platform”, to “fully delegated and interfaced”. Interfacing can be done towards a specific system or following a (standardized) protocol (APIs, data exchange structures,...), in which case the choice of the target system is theoretically irrelevant. However most standardized protocols foresee some space for flexibility (extensions,..) and, while choosing a standard clearly eases the process, it doesn’t solve everything.
Interfacing can also be done at various levels: database (data exchanges), api calls (online or batch) and even frontend (integrated UI - user interfaces).

Generally speaking, custom implementation tends to cost more, provide a smaller set of features/options and be less flexible,... but, on the other hand, custom code tends to be better focussed on the essential (“keeping it simple”), integrates with user interfaces more effectively and keeps the platform free from any additional dependency with other systems/protocols and their own evolution path... which leads to improved and controlled maintenance costs (not ‘forced to upgrade’...).

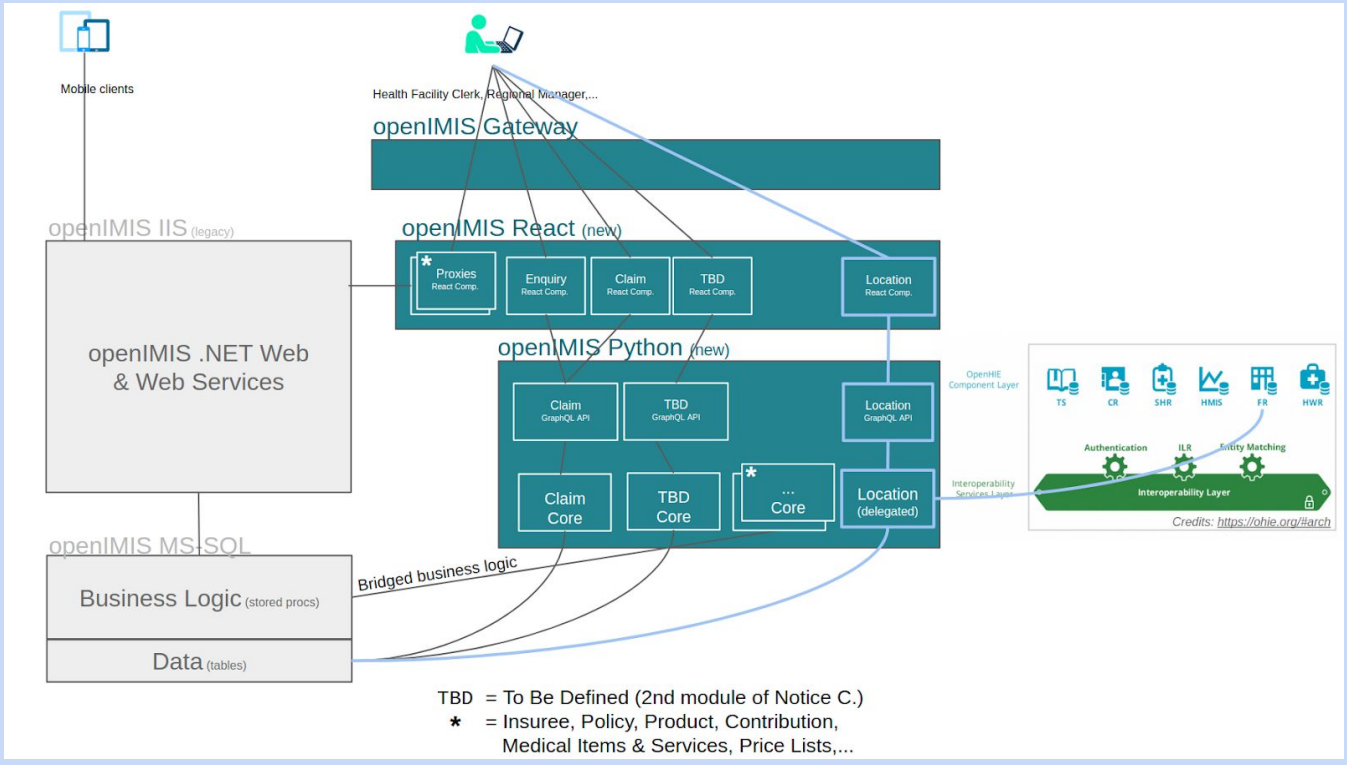
Choosing to delegate a module to an external system (via standard protocol or not) is not an easy process and doing so effectively requires the following:

- choose the target system (or, when possible, standard protocol);
- define the desired level of integration (data exchanges, API calls,...);
- develop the ‘connectors’ (from ETLs for data exchanges... to “connected widgets” in case of UI integration);
- except for pure data integration (where data managed in external system is ‘injected’ in openIMIS), develop the necessary “proxies” towards the delegated system (while, ideally, only the ID to externalized entity should be enough, we often “need more”: capability to ‘show’ the entity on the screen or in reports, cache for batch calculations... ; and
- develop the necessary ‘callback flows’ from the target system (example: impact on policy if a family composition is changed in external system, deletion/de-duplication of entities...).

The drawbacks of delegating to an external system can be relatively well attenuated by passing through a service broker platform. As identified in the RFA, OpenHIE is clearly a good candidate for this. For Notice D: “Patient-Level Indicator Reporting” the proposal we participated in aims to set up a connector between openIMIS FHIR API and the OpenHIE platform. In other words, if

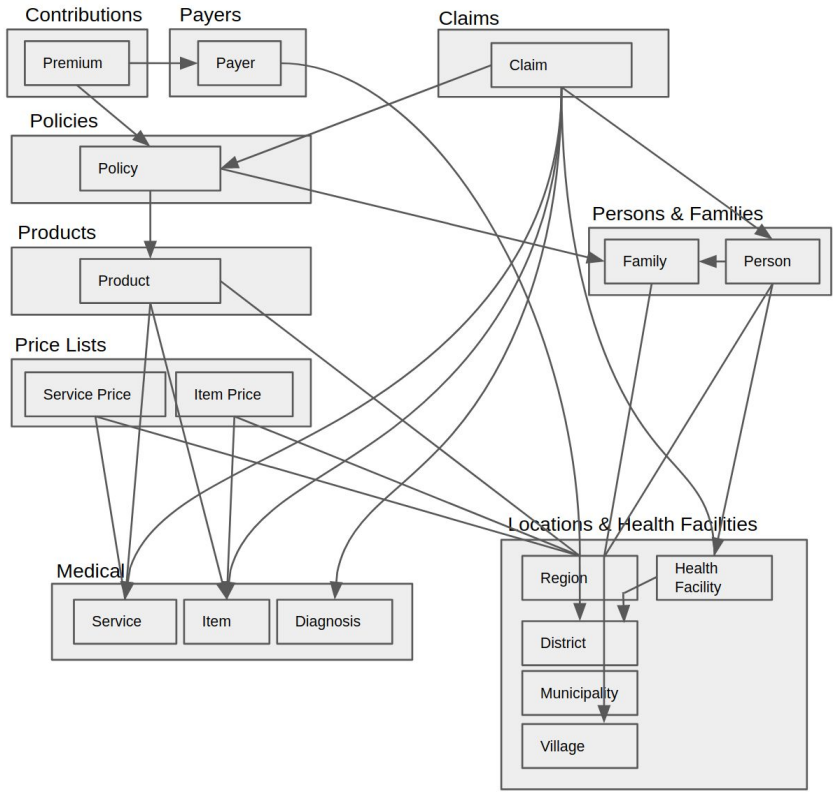
we are awarded that work assignment, we would probably be able to reuse most of the connector work and extend it to several openIMIS modules.

Note: the Notice D proposal places openIMIS as source (only) PoS for patient-level data. In the context of this RFA, we will however need a bi-directional connector (i.e. openIMIS also as client application) to enable module delegation.



## Module Dependencies

The diagram below depicts the static dependencies (as they are written down in a database) of the various modules and their main entities:

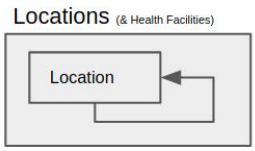



When migrating a module from the legacy platform to the new architecture, **dependencies don't matter**. This is mainly because we keep the data model unchanged. Whatever user interface/application the user uses to write into the database (and, behind it the backend code), as long as the involved component writes the data the same way the legacy code does, there is no impact.

Dependencies start to matter if, during the migration, we start changing the structure and/or the 'business logic'.

Example (not hypothetical at all), if we migrate the Locations and Health Facilities to a system that:

- Manages them without a real distinction between Location and Health Facility (meaning, a Health Facility is just one 'kind of Location');
- Provides an unlimited number of levels; and
- Where Health Facilities can be registered at each "level" ("regional hospital", "district health center", "village facility",... ).




Suddenly, the way the claim (registered for a Health Facility) amount is calculated may need to be adapted: it has to “bubble up” the Location hierarchy to the level where the price is defined. Of course a more generic model will allow the implementation of the current openIMIS: with good governance, we can ensure that the hierarchy has only four levels, that Persons are registered at Village (4th) level and Health Facility at District (2nd)... But the purpose of interfacing with an external system is to embrace more “interconnected” systems. In other words, the Facility Registry system may itself be connected to systems where openIMIS ‘constraints’ are not relevant… if it does not pose a problem.

Thus the further “down” the link is made, the more the “additional flexibility” of the delegated system will have an impact on openIMIS.

Alternatively, the further “up” the link happens, the less interesting it becomes… as it then ‘competes’ with openIMIS itself. Claim processing won’t be externalized (it is the very core added value of openIMIS). Policies? Probably not… it is too intertwined with (dependent/constrained) Claims. Persons & Families? Could be… provided that it provides for the flexibility needed for openIMIS (a Family, with a ‘head’ person)... and not more: in that system, can a Person be a member of several Families (stepfamilies,...)?

In other words, interfacing OpenCRVS (for Persons and Families), or the Facility and Product Registries requires a dedicated analysis, to determine which are the impacts and associated costs… but, at this stage, we have limited insights on the associated complexity.

## Architecture Guidelines

The new openIMIS platform has been designed, from the start, for a modular architecture. In order to prevent jeopardizing the benefits of the modular architecture, component additions (be they custom code or integration strategy) must **follow several guidelines which, while useful to ensure a qualitative and aligned result, make development more time consuming**. Some major highlights of the constraints on how development should be conducted:

- the loose coupling of modules via **event-driven** pattern… and this on backend side (via django signals) and frontend side (via Redux dispatch/reducers);
- the **contributions** pattern to assemble final result in a coherent, yet flexible way… and this on the backend side (cfr. the implemented GraphQL IOC pattern) and frontend side (using React HOC for Contributions and PublishedComponents);
- localization, and not only for the frontend screens (where react-intl is put in place) but also for date management (and very specifically the calendar hotswap mechanism in place to cope with Nepali calendar); and

- the centralized module configuration mechanism (frontend and backend) which allows each implementation to unplug/plug modules, (de-)activated contributions, replace reference (claim filter,...) components by country-specific ones.

# Evaluation of the Requested Features

## Improved Claim Review

The Improved Claim Review requirement describes two distinct mechanisms aiming at Claim processing automation. Each mechanism can be addressed in isolation:

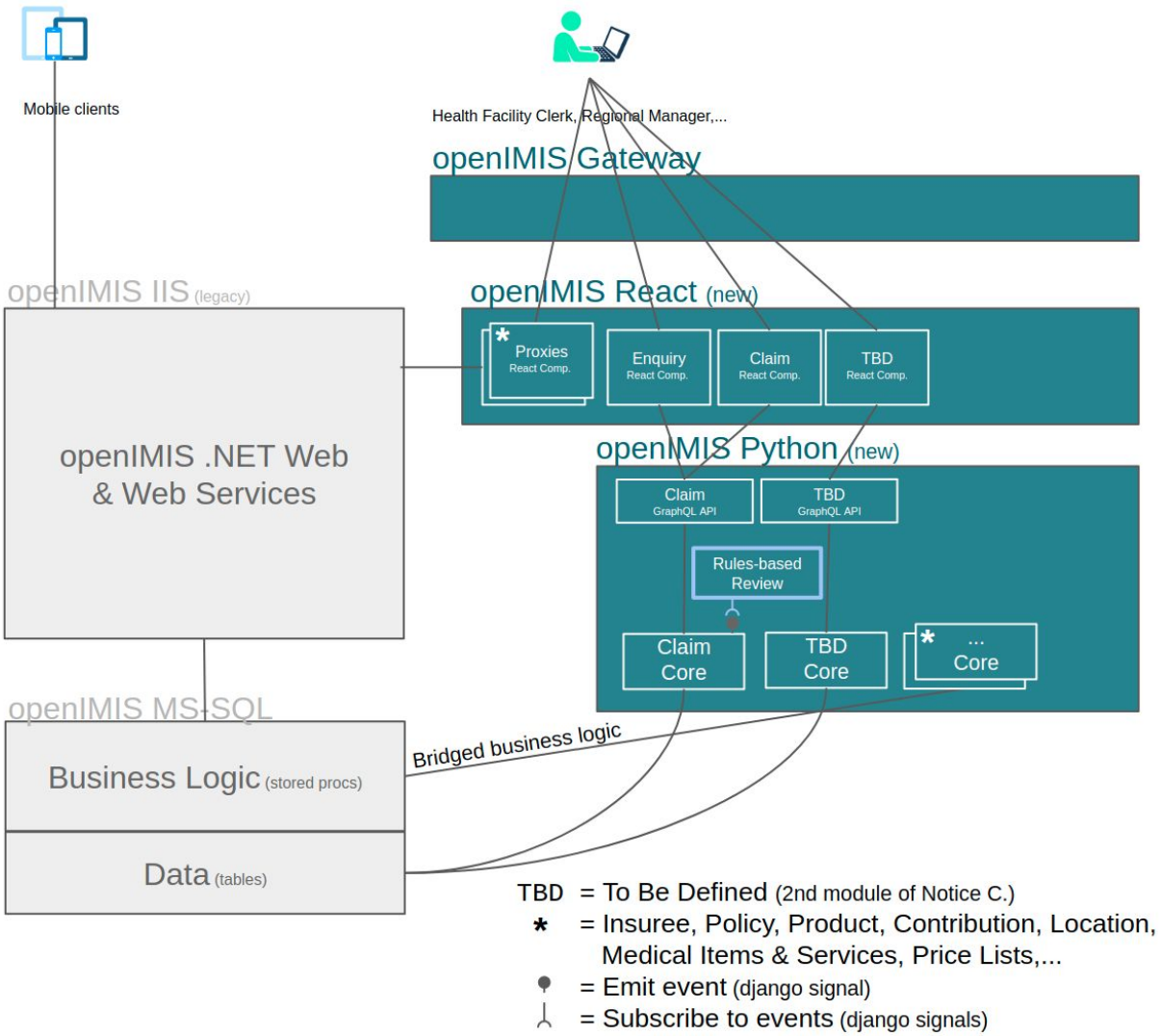
- A Configurable Claim Review Engine, which can be used at Claim entry/submit to validate a claim prior to any further treatment or, once the claim is successfully submitted, for (automated) review / adjudication.
- An AI-based Automated Claim Adjudication support, including fraud detection.

### Configurable Claim Review Engine

The new openIMIS architecture already integrates django-rules: a rule engine primarily used for access management… but which can clearly be used in 'other contexts' (incl. claim review).

The Claim module rewriting is performed according to the new architecture standard and integrates events (django signals) at each claim status change. The django signals can (but don't have to) be "in transaction": in other words we can plug a django-rules execution context (with the rules to apply) and allow it to block the transition in case of rule violation or emit 'alerts'.

In other words, thanks to the modularized architecture, we can plug a rule-engine based claim validation module as an extension to current claim module. This module would allow countries to dynamically change the validations performed (via rules) when a claim transits from entered to submitted (and/or other transitions if useful):




**This is however not the end of the story**:

- Rules have to be configured by (admin) users, tested (errors in the rules can have a significant impact), versionned (probably with dependence on several models: obviously the claim… but probably also the insuree, the health facility,...).
- Once 'live', the rule engine will have to 'explain' errors (why it rejected a claim) to users so that they can take corrective actions.
- Rules may have a real impact on performance: if claims are submitted in batch via the FHIR API, the rule engine will trigger the rules validation on the complete dataset. So if the rule requires the insuree gender, date of birth, the health facility type, available services,... all necessary data must be loaded from database when rules get evaluated… in batch (and just by a 'subtle' change in the validation rules by users).

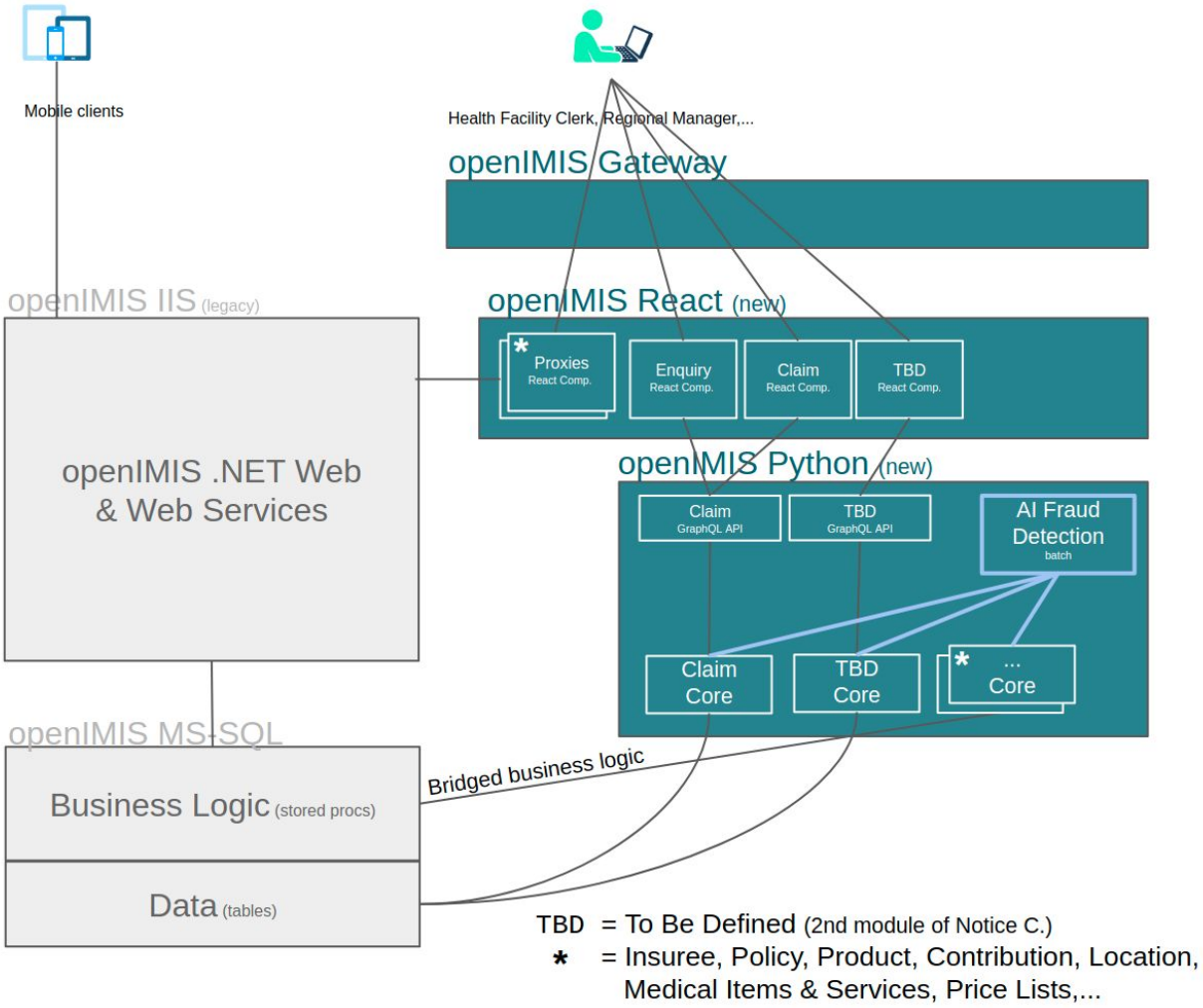
In other words, adding a rule engine for claim review is a very low complexity task on the technical side (ranking 1 if the one already in place is suitable, a little more if we want a nice UI to define rules,...). It has a much greater impact on the implementers side however (ranking 5-6).

**Note:** following the same concept (intercept, in transaction, the claim state changes events /signals), a simple (pure python code) country-specific module can be developed and added to deployment. This already allows countries to specialize the validations, without the need for a rule-based setup...

### AI-based Automated Claim Adjudication

AI technologies can be seen as an extension of the rule-based mechanisms… and it *could* plug onto the claim processing in the same way (intercept signals,...).
However the background technology, is by nature, even more data-consuming (and hence performance impacting)... and probably better suited for batch (autonomous) mode.
Furthermore, while the rule engine can actually 'explain' to users why it rejected a claim transition by pointing to the violated rule, machine learning algorithms are better suited in a 'I see something weird here, could you double check' approach since they provide no further explanation. In such an approach, the final decision will always be left to users. The obligation to provide a justification (based on explicit rules) can be enforced legally, as is the case in Europe under GDPR.

This must be refined, but it is potentially better suited to add a new state to claims, prior to being 'approved' in order to help depict the 'fraud detection status'. This flag (transition) would be taken in charge by the fraud detection batch. For countries not using the AI fraud detection module a default (no-op / pass thru) implementation will also be created…




There will be two very distinct steps to realize this requirement:

1. open the “hook” for the fraud detection management in the current claim state diagram
2. configure/fine-tune/train the AI-based fraud detection module

Point 1 can be a rather easy task on both the technical and business sides (ranking 1-2): the complexity will probably be to ensure that adding a claim state doesn’t have any side effect on the existing code base… but since the module will have been ported to the new architecture, it should not be a big issue.
Point 2 is more complex (ranking 4-5) and requires specific skills somewhere in between the technical and business sides.

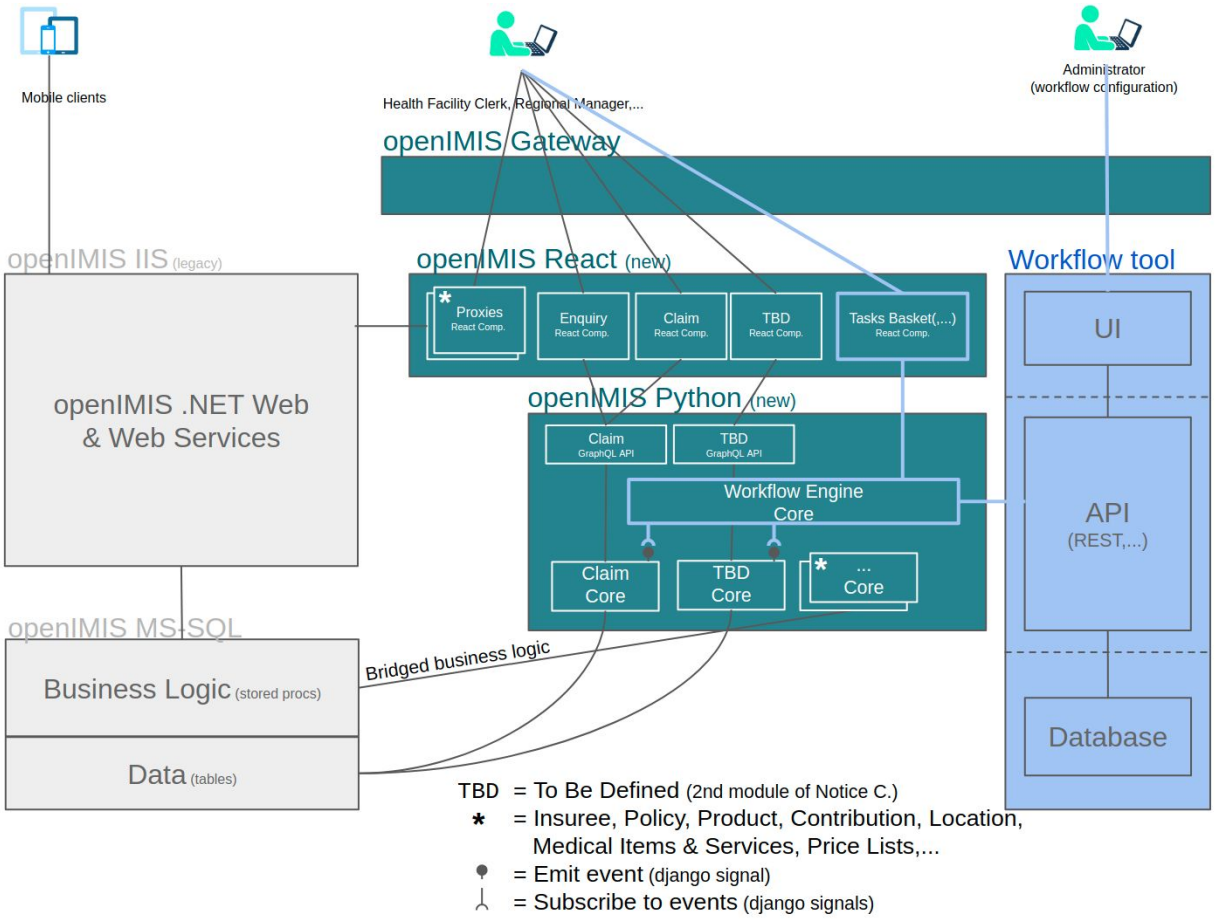
## Configurable Workflows

Business Workflow management is a very large topic, ranging from simple task flow assignment tasks (to user, automated process, bot, …) to “full scale” BPEL support with integrated governance tool (who edit/test/rollout the business processes configurations), process monitoring (SLA tracking, workload alerts,... ), process optimization,...

The expressed need goes beyond a simple task management tool… so we believe it should not be addressed via custom code but via (open source) tool integration. http://viewflow.io/ (as mentioned in the wiki) is a clear candidate, but there are others: https://ode.apache.org/, https://www.jbpm.org/,...

Such tools can be integrated at various levels in openIMIS stack: from a pure backend module that update entities in the background to user interface-integrated screens where users can be alerted for actions (and directly access the page where the action has to be undertaken). The required level of integration greatly influences the workload to be assigned to this work item.

Whatever Business Workflow management tool is chosen, its integration in openIMIS is clearly better suited in the new architecture: thanks to its event-driven core concept, a business process engine can autonomously react on “things that happen” (claim submitted, claim rejected, policy invalidated,...) and trigger (mark progression in,...) the business processes. Business process engines also usually require clear APIs to interact with the core components. The GraphQL generic (built by IOC) REST API provided by the new architecture (and more specifically its “mutations”) is particularly well suited for this.




Yet, since most of the modules are still in the .NET codebase (where offering events and standardized API is not available), implementing a configurable business workflow engine will be restricted to interact with migrated modules only.

For example, managing the payments (contributions) followup via a configurable workflow engine would greatly be facilitated when the Contribution module is in place in the new openIMIS.

Adding a configurable workflow management tool (in the new architecture) is not perceived as technically complex in the new openIMIS software architecture. It will however generate much more work on the business side: beside the (admin) user guides to configure the workflows themselves, the (end) user guides will also have to somehow integrate the (implementation-specific) workflow configurations.

With all these considerations in mind, we estimate this topic to be:
Technology rank: 2-3 (choosing the appropriate tool and plugging it)
Business rank: 6-7 (setup of workflow governance and documentation for it)

## Claim Management Workflow

Beside Improved Claim Review section, the RFA also mention that Claims should be managed via a configurable workflow. Since the Claim module is (or soon will be) migrated to the new platform by the time this award is attributed and provided that the Configurable Workflows module is part of the scope of this RFA, the Claim Management via the Workflow engine should not require any additional technical work and should 'only' be a business configuration concern.

## Beneficiary Enrollment Workflow

The "Beneficiary Enrollment" as a set of business processes to manage insurees, families and their policies requires features from several modules: Persons & Families, Policies and potentially Contributions.

To address the beneficiary enrollment, **it is thus advised to first migrate these underlying modules**, add the Configurable Workflow module and configure the beneficiary enrollment as a managed process of that workflow engine.

With this approach, the Beneficiary Enrollment should not require any additional technical work and should 'only' be a business configuration concern.

## Insurance Scheme for Formal Sector

The Insurance schemes for Formal Sector requires data model modifications and the provided documentation depicts two contradictory situations: the RfC_XX_support_of_formal sector_20190701_PD.docx seems to link the Policy to the Employer, while the schema in the comment describes the Policy as linked to the insuree (Person) in the scope of an Insurance Product.

Whatever model is finally adopted, in the current model a Policy is bound to a Family, composed of family members (the insurees). In other words for the tool to support formal sector introduces a major change to the core Policy/Person model.
Many other core entities will also have to be extended (record wages,...) and most of the core processes will have to change: renewals, valuations (and related contributions), specific reporting for the Insurance Taker (aka. the employer).
We can also expect the business processes themselves to be rather different: probably more based on bulk actions (the list of employees - and updates - as CSV,...) with a process by lot... which more than likely requires an adapted UX (User Experience) design.

One major question (asked during the August 16, 2019 Q&A Call #1) is the necessity (or not) to have both models within the same openIMIS instance. We acknowledge that best would be to have one openIMIS package able to support both 'worlds' in one instance. Yet, given the necessary changes mentioned above, the risk this approach brings is quite high. A probably more pragmatic (and cautious) approach with two instances would allow us to:

- Reuse (re-assemble) modules that can easily work for both “worlds” (maybe at the cost of some additional parameterizing). These could include Claim processing, Medical Items/Services (and Price lists), Location and Health Facility (and the related reports),…
- Re-think (either migrate/enhance… or delegate) the modules that are difficult to “combine” (like Products, Policies and their , Enrollment process,...).

In the longer term (once all modules in new architecture and database are switched to profit from JSONB flexibility at entity level) we will more easily:

- Merge modules from the informal and formal sector where relevant (profit from formal sector product options in informal sector,...).
- Ensure non-conflicting cohabitation of modules that cannot (should not) be merged.

In any case it is very difficult to  evaluate this topic adequately and, should this topic be on high priority, we suggest to start the work by establishing a clear roadmap.

The ranks for the development workload is to be considered  high (rank 7) as well as the business workload (rank 7).

## Communication platform

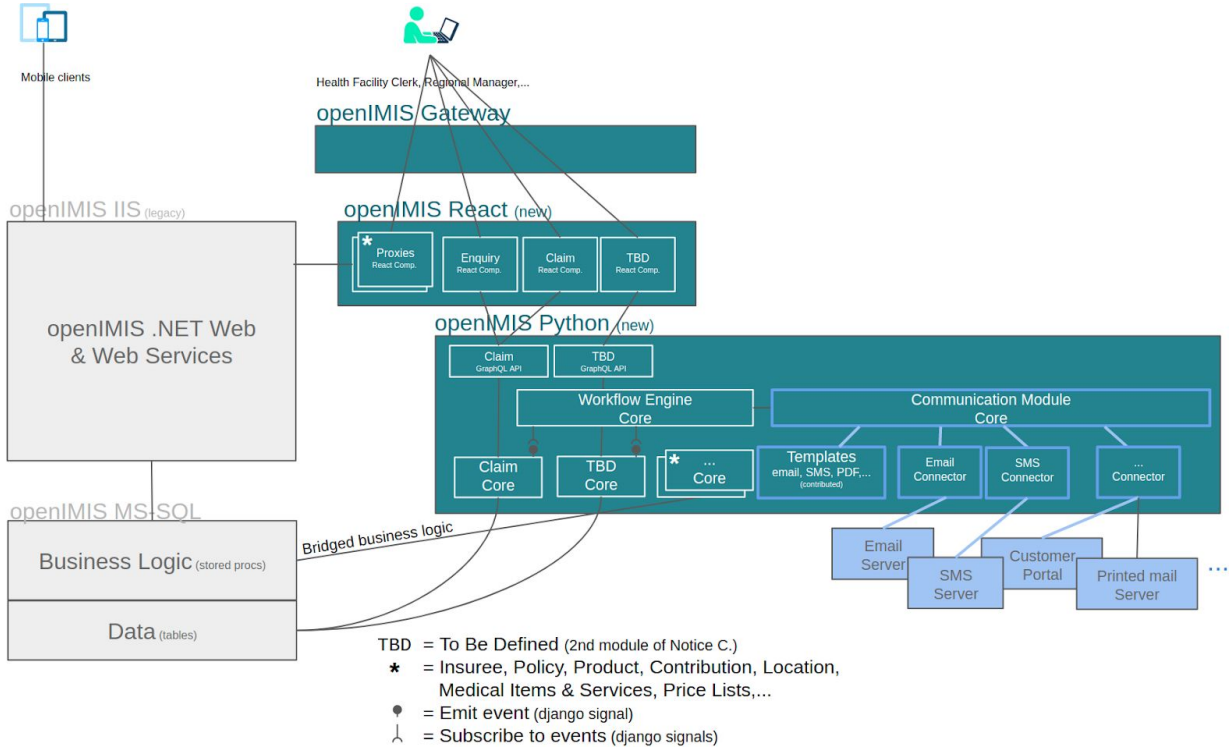
The Nepali requirements include a communication platform towards the users, insurees and beneficiaries.
We strongly believe that communication towards users and customers (insurees and beneficiaries... but also payers,...) are two separate things that should be addressed independently.
Communication to users should be natively integrated in the Configurable Workflow management solution.
What we will look at more closely though is how to manage the communication towards customers.

Communicating with customers involves several dimensions:

- the content and representation of the communication: email subject/body, printed document, SMS notification content, customer portal private message,...
- the technical platform(s) taking the communication in charge (email server, regular post services, customer portal site,...)
- the workflow associated with the communication: from ‘fire and forget’ notifications to complex communication with escalation process when communication didn’t result in expected actions from targettee.
- the internal management of the produced communication items (archiving and auditing capabilities,...)

Communication towards customers will also probably be very country specific and achieving high flexibility is a must.

As a result, one possible approach is to provide a small core generic module, within the openIMIS platform, that would standardize the way each openIMIS module communicates with “the outside world”. This standardized module would then be mapped to country-specific platforms.
The triggering and follow-up of the communication would be implemented in the chosen Configurable Workflow management solution which would include (wherever needed) interfaces towards archiving (EDM) solutions,...




This would require us to split this step into separate work items:

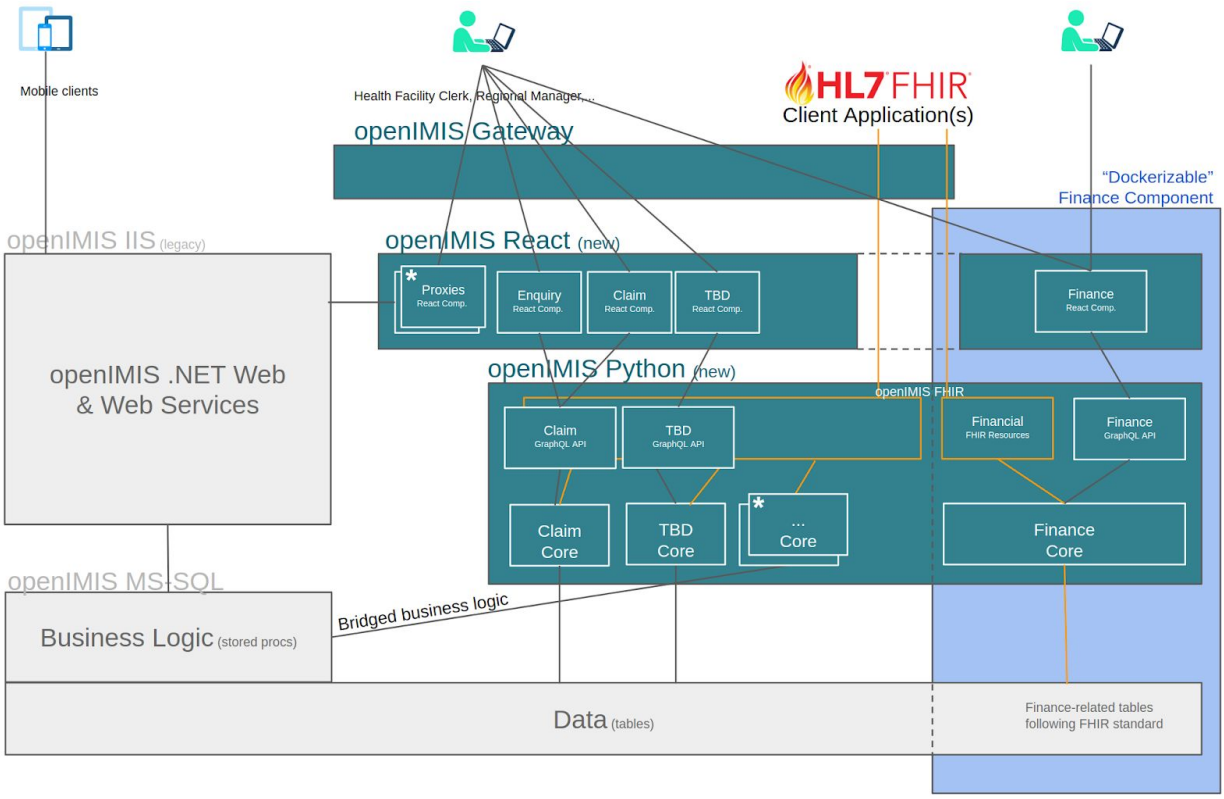
a. Develop a module that would allow decoupling of openIMIS business modules communications from its final form it is sent. Development of this module should follow the contribution principle in place in the new openIMIS architecture and allow each module (communication ‘extension’) to register its communication “templates” (customer language sensitive “placeholder” email bodies,...).
b. Identify current (customer) communications and provide a default (email?) representation.
c. Connect that module to one chosen (default) communication platform (email server?) via the Configurable Workflow solution.
d. Connect the communication platform to an EDM solution.

a. and c. are purely technical work items with medium-to-high complexity: rank 4-5
b. is more business-oriented and requires to look beyond a single country-specific case to reach effective flexibility. It is however not identified as a very complex activity: rank 1-2

d. is optional and is not perceived as very complex from a technical point of view (rank 1-2), maybe a little more on the business point of view (reaching the right document classification/taxonomy is often much more complex than expected).
Though this item is not dependent it will be more easily addressed after the Configurable Workflow is in place.

## (HL7 FHIR compatible) Payment module

Since payment management is not identified as primary focus area for the openIMIS platform, the Integration of a Payment Layer would preferably be addressed by searching for an (open source) system that could be delegated/interfaced directly to it. Ideally that system should be based on HL7 FHIR financial resources.
However, if no such system exists, developing such a module as a standalone service would greatly benefit from the new openIMIS development framework: the various mechanisms in place (contributions, events,...) are clearly of interest in a design where the core processing/logic has to be interfaced with (country-specific) external systems (using the same architecture/reasoning as the Communication platform). Thanks to the new “containerized” assembly/deployment, such a standalone service could also be deployed as a micro-service, with its own components assembly (lifecycle,...) and, as such, it will be reusable in environments where openIMIS (as an insurance management product) is not involved.



In any case it is very difficult to provide an accurate evaluation for this topic and, should this topic be of high priority, we suggest identifying a partner with experience in the field. The development workload is most likely high (rank 7) as well as the business workload (rank 7).

## Not covered

The Nepali requirements also include items of note (Sickness & Maternity benefit, customer portals,...) which will require further scoping/analysis before determining the best fit solution.

## Topic Dependencies and Suggested Roadmap

As stated in the current situation description, there are many good reasons to push migration of legacy modules into the new platform. Our proposed approach is to be conservative on the challenges proposed in the changes requested in this RFA and ensure most of the work to build the base of the structure gets done to ease further adoptions and feature development.

Improved Claim Review via a Rule engine and/or AI-based algorithms are within reach: they can already benefit from the work accomplished thanks to Notice C. and have no further dependencies. The necessary budget to develop the solution is perceived as rather low… but will clearly impact the community support work (Stream C).

The Configurable Workflow engine and related Claim Management / Beneficiary Enrollment processes refactorings seems to be premature and its roll out would be best suited once the underlying major core components are migrated to the new platform.

The Insurance Scheme for Formal Sector topic seems like it will be very challenging in the current situation. To answer this need we see the necessity to change the core model which would require a new code base split between the informal and formal sectors solutions. This split, in a cartesian product with the current master (Tanzanian?)/Nepali code base is problematic.

The new modular architecture would ease flexibility, but it is not yet in place for most of the necessary modules (Products and Policies). Should the Insurance Scheme for Formal Sector be considered a very high priority by the community, we would recommend reusing ‘as is’ where possible (Location & Health Facilities, Medical Services and Items, Claims,... )... and restarting from ‘scratch’ for the other modules (via delegation to external systems wherever it is possible).

The Communication Platform would clearly profit from a Workflow engine so it also seems premature to address this topic.

Finally the (HL7 FHIR Compatible) Payment platform, if not developed as an interface with an existing solution, will probably not fit in this RFA budget… However, it could be approached as

an independent product (sharing the same technology stack as openIMIS) and be co-funded by other programs where such a platform is also needed.

# Stream C.

The work to be delivered along Stream C. is highly dependent on the choices made for Streams A. and B. and will need to be totally aligned with what is currently being done as part of work on Notice C.

One major challenge that needs to be anticipated is the move from an “all-in/standalone” solution to more and more spread out/country-specific installations. This will be true on several levels:

- Intra-module: with the contributions mechanism of the new platform as well as the use of independent (/plugged on events) modules such as rule engines ...
- Inter-module: with the move towards more transversal/configurable tools such as the Workflow Engine and the Communication Platform.
- Inter-systems: when openIMIS will delegate to external systems.

All the community support in place (from user guides to issue queues and knowledge base, wiki documentation) has to be reviewed according to this need for flexibility.

Finally, our description of the current situation (above) also clearly indicates that any new implementations should be approached very carefully, first gathering requirements to identify the future flexibility needs  (and integrate them, where/when possible in the rewriting). Should a new opportunity become more concrete, a specific roadmap, probably putting migration of existing modules at high priority, would need to be elaborated (with the associated budget).