RFC50: Add support for additional arbitrary variant annotation

Title:	Add support for additional arbitrary variant annotation
Proposed by:	Philipp Unberath (MIRACUM)
RFC Type:	Exploratory
Date Proposed:	2019-07-11

1. What central problem does this RFC address?

Adding support for additional arbitrary variant (mutation) annotation.

2. What feedback are you seeking from the Community?

Feedback on the implementation concept.

3. What is your proposed solution?

Described in detail in the implementation proposal. In a nutshell, it consists of adding a JSON column to the mutation table to store arbitrary variant annotation and extending the backend to fetch and the frontend to display these annotations.

4. How is the proposed RFC useful for the public cBioPortal?

Right now the annotations that can be stored in the mutation table are fixed although the MAF format can contain arbitrary annotations for mutations.

With the proposed JSON column it would be possible to store all the information contained in the MAF file to later support the display of useful information (e.g. quality scores).

There is also the possibility to utilize this approach to identify the most used and useful additional annotations to implement them as fixed columns in the mutation table.

5. How is the proposed RFC useful for local cBioPortal instance, e.g. at academic centers or pharmaceuticals?

Institutions like hospitals or pharmaceuticals would benefit from this approach as they could upload and display all the information their custom variant annotation pipeline produces.

6. What is the impact of this RFC on the current refactoring efforts?

None expected.

7. How will the work proposed within your RFC be maintained over the long-term?

We will maintain this functionality and if necessary extend/adapt it during the supporting phase of MIRACUM at least until the end of 2021.

We're also hoping this becomes core functionality and other institutions will likely use, maintain and extend it beyond this time period.

Implementation Proposal

Files & Import

MAF

No changes are necessary as the MAF format already support arbitrary columns.
In order to be able to link concepts one would need to define standards that should be used for all/some columns (cf. clinical data e.g. gender).

Import

Adapting the import routine for mutations to merge all currently unused MAF columns to a JSON document and store it in the mutation table. This includes changes to the mutation model and DAO, as well as the import script.

Table 1: Examples for files to be modified for the import routine.

File	**Comment**
DaoMutation	Use JSON field in addMutation
MySQLbulkLoader	Potentially adaptations necessary because of the JSON format
ExtendedMutation	New JSON field
ImportExtendedMutationData	Build JSON from all unused columns and set JSON field

Database

A new column with the type JSON will be added to the mutation table which can hold all the additional annotations.

Back End

Extending the data fetching for mutations to return the additional annotations. This includes a substantial amount of adapting multiple files and classes responsible for the internal WebAPI.

Front End

Extending the mutation table in the patient view and the plot view to support the display of the attributes in the JSON column.

Table 2: Examples for files to be modified in the front end.

File	Comment
MutationTable	New concept for the generic display of arbitrary annotations (as for now the columns that can be displayed are hard-coded)
PatientViewMutationTable	Mutation table in patient view
ResultsViewMutationTable	Mutation table in results view

Mockups

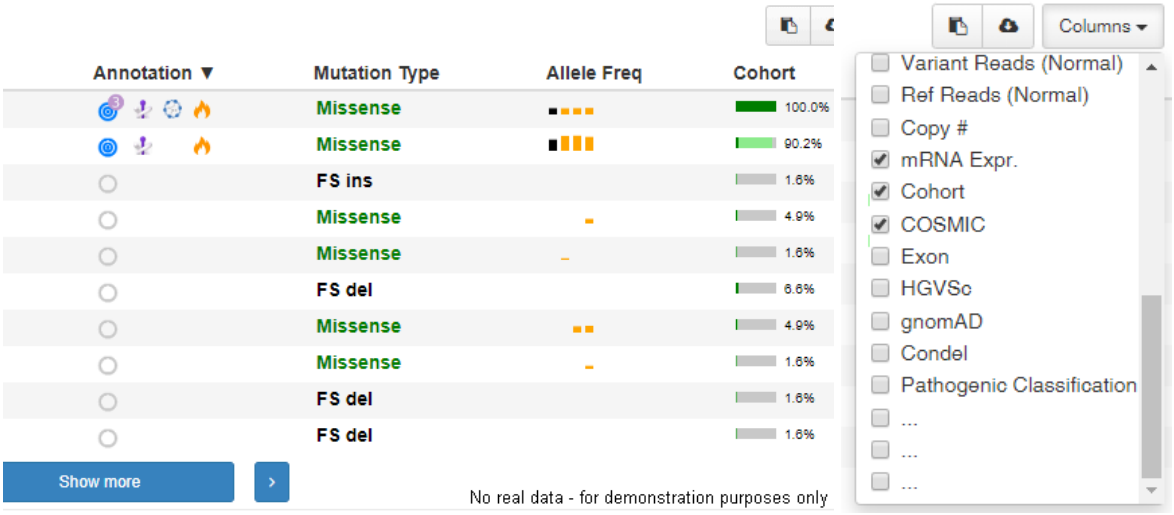

Figure 1: The current mutation table with the defined set of annotations. The dropdown menu "Columns" contains all annotations that were included in the MAF file and therefore imported in cBioPortal.

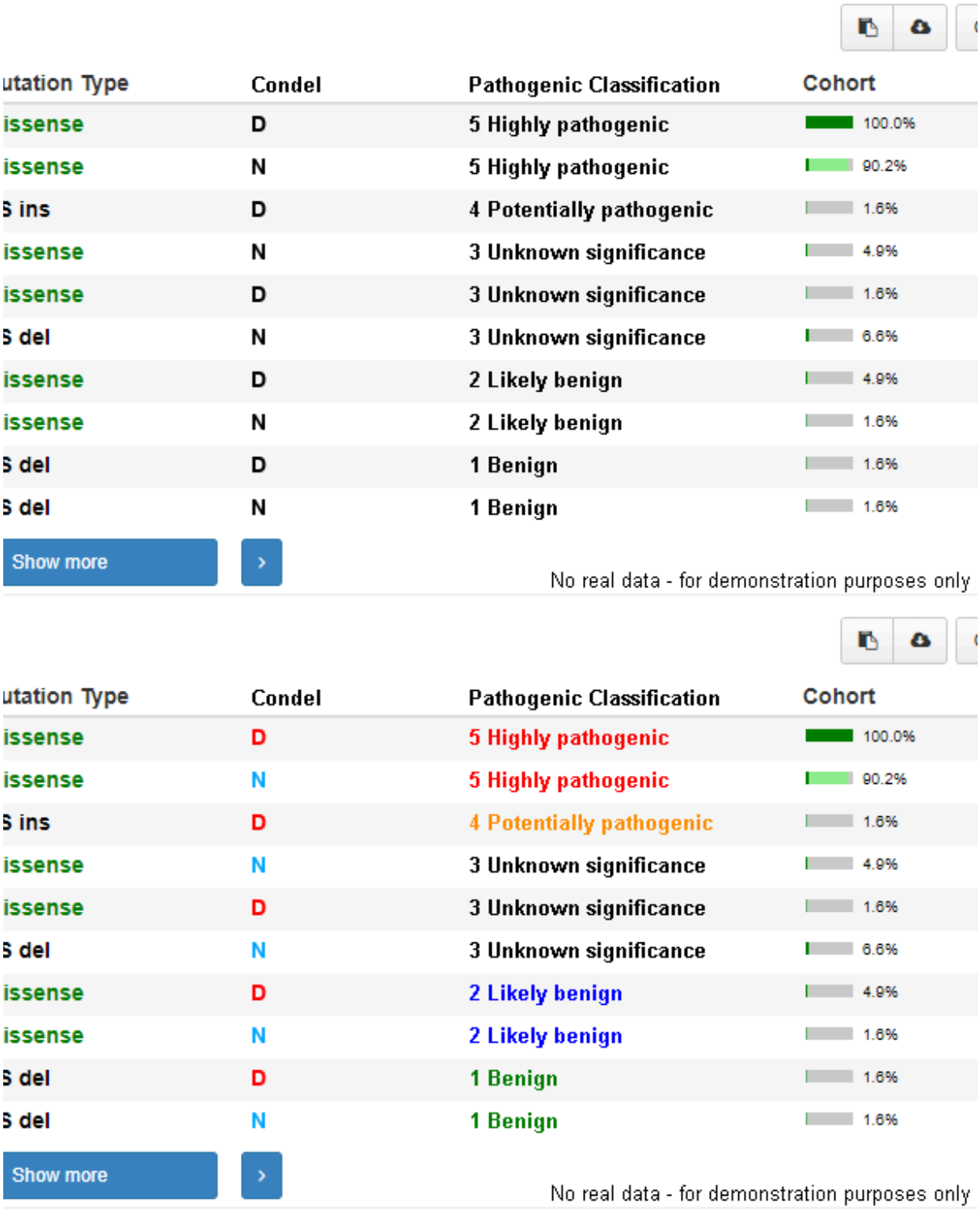

Figure 2: Further annotation of mutations included as additional columns in the mutation table.
Option I (top): Annotations are displayed as stored in the database - i.e. plain text or numbers.
Option II (bottom): The annotations are augmented with a means of customizing the visualization - e.g. colors for different categories or numbers exceeding a threshold value. Another option would be a link that is used to display additional information about that annotation on mouseover.

Pending/Potential Issues

Here we describe known potential issues as a basis for discussion. Some of the issues need to be analyzed with a prototypical implementation.

- Issues regarding the system design of the JSON document:
    - It may be difficult to store complex annotations, that consist of more than just numbers or strings, and to visualize them in a generic way.
    - The option for customization in Figure 2 Option II introduce a lot of complexity and overhead designing.
- One has to identify or define standards for additional annotations to enable comparability and interoperability.
- There could be performance bottlenecks.
    - Extensive annotation in the JSON column could lead to high network load when fetching mutation annotation.
    - Later on: Without proper indexing of the JSON annotation, searching within those annotation requires full table scans.
- The mutation table could require a substantial amount of additional storage, depending on the amount of annotation.
- How to update the mutation annotation to avoid overlapping? Namespace declaration.
- How to include the backend code with a dynamic data structure defined in database but requires backend support, for instance, adding a new endpoint for a specific data in annotation column.
- How will the frontend client look like? The swagger client supports LONGTEXT which means the logic of deciding whether curtain component is enabled is after fetching the mutations data. And when mutations fetched do not have that particular annotation but does not mean the database does not have that data stored. Provide the default?
- How does this integrate with Genome Nexus? We could for instance:
    - Also store genome nexus annotations in the proposed JSON column
    - or instead of extending the mutation table, extend integration with genome nexus such that people can add their own annotations more easily
    - it could also be that a dual approach makes sense where some portion of core JSON fields are stored in one and extended annotations in another

Current workflow to support website feature for an additional mutation data

1. Update the migration code

2. Update JAVA model
3. Migrate database(is this very time consuming?)
4. Deploy backend
5. Generate frontend client
6. Development feature

Options to support additional arbitrary variant

Use Cases:

- OncoKB analysis. We decided to add oncokb data in clinical table for sample analysis in study view. But also like to keep the mutation level data in the annotation column to prevent data discrepancy(comparing to the current real time annotation). In this case, a nested structure is needed.
- Additional sample-specific annotation. e.g. zygosity classification ("heterozygous"/“homozygous”) or value for the quality of the reads at this specific position

Option 1: Add annotation JSON column in mutation table

Preferred if each additional data(namespace) has a different structure and will not be used in service layer(meaning we do not convert the String to a JAVA model)

Pros:

- Flexible.
    - No migration code
    - No update JAVA model
    - No need to migrate the database
    - No backend deployment needed.
- Only update the mutation/study if necessary.
- Any other institution could be able to plugin any data they have to the JSON column and develop frontend feature when needed. We could create a generic plugin model in the frontend to accept any visualization as long as the data exists. (potentially, not even need to add the type by the core team? Only the institution who wants to use the new data?)

Cons:

- Frontend needs to make a type for each namespace added in the database
- Other service layers can not use these JSON data

Issues needs to be addressed:

1. The namespace needs to be declared in JSON.
2. The namespace format?

3. How to let the frontend knows certain data exists? For instance, a subset samples in a study do not have oncokb data in annotation column but other samples do. When the study view asks for the information about the subset, the page will think the oncokb annotation does not exist, so visualization will be hidden, but after resetting the filter and look at the whole study, the visualization will show up?

Option 2: Add annotation table

Preferred if the additional data is key-value pair

Pros:

- Data is structured
- Frontend does not need to generate a type to understand the data, it's autogenerated.
- Other service layers can use the data

Cons:

- Not flexible
    - Need to write migration code
    - Need to add JAVA model
    - Need to migrate the database
    - Need to redeploy database
- Maintaining the columns in the annotation table even if some columns are not needed for some instances
- Discussion needed with the consortium whether the column is needed in the annotation table