Environment

The application is composed of three elements: the user interface which includes the web pages and scripts that implement the web site, the backend which encapsulates all the logic and data structure into a set of web services and the database that physically stores the data.

The user interface never interacts directly with the database, it exchanges information with the backend that provides a high level view of the operations and data.

The majority of the forms should be dynamic, this means that the user interface will request a specific form, the backend will return a structure that will contain all the information needed to display the labels, construct the input components and validate the content. This means that the core of the front end work will be to create components that deploy themselves according to the data received from the backend.

Technology

The database is ArangoDB, it is a multi-model database that supports document, relational and graph structures, it communicates with the backend with a JavaScript framework called Foxx.

The preferred technology for implementing the front end would be React with JSX. React would be a good choice, because it enforces creating independent components, which follows the logic of the application.

Other frameworks and technologies can be used, if React is unavailable, they should be either written in JavaScript or PHP, which are the two languages that I am familiar with.

Application

The application should essentially allow the harmonisation of uploaded datasets with a centralised dictionary, provide access to the harmonised data with user queries and allow visualising and downloading the results of these queries. The application is structured into a set of components composed of input and output forms covering the following areas.

Typical workflow

The typical workflow for the majority of operations involves the following steps:

1. The front end requests a form from the back end by providing an identifier and eventual data for initialising the form.
2. The back end compiles a structure containing the list of fields comprising the form, each having the necessary information so that the front end can display the fields. The structure elements will contain:
    1. The field identifier.
    2. The field data type.
    3. The presentation mode (read-only, read-write, hidden, etc.).
    4. The label, definition, description, examples and notes. The label will be used as the control label, while the other fields will be used as additional information. The

definition might be displayed when the used hovers over the field label and the other fields could be displayed by having an information button next to the field.

5. The eventual initial data.
6. Other fields that provide additional information depending on the type of the field.

3. The front end should then display the form and validate the user input as it is provided. The validation can be performed by the back end by providing the field identifier and the filled data.
4. Once the user is done and presses the submit button, the front end should extract from the form the field identifiers and their value and return this information to the back end, which will perform a final validation, execute whatever action is needed and return a status report to the front end.

Components

Since the main activity of the front end will be to generate controls in an input or output form, the first elements that should be developed are the components or controls that will be placed in the layout. The initial choice of React as the front end technology is precisely to allow creating self contained and independent components that know how to behave according to the data they are provided, any other technology that allows this modularisation will do.

The components to be developed depend on the data type of the field, there are a number of different types, I will list here the main data types according to the different component types to be developed.

One common characteristic of all these components should be the ability to display a definition when the control is hovered and have an information button that can open a window that provides the full set of descriptions for the field and an area where status messages can be displayed, such as when the user inputs incorrect data.

Boolean

This is a simple radio button for true/false values.

Text

This is scrollable area where to insert long texts.

String

This is the default text input control, it is a one liner text input which may have a maximum character length. This type comes with several sub-types, such as password, e-mail and other variants.

Numeric

This can be implemented with the string input, except that it must contain numeric data, this type is divided into two sub-types: integer and float. The first should accept numbers without decimals, the second should enforce decimals.

Reference

This represents a reference to a record in the database, the value is a string that identifies a specific record in a specific database collection. The peculiarity of this kind of control is that the user should be presented the target record, rather than the reference itself. Database records or objects should have at least three output forms, one for tabular views, one that can be considered as a card which shows the main information regarding the object (this would be used in the reference case) and a full view that contains all the information. References will generally be set programmatically, or may be selected from a list of choices, such as enumerations that we will cover now.

Enumeration

This represents a controlled vocabulary choice, such as a country selected from a standard list. The value is the key to a specific collection and it must be selected among a predefined list of choices. Enumerations are hierarchical lists whose root is the element that defines the controlled vocabulary. For instance, suppose we have a generic “location” enumeration, this would start with the element that can be used to reference the enumeration as a whole. The root element may have “continent” children, continents may have “country” children and countries may have administrative unit subdivisions. The nodes of the hierarchy that contain children may represent valid choices, for instance, we may know that somebody lives in Italy, or we may know the person lives in southern Europe: Italy belongs to southern Europe and both choices are valid. There should be a mechanism to select recursively all children of a specific tree branch. These nodes may also not represent valid choices, but be used as categories to subdivide the list of choices to make them easily searchable. In the last example, for instance, southern Europe could be a collapsable category, but not a selectable option. An enumerated value is a string, or a set of strings, that are the key of records in a specific collection, the user should be presented with a set of enumerated structures (the list is provided in the form field record) and should select one or more elements. This is a fundamental component, since all controlled vocabularies and other elements, such as the forms themselves, are structured this way. An enumerated field in a form will have a set of enumerations (root element of these structures) that the front end will display to the user, the back end will provide the necessary services to retrieve the elements of these controlled vocabularies.

Enumerated list

This type of field contains a structure in which the property name is an enumeration element and the value can be of several types. For instance, language strings are widely used in the data dictionary, these are structures where the property name is the language code and the value is a text in that language, labels, definitions and descriptions are implemented this way in order to provide content in different languages.

Range

Ranges are used in the data dictionary to indicate minimum and maximum values, these are used, for instance, to indicate the size of fields, or the legal range of numeric values. Internally they are implemented as an array in which the first two elements represent respectively the minimum and maximum value and the following two elements are booleans that indicate respectively if the minimum and maximum values are inclusive.

Other

There will be other types that are derivatives of the above main types which implement custom variations. For instance time stamps, dates,

Implementation

Once all the components are developed and tested, these can be placed in the user interface windows according to the form record requested from the back end. Note that the data dictionary is dynamic, forms contain only those fields that are required or suggested, but they can accept any other type of fields, so the windows should be organised in such a way that there is a scrollable and collapsable area that displays all the other available fields from which the user may select and fill data. The basic units of the application are as follows:

Credentials

This unit deals with everything that has to do with users and authentication. There are three main types of users:

- *Administrator*. This is the user that can do pretty much everything in the application, it can only be created once, when the users collection is empty.
- *User*. This is the standard user, the majority of the application operations require a user to be logged in. Users have a field called “roles” which is an enumerated set of codes that define which operations the user is allowed to perform. Since menus are organised as forms, the user will only be presented those that he/she is allowed to use.
- *Guest*. The guest user may have no roles, in which case the application can only be used by regular users, or may only be able to browse or search. There is only one guest user which will be automatically logged at the first connection with the application.

All front end requests involving users involves two main parameters:

- *Token*. This is an encoded string that is used to authenticate the front end, it is a way to ensure that the requestor is authenticated.
- *Form*. This is the form identifier.

The administrator is the first user to be created, the process follows these steps:

1. The front end requests the administrator creation form.
2. The back end asserts that the users collection is empty and sends back the form.
3. The front end displays the form, gathers its data and sends it back to the back end.
4. The back end validates and processes the data, created the administrator user, creates also the guest user and logs the administrator in.

Regular users are created by other users that we will refer to as managers. All users in the system must have a manager, except the administrator, the manager decides which roles the user can have, it can revoke them or disable the user. This manager relationship is a hierarchic tree, managers have access to all their siblings, at the root of the tree we have the administrator that will have access to all users.

The process of creating regular users involves two main steps: *sign-up* and *sign-in*. The sign-up process creates the user in the users collection and sends an e-mail notification to the created user. The user created in the database is disabled, so log in is not allowed, the only way to activate the user is to respond to the e-mail notification. The sign-in process is performed by the user when it responds to the e-mail, once completed, the user is allowed to log in. The steps are as follows:

1. The front end requests the sign-up form.
2. The back end checks if the current user is allowed to manage other users, if that is the case it sends the sign-up form.
3. The manager sets the required roles and other information necessary to register the user, then sends this data back to the back end.
4. The back end validates the data, creates a random username and password and registers the user in the database as an inactive user. It then encodes the user information into a string token and sends it back to the front end.
5. The front end compiles a sign-in e-mail containing a link that includes the received token and sends it to the new user.
6. The new user presses the link in the e-mail.
7. The front end receives the link request and sends the token to the back end.
8. The back-end processes and validates the token and sends back to the front end the sing-in form.
9. The user fills his/hers personal information and submits the form.

10. The back end validates the form data, replaces the username with the provided one and updates the user record setting it to active. The user is logged in and the back end sends to the front end the user record.

There will be other available services, such as login and whom which returns to the front end the currently logged in user record. In addition there will be services allowing manages to manage their siblings.

Study registration

The main function of this application is to harmonise datasets with the data dictionary and store their data in a single collection. By *study* we mean a survey or other project that is composed of many files and one or more datasets. By *dataset* we mean a table of data belonging to a specific study.

The first step is to ***register a study***:

1. The front end sends a request for the study registration form.
2. The back end asserts that the current user can register studies and sends back the form.
3. The front end displays the form, the user fills all the necessary information to document the study and returns the form data.
4. The back end validates and registers the study, it then adds the study identifier to the user record and returns it to the front end.

The next step is to ***upload all the annex files*** comprising the study, this would include the survey questionnaire, eventual reports and the original version of the dataset files. The upload and storage process should be handled by the front end, the services exchange would typically follow these steps:

1. The front end requests the upload form by providing both the form identifier and the concerned study identifier.
2. The back end checks if the user is authorised and sends the form record.
3. The front end presents the form to the user who provides the type of the file and uploads it. Once uploaded, the front end stores the file and computes either an identifier, or uses the file path as the identifier. It then returns the form data, the uploaded file information and the study identifier.
4. The back end registers the upload, attaching it to the study and returns the upload identifier.

This process should be repeated for each annex file comprising the study. Note that these files should be stored as-is and never modified, they represent the original copy of the study and will be made available for download to allowed users.

Dataset registration

Once the study has all its elements set, it is time to ***upload the datasets***. For the moment we should only accept CSV files. These datasets will actually be CSV versions of the uploaded datasets, so in the future other formats could be supported making the dataset upload step redundant.

The registration process should follow these steps:

1. The front end requests the dataset upload form.
2. The back end checks if the user is authorised and sends the form.
3. The user fills the information and uploads the file. The front end send back the form data and a reference to the uploaded file. *It could also send the uploaded file itself, but since in most cases the database and the web site will be sharing the same server it is easier to share the file path so that the back end can directly manipulate it*.
4. The back end validates the form, registers the dataset upload and gets the file, it will then do the following things:

1. It will create a database collection,
2. associate the collection with the dataset upload registration
3. and load the uploaded file's data into that collection;
4. it then returns the dataset upload identifier.

This process should be repeated for all the datasets comprising the study. Once completed, the study will have a set of collections containing the data of each dataset as it was provided by the user.

Dataset parsing

The datasets will be tables in which the rows represent observations and the columns represent the variables of the data. This process involves selecting the rows that represent the data and the row that represents the data variable tag.

The first steps are as follows:

1. The front end requests the dataset parsing form.
2. The back end asserts the user is allowed and returns the first 10 or 50 rows of the dataset.
3. The front end displays the dataset to the user:
    1. The user must first select which row represents the first row of data, we assume all subsequent rows to be data.
    2. Then the user must select which row is the header row, which should contain the labels or tags identifying the variable for that column.
    3. The front end returns the selections made by the user.
4. The back end performs the following steps:
    1. It removes all the rows that are neither the header or the data.
    2. It compares the values of the selected header row with the data dictionary.
    3. For each column that matches entries in the dictionary it compiles a list of choices.
    4. It then initialises a metadata record which associates dataset columns to

At this point the dataset collection will have one header record and all the data records and there will be a metadata record associated to the dataset containing the initial data dictionary matches.

Dataset harmonisation

This process involves matching each column with a single data dictionary descriptor, the outcome should ideally be that all the columns of the dataset are associated with a specific entry in the data dictionary. This will be an iterative process in which each column is processed as follows:

1. The user selects the column.
2. The user browses the data dictionary and selects the element that corresponds to the column data.
3. The user submits the choice.
4. The back end validates the data according to the selected descriptor.
5. The back end returns the validation results.
6. If there are errors:
    1. the user either corrects these errors using application featured processes, or downloads the dataset, loads it in their preferred statistical package and corrects the values.
    2. The user uploads the dataset again, except that in this case only the data is updated, the metadata record should still apply.

7. Once there are no more errors, the user submits a validated status for the dataset.
8. The back end transfers the data in the final destination collection and deletes all the temporary collections used during the harmonisation process.

These steps have been described as an overview, they involve more operations and due to the limited time it will not be possible to implement them, in particular, the data dictionary management is too complex to be implemented at least in its complete form. These are some of this functionality:

- *Enumerated data*.
    - When validating controlled vocabularies there might be invalid entries that are simply different codes for existing entries, in this case the back end could replace the invalid values with the correct ones.
    - Some invalid controlled vocabulary codes might have to be added to the data dictionary enumerations, an interface to add elements to a controlled vocabulary could be used here.
- *New descriptors*. Some variables may not exist in the data dictionary, an interface should be provided to insert a new descriptor and associate it to the column. Note that this is a delicate operation, since managing the data dictionary should be done by users that are familiar with the subject and added descriptors should go through a revision phase before becoming "official".
- *Unknown descriptors*. Not all of the columns must be matched, some columns may not represent relevant data or their nature might be unknown at the moment, this means that there will be a property in each observation record that will contain a set of unknown variables which may, in the future, be subjected again to the harmonisation process.
- *Bulk changes*. Correcting values should be made an easy process, a set of operations should be made available to correct existing data in a bulk mode. For instance applying regular expressions to string data, removing values that are out of bounds, etc.
- *Data quality*. While validation ensures data follows the descriptor format definitions, a set of data quality checks should be implemented to provide an evaluation of the quality of the dataset. The results of this data quality check should be stored in the study and dataset records and made available for consultation.

Data queries

Once the data is transferred to the final collection, it is available for querying. The query form will contain a set of predefined fields and should allow selecting other fields for search. The process is as follows:

- The user selects the field, sets the operator and sets the query value. For strings the operators could be "starts with", "contains", etc. Other types should be developed according to the data type of the descriptor.
- Once the form is submitted, the back end compiles a database query and runs it. There will be two distinct query results displayed in two distinct tabs:
    - *Study*. The set of studies that satisfy the query. For instance we could have a query requesting all elements containing a specific variable, in this case all the studies that have datasets containing that variable will be returned.
    - *Data*. The set of data records that satisfy the query. This will be a tabular output of the query results from the final data collection.

The displayed results should be displayed in a table component, developed as the other form components. The table should have the following features:

- *Pagination*, the ability to display a provided number of rows and to jump to any page.

- *Sorting*, clicking on a column header should allow sorting the data accordingly.
- *Collapsing and expanding*, each row, especially when displaying studies, represents more data than can be presented as a table row, it should be possible to click the row and have it expand into a card view.
- *Download*, it should be possible to download the data displayed in the table as a CSV file. In the case of studies, downloading a study would generate a zipped archive containing the study metadata record and all of its annex documents; individual annex files should also be displayed and made downloadable.

Framework

There is no specific requirement for the front end framework, except for the fact that it must be able to handle the developed components described above. Twitter Bootstrap is a very popular library that provides a lot of features useful to this application. There is another library, Ant Design, which implements a lot of the features that are needed by this application, in particular, the tables have the required functionality and the enumerations can be implemented with a component that has most of the required functionality.

Work plan

There is limited time and not all components will be able to be finished, but at lest this could be the order in which the work could proceed:

1. Develop a parser for the form record elements.
2. Develop the base components used in the forms, each data type should be implemented as its specific input control.
3. Once all base components are developed, implement a form parser that places these components according to the elements of the form record. This can be a generic element, since all forms share the same logic.
4. Start assembling that units of the application by implementing the workflows involving service requests and responses.