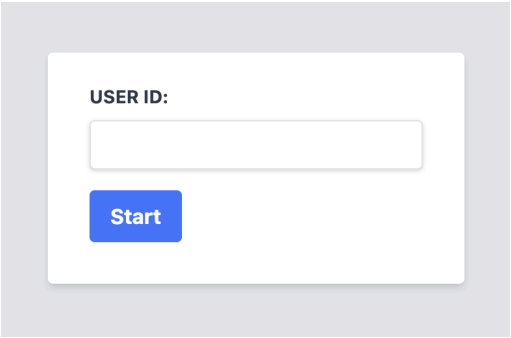
A

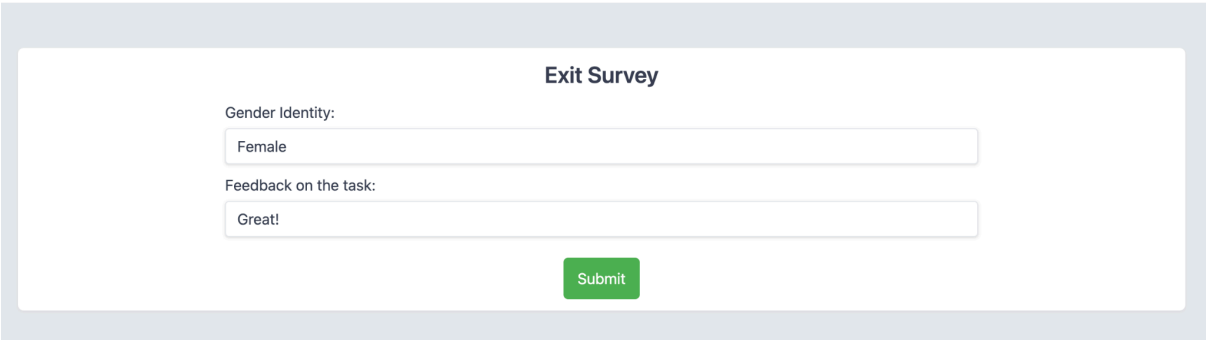
UI Functionalities

Collection of User Data

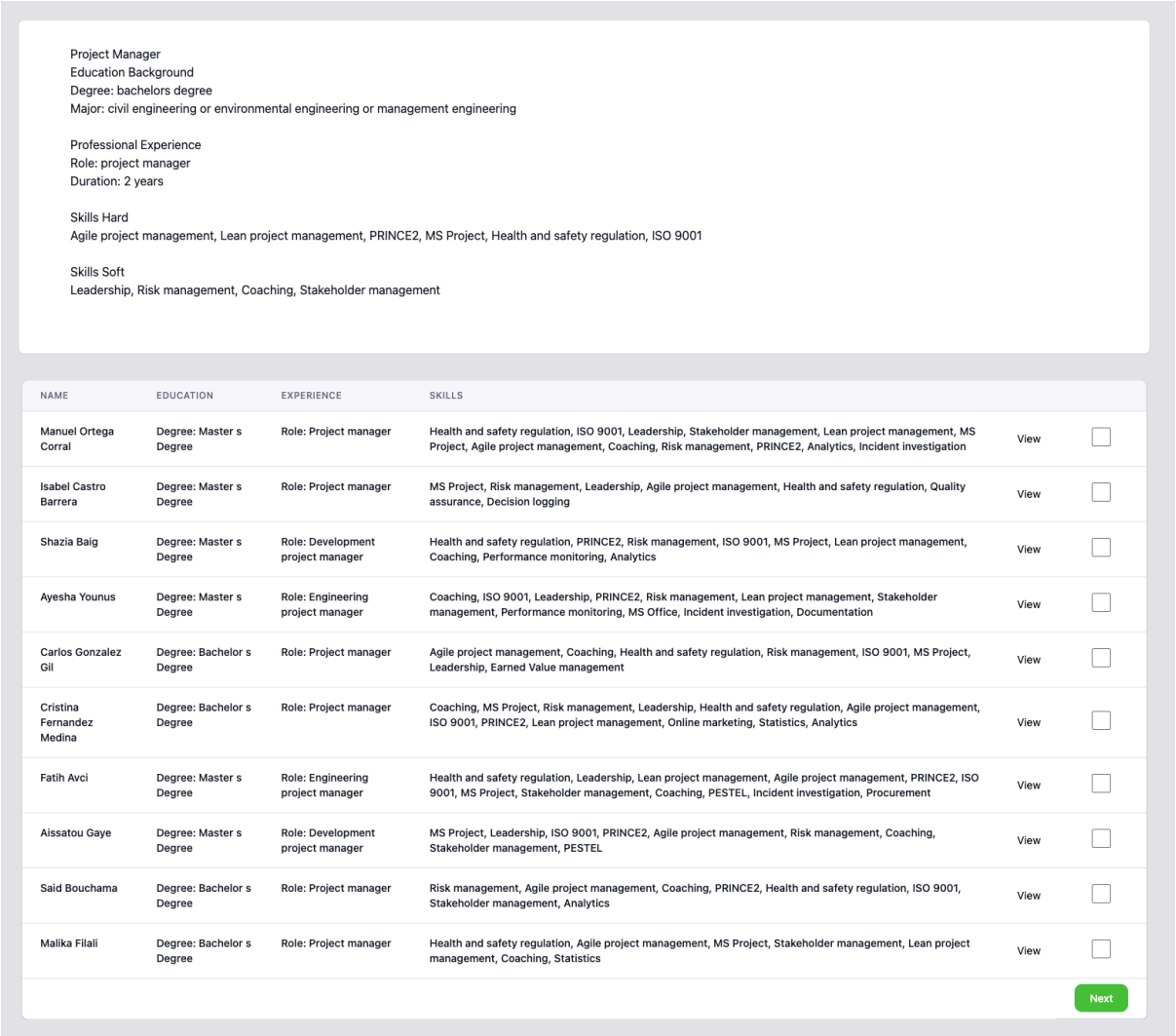
When interacting with the tool the users will be first asked to provide an ID with which they will be registered in the database. The user can interact with the UI by typing in the empty field the ID and then clicking start. All interactions will be collected during the session for each user ID.



At the end of the interaction with the tool, the users can be asked to complete an exit survey in which one can collect feedback or demographic data about the user. An example of an exit survey can be seen below. This can be configured to include other fields. More details about the configuration of the Exit Survey can be found under the section Configuration File. The user can interact with the UI by typing in the empty fields (e.g. feedback field) or by choosing an option from a drop-down (e.g. gender field), then when clicking submit all the collected values are stored in the database for the corresponding user ID.

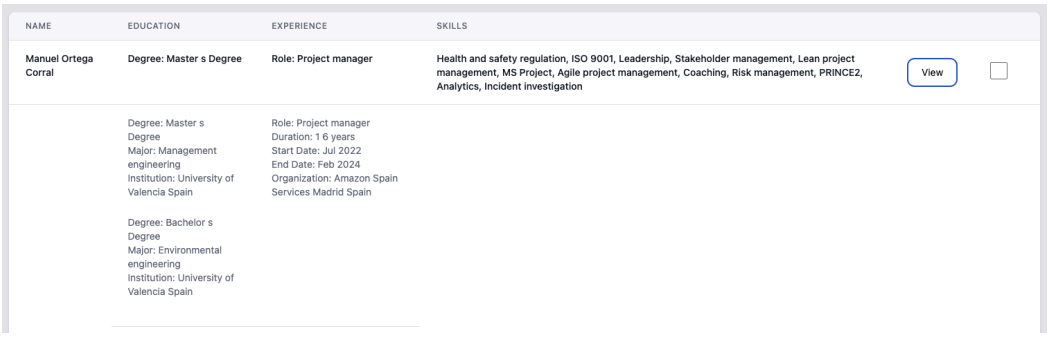


Display Interaction Annotate UI



In the option for displaying a query-ranking pair the UI will show in the top of the page the query, and in the bottom of the page the list of documents. The user can interact with the UI in the following ways:

View Button: the user can expand the table to view more information about the document. When the user clicks on another View Button, the current expand will close. Only, one expand can be open at a time.



Check Box: the user can select top-k documents that are the most relevant given the displayed query. If the predefined minimum and maximum of selected documents is not met, the UI will display an alert to the user.

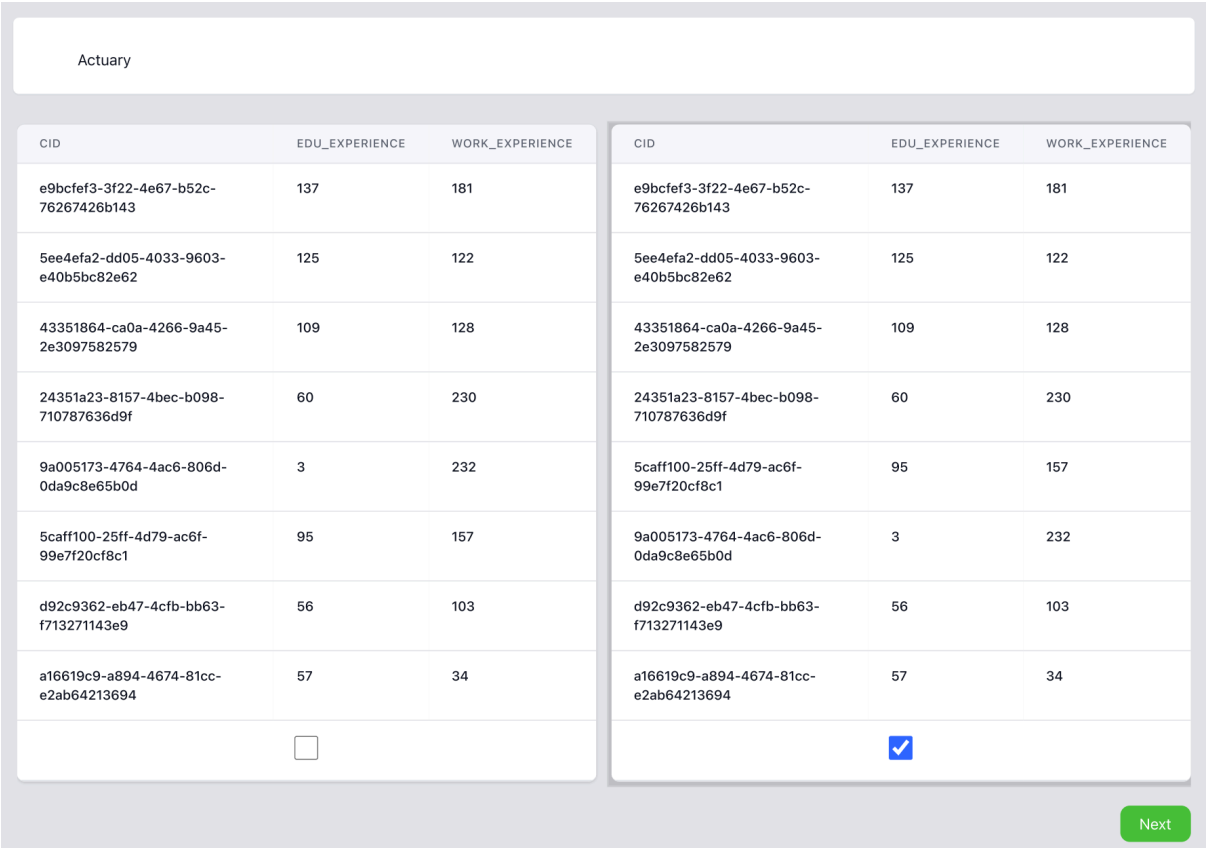
Next Button: the user can navigate to the next query using the Next Button.

For each user, the following interactions can be collected:

- **Clicks:** number of clicks for each document on the View Button
- **Timestamps:** for each click the timestamp of the click to open the expand and the timestamp of closing the expand are collected.
- **Top-k Documents:** the list of the selected documents in the order in which the user checked the box
- **Heatmap:** of the user mouse movements generated using hotjar

Display Ranking Comparison UI

Ranking Compare Annotate UI



In the option for displaying a ranking comparison, the UI will show in the top of the page the query, and below the two rankings to be compared.

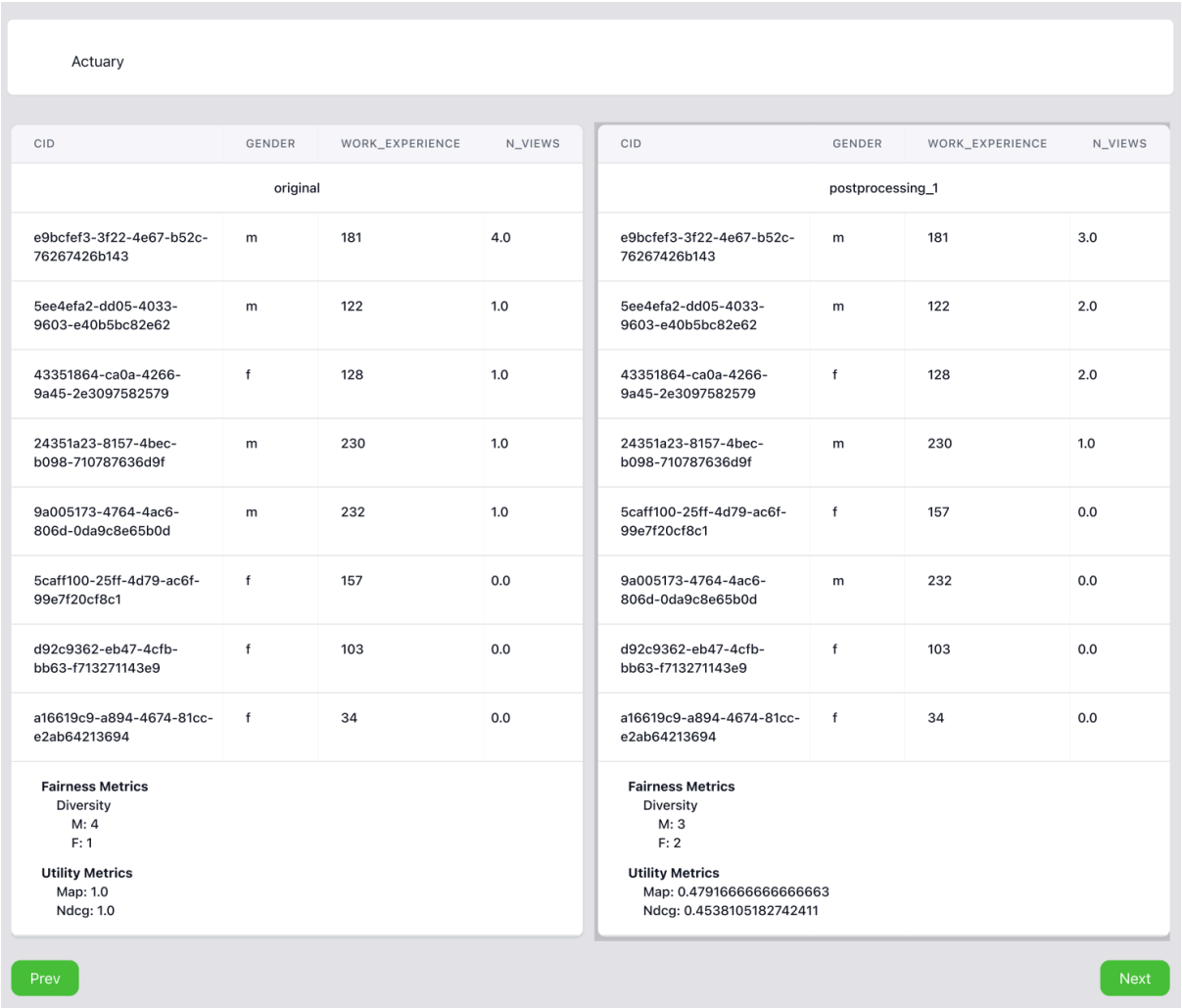
The user can interact with the UI in the following ways:

- **Check Box:** depending on the task, the user can choose which ranking is more suitable for the displayed query.
- **Next Button:** the user can navigate to the next query using the Next Button.

For each user, the following interactions can be collected:

- **Best Ranking:** the chosen ranking by the user to be more suitable for the displayed query.

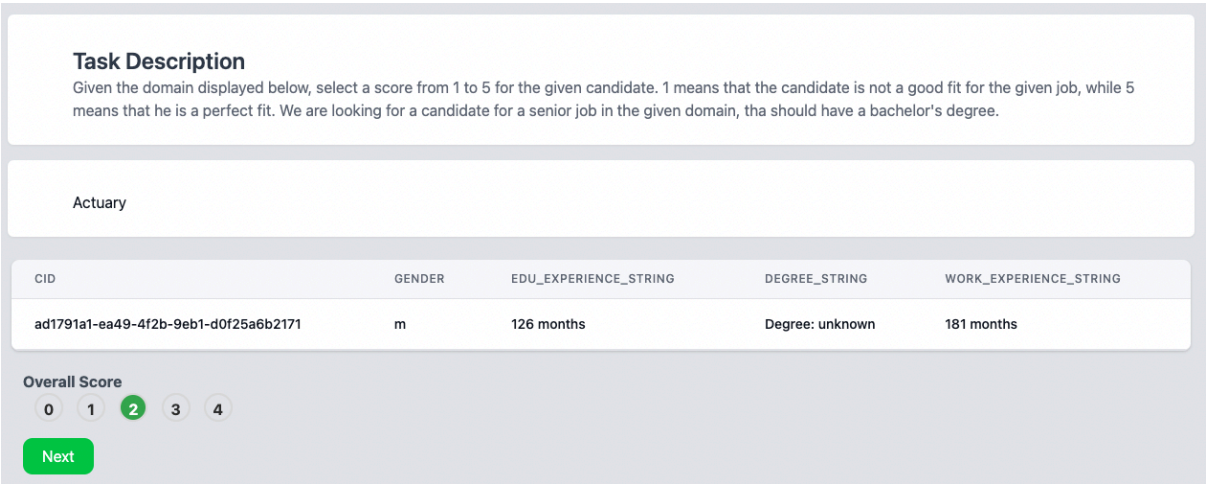
Ranking Compare Visualise UI



On top of this, this UI can be used by the researcher to compare rankings produced using different ranking methods or evaluate the impact of a fairness intervention. The UI has the option to display metrics for each ranking, as shown in the bottom of the UI. Moreover, if interaction data is available, the UI can display the average of the interactions collected by the users. The navigation between several ranking comparison can be performed using the **Prev/Next Button**.

The utility metrics that can be displayed are computed using the Pytrec eval library[4] on the displayed ranking. The ready to use fairness metrics that the tool offers are: selection parity, parity of exposer, and IGF (in group fairness).

Score Annotate UI



In the option for annotating a document by giving it a score, the UI will show in the top of the page the query, and in the bottom of the page document. The user can interact with the UI in the following ways:

- **Score Bar:** the user can give a score to the document by clicking on the numbers in the score bar.
- **Next Button:** the user can navigate to the next query using the Next Button.

For each user, the following interactions can be collected:

- **Score:** the score the user things the document should receive given the displayed query and the task description.

Displaying a new Dataset

To use the UI with a new dataset, the following steps should be followed:

1. Under ./datasets, create a new folder called `<data_name>`/data and save the dataset there. Next, under ./datasets create a folder called experiments in which you can define the query-documents pairs to be displayed when running the UI. More details can be found in the following section: Experiment File.

2. Under ./src/data_readers create a python file `data_reader_<data_name>.py`. Inside create a class `DataReader<Data_name>`.

```
class DataReader<Data_name>(DataReader):

   def __init__(self, configs):
       super(DataReader<Data_name>,
self).__init__(configs=configs)

   def transform_data(self):
      # read the dataset file from self.data_path
            . . .
```

```
# transform the dataset into two dataframes
dataset_queries = pd.Dataframe(columns=['title', 'text'])
dataset_documents = pd.Dataframe(columns=['docID',
'query', 'score', 'group', ...])
      . . .

# if needed split the dataframe for documents into
data_train and data_test

from sklearn.model_selection import train_test_split

data_train, data_test =
train_test_split(dataset_documents, test_size=0.2)
      . . .
# if not needed return None for data_train
data_test = dataset_documents
data_train = None

return dataset_queries, data_train, data_test
```

The `DataReader<Data_name>` should implement the `transform_data` method that should return the following:

- `dataframe_queries` – dataframe describing the queries. It should have the following mandatory columns:
  - `'title'` – this is the title of the query
  - `'text'` – this will be displayed in the UI

  For example, if the query is a job description, the `'title'` will be the job title and the `'text'` will be the job description, which includes the job title. If the query is represented by a word/s set both `'title'` and `'text'` to the same value.
- `data_test` – dataframe describing the documents to be displayed.
- `data_train` – dataframe describing the documents to be used when training a ranker or a fairness intervention.

The formatted dataset will be saved under ./datasets/`<data_name>`/format_data

3. Under ./configs create json files following the naming conventions:
   - `config_shortlist_<data_name>.json` → to run the Interaction Annotate UI
   - `config_compare_<data_name>.json` → to run the Ranking Compare Visualise UI
   - `config_compare_annotate_<data_name>.json` → to run the Ranking Compare Annotate UI

   in which one should define the configuration to run the tool with. More details about the configuration file can be found in the following section: Configuration File.

Configuration File

Under ./configs create a josn file which represents the configurations with which to run the tool with the desired dataset.

"data_reader_class" – Configuration for the data.

"name" – Specify the dataset name you want to run the tool with.

"score" – Column to be used for ranking the documents.

"group" – Column to be used by the fairness interventions representing the sensitive information of the documents (e.g. gender).

"query" – Column representing the query to which the document is linked to. This should contain the same values found in dataframe_queries under 'title'.

"docID" – Column representing the document IDs.

"ui_display_config" – Configuration for what to display in the UI.

"display_fields" – List of columns to display.

"task_description" – guidelines of how the user should perform the assessment.

"exit_survey" – List of questions to be asked in the exit survey.

"question" – The question to be displayed.

"field" – The field name that will be used to store the value in the database.

"options" – If a drop-down is wanted, list of options to be shown in the drop-down, otherwise set the value to "text" to display a free text input.

"mandatory" – Boolean to be set if the field is mandatory to be completed by the user or not.

Specific fields to declare for the Interaction Annotation UI:

"shortlist_button" – Boolean value for whether to display the shortlist button.

"shortlist_select" – List representing the range [min, max] of top-k items the user has to select as being the most relevant for the given query.

"view_button" – Boolean value for whether to display or not the view button.

"view_fields" – List of columns to display when clicking on the view button.

Specific fields to declare for the Score Annotation UI:

"score_range" – range for the score bar (e.g [1,5]), with the first value being the start of the range and the second value being the last value from the range. The score bar will have 5 circles starting with values from 1 to 5.

Specific fields to declare for the Ranking Comparision UI:

"annotate" – Boolean to be set to true if wanting to run the Ranking Compare Annotate UI, or set to false if wanting to run Ranking Compare Visualise UI.

"avg_interaction" – Configuration for the interactions to be displayed.

"experiment_id" – Experiment ID out of which we want to visualise the collected interactions.

"interaction" – Interaction type to be displayed (e.g. "n_views" - clicks on the view button).

"display_metrics" – Configuration for metrics to be displayed.

"top_k" – Compute the metric at @k.

"utility_metrics" – List of utility metrics to be computed and displayed.

"fairness_metrics" – List of fairness metrics to be computed and displayed.

"train_ranker_config" – Configuration for applying a ranker on the original ranking. If set to null, no ranker will be applied to the original ranking, otherwise, the documents will be ranked based on the predicted score by the defined ranker.

"name" – Name of the ranker to be applied.

"model_path" – Path to load a pre-trained model. If no pre-trained model is available, a new model will be saved at the specified path.

"settings" – List of configurations to run the ranker (specify your own fields used by the ranker class). The ranker class should be initialised with the settings specified.

"ranking_type" – Specify under which tag the ranking of the documents should be saved in the database.

- ranker_<name> - for running a ranking model

"pre/in/post_processing_config" – Configurations to apply a pre-processing fairness intervention on the data. If set to null, no ranker will be applied to the original ranking, otherwise, the documents will be ranked based on the predicted score by the defined ranker.

"name" – Name of the method to be applied.

"model_path" – Path to load a pre-trained model. If no pre-trained model is available, a new model will be saved at the specified path.

"settings" – List of configurations to run the method (specify your own fields used by the fairness intervention class). The fairness intervention class should be initialised with the settings specified.

"ranking_type" – Specify under which tag the ranking of the documents should be saved in the database. It is important to keep the following naming convention for the ranking type:

- preprocessing_<name> - for running a preprocessing fairness intervention.
- postprocessing_<name> - for running a postprocessing fairness intervention.
- inprocessing_<name> - for running an in-processing fairness intervention.

Experiment File

Experiment File for the Interaction Annotation UI

"exp_id" - id of the experiment.
"description" - description of the experiment to run.
"tasks" - list of assessments.

- "query_title" - the title of the query to be displayed.
- "setting" - an extra requirement that the user should take into account when doing the assessment.
- "ranking_type" - the order in which to display the ranking of the documents.
  - "ranking_type" could have the following values:
    - "original" - sorted by the "score".
    - "ranker_<name>" - sorted by the predicted score of a pre-trained model.
    - "pre/in/postprocessing_<name>" - displays the ranking generated by the fairness intervention.

    The "ranking_type" should match the "ranking_type" defined in the configuration file.

Experiment File for the Score Annotation UI

"exp_id" - id of the experiment.
"description" - description of the experiment to run.
"tasks" - list of assessments.

- "query_title" - the title of the query to be displayed.
- "setting" - an extra requirement that the user should take into account when doing the assessment.
- "index" - the index of the item to be annotated.

Experiment File for the Ranking Comparison UI

"exp_id" - id of the experiment.
"description" - description of the experiment to run.
"tasks" - list of assessments.

- "query_title" - the title of the query to be displayed.
- "ranking_type_1" - the order in which to display the ranking of the documents that is displayed on the left side.
- "ranking_type_2" - the order in which to display the ranking of the documents that is displayed on the right side.
  - "ranking_type" could have the following values:
    - "original" - sorted by the "score".
    - "ranker_<name>" - sorted by the predicted score of a pre-trained model.

"pre/in/postprocessing_<name>" - displays the ranking generated by the fairness intervention.

The "ranking_type" should match the "ranking_type" defined in the configuration file.

Database

The tool works with a MongoDB database. The diagram below describes the collections present in the database. For each dataset, a new database is created automatically together with the required collections. The name of the created dataset will be the name specified in the configuration file as the "data_reader_class". Below the use of each collection is described:

- documents - stores the dataframe describing the data to be displayed, meaning the data stored in data_test. On top of the fields displayed in the diagram, the tool automatically adds the rest of the columns present in data_test. The preprocessing field stores a list with preprocessed values given the configurations of the ranking_type.


```
_id: ObjectId('65aec8ea0d22641b6cd1e125')
qualification: 0.2688888728417616
gender: "m"
work_experience: 230
edu_experience: 60
hits: 1926
originalQualification: 558540
cid: "24351a23-8157-4bec-b098-710787636d9f"
title: "Actuary"
preprocessing: Array (1)
  0: Object
    ranking_type: "preprocessing_1"
    edu_experience_fair: 19.956521739130395
    work_experience_fair: 230
    hits_fair: 1926
    qualification_fair: 0.2448788728417616
    _cls: "PreDocRepr"
```


- queries - stores the dataframe describing the queries, meaning the data stored in dataframe_queries.


```
_id: ObjectId('65aec8ea0d22641b6cd1e124')
title: "Actuary"
text: "Actuary"
```

```
_id: ObjectId('65b038abf3c7907e51baf8cf')
title: "project_manager"
text: "Project Manager
      Education Background
      Degree: bachelors degree
      Major: civil engineering or environmental engineering or management

      Professional Experience
      Role: project manager
      Duration: 2 years
```


- dataset - stores the query-rankings pairs. For each pair, it stores the id of the query and a list of rankings. Each ranking stores an ordered list of the IDs of the documents linked to this query. The lists are ordered based on the ranking_type that was defined in the configuration file.

```
_id: ObjectId('65aec8ec0d22641b6cd1e308')
query: "65aec8ea0d22641b6cd1e109"
rankings: Array (6)
  0: Object
    docs: Array (8)
    ranking_type: "original"
  1: Object
    docs: Array (8)
    ranking_type: "preprocessing_1"
  2: Object
    docs: Array (8)
    ranking_type: "ranker_1:qualification__qualification"
  3: Object
    docs: Array (8)
    ranking_type: "ranker_1:qualification_fair__qualification_fair"
  4: Object
    docs: Array (8)
    ranking_type: "postprocessing_1"
  5: Object
    docs: Array (8)
    ranking_type: "postprocessing_2"
```

- `experiments` - stores the data defined in the experiment file, including the list of tasks to be displayed when running the tool. On the left there is an example belonging to an experiment file used to run the Interaction Annotation UI, while on the right an example belonging to an experiment file used to run the Ranking Compare UI. The corresponding tasks can be seen under the next point: on the left for the Interaction Annotation UI and on the right for Ranking Compare UI.

```
_id: ObjectId('65aec8ed0d22641b6cd1e349')
_exp_id: "4"
_description: "test experiment"
tasks: Array (4)
  0: "65aec8ed0d22641b6cd1e345"
  1: "65aec8ed0d22641b6cd1e346"
```

```
_id: ObjectId('65aec8ed0d22641b6cd1e344')
_exp_id: "3"
_description: "compare rankings side by side"
tasks: Array (3)
  0: "65aec8ed0d22641b6cd1e341"
  1: "65aec8ed0d22641b6cd1e342"
```

- `tasks/tasks_compare/tasks_score` - stores the data described in the experiment file for each task, together with the ID of the query-ranking pair stored in the dataset collection. On the left, there is an example belonging to collection `tasks`, while in the middle an example belonging to collection `tasks_compare`, and on the right an example belonging to collection `tasks_score`.

```
_id: ObjectId('65aec8ed0d22641b6cd1e345')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type: "postprocessing_1"
query_title: "Actuary"
```

```
_id: ObjectId('65aec8ed0d22641b6cd1e346')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type: "original"
query_title: "Actuary"
```

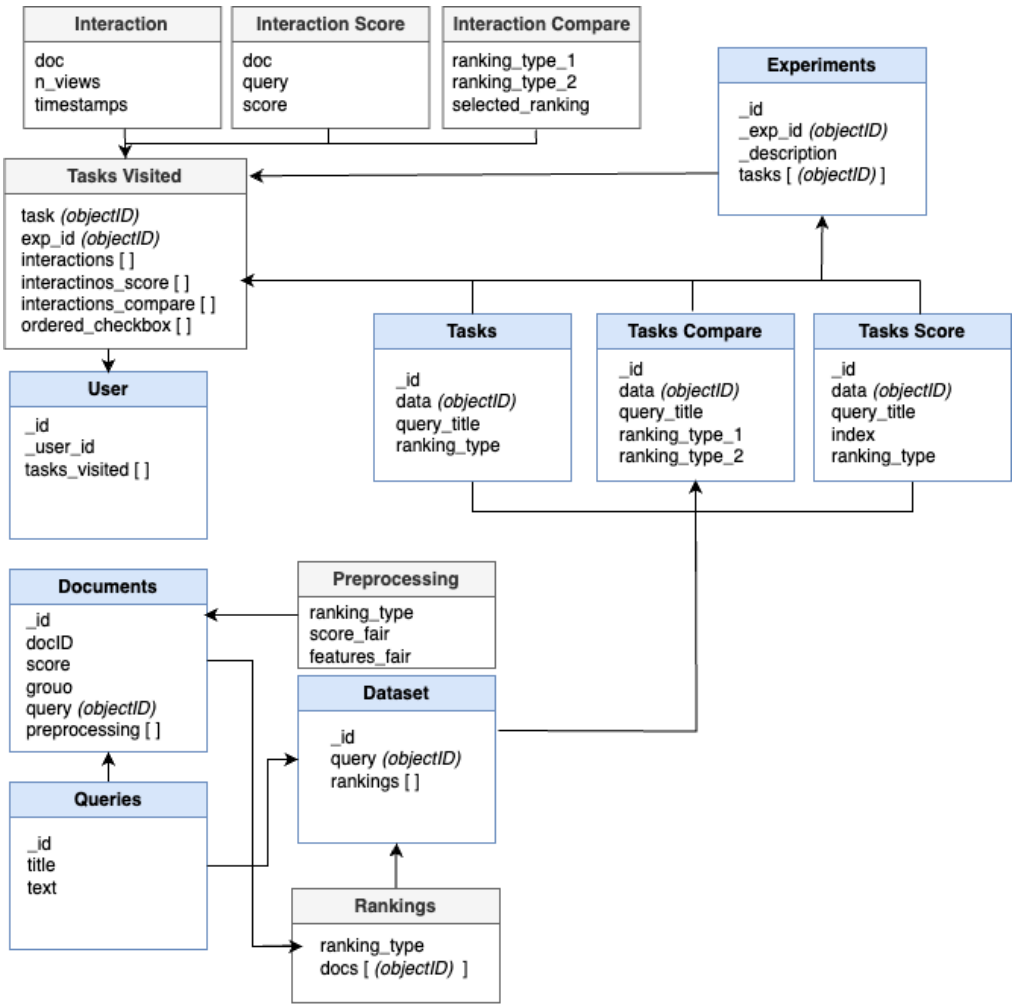
```
_id: ObjectId('65aec8ed0d22641b6cd1e341')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type_1: "original"
ranking_type_2: "postprocessing_1"
query_title: "Actuary"
```

```
_id: ObjectId('65aec8ed0d22641b6cd1e342')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type_1: "original"
ranking_type_2: "ranker_1:qualification__qualification"
query_title: "Actuary"
```

```
_id: ObjectId('65c0e3fa193007bae2a8c54b')
data: "65c0df6ad4bfa872c5efdff9"
index: "1"
setting: ""
query_title: "Actuary"
ranking_type: "original"
```

- `user` - stores the ID provided by the user on the first page of the tool and the list of tasks the user interacted with in the tool. As the user interacts with the tool, for each task viewed we store the ID of the tasks together with the list of collected interactions for each document.

```
_id: ObjectId('65aa7c9f40a7cbdf2e56a02e')
_user_id: "1289"
tasks_visited: Array (4)
gender: "Female"
feedback: "Great!"
```



Additional Support

Ranklib Rankers

Ready to Use

The tool offers support for using any model implemented using the ranklib library[3]. It is to be considered that the ranklib library requires java to be installed in the docker container, thus, on the first run it might take longer for the app to start as it first needs to install java. It can be used by defining in the config file the following:

```
"train_ranker_config": {
 "name": "Ranklib",
 "model_path": "./dataset/<data_name>/models/Ranklib", // path to save/load the model

 "settings": [{
     "features": ["feature_1", "feature_2", "feature_3"], // list of features to be considered during training
     "pos_th": 0, // threshold to consider relevant documents
```

```
     "rel_max": 500, // maximum relevance that is assigned based
on the 'score' column
     "ranker": "RankNet", // ranker name
     "ranker_id": 1, // ranker id

     "metric": "NDCG", // metric to evaluate
     "top_k": 10, // evaluate at top-k
     "lr": 0.000001, // learning rate
     "epochs": 20, //number of epochs

     "train_data":["original"] // define the train data,
     "test_data": ["original"] //define the test data,
     "ranking_type": "ranker_1" }]}
```

For more information on how to use ranklib rankers check the ranklib documentation: https://sourceforge.net/p/lemur/wiki/RankLib%20How%20to%20use//.

The `"train_data"` and `"test_data"` can be set to the following values:

- `"original"` - the ground truth is set to be the original score
- `<ranking_type>` - if the ground truth is set to be the score computed using a fairness intervention

The `train` method saves the input files in the format expected by the ranklib library under ./dataset/`<data_name>`/models/Ranklib/`<train_data>_<test_data>`. The model and predictions are saved under ./dataset/`<data_name>`/models/Ranklib/`<train_data>_<test_data>`/ranklib_experiments/`<ranker_name>`.

The output of the `predict` method returns a new dataframe that is in the same format as the original one with an appended column representing the predicted score. The column with the predicted relevance should follow the convention `<train_column>__<test_column>`. For example if the train and test data is set to be "original", the predicted score column should be `"score"__"score"`, where `"score"` is the value defined in the config file under the `"score"` field. If the train and test data are pre-processed by a fairness intervention the predicted score column should be `"score"_fair__"score"_fair`.

The new ranking based on the predicted score is saved in the MonogDB database in the collection `dataset`, in the field rankings. Given the config presented above, the ranking type of saved in the database will be set to `ranker_1:"score"__"score"`. If a preprocessing fairness intervention is applied on the train/test data, the ranking type saved in the database will be set to `ranker_1:preprocessing_1:"score"_fair__"score_fair"`

Add a new ranker

In order to add a new fairness intervention the following steps should be followed:

1. Under ./src/rankers create the following python file: ranker_`<ranker_name>`.py
   Inside the python file create the class that implementes the fairness method:

```
class <ranker_name>Ranker(Ranker):
     def __init__(self, configs, data_configs,
model_path):
          super().__init__(configs,data_configs,
model_path)
```

   `model_path` – path to save/load the model
   `configs` – dictionary of hyperparameters needed to run the ranker. This is defined in the config file as “settings”.
   `data_configs` – dictionary of configs defined for the `data_reader_class`. This is needed to be able to access the required columns by the fairness intervention.

2. Implement the methods defined in the parent class `Ranker`, which can be found in ./src/fairness_interventions/ranker.py.

   `def train_model(self, data_train, data_test, experiment)`
   `data_train` - data to train the model
   `data_test` - data to evaluate the model during training
   `experiment` - tuple containing the <train_ranking_type> and <test_ranking_type> defined in the configuration file.

   The `train` method should save the trained model at the following path:
   `self.model_path/<train_ranking_type>__<test_ranking_type>`.

   `def predict(self, data, experiment)`
   `data` - data to run the model on and generate the predicted ranking
   `experiment` - tuple containing the <train_ranking_type> and <test_ranking_type> defined in the configuration file.

   The `predict` method should load the model from `self.model_path/<train_ranking_type>__<test_ranking_type>` and apply it on the `data`.

   The method should return a new dataframe that is in the same format as data with an appended column representing the predicted score. The column with the predicted score should follow the convention `<train_score_column>__<test_score_column>`.

For example if the `<train_ranking_type>` and `<test_ranking_type>` is set to be “original”, the predicted score column should be `”score”__”score”`. If the train and test data are pre-processed by a fairness intervention the predicted score column should be `”score”_fair__”score”_fair`, where `“score”` is the value defined in the config file under the `“score”` field.

3. If needed any files related to the fairness method can be saved under ./src/rankers/modules/`<ranker_name>`

4. Define the new ranker in ./src/constants in the dictionary containing the rankers.

5. Using the new ranker:

```
"train_ranker_config": {
      "name": "<ranker_name>" // same as the key defined in
the dictionary,
      "model_path":
"./dataset/<data_name>/models/<ranker_name>", // path to
save/load the model
      "settings": [{
            // define any configs needed to run the ranker
            "features": ["feature_1", "feature_2",
"feature_3"], // list of features to be considered during
training

            "train_data":["original"], // define the train
data
            "test_data": ["original"], //define the test data
            "ranking_type": "ranker_1"
            }]
      }
```

Fairness Interventions

Ready to Use

The tool supports applying:

- post-processing fairness intervention: FA*IR[2]. It can be used by defining in the config file the following:

```
"post_processing_config": {
 "name": "FA*IR",
 “model_path”: “”, //empty as there is no model to save
 "settings": [{
      "k": 10,
      "p": 0.7,
      "alpha": 0.1
      "ranking_type": "postprocessing_1"}]
 }
```

- pre-processing fairness intervention: CIF-Rank[1]. It can be used by defining in the config file the following:

```
"pre_processing_config": {
      "name": "CIFRank",
      "model_path": "./dataset/<data_name>/models/CIFRank", // path to
save/load the model

      "settings": [{
             "pos_th": 0, //threshold to consider positive documents
             "control": "group_value", //control group, the group
            towards which we convert all the data in a counterfactual
            world
             "features": ["field_1", "field_2", "field_3"]//list of
      features (columns from the dataframe) to be considered as
      mediators
}]
}
```

It is to be considered that the fairness interventions might require additional libraries to be installed in the docker container, thus, on the first run it might take longer for the app to start as it first needs to install the required libraries.

Add a new fairness method

In order to add a new fairness intervention the following steps should be followed:

1. Under ./src/fairnes_interventions create the following python file: fairness_method_`<method_name>`.py

   Inside the python file create the class that implementes the fairness method:

   ```
   class <method_name>(FairnessMethod):
         def __init__(self, configs, data_configs, model_path):
               super().__init__(configs, data_configs, model_path)
   ```

   `model_path` – path to save/load the model

   `configs` – dictionary of hyperparameters needed to run the fairness method. This is defined in the config file as `"settings"`.

   `data_configs` – dictionary of configs defined for the `data_reader_class`. This is needed to be able to access the required columns by the fairness intervention.

2. Implement the methods defined in the parent class FairnessMethod, which can be found in ./src/fairness_interventions/fairness_method.py.

   ```
   def train_model(self, data_train)
      data_train - data to train the model
   ```

   The `train_model` method should save the fairness intervention method to `self.model_path`.

   ```
   def generate_fair_data(self, data):
   ```

```
data - data to be used to generate the fair data
```

The `generate_fair_data` method should return a new dataframe that has the same columns as data, to which the fair columns are added following the name convention `<column_name>_fair`.

For example, when running a pre-processing fairness intervention like CIF-Rank[1], both the features defined under "`features`" and the score define under "`score`" will be changed. The fair columns should be "`score`"`_fair`, where "`score`" is the value defined in the config file under the "`score`" field, and same for all the features defined under "`features`". When running a post-processing intervention like FA*IR[2], which re-ranks the candidates, then the fair column should represent the new ranking, thus, the name of the returned column is `rank_fair`.

3. If needed any files related to the fairness method can be saved under ./src/fairness_interventions/modules/`<method_name>`

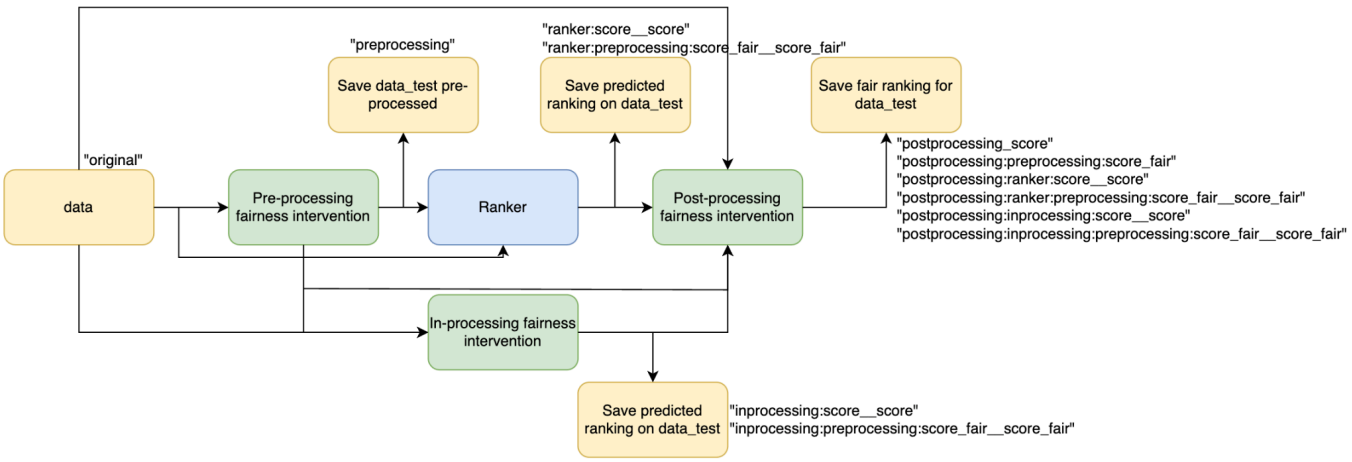
4. Define the new fairness method in ./src/constants in the dictionary containing fairness methods.

5. Using the new fairness method:

```
      "name": "<method_name>" // same as the key defined in the dictionary,
      "model_path": "./dataset/<data_name>/models/<method_name>", // path to
save/load the model
      "settings": [{
            // define any configs needed to run the fairness method
            "train_data":["original"], // define the train data for an
in-processing method
            "test_data": ["original"], //define the test data for an
in-processing method
            "ranking_type": // define the ranking type as according to the
naming convention specified in the Configuration File section
            }]
```

The figure below describes how the fairness interventions can be applied on the data and how their outputs can be used to train a ranking model, if the configuration file has enabled the use of both the fairness interventions and the use of a ranking model. The **Pre-processing fairness intervention** is applied on the data and saved in the database. As mentioned before, the fair values are saved in the documents collection, while the fair ranking is saved in the dataset collection. The **Ranker**, the **In-processing fairness intervention** and the **Post-processing fairness intervention** can be trained/tested on either the pre-processed data or on the original data. This can be set in the configuration file using the field `"train_data"` and `"test_data"`. The predicted rankings are saved in the database in the collection `dataset`. The post-processing method can be applied on any kind of ranking, including the ranking based on the original "`score`" column, the ranking

based on the output of the fairness interventions or of the ranker. The predicted rankings are saved in the database under the `dataset` collection.



Given the following example of a configuration for the post-processing intervention using FA*IR[2]:

```
"post_processing_config": {
 "name": "FA*IR",
 "model_path": "",
 "settings": [{
      "k": 10,
      "p": 0.7,
      "alpha": 0.1,

      "test_data": ["original", "preprocessing_1",
      "ranker_1:preprocessing_1:qualification_fair__qualification_fair",
      "ranker_1:qualification__qualification"],

      "ranking_type": "postprocessing_1"}]
```

The post-processing fairness intervention will be applied on the following rankings:

- `"original"` - the ranking produced by column “score”
- `"preprocessing_1"` - the ranking produced by the pre-processed score column (`qualification_fair`)
- `"ranker_1:preprocessing_1:qualification_fair__qualification_fair"` – the ranking produced by `ranker_1` which was trained on the `qualification_fair` column produced by the pre-processing method defined as `preprocessing_1`.
- `"ranker_1:qualification__qualification"` – the ranking produced by `ranker_1` which was trained on the `qualification` column.

The ranking_type of the ranking saved in the database are defined as it follows for the above use cases:

- `postprocessing_1:qualification`
- `postprocessing_1:qualification_fair`

- `postprocessing_1:ranker_1:preprocessing_1:qualification_fair__qualification_fair`
- `postprocessing_1:ranker_1:qualification__qualification`

Run the tool

Requirements:

Install Docker by following the steps presented here: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04 Select the operating system of the device you want to run the app on and proceed with the steps indicated.

Install Docker Desktop: https://www.docker.com/products/docker-desktop/

Install MongoDB Compass: https://www.mongodb.com/products/tools/compass to view the dataset created and its collections. The connection should be set as mongodb://<IP>:27017. <IP> should be set to IP address for of the machine where the docker container is running.

Run

Add the paths to the configuration files needed to run the app in: **apps.docker.sh.**
To run the app, run the following script **run_apps.sh**

Access

To access the Interaction Annotation UI go to the following link:
http://localhost:5000/start_ranking/<exp_id>/

To access the Ranking Comparison Visualise UI go to the following link:
http://localhost:5001/start_compare/<exp_id>/

To access the Ranking Comparison Annotate UI go to the following link:
http://localhost:5002/start_compare_annotate/<exp_id>/

To access the Score Annotate UI go to the following link:
http://localhost:5003/start_annotate/<exp_id>/

Where `<exp_id>` is the id of the experiment that you want to run, which was defined in the experiment file and stored in the collection experiment.

Tutorial

Example on the XING dataset.

1. Download the XING dataset from: https://github.com/MilkaLichtblau/xing_dataset
2. Create the following folder ./datasets/xing/data and save the dataset there

3. The data reader implemented for this dataset can be found in the following python file ./src/data_readers/data_reader_xing.py.

4. The experiment files can be found at the following locations:
   ./datasets/xing/experiments/experiment_shortlist.json → to run the Interaction Annotation UI
   ./datasets/xing/experiments/experiment_compare.json → to run the Ranking Comparison UI
   ./datasets/xing/experiments/experiment_annotate_score.json → to run the Score Annotate UI

5. The config files can be found under ./configs/xing_tutorial/

- `config_create_db_xing.json` → configuration used to add data in the database the data

  The configuration files specifies that a pre-processing fairness intervention and a post-processing fairness intervention is applied on the data. It also specifies to train a ranker on the data and store its prediction in the database. An example of how the data is stored in the database can be seen below.

  On the left it can be seen that for each document and for each fairness intervention/ranker the predictions are saved in the database. On the right it can be seen that the ordering of all defined configuration is stored in the `dataset` collection.

- `config_shortlist_xing.json` → to run the Interaction Annotate UI

  This configuration file should create the following UI:

  In the configuration file, the "`shortlist_button`" is enabled together with the "`view_button`". The "`shortlist_select`" field defines what is the minimum and the maximum items that the user needs to select as being the top most relevant items given the query. Under the "display_fields" you can define which fields to be shown first to the user, while under the "`view_fields`" you can define which fields to be shown when clicking on the "View" button. The "`exit_survey`" field defines what questions the exit survey should contain at the end of the assessments.

- `config_compare_annotate_xing.json` → to run the Ranking Compare Annotate UI

  Similarly, one can define which fields to be displayed and what the exit survey should contain. It is important to set "`annotate`" to be true, otherwise, no user ID will be

collected and no checkboxes to select the most suitable ranking given the query and the requirements will not be present.

- `config_compare_xing.json` → to run the Ranking Compare Visualise UI

  Similarly, one can define which fields to be displayed. It is important to set "`annotate`" to false. In the configuration file one can select which metrics to be displayed for each ranking and which interaction data.

- `config_annotate_score_xing.json` → to run the Annotate Score UI

  Similarly, one can define which fields to be displayed. It is important to set the range for the scores to be displayed using the field `"score_range"`.

6. Requirements:
   Install Docker by following the steps presented here: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04 Select the operating system of the device you want to run the app on and proceed with the steps indicated.

   Install Docker Desktop: https://www.docker.com/products/docker-desktop/

   Install MongoDB Compass: https://www.mongodb.com/products/tools/compass to view the dataset created and its collections. The connection should be set as mongodb://<IP>:27017. <IP> should be set to IP address for of the machine where the docker container is running.

7. Run the following script **run_apps.sh**

8. To access the Interaction Annotation UI go to the following link:
   http://localhost:5000/start_ranking/4

   To access the Ranking Comparison Visualise UI go to the following link:
   http://localhost:5001/start_compare/3

   To access the Ranking Comparison Annotate UI go to the following link:
   http://localhost:5002/start_compare_annotate/3

   To access the Score Annotate UI go to the following link:
   http://localhost:5003/start_annotate/5

References

[1] Ke Yang, Joshua R. Loftus, and Julia Stoyanovich. 2021. Causal intersectionality and fair ranking. In Symposium on Foundations of Responsible Computing (FORC).

[2] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. 2017. Fa* ir: A fair top-k ranking algorithm. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. ACM, 1569–1578.

[3] Dang, V. "The Lemur Project-Wiki-RankLib." Lemur Project,[Online]. Available: http://sourceforge.net/p/lemur/wiki/RankLib.

[4] Van Gysel, C., & de Rijke, M. (2018, June). Pytrec_eval: An extremely fast python interface to trec_eval. In The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (pp. 873-876).