Implementation documentation

Outlr.

A web application for analyzing subspace-outliers on high dimensional datasets

Bennet Alexander Hörmann, Salomo Hummel,
Simeon Hendrik Schrape, Erik Bowen Wu, Udo Ian Zucker

10.02.2023

Contents

1 Introduction

This document intends to provide a basic overview of the implementation phase. In the phase, which this document covers, we implemented our implementation plan from the design phase, which was based on the software requirements specification we created in the first phase.

The document first explains our workflow in this phase, then covers our planned procedure of implementing "Outlr." for which we assigned each package a desired starting and ending date. This is followed by a Gantt chart displaying the actual procedure by displaying the actual starting and ending dates of each package implementation. After that, data related to our implementation is presented.

Then the product functions and requirements defined in the software requirements specification are covered. For the product functions as well as for the requirements this document lists, which of these were implemented and which were not - looking at mandatory and optional product functions/ requirements separately. For the product functions/requirements, which were not implemented, the reason is explained.

Lastly, after listing the libraries/frameworks used as well as giving an overview of unit tests the faced problems and deviations from the implementations are covered.

2 Workflow

Project management

Our project is divided into tasks (see 3). Most tasks are the implementation of a package or sometimes individual classes. The goal is that all tasks need a similar amount of time to complete.

The dev branch is the main branch of the project and it is protected. Each task is implemented on a separate feature branch that gets merged into the dev branch using a merge request.

Each task goes through the following phases:

1. Not started: Task is not currently worked on
2. In progress: The assigned team member is currently working on the task on a feature branch. Once the task is ready a merge request is created.
3. Review: The task is ready, but needs to be reviewed by the assigned reviewer. The following requirements must be fulfilled before the merge request is accepted:
    - The implemented feature should work as defined in the design phase
    - Type annotations should be used in method signatures
    - Documentation in the form of docstrings or TSDoc should be present on all public classes, attributes, and methods
    - Basic unit tests should be implemented and all unit tests must pass
    - Linter tests must pass
    - There should be no merge conflicts

    If a task does not fulfill these requirements it goes back to the previous phase.
4. Done: The task is completed

We use kanban boards in Notion to manage this workflow.

Continuous integration

Our git repository is managed on GitLab. Continuous integration automates some parts of the review process. Once a commit is pushed to a feature branch of the repository the app is built, the database is set up, and unit tests, as well as a lint test, is run automatically. If any of these steps fail the merge request of the feature branch cannot be merged to the dev branch.

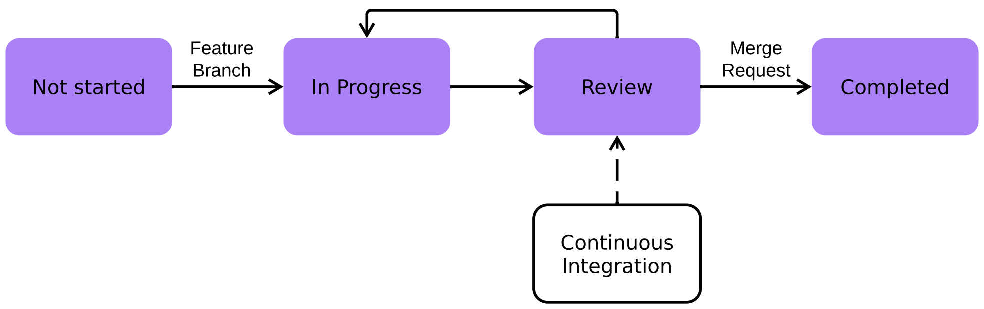

Figure 1: Task phases

3 Planned and actual timeline

3.1 Planned timeline

Back-end

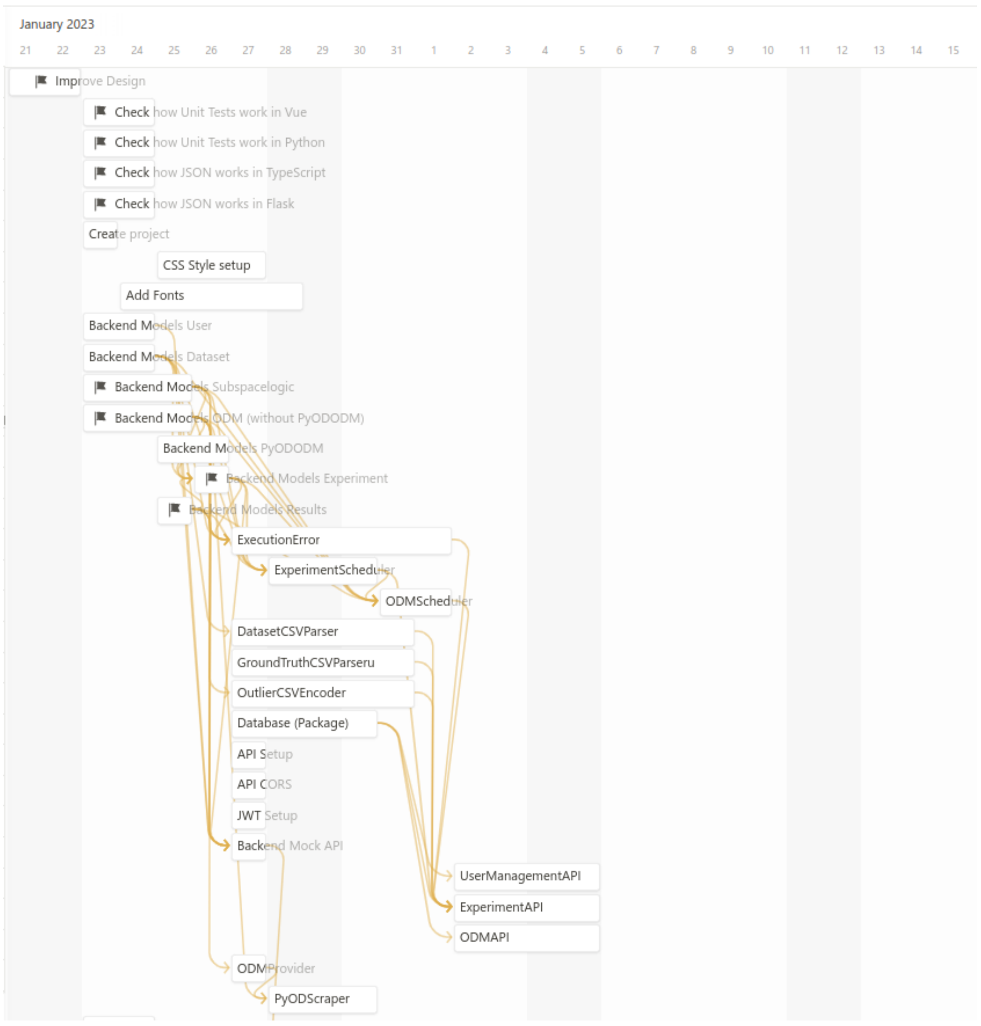

Figure 2: Planned timeline of the back-end tasks

Front-end

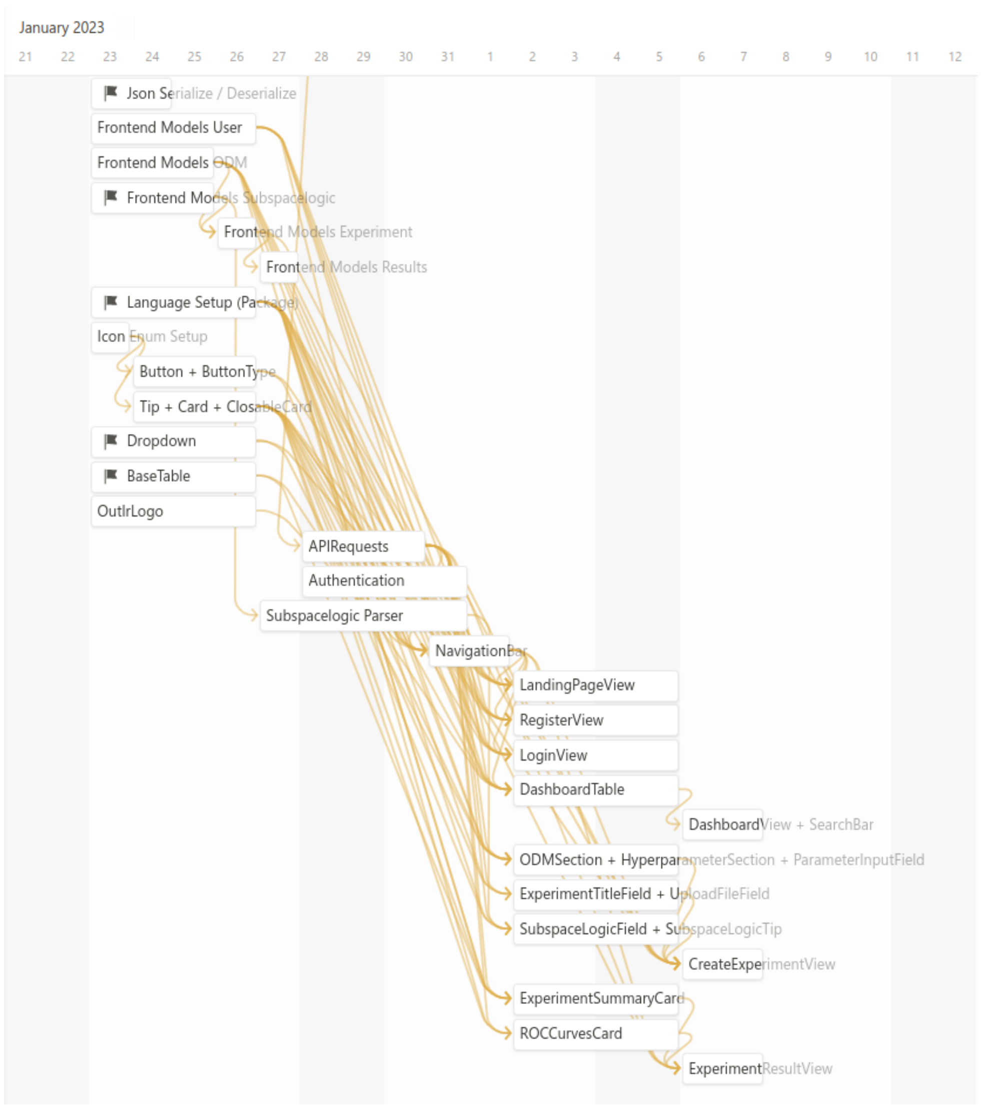

Figure 3: Planned timeline of the front-end tasks

3.2 Actual timeline

Back-end

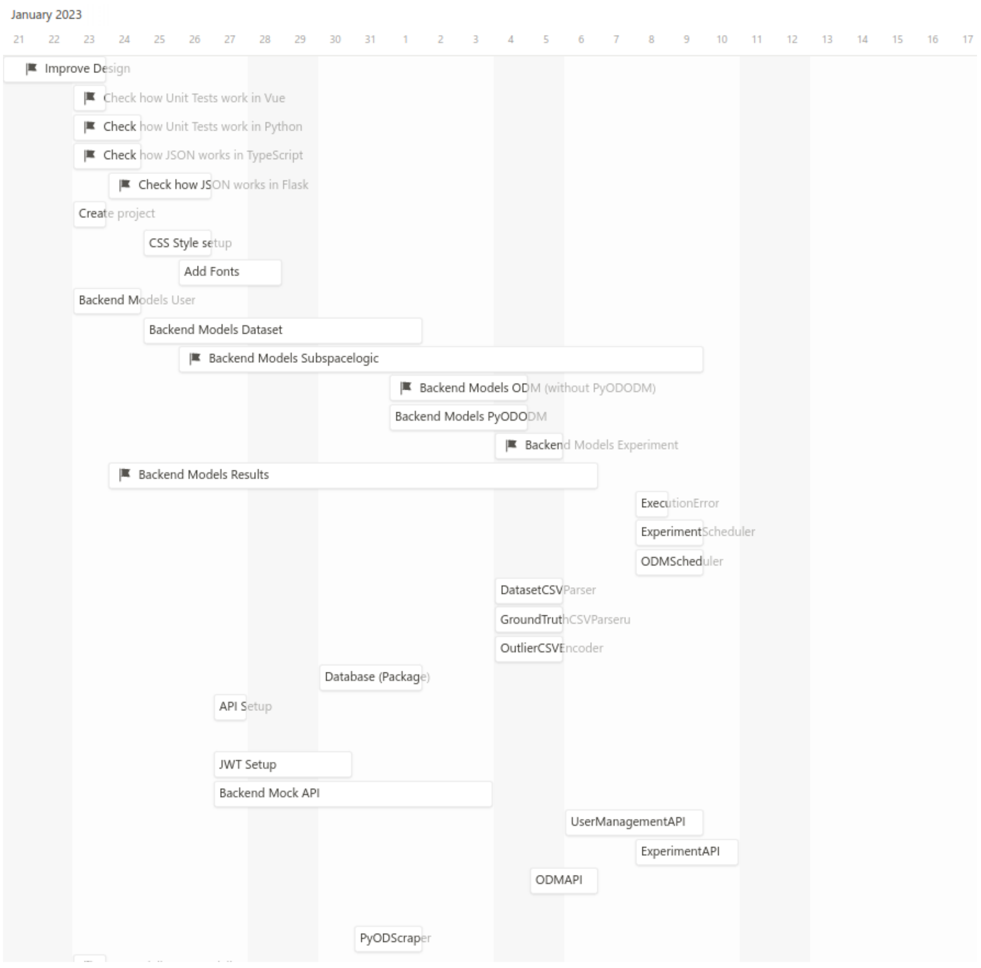

Figure 4: Actual timeline of the back-end tasks

Front-end

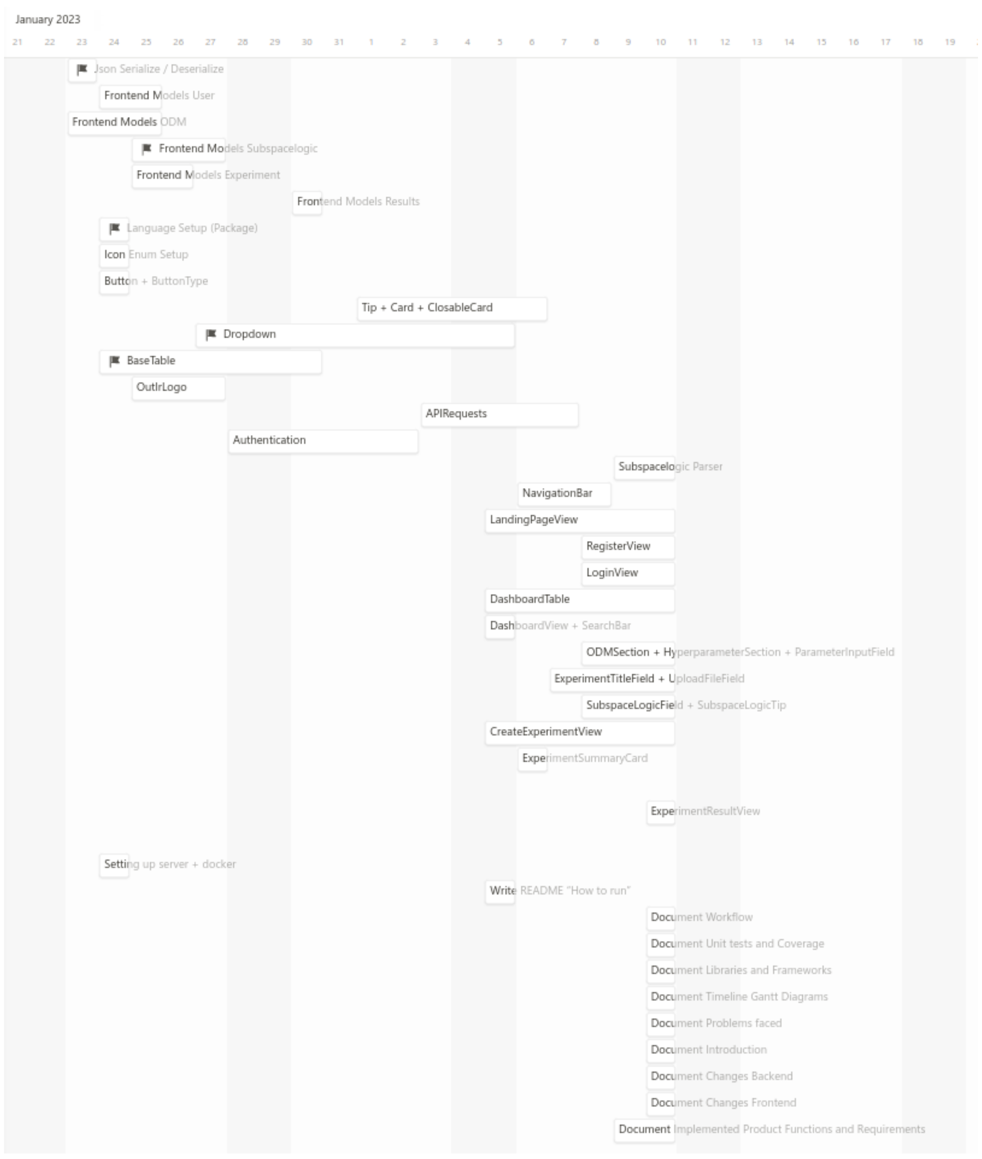

Figure 5: Actual timeline of the front-end tasks

4 Statistics related to our implementation

Lines of code in total: ≥ 6400

5 Overview of product functions, requirements defined in the Pflichtenheft

Product Functions

Mandatory

Looking back at the software requirements specification we met our goal of providing all the defined mandatory product functions:

- **FM1:** User Management: allowing a user to create an account, log in and logout
- **FM2:** Dashboard: providing an overview of all experiments
- **FM3:** Create Experiment: allowing a user to create an experiment, that is select an ODM, the hyperparameters for it and specify subspace logic for it.
- **FM4:** Run Experiment: run the experiment the user created
- **FM5:** Experiment Results: providing the user an overview of his run experiment

Optional

As for the optional product functions we implemented:

- **FO1**: providing the user with an appealing landing page

However, we did not implement the other optional product functions, which were:

- **FO2**: providing the user with an about us page giving users information about us
- **FO3** providing the user with a page, which compares the experiments the user selected before

The reason for not implementing these optional product functions was that implementing the other mandatory pages was time-consuming, not leaving us with enough time to work on the listed optional product functions.

Requirements

Mandatory

As for the mandatory requirements we implemented all the defined mandatory requirements, which were:

- **RM1:** Allowing users to create an account
- **RM2:** Allowing users to log into their created accounts
- **RM3:** Allowing users to log out of their account
- **RM4:** Providing the user with a dashboard giving an overview of all experiments
- **RM5:** Allowing the user to click on the experiments in the dashboard, redirecting the user to the experiment's result
- **RM6:** Allowing users to name their experiment
- **RM7:** Allowing users to upload a dataset (.csv) for an experiment
- **RM8:** Allowing users to upload ground-truth to the provided dataset for an experiment
- **RM9:** Allowing users to select subspaces from the dataset, which shall be processed by the subspace logic
- **RM10:** Allowing users to specify subspace logic using logical or and logical and
- **RM11:** Allowing users to customize the hyperparameters of the ODM they selected
- **RM12:** Allowing users to select an ODM from a significant subset of the ODMs provided by PyOD
- **RM13:** Allowing users to a created experiment
- **RM14:** Allowing users to view the result of their experiment
- **RM15:** Allowing users to download a .csv file containing the indices of the detected outliers
- **RM16:** Outlr. is by default an English website
- **RM17:** Passwords are somewhat securely stored as the password are not stored in plain text but hashed. To achieve superior security these could be salted user-wise.
- **RM18:** Notifying the user of (unexpected) errors. The app mustn't crash when errors happen.
- **RM19:** "Outlr." is reliable and mature.

- **RM20:** "Outlr." is from our point of view easy to learn and provides efficient workflows.
- **RM21** For good resource management, "Outlr." releases all resources of an experiment after it is run.
- **RM22** "Outlr." is well documented and the code is easy to read.
- **RM23** From our point of view, "Outlr." has an appealing and modern design.

The optional requirements we implemented were:

- **RO7:** Allowing the user to search experiments from his dashboard and clear the search term if wanted
- **RO8:** Allowing the user to sort the dashboard by clicking on the table headers
- **RO13:** Allowing the user to nest logical operators in the subspace logic defined for an experiment
- **RO18:** Allowing the user to run experiments without passing a ground-truth file

However, we did not implement RO1-RO6, RO9-RO12, RO14-RO17, RO19-RO28. The reason for not implementing these optional requirements was that we did not have enough time to work on the listed optional requirements.

6 Libraries/Frameworks used

Libraries and frameworks used on front-end side:

- axios 1.2.3
- jest 29.4.2
- vue 3.2.45
- vue-i118n 9.2.2
- vue-router 4.1.6
- vuex 4.1.0
- vuex-persistedstate 4.1.0

Libraries and frameworks used on back-end side:

- unittest
- Flask 2.2.2
- Flask-JWT-Extended 4.4.4
- Flask-CORS 3.0.10
- pandas 1.5.3
- pyod 1.0.7
- SQLALchemy 2.0.0
- psycopg2-binary 2.9.5

7 Overview of unit tests

On front-end side;

- Subspace Logic Parser tests
- Front-end model tests
  - User tests
  - Subspace tests
  - Subspace logic tests
  - Hyperparameter tests
  - Experiment tests

On back-end side:

- Configuration (JWT secret key and database URL)test
- Experiment execution test
- Back-end model tests:
  - User tests
  - ODM tests
  - Hyperparameter tests
  - Experiment, Experiment result, Subspace tests (also checking database relations)
  - PyOD ODM tests
  - Subspace logic tests
  - Tests, where objects are added to the database
  - CSV functions tests

8 Problems faced

- Difficulties setting up the CI/CD pipeline

  Due to not having enough permissions on the SCC server and our Gitlab repository, our team faced difficulties setting up a proper CI/CD environment. This resulted in the postponed completion of some previously scheduled tasks.

- Missing definition of JSONs

  We did not precisely define JSON requests and responses between the front-end and the back-end in the design phase. This was made up for in the implementation phase. Since the models have many references to each other defining the JSON structure was a difficult task. For these reasons, implementing the models took more time than expected.

- Difficulties estimating how long tasks take to implement

  Tasks took longer than expected as can be deduced from the difference between the planned timeline and the timeline depicting the actual procedure

- Getting used to new tools

  Although Python and TypeScript are relatively easy to learn it took the team some time to get used to. Additionally, working with frameworks like Vue.js, Flask, and SQLAlchemy made it difficult to predict how long tasks take. Implementing the models using SQLAlchemy took much more time than expected.

9 Design changes

9.1 General

Since both Python and TypeScript are not strictly object-oriented, we implemented some static utility classes or singleton classes from the class diagram by just writing the attributes and methods in the global scope.

9.2 Back-end

Package backend.models.odm

`ODM` is now an abstract class with an abstract `run_odm` method. `PyODM` implements `ODM` and calls the PyOD library in the `run_odm` method. `check_params` was removed.

Package backend.models.experiment

This package now contains the `Experiment`, `ExperimentResult`, `Subspace`, and `Outlier` classes.

`Subspaces` are now owned by the an `Experiment` directly instead of by an `ExperimentResult`. This is necessary because `Subspaces` exist before an `ExperimentResult` exists.

`Experiment` now has additional attributes:

- `dataset_name: str`
- `error_json: dict | None` contains an error if the experiment execution failed
- `ground_truth: np.NDArray | None` instead of `true_outliers` contains the ground truth array of zeros and ones, is not stored in the database

Package backend.execution

No `QueueSchedulers` were implemented. Instead, a `CoroutineExperimentScheduler` and a `SequentialODMScheduler` were implemented.

Package backend.parsers

Moved all functions to module `util.data` and renamed them to the following:

- `csv_to_dataset(name: str, dataset: str): Dataset`
- `csv_to_numpy_array(csv: str): NDArray`

- `write_list_to_csv(data: list[int], path: str): BytesIO`

Package backend.api

The endpoint `/experiment/upload-files` was created to upload the dataset and ground-truth files before experiment creation.
This was necessary as http only allows the content type to be either form-data or text/json. The former can contain raw files, the latter is used to give more information about the experiment to be created.

Package backend.database

Added more helper functions which were required in other places to `DatabaseAccess`.

9.3 Front-end

Package frontend.components.basic

The `Icon` enum was removed. Icons are now imported as a font.

Package frontend.api

`requestTokenCheck()` was renamed to `requestTokenIdentity()` and returns a JSON containing the username to the token if valid. If not valid as all other JSONs the JSON contains an error key.

To `AuthServices` the method `intialValidityCheck()`, which, when the HTTP request has a valid token, sets the Vuex state, was added. Furthermore `validateUsername()` and `validatePassword()` were added.

Package frontend.components.views.dashboard

Added a `DashboardSortColumn` enum for sorting the dashboard table.

Package frontend.components.views.createExperimentView

The `ExperimentTitleField` vue component was removed and replaced by plain HTML code.

Package frontend.models

Because of the changes to our JSON files we had to implement different versions of the `fromJSONObject()` method with different function headers. And therefore we could not use the `JSONDeseializable` in all of our frontend modles.

Package frontend.components.views

The `LoginForm` and `RegisterForm` vue components also got removed and replaced by plain HTML code.

Appendices

Glossary

Flask Python web framework. 16

Git A distributed version control system. 20

GitLab Web application for version control based on Git. 4

Notion An app for productivity and note taking, that also provides features for project management. 4

Object-Relational Mapping ORM (Object-Relational Mapping) is a programming technique that maps relational databases to objects in an object-oriented language. It provides an abstraction layer between the database and the application, and provides a set of tools to interact with databases using an object-oriented API.. 20

PyOD Python library providing methods for outlier detection. 17

Python Programming language commonly used in the field of machine learning and outlier detection. 16, 17, 20

SQLAlchemy Python library for working with relational databases. It provides a set of high-level APIs for interacting with databases in an object-oriented way, as well as a powerful ORM (Object-Relational Mapping) system for mapping Python objects to database tables.. 16

TypeScript Programming language that can be used for web development. 16, 17

Vue.js Front-end framework that can be used for building websites. 16