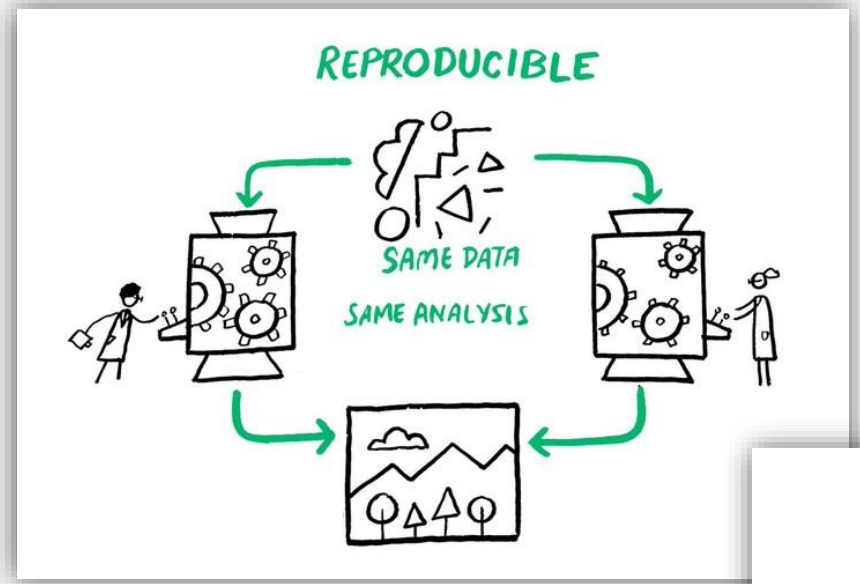
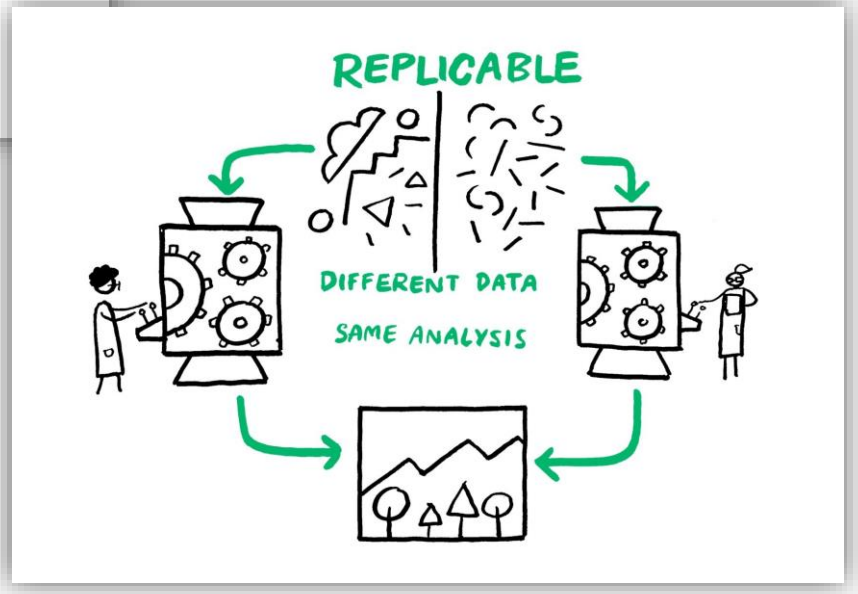
JSON Schema Discovery Reproduction Packages





https://oliviergimenez.github.io/reproducible-science-workshop/slides/0_introduction.html#2

Project Description

UNIVERSITY OF PASSAU

JSON Schema Discovery

GitHub

Original Project: Hypothesis

UNIVERSITY OF PASSAU

JSON Document

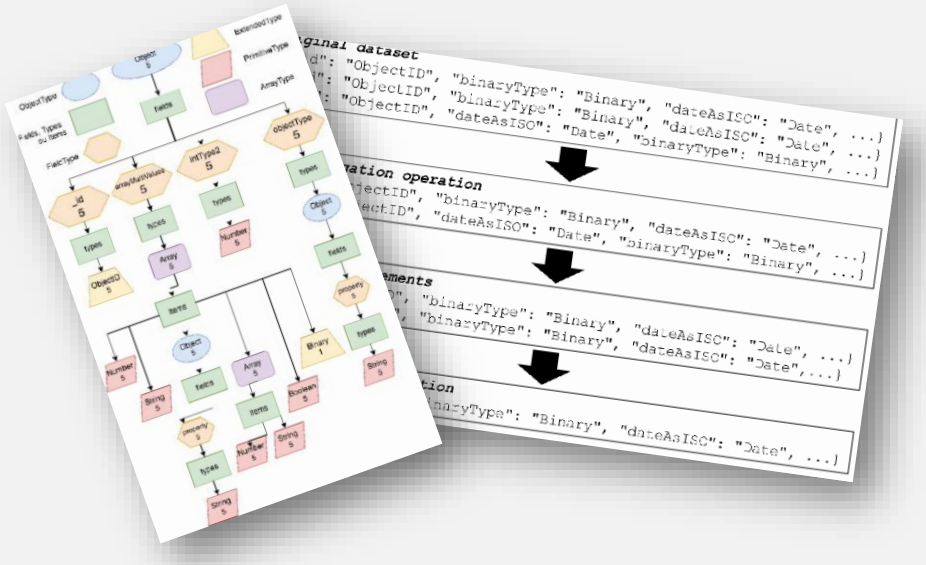
```
{
   "_id": ObjectId("50319491fe4dce143835c552"),
   "binaryType": BinData(2, "dGVzdGFuZG8="),
   "dateAsISO": Date("2014-01-31T22:26:33.000Z"),
   "timestampType": Timestamp(1421006159, 4),
   "dbRefType": DBRef("some_collection",
         "50319491fe4dce143835c552"),
   "undefinedType":undefined,
   "longType": NumberLong("9223372036854775807"),
   "booleanType": false,
   "nullType": null,
   "regexType": /\/^S\//m,
   "stringType": "some_value",
   "objectType": { "property": "some_value" },
   "arrayMultiValues": [ 48, "50",
      {"property": "teste"},  [1,"some_value"],
      true, BinData(2, "dGVzdGFuZG8=")   ],
   "arrayOfNumber": [ 1, 2 ],
   "javascriptType": { "$code":
      "var a = function(param){console.log(param);};"},
   "javascriptWithScopeType": {
      "$code": "int x = y", "$scope": { "y": 1 } },
   "maxKeyType": { "$maxKey": 1 },
   "minKeyType": { "$minKey": 4 },
}
```

JSON Schma

```
  "_id": "ObjectID",
  "binaryType": "Binary",
  "dateAsISO": "Date",
  "timestampType": "Timestamp",
  "dbRefType": "DBRef",
  "undefinedType": "Undefined",
  "longType": "Long",
  "booleanType": "Boolean",
  "nullType": "Null",
  "regexType": "RegExp",
  "stringType": "String",
  "objectType": { "property": "String" },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean", "Binary"    ],
  "arrayOfNumber": [ "Number" ],
  "javascriptType": "Code",
  "javascriptWithScopeType": "Code",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
}
```

Presenting an approach that extracts a schema from a JSON or Extended JSON document collection stored in a NoSQL document-oriented database or other document repository

Original Project: Technology



Original Project: Artefacts

2018 IEEE International Conference on Information Reuse and Integration for Data Science

An Approach for Schema Extraction of JSON and Extended JSON Document Collections

Angelo Augusto Frozza
Catatarinense Federal Institute
Camboriú – SC, Brazil
angelo.frozza@ifc.edu.br

Ronaldo dos Santos Mello, Felipe de Souza da Costa
Federal University of Santa Catarina
Florianópolis – SC, Brazil
r.mello@ufsc.br, feekosta@outlook.com

***Abstract*—JSON documents are raising as a common format for data representation due to the increasing popularity of NoSQL document-oriented databases. One of the reasons for such a popularity is their ability to handle large volumes of data at the absence of an explicit data schema. However, schema information is sometimes essential for applications during data retrieval, integration and analysis tasks, for example. Given this context, this paper presents an approach that extracts a schema from a JSON or Extended JSON document collection stored in a NoSQL document-oriented database or other document repository. Aggregation operations are considered in order to obtain a schema for each distinct structure in the collection, and a hierarchical data structure is proposed to group these schemas in order to generate a global schema in JSON Schema format. Experiments conducted on actual datasets, like DBPedia and Foursquare, demonstrate that the accuracy of the generated schemas is equivalent or even superior than related work.**

Keywords: NoSQL; JSON; Extended JSON; Schema Extraction; JSON Schema; Document-Oriented Database.

I. Introduction

The *Big Data* market explosion has led large companies to demand databases (DB) that are able to store and process large volumes of data effectively. In this context, traditional relational DB present several limitations as they prioritize strong consistency for read and write operations instead of high availability and horizontal expansion to better deal with increasing data volumes [1].

One family of new DB systems that have emerged to cope with these relational DB limitations is called NoSQL DB [2]. Basically, they avoid the overhead with the traditional ACID properties for transaction management, providing, instead, an eventual data consistency, a strong availability and elasticity capabilities. A common feature of NoSQL DB is that they are *schemaless*, i.e., they allow the storage of data without prior knowledge of their structure [3]. Document-oriented DB, one of the NoSQL database categories, for example, store and retrieve documents with simple and complex attributes mainly in JSON (*JavaScript Object Notation*) or Extended JSON[1] formats ([1], [4]), and documents do not necessarily share a common structure. In fact, schemaless DB avoid the problem of records with several attributes without a value when they are not uniform, allowing each record to contain only what is necessary.

However, the lack of information regarding the schema makes difficult to perform data integration processes, as well as several data processing tasks, such as data retrieval, validation and analysis [5]. Nevertheless, the absence of an explicit schema does not mean the total absence of a schema, since there is usually an implicit schema in the application code that access the database [6]. In the NoSQL DB context, to be aware of a data collection schema is very important during an application development. For example, several applications dealing with geographic data, such as *Foursquare*, retrieve data in JSON format, but do not define a schema for them, being difficult for users to query such data because they are unaware of the documents structure.

Based on this motivation, this paper presents an approach that aims to generate a single schema from a collection of JSON or Extended JSON documents. The generated schema is defined in the *JSON Schema* recommendation, which is establishing as a standard for specifying schemas of JSON documents [7]. The proposed approach, called *JSON Schema Discovery*, intends to mainly aid applications that need to be aware of NoSQL document-oriented DB schema for several purposes, like NoSQL schema and data integration. We give emphasis here to NoSQL document-oriented DB due to their popularity, but, in fact, our approach is able to produce a JSON Schema for any JSON document collection that it receives as input.

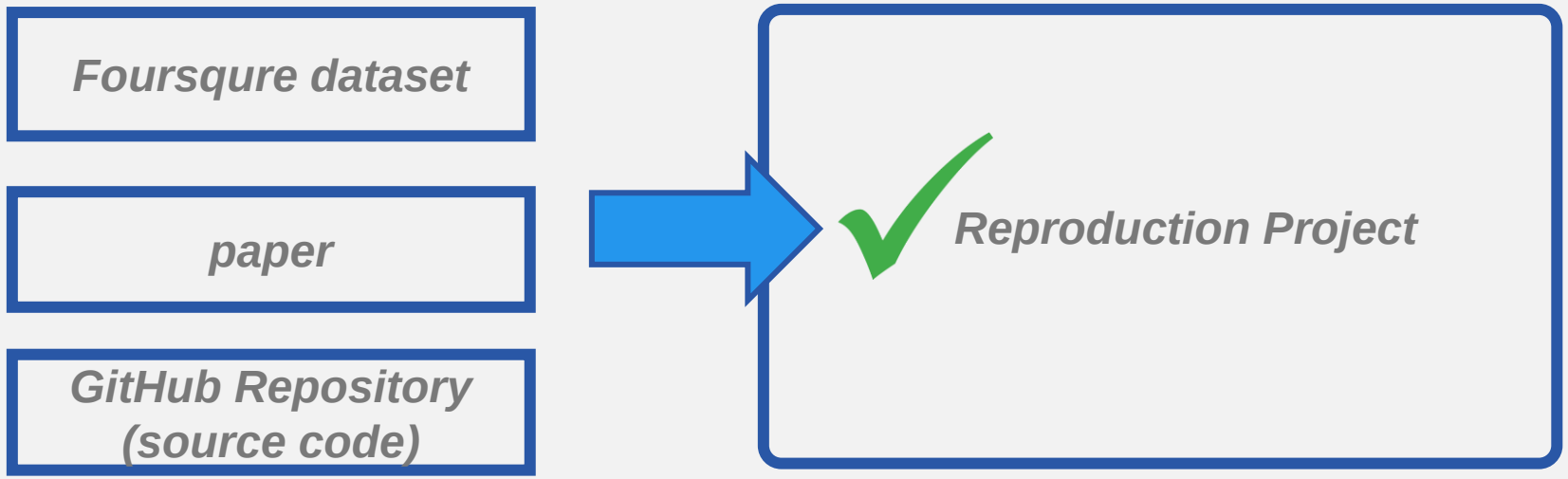
Different from related work, our approach provides the generation of a schema not only for a set of JSON documents, but also for a set of Extended JSON documents, and defines a hierarchical data structure that hold several metadata information useful for the schema generation. This hierarchical structure is manipulated by a *Model Driven Engineering (MDE)*-based process, which is a suitable software development technique for dealing with data model transformations [8]. Besides, our solution is available as a web tool that allows the generation, persistence, view and download of schemas in JSON Schema format.

Experiments to verify the quality of the schemas generated

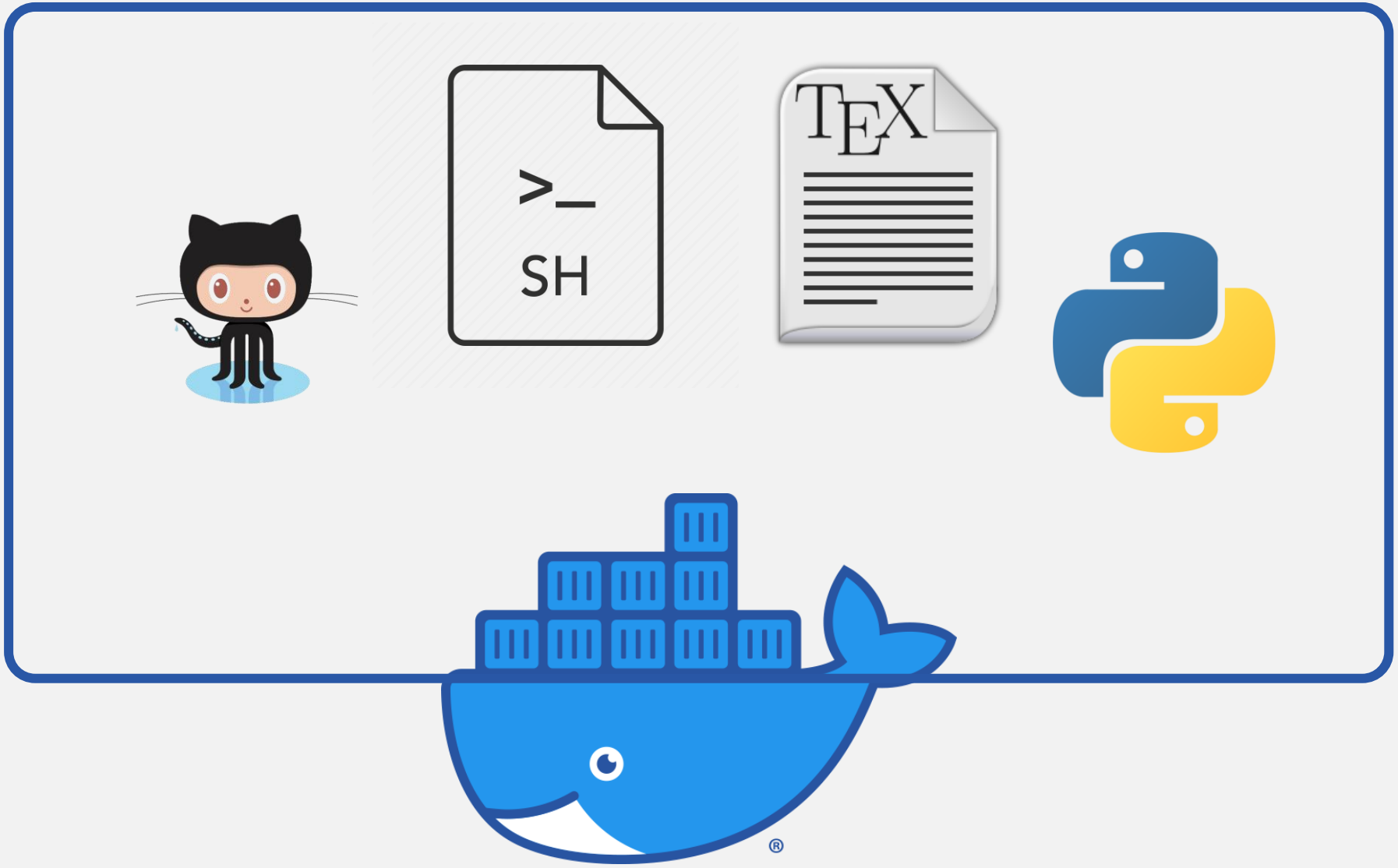
[1]https://goo.gl/1QA7EY

978-1-5386-2659-7/18/$31.00 ©2018 IEEE
DOI 10.1109/IRI.2018.00060

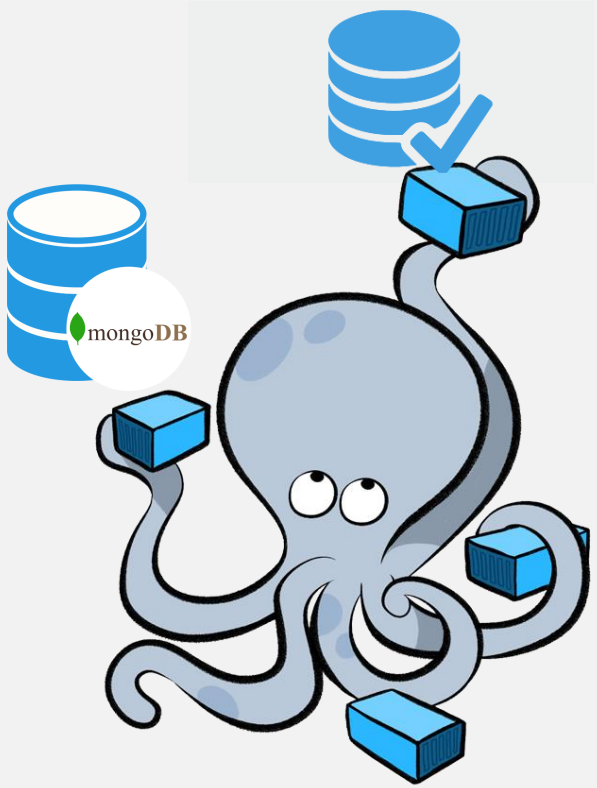
Packaging: Artefacts

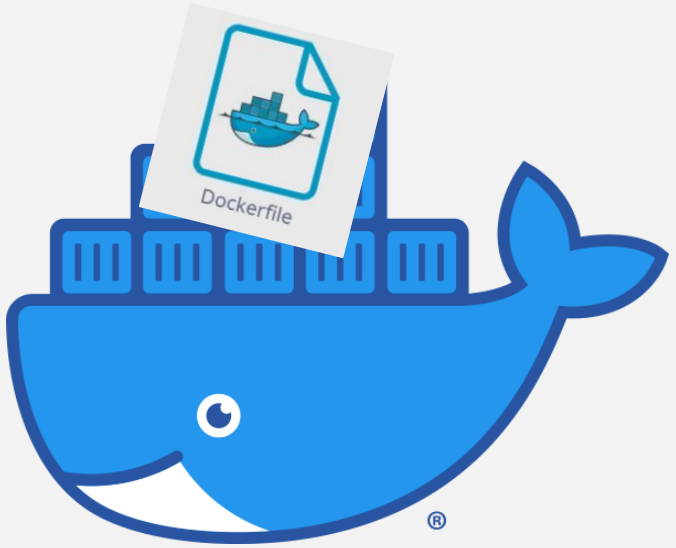


Packaging: Technologies



Packaging Steps

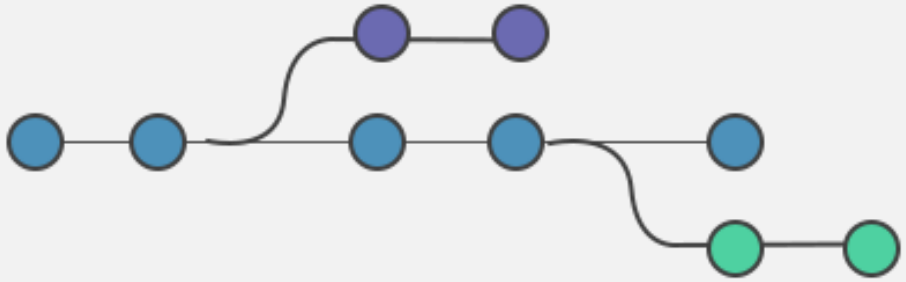




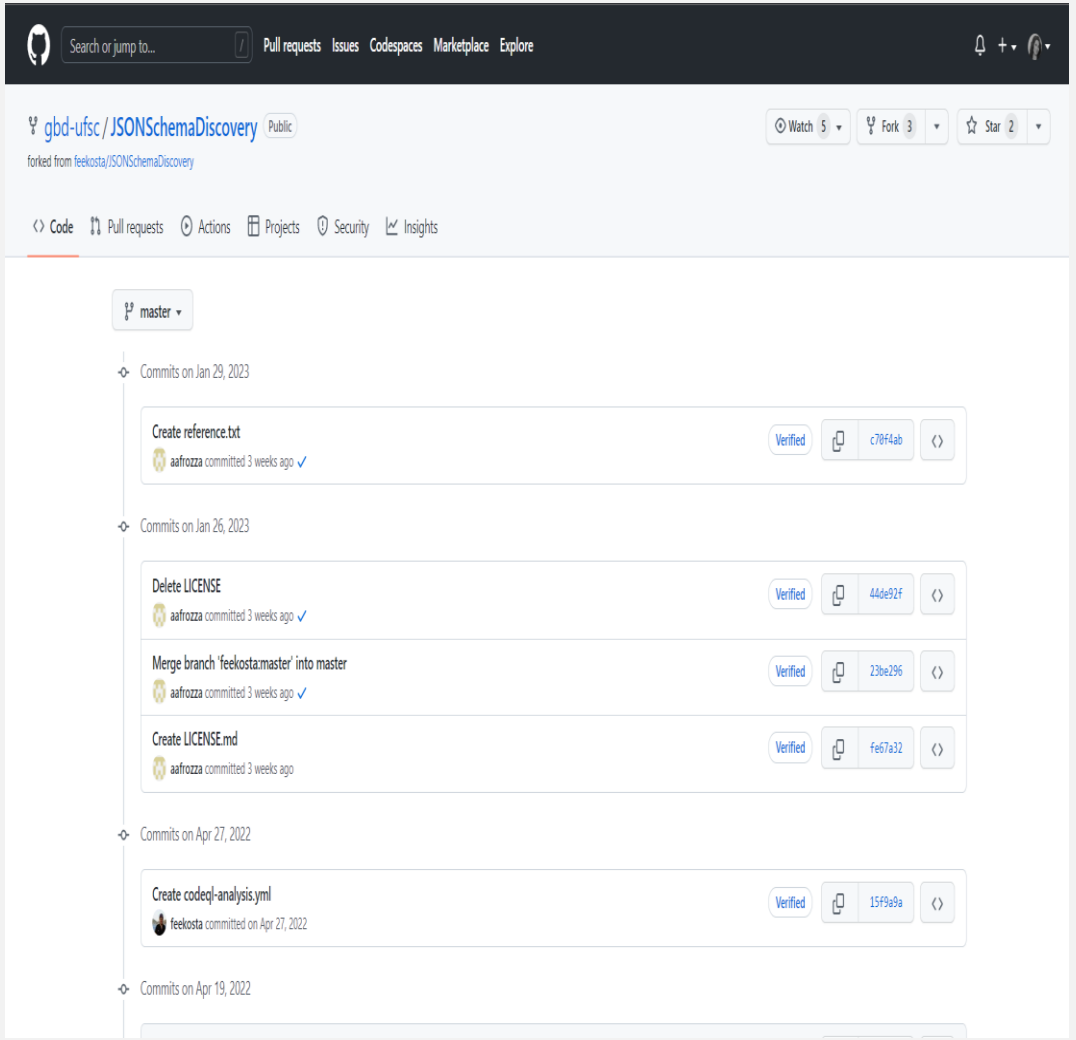
Packaging Steps: Clone

GitHub Repository (source code)

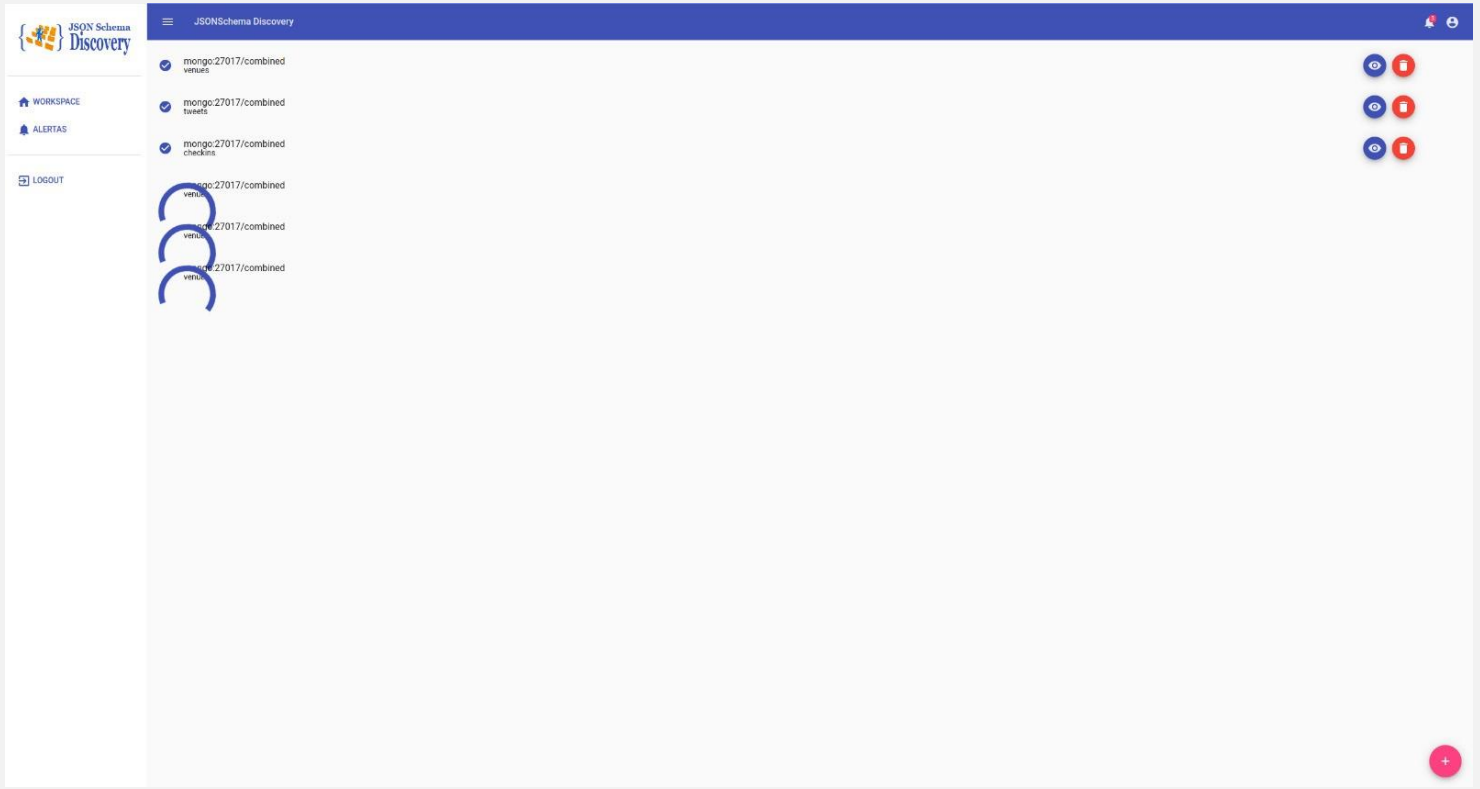
Git Clone https://github.com/gbd-ufsc/JSONSchemaDiscovery.git



Check out Branch 5f819f8fe60f29439d9cc34476daefe35645a840



Packaging Steps: Automation



„Automating chaos just gives faster chaos“ (Dorothy Graham)

Packaging Steps: Automation

Import JSON collections

dataset > $ import.sh

```bash
#!bin/bash
if [ $# -eq 0 ]
  then
    echo "No collectionName supplied"
fi

if [ $1 = "doall" ]
then
    mongorestore  --db  jsonschemadiscovery --collection venues /dataset/venues.bson --uri "mongodb://mongo:27017/jsonschemadiscovery"
    mongorestore  --db  jsonschemadiscovery --collection checkins /mongo-seed/checkins.bson --uri "mongodb://mongo:27017/jsonschemadiscovery"
    mongorestore  --db  jsonschemadiscovery --collection tweets /mongo-seed/tweets.bson --uri "mongodb://mongo:27017/jsonschemadiscovery"
fi
if [ $1 = "venues" ]
then
    mongorestore  --db  jsonschemadiscovery --collection venues /dataset/venues.bson --uri "mongodb://mongo:27017/jsonschemadiscovery"
fi
if [ $1 = "checkins" ]
then
    mongorestore  --db  jsonschemadiscovery --collection checkins /mongo-seed/checkins.bson --uri "mongodb://mongo:27017/jsonschemadiscovery"
fi
if [ $1 = "tweets" ]
then
    mongorestore  --db  jsonschemadiscovery --collection tweets /mongo-seed/tweets.bson --uri "mongodb://mongo:27017/jsonschemadiscovery"
fi
```

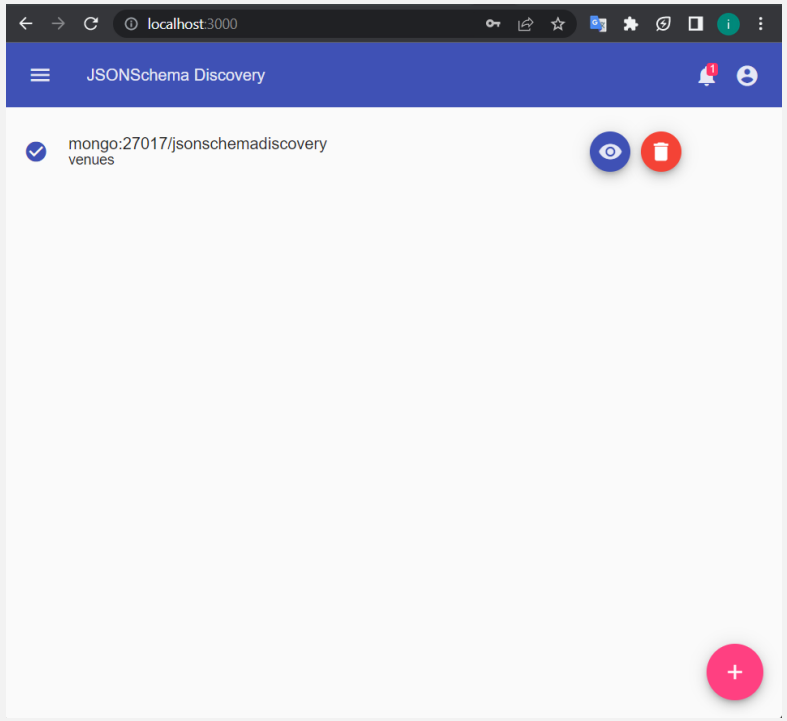
Packaging Steps: Automation

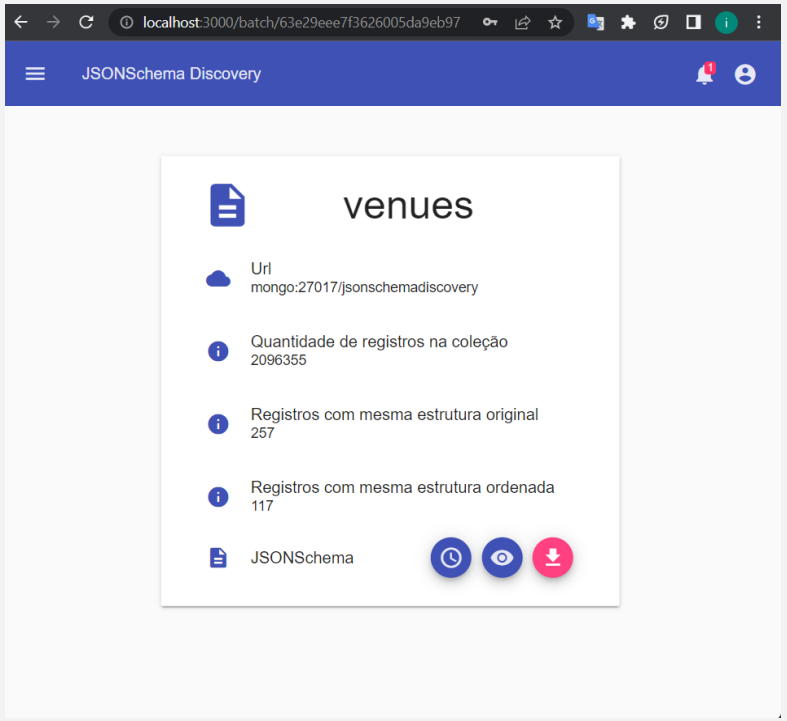
Automation all steps

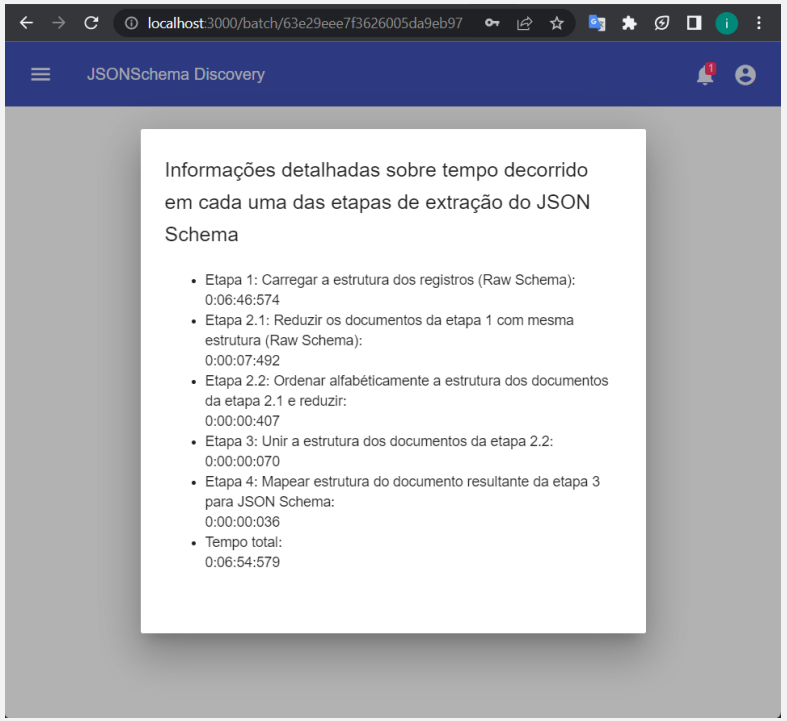
scripts > $ doallsteps.sh

```
#!/bin/sh
host=localhost
port=3000

rm /usr/bin/reproresults/*.txt
rm /usr/bin/reproresults/*.json

# Create an account
curl $host:$port/api/register -X POST -H 'Content-Type: application/json' -d '{"username": "tayebi", "email": "tayebi@tayebi.de", "password": "tayebi107323"}' > /usr/bin/
# Login and get back the Token
curl $host:$port/api/login -X POST -H 'Content-Type: application/json' -d '{"email": "tayebi@tayebi.de", "password": "tayebi107323"}' > /usr/bin/reproresults/token.txt
token=$(jq .token -r /usr/bin/reproresults/token.txt)
header1="Content-Type: application/json"
header2="Authorization: Bearer"
curl $host:$port/api/batches -H "$header1" -H "$header2 $token" > /usr/bin/reproresults/batches.txt

# Collection venues
curl $host:$port/api/batch/rawschema/steps/all -X POST -H 'Content-Type: application/json' -H "$header2 $token" -d '{"authentication":{"authMechanism":"SCRAM-SHA-1"},"por

curl $host:$port/api/batches -H 'Content-Type: application/json' -H "$header2 $token" > /usr/bin/reproresults/allbatches.txt

jq '.[] | select( .status == "DONE" ) |  select( .collectionName == "venues") | ._id' -r /usr/bin/reproresults/allbatches.txt > /usr/bin/reproresults/batchidlist.txt

# REDUCE_DOCUMENTS
while read -u 10 id; do
    curl $host:$port/api/batch/$id -H 'Content-Type: application/json' -H "$header2 $token" > /usr/bin/reproresults/batch_$id.json
done 10< /usr/bin/reproresults/batchidlist.txt

# Collection checkins
curl $host:$port/api/batch/rawschema/steps/all -X POST -H 'Content-Type: application/json' -H "$header2 $token" -d '{"authentication":{"authMechanism":"SCRAM-SHA-1"},"por

curl $host:$port/api/batches -H 'Content-Type: application/json' -H "$header2 $token" > /usr/bin/reproresults/allbatches.txt

jq '.[] | select( .status == "DONE" ) |  select( .collectionName == "checkins") | ._id' -r /usr/bin/reproresults/allbatches.txt > /usr/bin/reproresults/batchidlist.txt

# REDUCE_DOCUMENTS
while read -u 10 id; do
    curl $host:$port/api/batch/$id -H 'Content-Type: application/json' -H "$header2 $token" > /usr/bin/reproresults/batch_$id.json
done 10< /usr/bin/reproresults/batchidlist.txt

# Collection tweets

```

Packaging Steps: Automation







Packaging Steps: comparison

Table III
RESULTS FOR FOURSQUARE DATASETS

Collection	N_JSON	RS	ROrd	TB	TT	TB/TT
venues	2 million	257	117	7,47 min	7,52 min	99,33%
checkins	11 million	2	2	35,27 min	35,52 min	99,29%
tweets	17 million	23	16	53,11 min	53,44 min	99,38%

N_JSON - Number of JSON documents. RS - Raw schemas.
ROrd - Raw schemas with ordered structure.
TB - Time to obtain the raw schemas. TT - Total time.

Collection	N-JSON	RS	TB	TT	T B/TT
$venues	2	257	17	74	99.33
$checkins	11	2	2	35	99.29
$tweets	17	23	16	53	99.38

Table 1: RESULTS FOR FOURSQUARE DATASETS (N-JSON: Number of JSON documents.RS:Raw schemas. ROrd:Raw schemas with ordered structure. TB:Time to obtain the raw schemas per minitue.TT:Total time per min.)

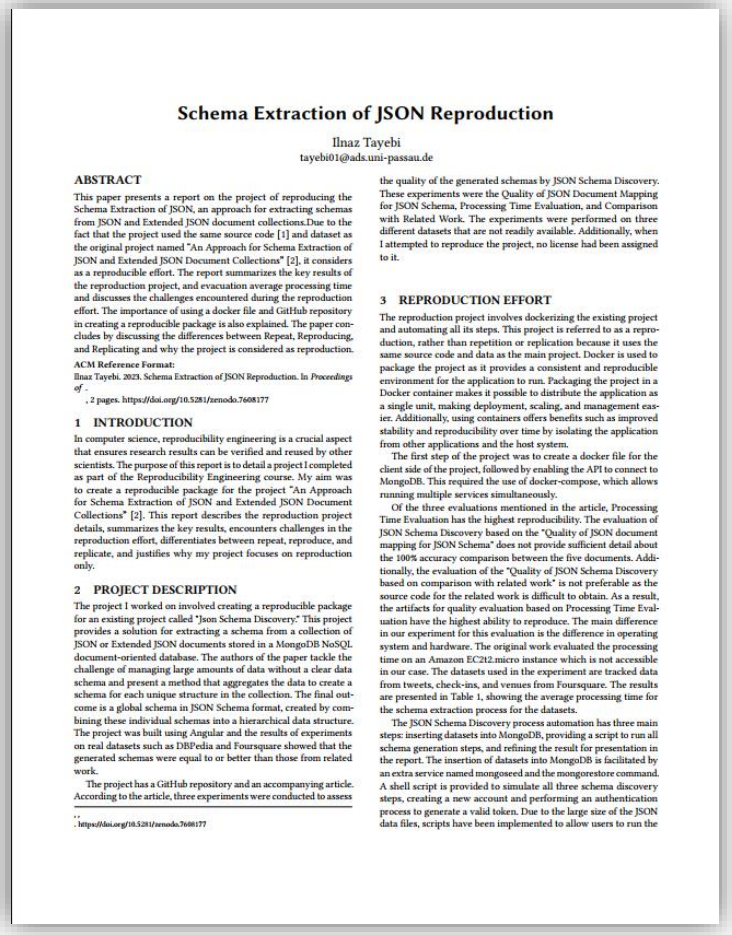
Packaging Steps: Generating PDF

```
root@5d6d2469af85:/home/repro/git-repos/JSONSchemaDiscovery# prepare_data.sh
This is pdfTeX, Version 3.141592653-2.6-1.40.22 (TeX Live 2022/dev/Debian) (preloaded format=pdflatex)
 restricted \write18 enabled.
entering extended mode
(/home/repro/paper/report.tex
LaTeX2e <2021-11-15> patch level 1
L3 programming layer <2022-01-21>
(/usr/share/texlive/texmf-dist/tex/latex/acmart/acmart.cls
Document Class: acmart 2021/12/05 v1.81 Typesetting articles for the Associatio
```

…./JSONSchemaDiscovery# prepare_data.sh

```
Package rerunfilecheck Warning: File `report.out' has changed.
(rerunfilecheck)                Rerun to get outlines right
(rerunfilecheck)                or use package `bookmark'.

 )
(see the transcript file for additional information)</usr/share/texlive/texmf-d
ist/fonts/type1/public/amsfonts/cm/cmbx10.pfb></usr/share/texlive/texmf-dist/fo
nts/type1/public/amsfonts/cm/cmbx9.pfb></usr/share/texlive/texmf-dist/fonts/typ
e1/public/amsfonts/cm/cmr10.pfb></usr/share/texlive/texmf-dist/fonts/type1/publ
ic/amsfonts/cm/cmr12.pfb></usr/share/texlive/texmf-dist/fonts/type1/public/amsf
onts/cm/cmr6.pfb></usr/share/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/
cmr9.pfb></usr/share/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/cmssbx10
.pfb>
Output written on report.pdf (2 pages, 126119 bytes).
Transcript written on report.log.
root@5d6d2469af85:/home/repro/git-repos/JSONSchemaDiscovery#
```

Packaging Steps: Generating PDF



Schema Extraction of JSON Reproduction

Ilnaz Tayebi
tayebi01@ads.uni-passau.de

ABSTRACT

This paper presents a report on the project of reproducing the Schema Extraction of JSON, an approach for extracting schemas from JSON and Extended JSON document collections.Due to the fact that the project used the same source code [1] and dataset as the original project named "An Approach for Schema Extraction of JSON and Extended JSON Document Collections" [2], it considers as a reproducible effort. The report summarizes the key results of the reproduction project, and evacuation average processing time and discusses the challenges encountered during the reproduction effort. The importance of using a docker file and GitHub repository in creating a reproducible package is also explained. The paper concludes by discussing the differences between Repeat, Reproducing, and Replicating and why the project is considered as reproduction.

ACM Reference Format:
Ilnaz Tayebi. 2023. Schema Extraction of JSON Reproduction. In *Proceedings of* .
, 2 pages. https://doi.org/10.5281/zenodo.7608177

1 INTRODUCTION

In computer science, reproducibility engineering is a crucial aspect that ensures research results can be verified and reused by other scientists. The purpose of this report is to detail a project I completed as part of the Reproducibility Engineering course. My aim was to create a reproducible package for the project "An Approach for Schema Extraction of JSON and Extended JSON Document Collections" [2]. This report describes the reproduction project details, summarizes the key results, encounters challenges in the reproduction effort, differentiates between repeat, reproduce, and replicate, and justifies why my project focuses on reproduction only.

2 PROJECT DESCRIPTION

The project I worked on involved creating a reproducible package for an existing project called "Json Schema Discovery." This project provides a solution for extracting a schema from a collection of JSON or Extended JSON documents stored in a MongoDB NoSQL document-oriented database. The authors of the paper tackle the challenge of managing large amounts of data without a clear data schema and present a method that aggregates the data to create a schema for each unique structure in the collection. The final outcome is a global schema in JSON Schema format, created by combining these individual schemas into a hierarchical data structure. The project was built using Angular and the results of experiments on real datasets such as DBPedia and Foursquare showed that the generated schemas were equal to or better than those from related work.

The project has a GitHub repository and an accompanying article. According to the article, three experiments were conducted to assess the quality of the generated schemas by JSON Schema Discovery. These experiments were the Quality of JSON Document Mapping for JSON Schema, Processing Time Evaluation, and Comparison with Related Work. The experiments were performed on three different datasets that are not readily available. Additionally, when I attempted to reproduce the project, no license had been assigned to it.

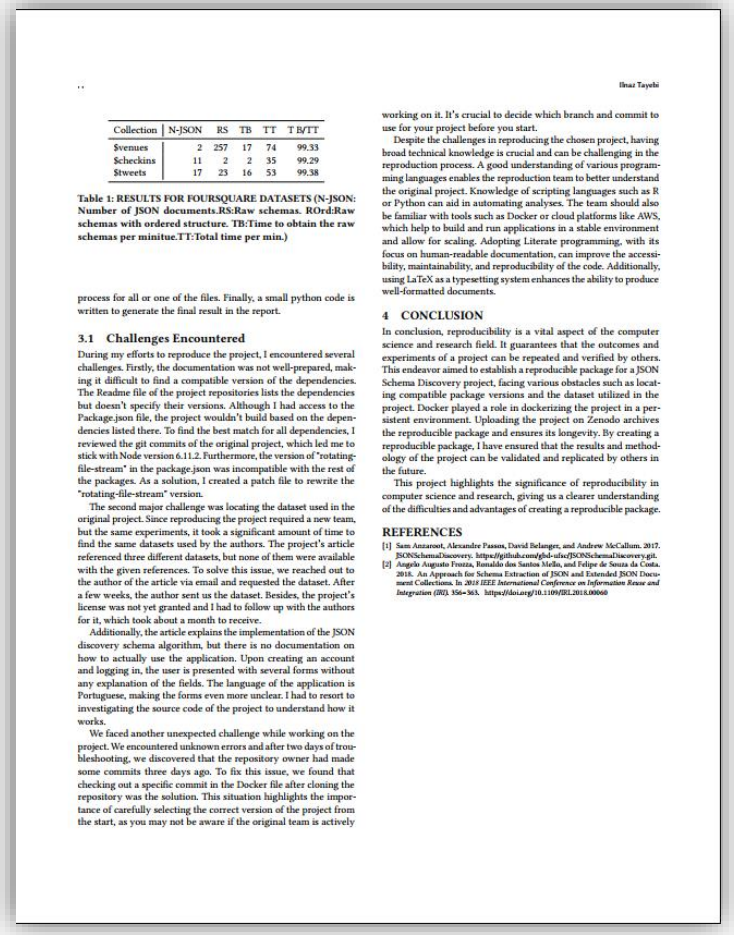
3 REPRODUCTION EFFORT

The reproduction project involves dockerizing the existing project and automating all its steps. This project is referred to as a reproduction, rather than repetition or replication because it uses the same source code and data as the main project. Docker is used to package the project as it provides a consistent and reproducible environment for the application to run. Packaging the project in a Docker container makes it possible to distribute the application as a single unit, making deployment, scaling, and management easier. Additionally, using containers offers benefits such as improved stability and reproducibility over time by isolating the application from other applications and the host system.

The first step of the project was to create a docker file for the client side of the project, followed by enabling the API to connect to MongoDB. This required the use of docker-compose, which allows running multiple services simultaneously.

Of the three evaluations mentioned in the article, Processing Time Evaluation has the highest reproducibility. The evaluation of JSON Schema Discovery based on the "Quality of JSON document mapping for JSON Schema" does not provide sufficient detail about the 100% accuracy comparison between the five documents. Additionally, the evaluation of the "Quality of JSON Schema Discovery based on comparison with related work" is not preferable as the source code for the related work is difficult to obtain. As a result, the artifacts for quality evaluation based on Processing Time Evaluation have the highest ability to reproduce. The main difference in our experiment for this evaluation is the difference in operating system and hardware. The original work evaluated the processing time on an Amazon EC2t2.micro instance which is not accessible in our case. The datasets used in the experiment are tracked data from tweets, check-ins, and venues from Foursquare. The results are presented in Table 1, showing the average processing time for the schema extraction process for the datasets.

The JSON Schema Discovery process automation has three main steps: inserting datasets into MongoDB, providing a script to run all schema generation steps, and refining the result for presentation in the report. The insertion of datasets into MongoDB is facilitated by an extra service named mongoseed and the mongorestore command. A shell script is provided to simulate all three schema discovery steps, creating a new account and performing an authentication process to generate a valid token. Due to the large size of the JSON data files, scripts have been implemented to allow users to run the process for all or one of the files. Finally, a small python code is written to generate the final result in the report.



Collection	N-JSON	RS	TB	TT	T B/TT
$venues	2	257	17	74	99.33
$checkins	11	2	2	35	99.29
$tweets	17	23	16	53	99.38

Table 1: RESULTS FOR FOURSQUARE DATASETS (N-JSON: Number of JSON documents.RS:Raw schemas. ROrd:Raw schemas with ordered structure. TB:Time to obtain the raw schemas per minitue.TT:Total time per min.)

3.1 Challenges Encountered

During my efforts to reproduce the project, I encountered several challenges. Firstly, the documentation was not well-prepared, making it difficult to find a compatible version of the dependencies. The Readme file of the project repositories lists the dependencies but doesn't specify their versions. Although I had access to the Package.json file, the project wouldn't build based on the dependencies listed there. To find the best match for all dependencies, I reviewed the git commits of the original project, which led me to stick with Node version 6.11.2. Furthermore, the version of "rotating-file-stream" in the package.json was incompatible with the rest of the packages. As a solution, I created a patch file to rewrite the "rotating-file-stream" version.

The second major challenge was locating the dataset used in the original project. Since reproducing the project required a new team, but the same experiments, it took a significant amount of time to find the same datasets used by the authors. The project's article referenced three different datasets, but none of them were available with the given references. To solve this issue, we reached out to the author of the article via email and requested the dataset. After a few weeks, the author sent us the dataset. Besides, the project's license was not yet granted and I had to follow up with the authors for it, which took about a month to receive.

Additionally, the article explains the implementation of the JSON discovery schema algorithm, but there is no documentation on how to actually use the application. Upon creating an account and logging in, the user is presented with several forms without any explanation of the fields. The language of the application is Portuguese, making the forms even more unclear. I had to resort to investigating the source code of the project to understand how it works.

We faced another unexpected challenge while working on the project. We encountered unknown errors and after two days of troubleshooting, we discovered that the repository owner had made some commits three days ago. To fix this issue, we found that checking out a specific commit in the Docker file after cloning the repository was the solution. This situation highlights the importance of carefully selecting the correct version of the project from the start, as you may not be aware if the original team is actively working on it. It's crucial to decide which branch and commit to use for your project before you start.

Despite the challenges in reproducing the chosen project, having broad technical knowledge is crucial and can be challenging in the reproduction process. A good understanding of various programming languages enables the reproduction team to better understand the original project. Knowledge of scripting languages such as R or Python can aid in automating analyses. The team should also be familiar with tools such as Docker or cloud platforms like AWS, which help to build and run applications in a stable environment and allow for scaling. Adopting Literate programming, with its focus on human-readable documentation, can improve the accessibility, maintainability, and reproducibility of the code. Additionally, using LaTeX as a typesetting system enhances the ability to produce well-formatted documents.

4 CONCLUSION

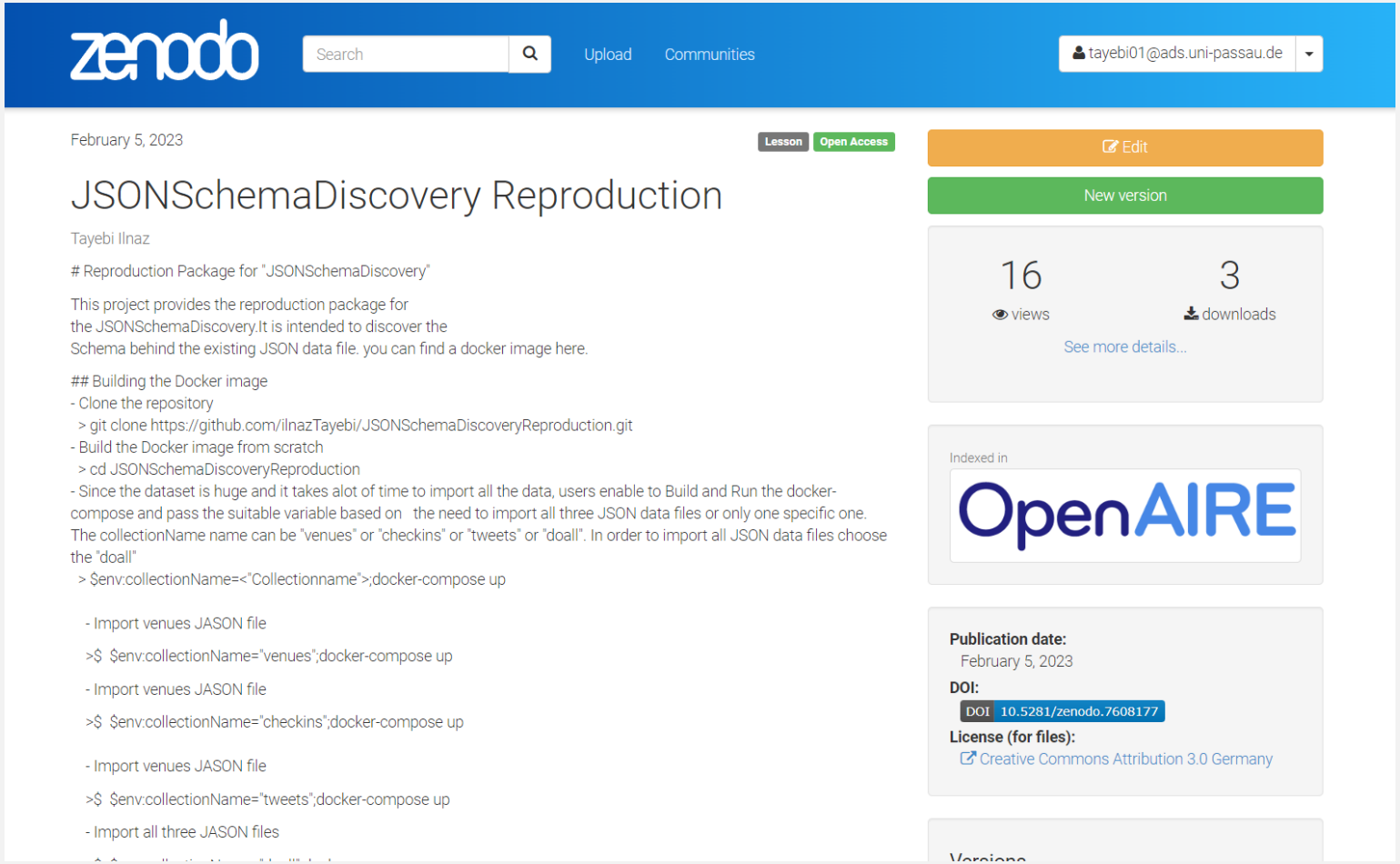
In conclusion, reproducibility is a vital aspect of the computer science and research field. It guarantees that the outcomes and experiments of a project can be repeated and verified by others. This endeavor aimed to establish a reproducible package for a JSON Schema Discovery project, facing various obstacles such as locating compatible package versions and the dataset utilized in the project. Docker played a role in dockerizing the project in a persistent environment. Uploading the project on Zenodo archives the reproducible package and ensures its longevity. By creating a reproducible package, I have ensured that the results and methodology of the project can be validated and replicated by others in the future.

This project highlights the significance of reproducibility in computer science and research, giving us a clearer understanding of the difficulties and advantages of creating a reproducible package.

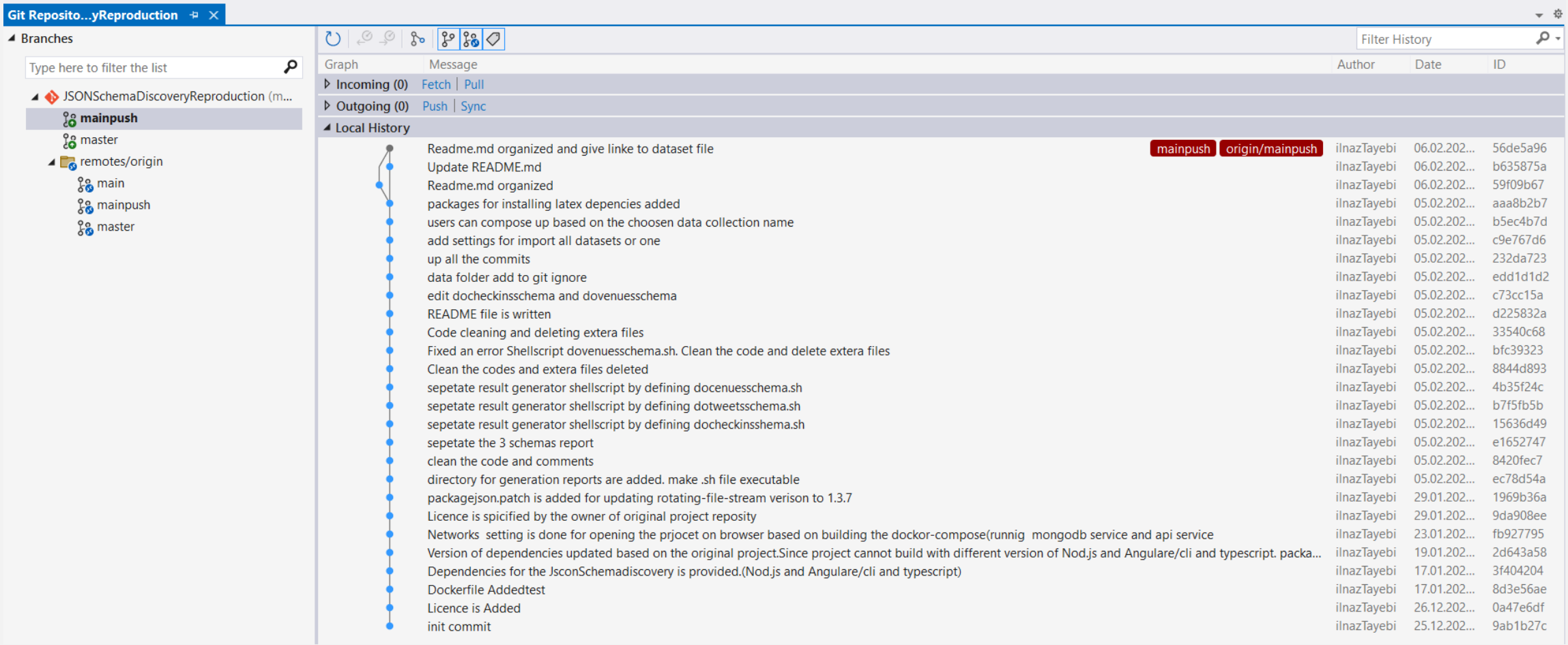
REFERENCES

[1] Sam Anzaroot, Alexandre Passos, David Belanger, and Andrew McCallum. 2017. JSONSchemaDiscovery. https://github.com/fdbl-ufes/JSONSchemaDiscovery.git.
[2] Angelo Augusto Frozza, Ronaldo dos Santos Mello, and Felipe de Souza da Costa. 2018. An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. 356–363. https://doi.org/10.1109/IRI.2018.00060

Packaging Steps: Zenodo + DOI



Packaging Steps: GitHub History

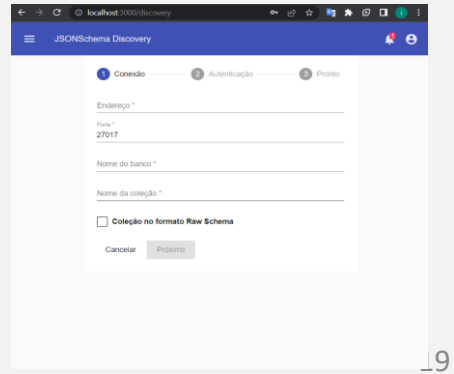


Packaging : Challenges

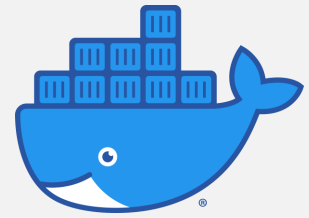
Datasets

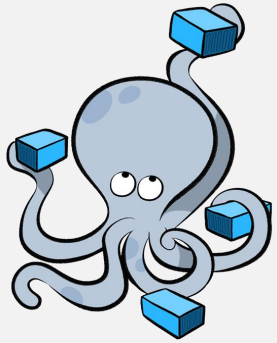
Version

Application !!!



Challenges: Broad Technical Knowledge





Summary

JSON Schema Discovery