HELLENIC REPUBLIC
UNIVERSITY OF THE PELOPONNESE
FACULTY OF ECONOMY AND TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS
M.Sc. IN COMPUTER SCIENCE AND TECHNOLOGY

M.Sc. Thesis

Process and correlation between large data sets

THEODOROS GIANNAKOPOULOS
A.M.: 2022201602004

Supervisor:
Costas Vassilakis

Tripoli, October 2019

Contents

Table of figures

Abstract

The goal of this thesis is to implement an easy-to-use solution to the end user, to query data from a given set of databases through a user interface. The user interface would provide a graphical representation of given databases and the ability to upload data and validate the data via a GUI. Hence, the solution was implemented as a web-based application, and the Laravel framework was used for the implementation.

A survey was conducted to identify similar tools and no applications providing the aforementioned functionalities were identified. In order to be able to query an information repository (a database, data warehouse, Google Cloud/ S3 buckets, Google Big Query tables, …) specific access rights should been given beforehand to the corresponding parties. Process-wise, this can be really time consuming, and impractical since involves communication between parties, requesting access, stating why they need access and then, if approved, then they are able to query corresponding datasets. Furthermore, users should familiarize themselves with diverse user interfaces.

Considering the above, the tool implemented in the context of this thesis provides a manageable solution to provide access to users to multiple resources under a unified process and from within a single, comprehensive user interface, so the end user will be able to query multiple given datasets.

Περίληψη

Ο στόχος της παρούσας εργασίας είναι να υλοποιηθεί μία εύχρηστη λύση προκειμένου ο κάθε χρήστης να έχει τη δυνατότητα να υποβάλει ερωτήματα σε ένα σύνολο από βάσεις δεδομένων διά μέσου μιας διεπαφής χρήστη. Η διεπαφή χρήστη θα πρέπει να παρέχει μία γραφική αναπαράσταση των βάσεων δεδομένων και τη δυνατότητα να μεταφορτώνονται προς την πλατφόρμα και να επικυρώνονται τα δεδομένα μέσω μιας γραφικής διεπαφής. Προς την κατεύθυνση αυτή, δημιουργήθηκε μία εφαρμογή ιστού (web-based application), και για την υλοποίησή της χρησιμοποιήθηκε το πλαίσιο ανάπτυξης Laravel.

Αρχικά, διεξήχθη έρευνα για τον εντοπισμό εργαλείων με παρόμοια λειτουργικότητα, ωστόσο προέκυψε ότι δεν υπάρχουν διαθέσιμα σχετικά εργαλεία. Προκειμένου να είναι δυνατόν -σύμφωνα με την τρέχουσα κατάσταση- να πραγματοποιηθούν ερωτήσεις σε ένα αποθετήριο πληροφορίας (βάση δεδομένων, αποθήκη δεδομένων, συλλογές δεδομένων σε μορφή Google Cloud/ S3 buckets, Google Big Query tables), ο χρήστης πρέπει να λάβει εκ των προτέρων την κατάλληλη εξουσιοδότηση. Σε διαδικαστικό επίπεδο, αυτό είναι χρονοβόρο και μη πρακτικό, καθώς περιλαμβάνει επικοινωνία μεταξύ των εμπλεκομένων μερών, αίτημα για πρόσβαση με προσδιορισμό του λόγου για τον οποίο ζητείται και -εφ' όσον παρασχεθεί η έγκριση- τότε μόνον ο χρήστης έχει τη δυνατότητα να υποβάλλει ερωτήματα. Επιπρόσθετα, ο χρήστης θα πρέπει να εξοικειωθεί με διαφορετικές διεπαφές χρήστη.

Λαμβάνοντας υπ' όψιν τα ανωτέρω, το εργαλείο που υλοποιήθηκε στο πλαίσιο της παρούσας διπλωματικής, παρέχει μία διαχειρίσιμη προσέγγιση για την παροχή στους χρήστες πρόσβασης, σε πολλαπλούς πόρους κάτω από μία ενιαία διαδικασία και διά μέσου μιας μοναδικής διεπαφής χρήστη, επιτρέποντας έτσι την υποβολή από μέρους των χρηστών ερωτημάτων σε πολλαπλές πηγές πληροφορίας.

Εκτεταμένη περίληψη

Ο στόχος της παρούσας εργασίας είναι να υλοποιηθεί μία εύχρηστη λύση προκειμένου κάθε χρήστης να έχει τη δυνατότητα να υποβάλει ερωτήματα σε ένα σύνολο από βάσεις δεδομένων διά μέσου μιας διεπαφής χρήστη. Η διεπαφή χρήστη θα πρέπει να παρέχει μία γραφική αναπαράσταση των βάσεων δεδομένων και τη δυνατότητα να μεταφορτώνονται προς την πλατφόρμα και να επικυρώνονται τα δεδομένα μέσω μιας γραφικής διεπαφής. Προς την κατεύθυνση αυτή, δημιουργήθηκε μία εφαρμογή ιστού (web-based application), και για την υλοποίησή της χρησιμοποιήθηκε το πλαίσιο ανάπτυξης Laravel.

Αρχικά, διεξήχθη έρευνα για τον εντοπισμό εργαλείων με παρόμοια λειτουργικότητα, ωστόσο προέκυψε ότι δεν υπάρχουν διαθέσιμα σχετικά εργαλεία. Προκειμένου να είναι δυνατόν -σύμφωνα με την τρέχουσα κατάσταση- να πραγματοποιηθούν ερωτήσεις σε ένα αποθετήριο πληροφορίας (βάση δεδομένων, αποθήκη δεδομένων, συλλογές δεδομένων σε μορφή Google Cloud/ S3 buckets, Google Big Query tables), ο χρήστης πρέπει να λάβει εκ των προτέρων την κατάλληλη εξουσιοδότηση. Σε διαδικαστικό επίπεδο, αυτό είναι χρονοβόρο και μη πρακτικό, καθώς περιλαμβάνει επικοινωνία μεταξύ των εμπλεκομένων μερών, αίτημα για πρόσβαση με προσδιορισμό του λόγου για τον οποίο ζητείται και -εφ' όσον παρασχεθεί η έγκριση- τότε μόνον ο χρήστης έχει τη δυνατότητα να υποβάλλει ερωτήματα. Επιπρόσθετα, ο χρήστης θα πρέπει να εξοικειωθεί με διαφορετικές διεπαφές χρήστη.

Λαμβάνοντας υπ' όψιν τα ανωτέρω, το εργαλείο που υλοποιήθηκε στο πλαίσιο της παρούσας διπλωματικής, παρέχει μία διαχειρίσιμη προσέγγιση για την παροχή στους χρήστης πρόσβασης σε πολλαπλούς πόρους κάτω από μία ενιαία διαδικασία και διά μέσου μιας μοναδικής διεπαφής χρήστη, επιτρέποντας έτσι την υποβολή από μέρους των χρηστών ερωτημάτων σε πολλαπλές πηγές πληροφορίας.

Για την πιλοτική εφαρμογή και επαλήθευση των λειτουργιών της εργασίας, χρησιμοποιήθηκαν δεδομένα από τις παρακάτω πηγές:

- https://www.penn.museum/collections/objects/data.php
- https://github.com/tategallery/collection

και συγκεκριμένα, τα δεδομένα που είναι σε μορφή csv.

Παρατηρήθηκε η δομή των δεδομένων μας και δημιουργήσαμε αντίστοιχες βάσεις δεδομένων, με συσχετίσεις έτσι ώστε να υπάρχει σύνδεση μεταξύ των δεδομένων μας. Δηλαδή, για τα δεδομένα με προέλευση από *tategallery* δημιουργήσαμε συσχετίσεις μεταξύ των έργων τέχνης και των καλλιτεχνών, που βασίζεται σε ένα id και για τα δεδομένα που έχουν προέλευση από το *penn.museum* δημιουργήσαμε συσχετίσεις μεταξύ των έργων τέχνης και της προέλευσης αυτών.

Για την ανάπτυξη της εφαρμογής, χρησιμοποιήθηκαν διάφορες τεχνολογίες όπως PHP, MVC, Laravel, Laravel HTML Collectives, Laravel Artisan Commands, MariaDB, ETL.

Μέσω της εφαρμογής ο χρήστης, αφού έχει δημιουργήσει λογαριασμό στην εφαρμογή, έχει τη δυνατότητα να καταθέσει ερωτήματα στις διάφορες βάσεις δεδομένων που είναι διαθέσιμες εκείνη τη χρονική στιγμή.

Επίσης, υπάρχει η δυνατότητα να οπτικοποιήσει αποτελέσματα είτε από ερωτήματα που έχει υποβάλει ο ίδιος είτε από προσχηματισμένα ερωτήματα.

Ο χρήστης μπορεί να ανεβάσει δεδομένα για εμπλουτίσει τα δεδομένα της βάσης δεδομένων *tategallery*. Συγκεκριμένα, όταν ανέβει ένα αρχείο, εκκινείται μια διαδικασία ETL και τα δεδομένα, αποθηκεύονται σε έναν προσωρινό πίνακα. Έπειτα, ο χρήστης έχει τη δυνατότητα να πιστοποιήσει δεδομένα που βρίσκονται στον προσωρινό πίνακα και μετά από ένα αριθμό θετικών είτε αρνητικών ψήφων η αντίστοιχη εγγραφή είτε μεταβαίνει στη αντίστοιχη βάση δεδομένων είτε διαγράφεται.

Η εφαρμογή διαθέτει έχει δύο επιπέδων χρηστών, τους απλούς χρήστες, που έχουν τις δυνατότητες που περιγράφηκαν στις παραπάνω παραγράφους και τους χρήστες-διαχειριστές. Οι διαχειριστές, έχουν τη δυνατότητα να δημιουργήσουν χρήστες, να ανανεώσουν πληροφορίες χρηστών, όπως κωδικούς, email και άλλα στοιχεία. Μπορούν ακόμη να διαχειριστούν τα δικαιώματα πρόσβασης στις διαθέσιμες βάσεις δεδομένων, προσδιορίζοντας πρακτικά σε ποιες μπορούν οι χρήστες να υποβάλλουν ερωτήματα και σε ποιες όχι.

Τέλος, για να πραγματοποιηθεί η υλοποίηση της εφαρμογής διάφορες τεχνολογίες έπρεπε να ερευνηθούν, όπως Laravel, SQL injection, D3.js κ.λπ. Αυτό είχε ως αποτέλεσμα να δημιουργηθεί ένα ευέλικτο γραφικό περιβάλλον για το χρήστη.

1 Introduction

The goal of this thesis is to implement an easy-to-use solution to the end user, to query data from a given set of databases through a user interface. The user interface would provide a graphical representation of given databases and the ability to upload data and validate the data via a GUI. Hence, the solution was implemented as a web-based application, and the Laravel framework was used for the implementation.

A survey was conducted to identify similar tools and no applications providing the aforementioned functionalities were identified. In order to be able to query an information repository (a database, data warehouse, Google Cloud/ S3 buckets, Google Big Query tables, …) specific access rights should been given beforehand to the corresponding parties. Process-wise, this can be really time consuming, and impractical since involves communication between parties, requesting access, stating why they need access and then, if approved, then they are able to query corresponding datasets. Furthermore, users should familiarize themselves with diverse user interfaces.

Considering the above, the tool implemented in the context of this thesis provides a manageable solution to provide access to users to multiple resources under a unified process and from within a single, comprehensive user interface, so the end user will be able to query multiple given datasets.

To validate our approach, data from two different sources were used. More specifically, both our data sources originate from artwork museums, namely the Tate Gallery and the Penn Museum; the data collections are publicly available via the following links:

- https://www.penn.museum/collections/objects/data.php
- https://github.com/tategallery/collection

For the purpose of the thesis only the CSV format files were used.

We examined the different data structures from our sources and created corresponding database structures in order to store our data; this includes the creation of associations between the data, i.e. for the data that originate from the *tategallery* collection, we have created an association between the artist ids and the corresponding artwork, and for the data that originate from the *penn.museum* collection we have a created an association between the origin of the artworks and the artworks.

2 Technologies that were used

For the development of the project various technologies were used. Technologies such as PHP, MVC, Laravel, Laravel HTML Collectives, Laravel Artisan Commands, MariaDB, ETL.

2.1 PHP

PHP (PHP ,2019a) a recursive acronym for Hypertext Preprocessor. PHP is a widely-used open source scripting language that is general-purpose programming language originally designed for web development.

PHP code may be executed with a command line interface (CLI), embedded into HTML code, or it can be used in combination with various web template system, web content management systems and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in a web server or as a Common Gateway Interface (CGI) executable.

2.2 MVC

2.2.1 What is MVC?

MVC (Leff and Rayfield, 2001; Majeed and Rauf, 2018) is a design pattern and the "MVC" acronym stands for "Model View Controller". With the MVC pattern we look at the application structure with regards to how the data flow of our application works.

2.2.2 Why use MVC?

When building PHP application, it may be ok to have files "flying" around in very small projects. However, when the project starts to grow, having a structure can drastically improve maintainability.

On a more technical note, when we use the MVC pattern we expect to see the following

- Controllers to handle the requests and retrieve data by leveraging Models
- Models to interact with our database and retrieve information objects
- Views to renders pages

Additionally, routes can be used to map URLs to designated Controller actions. To clarify the request processing scenario, we provide the following example:

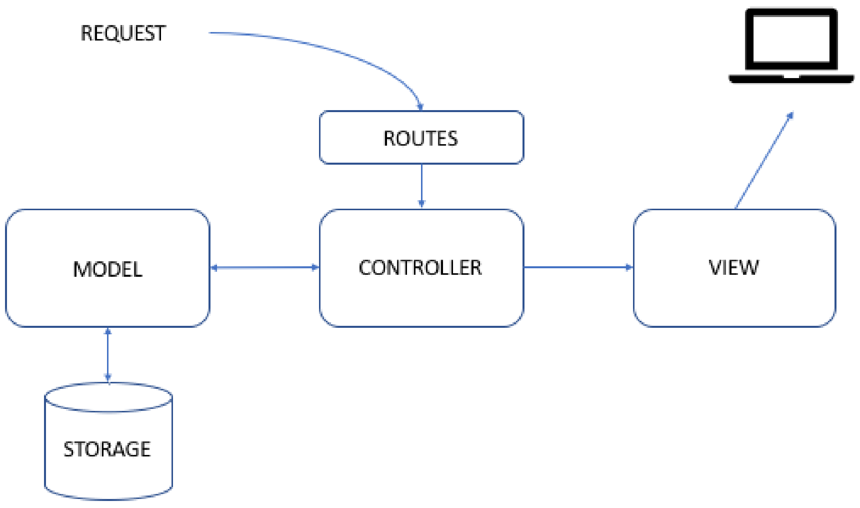


Figure 1. Request processing under the MVC paradigm

- A request is made, say a user enters a URL associated with the application or a program submits a relevant request.
- A route associated with that URL maps the URL to a Controller action
- The Controller leverages the necessary model(s) to retrieve information objects from the storage and passes the data off to a View
- The View renders the final page.

2.3 Laravel

Laravel (Laravel, 2019a) is a web application framework with expressive, elegant syntax. It lays the web application foundation, allowing the programmer to focus on the application logic and functionality, without devoting effort to the development of common and routine tasks.

2.3.1 Why use Laravel?

In the following paragraphs we outline the benefits of the Laravel framework for application development, through the description of features that the framework provides.

2.3.1.1 Authentication

A very important part of any web application is the authentication part. With the latest Laravel designs, the validation/authentication is included. Laravel has already been implemented in every software that uses controller in making route declarations by the use of syntax. Security-wise, it employs hashed passwords and does not save the password in plain text form; it has used the Bcrypt hash algorithm in creating encrypted passwords, which is adequately secure. Laravel also implements a simple method to escape user input, in order to confront SQL injection attacks.

2.3.1.2 Blade Templating Engine

Blade (Laravel, 2019a, LaravelCollective) is a simple, yet powerful templating engine provided with Laravel. Unlike controller layouts, Blade is driven by template inheritance and sections.

2.3.1.3 Support for multiple file systems

Laravel provides the native support for multiple file systems. Laravel uses third party package Flysystem to provide multiple file support. You can use any of Local or Cloud based storage by providing simple configuration. Someone is also capable to bypass the file system facade in the application and work directly with the disk facade.

2.3.1.4 Caching

Caching is a temporary data storage used to store data for a while and can be retrieved quickly. It is often used to reduce the times we need to access database or other remote services. Caching keeps your application fast and responsive. For more details on the Laravel caching implementation, the interested reader is referred to the relevant documentation (Laravel, 2019a).

2.3.1.5 Method or Dependency Injection

In Laravel Inversion of control (IoC) (Laravel, 2019a, Container) container is a powerful tool for managing class dependencies. Dependency injection is a method of removing hard-coded class dependencies. Laravel's IoC container is one of the most used Laravel features.

2.3.1.6 Modularity or Multi-app

Modularity is the degree to which a system's components may be separated and recombined. You split of the business logic into different parts, which belongs together. If you're into Domain Driven Design, you can consider a module an aggregate.

2.3.2 The "HTML" package of the Laravel Collective

When we are working with resources (i.e. different types of objects) in Laravel, it's very common to create forms to create/update those objects in your database.

And the most refined way to do this is using form model binding. Form model binding allows you to associate a form with one of your application's models, and automatically:

- Matches inputs named after model fields
- Populates the form's fields with an existing model object's data
- Repopulates the form with session data

2.3.3 Laravel Artisan Commands

Artisan is the command line interface, frequently used in Laravel and it included a set of helpful commands for developing an application.
Below, some examples of the utility commands that Artisan provides are listed:

```
php artisan serve
```

Starts the development server with the default options.

```
php artisan serve --host=host.app --port=8080
```

Changes the server address and server port

```
php artisan route:cache
```

When the command is called, an instance of `Illuminate/Routing/RouteCollection` is build. After being encoded, the serialized output is written to `bootstrap/cache/routes.php`. Application requests will always load this cache file if it exists. This speeds up our application performance.

In addition to the commands listed in Artisan, a user can also create custom commands which can be used in the application.

2.4 MariaDB

MariaDB (MariaDB, 2019a) server is one of the most popular database servers. It is made by the original developers of MySQL and guaranteed to stay open source.

MariaDB turns data into structured information in a wide array of application, ranging from banking to websites. It is an enhanced, drop-in replacement of MySQL. MariaDB is used because it is fast, scalable and robust, with rich ecosystem of storage engines, plugins and many other tools make it very versatile for a wide variety of use cases. The latest versions of MariaDB also include GIS and JSON features.

2.5 ETL Pipeline

The ETL (Qin H., Jin X. and Zhang X., 2013; Martins M., Abbasi M. and Furtado P., 2016) acronym stands for *extract, transform, load*. ETL is the general procedure of copying data from one or more sources into a destination system which represents the data differently from various sources.

2.5.1 Why do we need ETL Pipeline

It is essential to properly format and prepare data in an order to load it a destination system. The ETL pipeline provides crucial function that many times combined into a single application or a suite of tool that help in the following areas:

- Allows verification of data transformation, aggregation and calculation rules
- Allows sample data comparison between source and target system

A basic ETL process can be categorized in the following steps:

- Data Extraction
- Transformation & Data Cleansing
- Load to a target system

2.5.1.1 Data Extraction

The first part of an ETL process involves extracting the data from the source system(s). In most of cases, this step represents the most important aspect of ETL, since extracting data correctly sets the stage for success for the next processes.

2.5.1.2 Transform

In the data transformation stage, a series of rules are applied to the extracted data in order to prepare it for loading to the end target system.

Most important step is the data cleansing, which aims to pass only "proper" data to the target system.

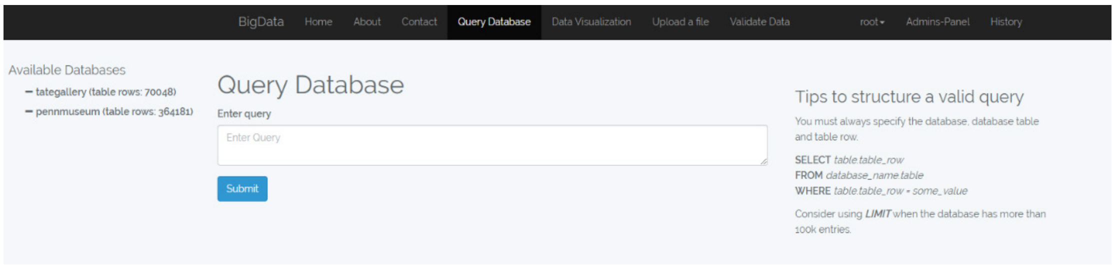
2.5.1.3 Load

The load phase loads the data into the final target source such as a data warehouse, Google Big Query, MySQL database etc.

3 The application

3.1 Query the database from the UI

The user is able to query different available databases from the interface. The interface is constructed as the following picture indicates



Available Databases

- tategallery (table rows: 70048)
- pennmuseum (table rows: 364181)

Query Database

Enter query

Tips to structure a valid query

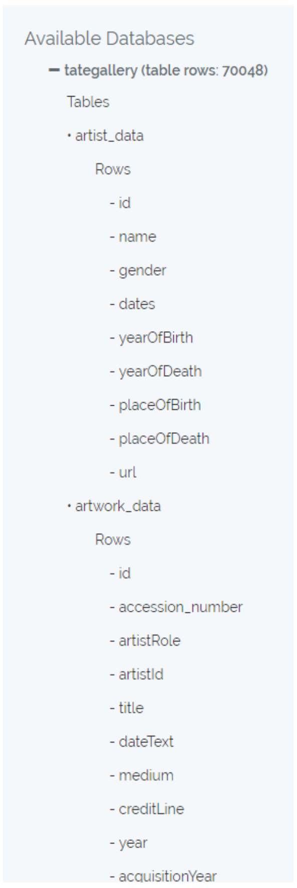
You must always specify the database, database table and table row.

SELECT *table.table_row*
FROM *database_name.table*
WHERE *table.table_row = some_value*

Consider using ***LIMIT*** when the database has more than 100k entries.

Figure 2. User interface for queries

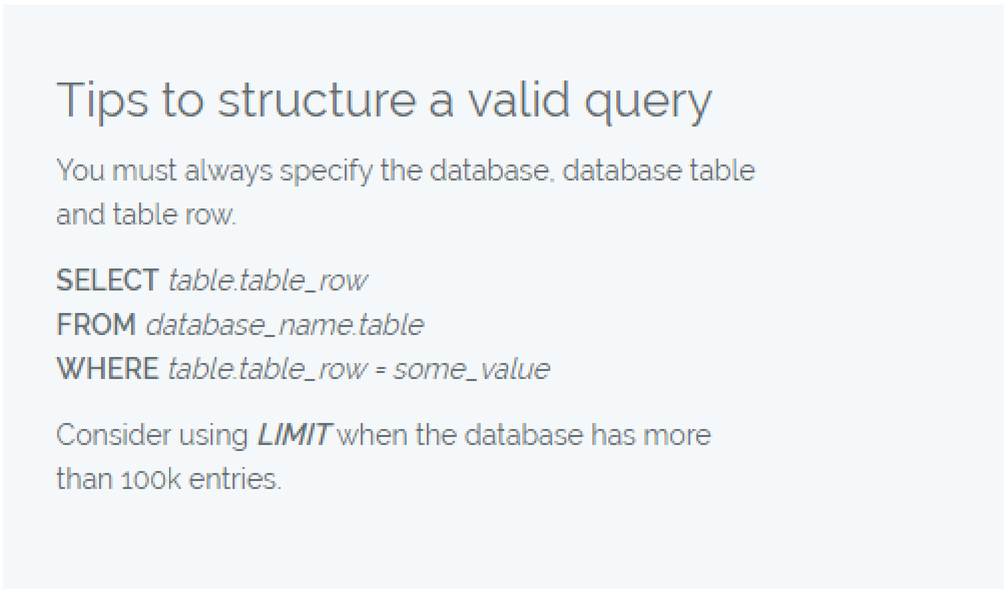
The user is able to see the structure of the available databases from the available dropdown located on the left of the screen. The dropdown contains information regarding the structure of the databases and number of the entries that they contain.



Available Databases
- tategallery (table rows: 70048)
  - Tables
  - • artist_data
    - Rows
      - - id
      - - name
      - - gender
      - - dates
      - - yearOfBirth
      - - yearOfDeath
      - - placeOfBirth
      - - placeOfDeath
      - - url
  - • artwork_data
    - Rows
      - - id
      - - accession_number
      - - artistRole
      - - artistId
      - - title
      - - dateText
      - - medium
      - - creditLine
      - - year
      - - acquisitionYear

Figure 3. List of available data collections, as provided by the application

The user must submit queries in a specific format in order to pass the validation. The valid format is displayed on the interface and it is the following



Tips to structure a valid query

You must always specify the database, database table and table row.

SELECT *table.table_row*
FROM *database_name.table*
WHERE *table.table_row = some_value*

Consider using ***LIMIT*** when the database has more than 100k entries.

Figure 4. Instructions provided to the end-user regarding the query formulation syntax

3.2 Data Visualization

To visualize our data, we used a JavaScript library that is known as D3.js. D3.js (D3js, 2019a) is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

We took into consideration the impact that has the screen clutter on the user experience when the presented screen is overload with information. In that regard we are presenting only eight random data entries from the selected data source from the user. However, the user is also has the possibility to submit its own queries and so the result will be visualized, but due to the fact that, there is no limitations applied by default at these kind of queries, the user should consider to limit the end result so the screen will not be cluttered with information.

To present our data we have used a dendrogram. A dendrogram is a diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from hierarchical clustering. The main use of a dendrogram is to work out the best way to allocate objects to clusters.

From the screenshots that follow, we able to recognize pretty easily the hierarchical connection between the presented objects despite if the origin of the random data selection (Figure 5) or a submitted query (Figure 6); the query used to retrieve the dataset depicted in Figure 6 is shown in Figure 7.

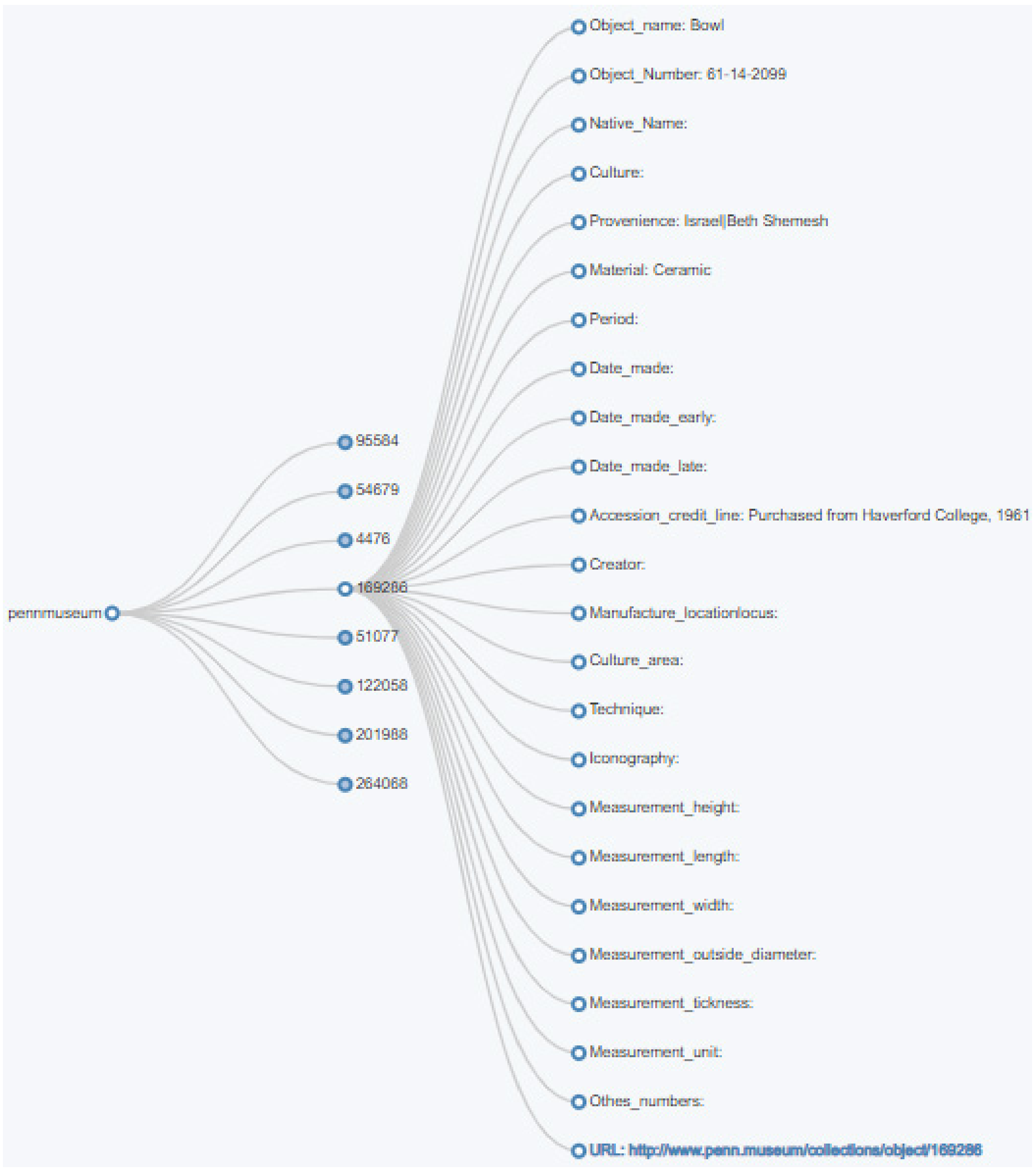


Figure 5. Hierarchical visualization of a randomly selected item collection

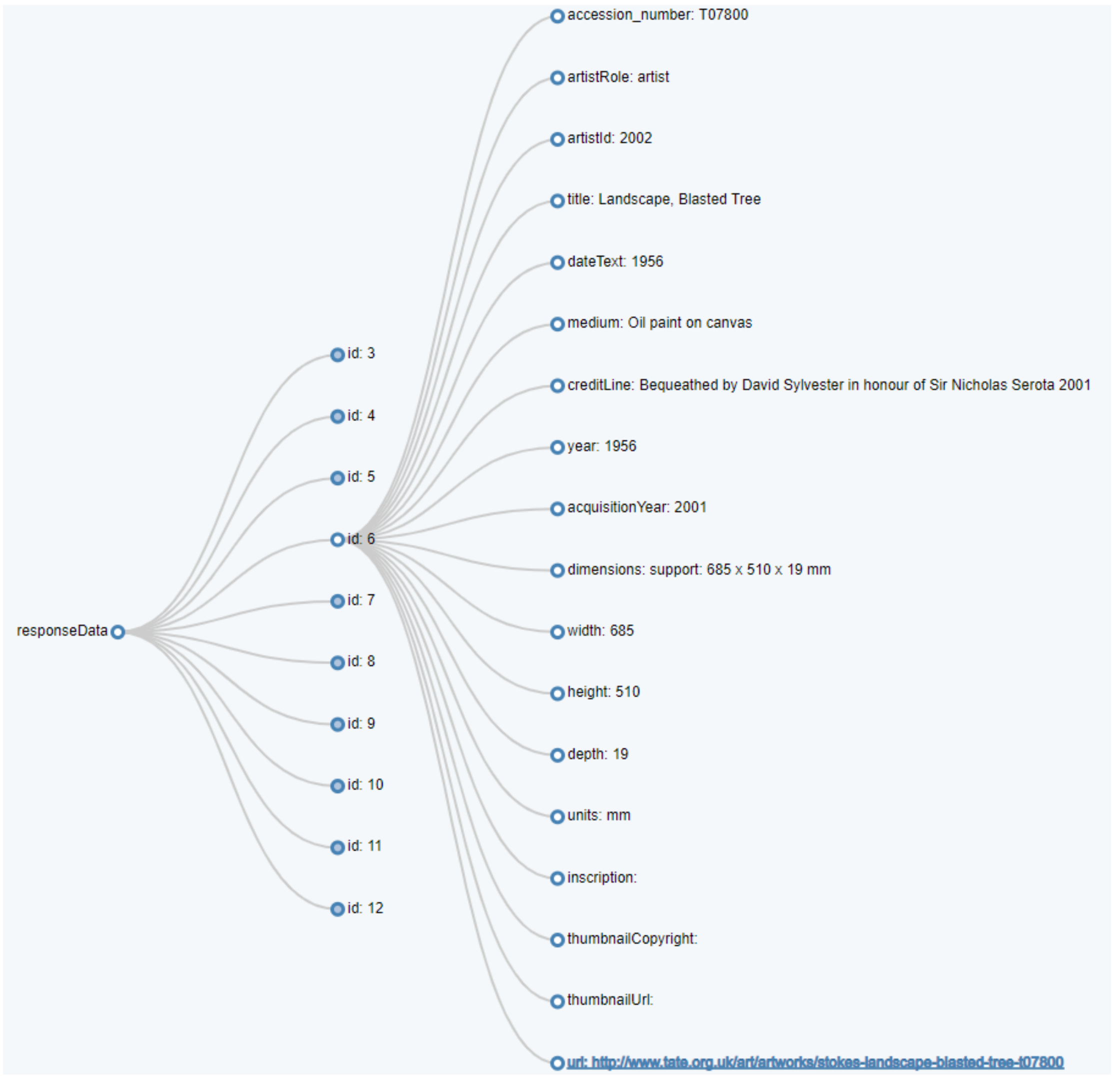
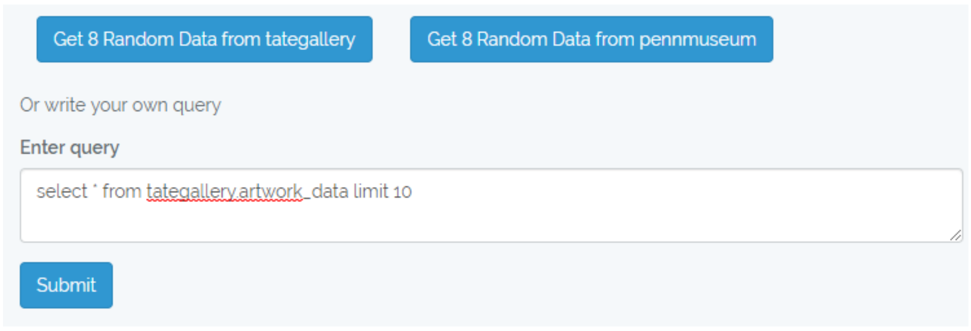

responseData
id: 3
id: 4
id: 5
id: 6
id: 7
id: 8
id: 9
id: 10
id: 11
id: 12
accession_number: T07800
artistRole: artist
artistId: 2002
title: Landscape, Blasted Tree
dateText: 1956
medium: Oil paint on canvas
creditLine: Bequeathed by David Sylvester in honour of Sir Nicholas Serota 2001
year: 1956
acquisitionYear: 2001
dimensions: support: 685 x 510 x 19 mm
width: 685
height: 510
depth: 19
units: mm
inscription:
thumbnailCopyright:
thumbnailUrl:
url: http://www.tate.org.uk/art/artworks/stokes-landscape-blasted-tree-t07800

Figure 6. Hierarchical visualization of the results of a user-provided query


Get 8 Random Data from tategallery
Get 8 Random Data from pennmuseum
Or write your own query
Enter query
Submit

```
select * from tategallery.artwork_data limit 10
```

Figure 7. User-provided query

3.2.1 Procedure to be followed in order to visualize new datasets

In the following, we report on the prescribed procedure to be followed in order to define the visualization of a new dataset. The realization of the visualization of a new dataset entails the following steps:

- Study of the structure of the new dataset, to gain insight on the most effective way to present the information to the users.
- Design of how to form correctly a formatted string, so the D3 library can render the visualization.
- Creation of an appropriate function in the visualization controller to handle the corresponding request from the user interface. In this step, the existing visualizations can be consulted, reused and tailored as needed.
- Introduce appropriate controls at user interface level, which will allow the visualization implementation to be called.
- add entry to the routes table to map the corresponding controller function and the user interface.

A sample format of a JSON string to be passed to the D3 visualization library, in order for a hierarchical visualization to be rendered is shown in Figure 8.

```
{"name":"parentNode",
   "children":[{
         "name":"childNode",
         "children":[{
               "name":"child1",
               "children":[{
                     "name":"string"
}]},
            {
               "name":"child2",
               "children":[
                  {
                     "name":"child3",
                     "children":[
                        {
                           "name":"string"
}]}]}]}]}
```

Figure 8. A sample format of a JSON string to be passed to the D3 visualization library

3.3 Data upload & validation

3.3.1 Preprocessing of data

Before populating the databases with the corresponding data, we performed the following steps

- removed redundant spaces from the entries
- replaced non utf-8 characters with the corresponding correct ones
    - characters such "Ã©" will be replaced with "é"
- replaced all single quotes with double quotes

3.3.2 Data upload

When a file is uploaded an ETL process is triggered that extracts the data from the given CSV file, transforms the data and then uploading the data to the correct database table.

The only limitations that the user has are

- the file size needs to be less than 2 MB (configurable through server settings)
- the only accepted file format is csv

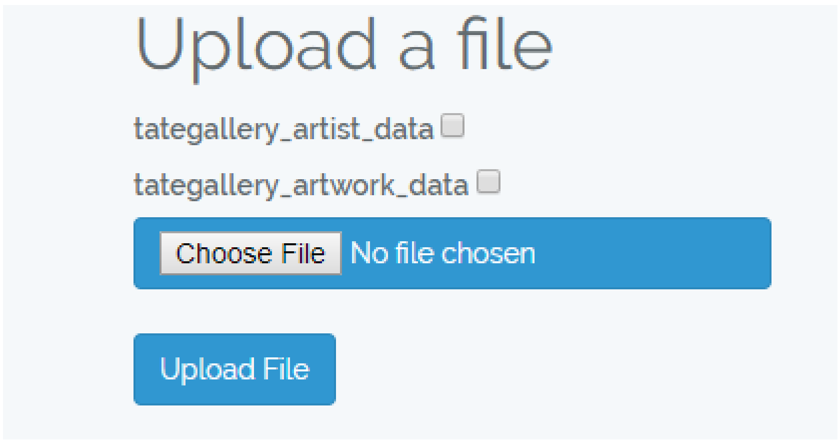


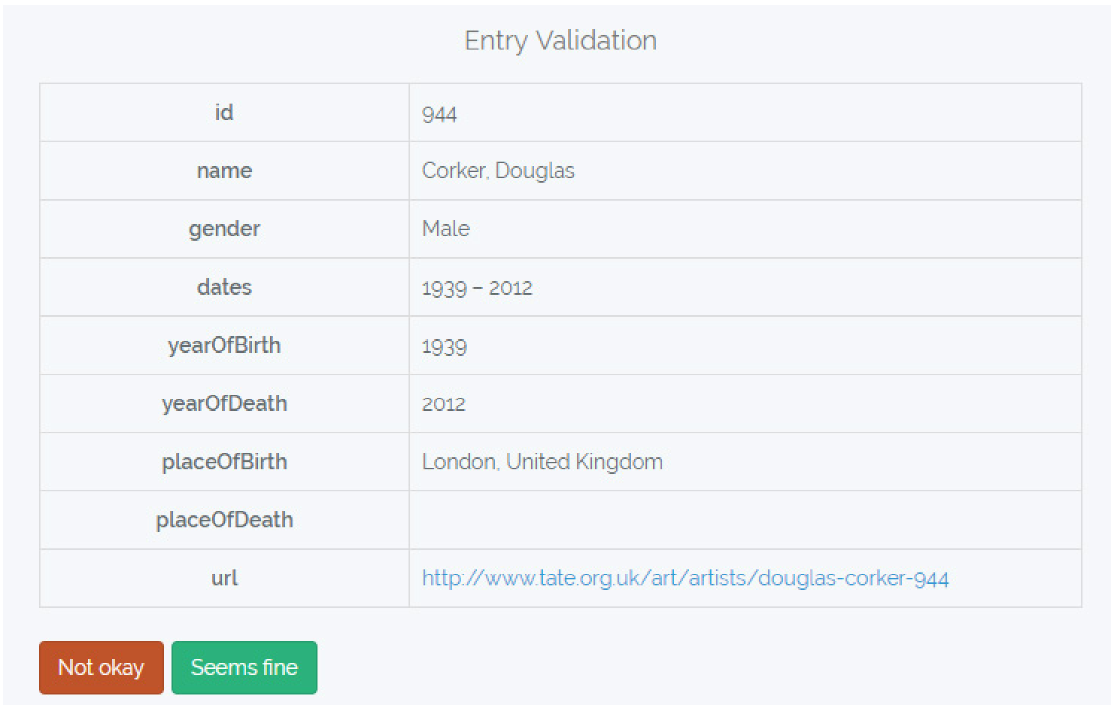
Figure 9. User interface to upload a dataset

3.3.3 Data validation

The data that the users upload does not end up to the actual database, but to a temporary one. The reasoning behind this, is because we are not able to determine in an automatic way, if the actual data that the users upload make sense or not. That is because, also, due to the nature of the data, artist names and artwork titles, we are not able to define if the provided input is actually valid or not.

In that regard we leave the data validation part to the users. We have created a panel that returns random entries, either from actual database either from the temporary one, and the users vote if the data that is presented to them make sense or not. The votes are taken into consideration only for data that originate from the temporary table. Depending on the number of negative or positive votes the corresponding entry is either removed from the temporary database or moved to the actual database.

The data is presented at the interface as is shown below



Entry Validation

id	944
name	Corker, Douglas
gender	Male
dates	1939 – 2012
yearOfBirth	1939
yearOfDeath	2012
placeOfBirth	London, United Kingdom
placeOfDeath	
url	http://www.tate.org.uk/art/artists/douglas-corker-944

Figure 10. User interface for dataset validation

3.3.4 Deepdive on the process of data validation

In order to let users, upload their own data to the *tategallery* database, we have created corresponding replicas of the database tables, that have the exact same schema with the only difference being one column that is responsible to count the validation votes from the users.

User submits a csv with the entries that wishes to end-up to the corresponding database table. When the user uploads the csv file, an ETL process is triggered which is handling the following

- the corresponding model of the table, that was chosen from the user, is used to validate the headers of the file
- corresponding entries are getting sanitized before ending up to the temporary database

After the successful upload of the data, users are able to go to corresponding validation screen within the UI and vote for the presented entries. The entry that is presented is randomly selected either from the main database or the temporary one which user entries are stored. After the user votes, if the corresponding displayed entry seems correct or not, and the entry is actually is stored in the temporary database, then a corresponding counter within the table is adjusted accordingly depending if the vote is either positive or negative. After a certain number of votes, the corresponding entry is either discarded from the corresponding table due to negative votes or moved to the main database due to positive votes.

3.3.5 Bypassing data validation

All data are uploaded to the platform via the procedure described in section 3.3.2; once the data have been uploaded, they are not released for public use until they are validated as described in section 3.3.3.

However, the following scenario may occur: a platofrm administrator uploads the data from a trusted source and wants to release the data for immediate use, bypassing the data validation step. This scenario is supported by the platform, albeit at database level only, i.e. without the support of a user interface. More specifically, the procedure is as follows:

1. When a new dataset is uploaded to the platform, a new dataset-specific database is created to host the validated data, while additionally appropriate tables are created in the base system database (the base system database is named *bigdata* by default), to host the data that are in need of validation before they are publicly released. The structure of these tables is identical to the ones created in the dataset-specific database, except the fact that they contain one one extra column named "faulty".
2. The system administrator may issue an SQL command that copies the data from the tables are created in the base system database tables into the tables in the dataset-specific database, followed by an SQL command to delete the data from the base system database tables.

To exemplify this procedure, let us consider the case of the Tate Gallery dataset. When this dataset is uploaded to the platform, a database `tategallery` is created, which hosts the tables `artist_data` and `artwork_data`, the schema of which is depicted in Figure 11 and Figure 12, respectively.

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| id           | int(11)      | NO   | PRI | NULL    |       |
| name         | varchar(255) | NO   |     | NULL    |       |
| gender       | varchar(6)   | YES  |     | NULL    |       |
| dates        | varchar(255) | YES  |     | NULL    |       |
| yearOfBirth  | int(4)       | YES  |     | NULL    |       |
| yearOfDeath  | int(4)       | YES  |     | NULL    |       |
| placeOfBirth | varchar(255) | YES  |     | NULL    |       |
| placeOfDeath | varchar(255) | YES  |     | NULL    |       |
| url          | varchar(255) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

Figure 11. Schema of table artist_data

```
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| id                 | int(11)      | NO   | PRI | NULL    |       |
| accession_number   | varchar(11)  | NO   |     | NULL    |       |
| artistRole         | varchar(255) | YES  |     | NULL    |       |
| artistId           | int(11)      | NO   | MUL | NULL    |       |
| title              | text         | YES  |     | NULL    |       |
| dateText           | varchar(255) | YES  |     | NULL    |       |
| medium             | varchar(255) | YES  |     | NULL    |       |
| creditLine         | text         | YES  |     | NULL    |       |
| year               | varchar(8)   | YES  |     | NULL    |       |
| acquisitionYear    | int(4)       | YES  |     | NULL    |       |
| dimensions         | varchar(255) | YES  |     | NULL    |       |
| width              | varchar(12)  | YES  |     | NULL    |       |
| height             | varchar(10)  | YES  |     | NULL    |       |
| depth              | int(4)       | YES  |     | NULL    |       |
| units              | tinytext     | YES  |     | NULL    |       |
| inscription        | varchar(255) | YES  |     | NULL    |       |
| thumbnailCopyright | text         | YES  |     | NULL    |       |
| thumbnailUrl       | varchar(255) | YES  |     | NULL    |       |
| url                | varchar(255) | YES  |     | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
```

Figure 12. Schema of table artwork_data

Correspondingly, in the `bigdata` database, two matching tables are created, named `tategallery_artist_datas` and `tategallery_artwork_datas`, which have identical schemas to the ones presented in Figure 11 and Figure 12, respectively, with the exception of an additional column named `faulty`, whose type is `int(11)`. Now, if the administrator wants to bypass the validation stage for the data in the `bigdata.tategallery_artist_datas` table, s/he may issue the following SQL command:

```
insert into tategallery.artist_data(id, name, gender,
        dates, yearOfBirth, yearOfDeath, placeOfBirth,
        placeOfDeath, url)
select id, name, gender,
        dates, yearOfBirth, yearOfDeath, placeOfBirth,
        placeOfDeath, url
from bigdata.tategallery_artist_datas;
```

Figure 13. Releasing data for public use, bypassing validation

Finally, the system administrator should remove the data from the `bigdata.tategallery_artist_datas` table, since this data has already been released; to this end, s/he may issue the following SQL command:

```
Delete from bigdata.tategallery_artist_datas;
```

Figure 14. Releasing data for public use, bypassing validation

3.4 Deepdive on how the query from the UI is working

On this section we will expand on in need topics that were used and investigated in order to make this functionality possible. More specifically, we discuss the issues of SQL injection handling and pagination.

3.4.1 SQL Injection

SQL Injection (Devi, Venkatesan, Koteeswaran, 2016; Sadegh Sajjadi and Pour, 2013) is a code injection technique, used to attack data-driven applications, in which SQL statements are inserted into an entry field for execution. SQL injection must exploit a security vulnerability in an application’s software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed.

SQL injection attacks allow attackers to spoof identities, tamper with existing data and databases, cause repudiation issues such as voiding transactions or changing balances, destroy the data, install backdoors to the database etc.

3.4.1.1 Common SQL injection cases

Always true statements

This injection method occurs when the input is not filtered for escape characters (characters such as ";") and is then passed into a SQL statement.

For example, let's assume we have the following SQL statement

```
SELECT * FROM table_users WHERE user_name = '" + name + "';
```

However, the designed query is poorly designed and the SQL statement may do more than the author intended. For instance, the following two examples effectively list all the entries from the table

```
SELECT * FROM table_users WHERE user_name = "" OR TRUE;
```

```
SELECT * FROM table_users WHERE user_name = "" OR 1=1;
```

and here is a more malicious way, the SQL-injected query deletes the corresponding table

```
SELECT * FROM table_users WHERE user_name = ""; DROP table_users;
```

3.4.1.2 Handling basic cases of SQL injection

In order to handle the basic SQL injections cases, some regular expressions are employed to validate the input queries from the users. A regular expression, is a sequence of characters that define a search pattern. Regular expressions are usually used by string searching algorithms to identify or identify and replace operations on string, or input validation. Effectively, the regular expressions correspond to textual patterns in query texts which typically indicate an SQL injection attack. If the query text matches any of the regular expressions, the query is not executed and a relevant error message is displayed to the user.

The following regular expression checks for statements such as INSERT INTO, CREATE DATABASE and CREATE TABLE

- `/([dD][rR][oO][pP](\s)[tT][aA][bB][lL][eE])|([dD][rR][oO][pP](\s)[dD][Aa][tT][aA][bB][aA][sS][eE])|([tT][rR][uU][nN][cC][aA][tT][eE](\s)[tT][aA][bB][lL][eE])/`

Checking for statements such as `where true`, `or true`, `where 1`, `or 1`, `“string” is not null`, `null is null`

- `/([iI][nN][sS][eE][rR][tT]\s[iI][nN][tT][oO])|([cC][rR][eE][aA][tT][eE]\s[dD][aA][tT][aA][bB][aA][sS][eE])|([cC][rR][eE][aA][tT][eE]\s[tT][aA][bB][lL][eE])/`

The following regular expression is checking for always true WHERE clauses between strings in statements.

- `/(\b\S+)\s*[\=]{1}\s*\1\b/`

To provide more context on this matter, the described expression is able to handle cases such as

```
- select * from table where 1=1
- select * from table where a=a
```

3.4.1.3 Limitations of the SQL handling methods

The described methods, are able to handle the basic SQL injection cases but there are also limited in their functionality. More specifically, we are able to track always true statements where the two parts of a where clause are exact same. So, we are able to detect cases like

```
  - select * from table where 1=1
```

but the developed functionality is not able to detect semantically identical cases. Hence, the system is not able to detect cases like

```
  - select * from table where 20 = 5*4
```

Some false positives may also be flagged, e.g. when the "drop table" or "drop database" literal occurs within a quoted string; such cases however are expected to be rare.

3.4.2 Pagination

Pagination, also known as paging, is the process of dividing digital content into discrete pages. Pagination, is a mechanism which provides users with additional navigation options for browsing through single parts of the digital content.

In the most cases, pagination refers to the automated process of adding consecutive numbers to identify the sequential order of pages.

3.4.2.1 Laravel's Pagination

Laravel's paginator (Laravel, 2019a Pagination) is integrated with the query builder and Eloquent ORM and provides convenient pagination of the database results out of the box.

There are several ways to paginate items. The simplest way is by using the paginate method on the query builder or an Eloquent Query. The paginate method automatically takes care of setting the proper limit and offset based on the current page being viewed by the user. By default, the current page is detected by the value of the page query string argument on the HTTP request.

In order to serve our needs, we have created a pagination instance manually, and to be exact a LengthAwarePaginator, that we pass an array of items. The LengthAwarePaginator constructor requires the following parameters in order to be initialized properly

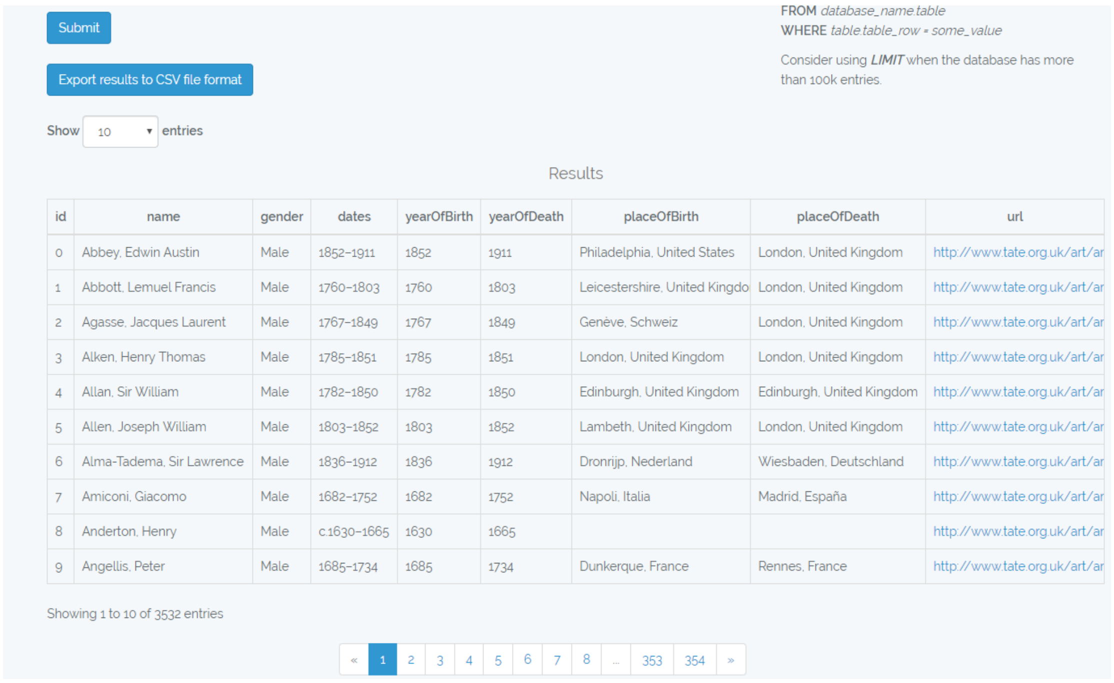
- an array of elements
- the number of elements within the array
- the items per page that we want to present
- an array of items that contain the URL and query parameters (this is the most essential parameter if we do not want to lose data when we will be navigating from the page to page of the paginated items)

3.4.2.2 Displaying Paginated Results

Due to the fact that the application does not limit the user to query certain datasets, our View display should be able to present the result set to the end user in a structured manner, regardless of the number of selected columns.

In that regard our View, iterates over the returned paginated data and creates an html table with the corresponding data and data headers. Also, the View provides helpful information such as the number of elements showing from the result set, the number of the active page etc. Last but not least, the user is able to change the number of items that is displayed and also extract the result to CSV file format.

To illustrate the above, let us check the following results from a submitted query:



FROM *database_name.table*
WHERE *table.table_row = some_value*

Consider using ***LIMIT*** when the database has more than 100k entries.

Submit

Export results to CSV file format

Show 10 entries

Results

id	name	gender	dates	yearOfBirth	yearOfDeath	placeOfBirth	placeOfDeath	url
0	Abbey, Edwin Austin	Male	1852–1911	1852	1911	Philadelphia, United States	London, United Kingdom	http://www.tate.org.uk/art/ar
1	Abbott, Lemuel Francis	Male	1760–1803	1760	1803	Leicestershire, United Kingdo	London, United Kingdom	http://www.tate.org.uk/art/ar
2	Agasse, Jacques Laurent	Male	1767–1849	1767	1849	Genève, Schweiz	London, United Kingdom	http://www.tate.org.uk/art/ar
3	Alken, Henry Thomas	Male	1785–1851	1785	1851	London, United Kingdom	London, United Kingdom	http://www.tate.org.uk/art/ar
4	Allan, Sir William	Male	1782–1850	1782	1850	Edinburgh, United Kingdom	Edinburgh, United Kingdom	http://www.tate.org.uk/art/ar
5	Allen, Joseph William	Male	1803–1852	1803	1852	Lambeth, United Kingdom	London, United Kingdom	http://www.tate.org.uk/art/ar
6	Alma-Tadema, Sir Lawrence	Male	1836–1912	1836	1912	Dronrijp, Nederland	Wiesbaden, Deutschland	http://www.tate.org.uk/art/ar
7	Amiconi, Giacomo	Male	1682–1752	1682	1752	Napoli, Italia	Madrid, España	http://www.tate.org.uk/art/ar
8	Anderton, Henry	Male	c.1630–1665	1630	1665			http://www.tate.org.uk/art/ar
9	Angellis, Peter	Male	1685–1734	1685	1734	Dunkerque, France	Rennes, France	http://www.tate.org.uk/art/ar

Showing 1 to 10 of 3532 entries

« 1 2 3 4 5 6 7 8 … 353 354 »

Figure 15. Paged query results

We can observe that the user is able to spot pretty easily the amount of data that is returned and the number of the page that currently is getting displayed.

4 System Administration Functionalities

4.1 User levels

The application has two different user levels

- admin user level
- basic user level

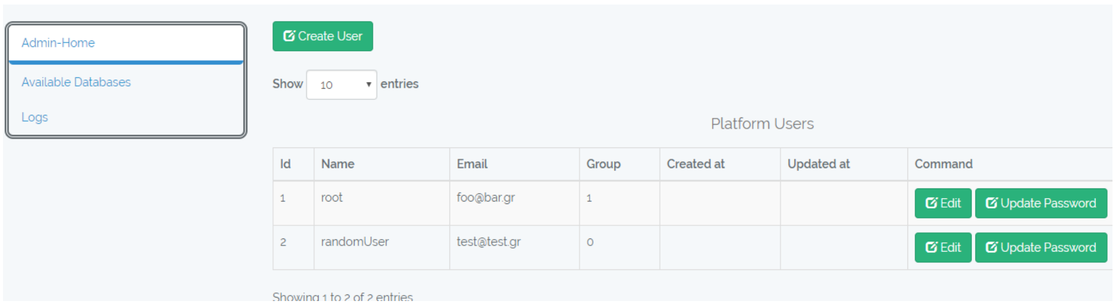
Both user levels have the ability submit queries to the available databases, upload data to the one of the databases, see a visualization dendrogram of the two available databases.

4.1.1 Admin user

The Admin user class has more options available within the application in comparison with the default users

4.1.1.1 User creation

The admin users are able to view list with all the users that are registered to the application; more specifically, admin users are provided with functionalities to create users, update users' passwords, emails, change user level.

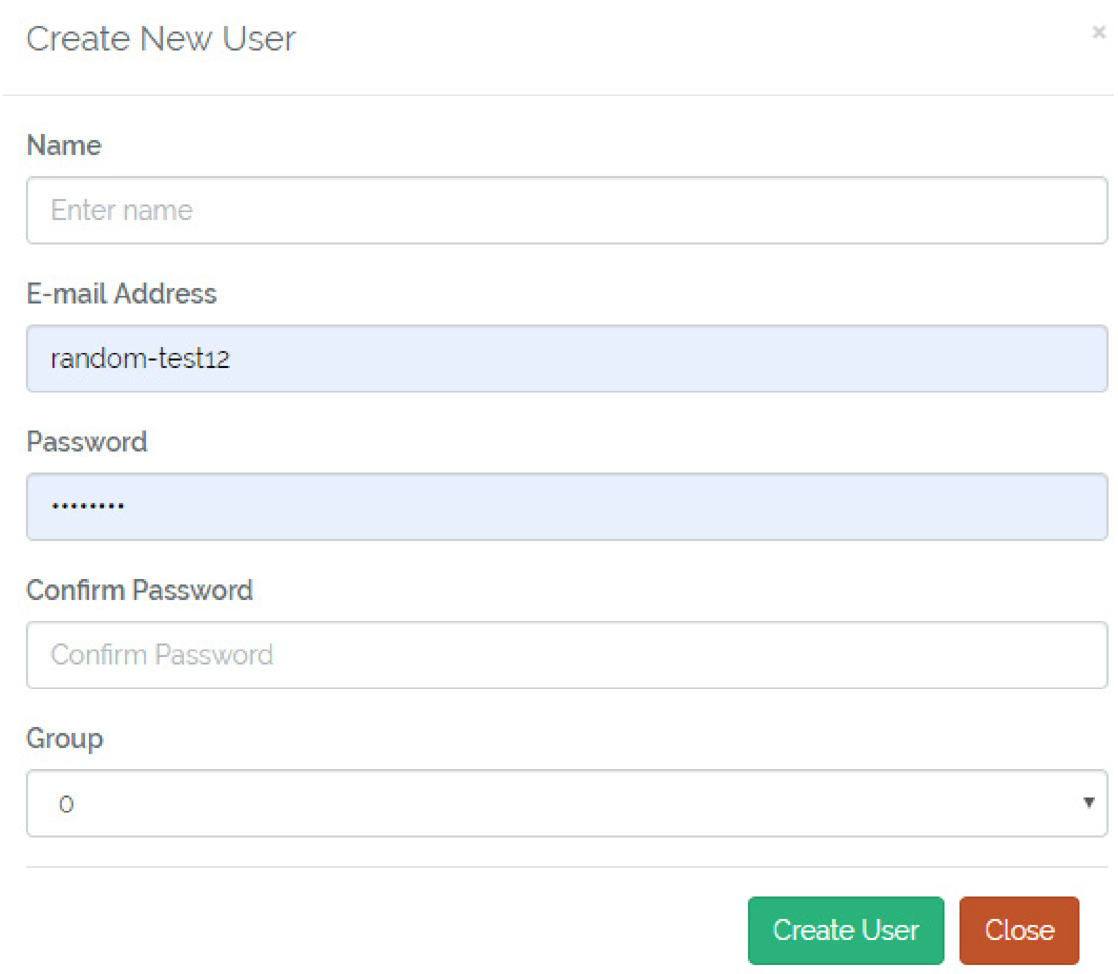


Platform Users

Id	Name	Email	Group	Created at	Updated at	Command
1	root	foo@bar.gr	1			Edit Update Password
2	randomUser	test@test.gr	0			Edit Update Password

Showing 1 to 2 of 2 entries

Figure 16. User management


Create New User

Name: Enter name

E-mail Address: random-test12

Password: ••••••••

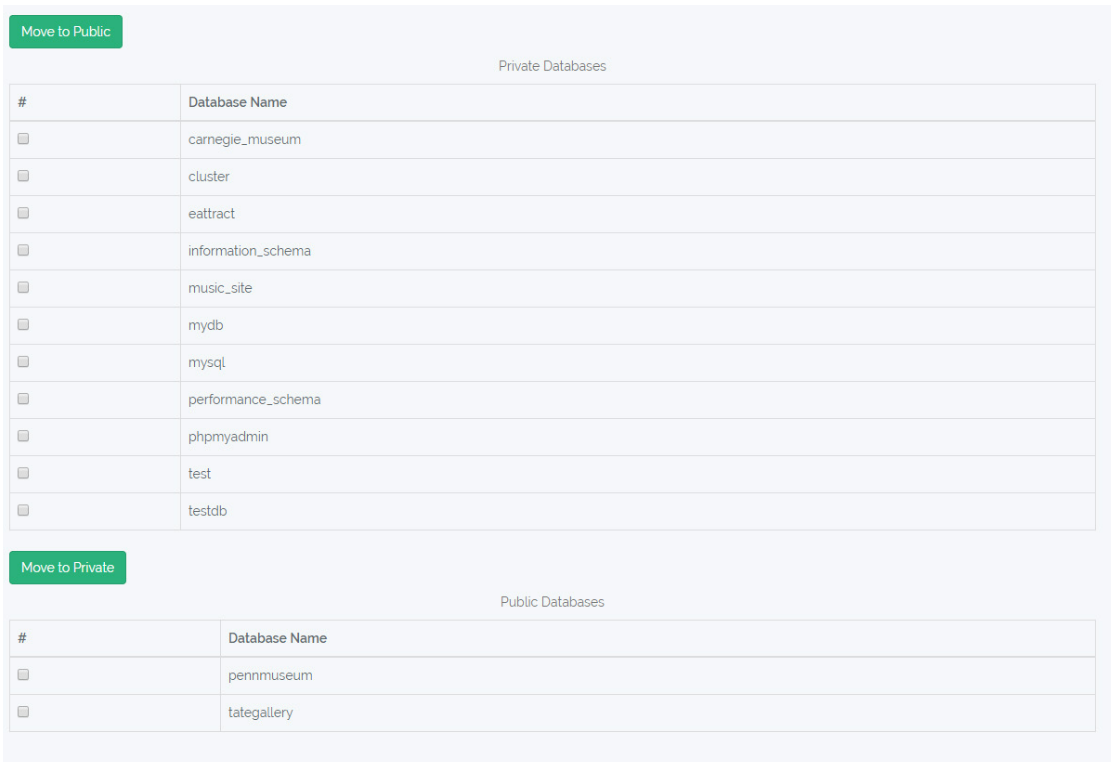
Confirm Password: Confirm Password

Group: 0

Figure 17. New user creation screen

4.1.1.2 Database access rights

The admin is able to specify which databases users are able to query. Admin is able to add or remove the database availability.



Private Databases

#	Database Name
☐	carnegie_museum
☐	cluster
☐	eattract
☐	information_schema
☐	music_site
☐	mydb
☐	mysql
☐	performance_schema
☐	phpmyadmin
☐	test
☐	testdb

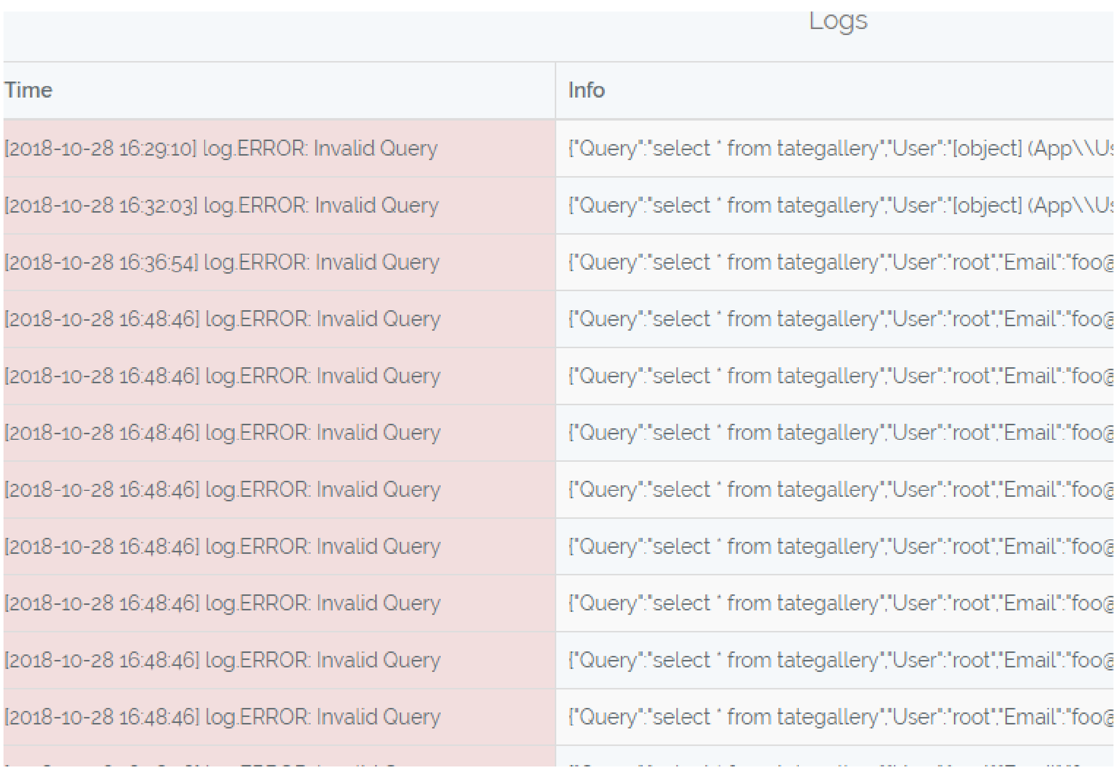
Public Databases

#	Database Name
☐	pennmuseum
☐	tategallery

Figure 18. Specifying database access permissions

4.1.1.3 Check logs

Admins have the option to check the logs of the submitted queries from all users. Malicious and suspicious queries are marked in red



Logs

Time	Info
[2018-10-28 16:29:10] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"[object] (App\\U
[2018-10-28 16:32:03] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"[object] (App\\U
[2018-10-28 16:36:54] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@
[2018-10-28 16:48:46] log.ERROR: Invalid Query	{"Query":"select * from tategallery","User":"root","Email":"foo@

Figure 19. Log examination by admin users

5 Conclusion

In order to achieve the outcome of the application, different technologies and concepts had to be investigated in order to make the different aspects of the application such as Laravel, SQL injection, D3.js and such. Thus, a flexible user interface was created to present a visualization of the two databases, Tate Gallery and Penn Museum, and also dynamic solutions were provided so the application user would be able to query the available databases from the interface and the data validation part.

We note here that there were no correlations between our two data sources due to the high degree of dissimilarity between them. In that regard no meaningful statistics could be extracted.

6 References

Devi, R. Venkatesan, R. Koteeswaran, R. (2016). A study on SQL injection techniques. International Journal of Pharmacy and Technology 8(4).

D3js (2019a). Available at https://d3js.org (accessed 2019-09-14)

Laravel (2019a). Available at https://laravel.com/docs/6.x (accessed at 2019-09-15)

Laravel (2019a). Blade. Available at https://laravel.com/docs/5.8/blade (accessed at 2019-09-15)

Laravel (2019a). Cache. Available at https://laravel.com/docs/5.8/cache (accessed 2019-09-12)

Laravel (2019a). Cache. Available at https://laravel.com/docs/5.8/cache (accessed 2019-09-12)

Laravel (2019a). LaravelCollective. Available at https://laravelcollective.com/docs/6.0/html (accessed 2019-09-15)

Laravel (2019a). Pagination. Available at https://laravel.com/docs/5.8/pagination (accessed 2019-09-15)

Leff A. and Rayfield J. T. (2001). Web-application development using the Model/View/Controller design pattern. Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, Seattle, WA, USA, 2001, pp. 118-127. doi: 10.1109/EDOC.2001.950428

MariaDB (2019a). Available at https://mariadb.com/kb/en/library/documentation/ (accessed 2019-09-15)

Martins M., Abbasi M. and Furtado P. (2016): ELASTIC PERFORMANCE FOR ETL+Q PROCESSING. Available at https://www.academia.edu/22964397/ELASTIC_PERFORMANCE_FOR_ETL_Q_PROCESSING (accessed at 2019-09-14)

Majeed A. and Rauf, I. (2018) MVC Architecture: A Detailed Insight to the Modern Web Applications Development. Peer Review Journal of Solar & Photoenergy Systems. Available at https://crimsonpublishers.com/prsp/pdf/PRSP.000505.pdf (accessed 2019-09-12)

https://ieeexplore.ieee.org/document/6406933 (accessed 2019-09-14)

PHP (2019a). Available at https://www.php.net/ (accessed 2019-09-15)

Sadegh Sajjadi S.M. and Pour B.T. (2013). Study of SQL Injection Attacks and Countermeasures, International Journal of Computer and Communication Engineering, Vol. 2, No. 5, September 2013

Qin H., Jin X. and Zhang X. (2013). Research on Extract, Transform and Load (ETL) in Land and Resources Star Schema Data Warehouse. Available at