



SCHOOL OF ENGINEERING
DEPARTMENT OF INFORMATION AND
ELECTRONIC ENGINEERING

Thesis
Research Publications Organizer



Student
Tsourdou Georgios Christos
Student ID: 185302

Supervisor
Antonis Sidiropoulos
Associate Professor

Thessaloniki 25/5/2024

Thesis Title Research Publications Organizer
Thesis Code 23243
Student's Full Name Georgios Christos Tsourdiou
Supervisor's Full Name Antonis Sidiropoulos
Date of undertaking 10-09-2023
Date of completion 25-05-2024

I hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance I received in its composition. I have, furthermore, noted the various sources from which I extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, I affirm the exclusive composition of this paper by me only, for the purpose of it being a thesis, in the Department of Information and Electronic Engineering of the I.H.U.

This paper constitutes the intellectual property of Georgios – Christos Tsourdiou, the student that composed it. According to the open-access policy, the author/composer offers the International Hellenic University authorization to use the right to reproduce, borrow, publicly present, and digitally distribute the paper globally, in electronic form and media of all kinds, for teaching or research purposes, voluntarily. Open access to the full text, by no means grants the right to trespass the intellectual property of the author/composer, nor does it authorize the reproduction, republication, duplication, selling, commercial use, distribution, publication, downloading, uploading, translation, modification of any kind, in part or summary of the paper, without the explicit written consent of the author.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Preface

The reason I chose this project was because it was an interesting one and large in scope, making it a good challenge. It allowed me to test and apply all my front-end and back-end skills. I learned a lot through this project, gaining hands-on experience with a variety of new technologies and development best practices. Additionally, the project has real-world applications, as the application will be used by my university department even after I graduate. This means I have contributed something lasting and useful to the university. Working on this project has been highly rewarding, providing me with valuable insights into software development and project management. It has also allowed me to collaborate with peers, further enriching my learning experience. Overall, this project has been a significant step in my academic and professional journey, equipping me with the skills and confidence to take on future challenges in the field of software development.

Περίληψη

Ο στόχος αυτής της διπλωματικής εργασίας είναι η ανάπτυξη ενός συστήματος που συλλέγει και οργανώνει με ημιαυτόματο τρόπο τις δημοσιεύσεις των ερευνητών, συλλέγει διάφορα προφίλ και στατιστικά δεδομένα για τους συγγραφείς, τους οργανώνει σε ομάδες και δημιουργεί στατιστικά δεδομένα για κάθε ομάδα. Οι κύριοι χρήστες του συστήματος είναι ερευνητές που ανήκουν σε διάφορες ομάδες, όπως εργαστήρια, τμήματα, ιδρύματα και τομείς. Ο στόχος είναι το σύστημα να χρησιμοποιηθεί από το Τμήμα Πληροφορικής, τα ερευνητικά εργαστήρια και τελικά από ολόκληρο το ίδρυμα ως ένα αξιόπιστο κεντρικό σύστημα καταγραφής δημοσιεύσεων.

Θα αναπτυχθεί μια διαδικτυακή εφαρμογή για την εμφάνιση των συγκεντρωτικών δεδομένων και την περιήγηση στη βάση δεδομένων από τους χρήστες. Αυτή η εφαρμογή θα λειτουργεί επίσης ως πύλη εισόδου για τους ερευνητές, οι οποίοι θα πρέπει να καταχωρίσουν βασικές πληροφορίες προφίλ και τις σχέσεις τους με τις ομάδες. Στη συνέχεια, οι ερευνητές θα εισάγουν τους μοναδικούς τους ταυτοποιητές, όπως το ORCID, το ResearcherID και το ScopusID. Χρησιμοποιώντας τα API αυτών των ερευνητικών βάσεων δεδομένων, το σύστημα θα ανακτά τη λίστα των δημοσιεύσεών τους και θα τις αποθηκεύει στη βάση δεδομένων. Για τις δημοσιεύσεις που δεν βρίσκονται μέσω αυτών των API, οι ερευνητές μπορούν να εισάγουν χειροκίνητα τα DOI και το σύστημα θα αντλεί τα υπόλοιπα στοιχεία μέσω του API του DOI.org.

Η διαδικασία ανάπτυξης χωρίζεται σε έξι φάσεις: διερεύνηση και καταγραφή της σχετικής βιβλιογραφίας και των υφιστάμενων συστημάτων· μελέτη της δομής και των μορφών αποθήκευσης βιβλιογραφικών δεδομένων· κατανόηση και οργάνωση των δεδομένων· σχεδιασμός του συστήματος και ανάλυση απαιτήσεων· υλοποίηση του συστήματος· και πιλοτική λειτουργία του συστήματος.

Ένα σημαντικό στοιχείο του συστήματος είναι ο αλγόριθμος συγχώνευσης δημοσιεύσεων, ο οποίος συνενώνει πληροφορίες από διαφορετικές πηγές, όπως το Scopus και το ORCID. Το σύστημα υλοποιείται χρησιμοποιώντας το πλαίσιο Laravel για το WebAPI, το React για τη διεπαφή χρήστη και άλλες συμπληρωματικές βιβλιοθήκες.

Research Publications Organizer

Giorgos Christos Tsourdiou

Abstract

The objective of this thesis is to develop a system that semi-automatically collects and organizes the publications of researchers, collects various profile and statistical data for the authors, organizes them into groups, and generates statistical data for each group. The primary users of this system are researchers who belong to various groups such as laboratories, departments, institutions, and sectors. The goal is for the system to be used by the Department of Informatics, research laboratories, and eventually the entire institution as a reliable central system for recording publications.

A web application will be developed to display aggregated data and allow users to browse the database. This application will also serve as the entry point for researchers, who will be required to enter basic profile information and their affiliations with groups. Researchers will then provide their unique identifiers, such as ORCID, ResearcherID, and ScopusID. Using the APIs of these research databases, the system will retrieve the list of their publications and store them in the database. For publications not found through these APIs, researchers can manually input the DOIs, and the system will retrieve the remaining information via the DOI.org API.

The development process is divided into six phases: investigating and documenting relevant literature and existing systems; studying the structure and storage formats of bibliographic data; understanding and organizing the data; designing the system and analyzing requirements; implementing the system; and conducting a pilot operation.

A significant aspect of the system is the publication merging algorithm, which consolidates information from different sources, such as Scopus and ORCID. The system is implemented using the Laravel framework for the WebAPI, React for the user interface, and other supplementary libraries.

Acknowledgments

In this section, I would like to express my sincere gratitude to those who contributed to the completion of this thesis. First and foremost, I would like to thank my supervisor, Antonis Sidiropoulos, for his unwavering support, invaluable advice, and guidance throughout my research. Without his scientific guidance and professional suggestions, this work would not have been completed in the same way.

Additionally, a big thank you to my fellow colleagues who supported me morally and encouraged me during difficult times. The discussions with them and the perspectives they shared were crucial for the progress of my thesis.

I would also like to thank my family for their unwavering support and understanding throughout my studies. Without their financial and emotional support, I would not have been able to dedicate myself to my studies with the same dedication and energy.

Lastly, I thank all my friends and colleagues for their understanding and support. Each of their contributions was valuable and significantly helped in the completion of this thesis.

Thank you all from the bottom of my heart.

Contents

Preface.....	iii
Περίληψη.....	iv
Abstract	vi
Acknowledgments	vii
Table of Figures	xii
List of Tables.....	xiv
Chapter 1 : Introduction	15
1.1 Organizing Research Publications.....	15
1.2 Utilizing a local Research Publication Database	15
1.2.1 Utilization by students.....	15
1.2.2 Utilization by academics	15
1.3 Wider Impacts and Benefits of the Local Database	15
1.4 Conclusions	16
Chapter 2 : Technological Stack	17
2.1 Database Selection	17
2.2 MariaDB vs MySQL	17
2.3 Back End	18
2.3.1 Laravel.....	18
2.3.2 Socialite	19
2.3.3 Eloquent.....	19
2.3.4 Routing	19
2.3.5 Form Requests	19
2.3.6 Conclusion.....	20
2.4 Front End.....	20
2.4.1 React.....	20
2.4.2 Tailwind CSS	20
2.4.3 Flowbite React.....	21
2.4.4 Material UI	21
2.4.4 Chart.js	21
2.4.5 Inertia.js.....	21
2.4.6 Lodash	22
2.4.7 Prop-types.....	22
2.4.8 CLSX.....	22

2.4.9 useHooks	22
2.4.9 React Icons	23
2.5 Data Sources.....	23
2.5.1 OpenAlex.....	23
2.5.2 ORCID.....	24
2.5.3 Scopus	25
2.5.4 Crossref / DOI	25
2.6 Conclusion.....	26
Chapter 3 : Database Design	27
3.1 Introduction	27
3.2 Defining the entities	27
3.3 User	28
3.3.1 User Categories	29
3.3.2 Permissions.....	29
3.3.3 User activities	29
3.3.4 How user actions affect the database.....	29
3.4 Author.....	30
3.5 Work.....	30
3.6 Group.....	31
3.7 Statistic	31
3.8 Type.....	32
3.9 Concept.....	32
3.10 Migrating the database	33
Chapter 4 : Database Management.....	35
4.1 Request Handling Infrastructure	35
4.1.1 Source-Specific Controller Classes and Configuration	35
4.1.2 OpenAlex API Controller.....	36
4.1.3 ORCID API Controller.....	37
4.1.4 DOI API Controller	38
4.2 Initialize Database Job.....	38
4.2.1 Managing Different Identifier Types.....	39
4.2.2 Processing the OpenAlex response	40
4.2.3 ORCID and Crossref Integration.....	40
4.3 Update Database Job	41
Chapter 5 : Back-End Infrastructure and API	42

5.1 Requests Utility Class	42
5.1.1 Success Response Function.....	42
5.1.2 Client Error Response Function	43
5.1.3 Server Error Response Function.....	43
5.1.4 Missing Parameter Error Function	43
5.1.5 Authentication Error Function.....	44
5.1.6 Authorization Error Function	44
5.2 API Resources	45
5.2.1 Author Resource.....	45
5.2.2 Work Resource	45
5.2.3 PaginatedWorkCollection Resource.....	46
5.2.4 WorkCollection Resource	46
5.2.5 User Resource.....	47
5.2.6 Group Resource	47
5.2.6 Statistic Resource	48
5.3 Form Requests.....	48
5.3.1 Add Group Member Request Validation.....	49
5.3.2 Remove Group Member Request	49
5.3.3 Create Group Request Validation.....	50
5.3.4 Delete Group Request Validation.....	50
5.3.5 Work Filter Request Validation.....	50
5.4 Page-Specific Controller Actions and APIS.....	51
5.4.1 Home page.....	51
5.4.2 Author page	52
5.4.3 Work page	52
5.4.4 Groups page.....	52
5.4.5 Successful login page	54
5.5 Common APIs: Shared across multiple components	56
5.5.1 General Search	56
5.5.2 Search Authors by Identifiers.....	57
5.5.3 Search Where Not In Group.....	58
5.5.4 Work Filter	59
5.5.5 Get All Groups	60
5.5.6 Get Group.....	61
5.5.7 Create Group	62

5.5.8 Delete Group	62
5.5.9 Add Group Member	63
5.5.10 Remove Group Member	63
5.5.11 Get Groups Min Info	64
5.5.12 Get OMEA Authors Stats	64
5.5.13 Get OMEA Type Stats	65
5.5.14 Get Works Metadata.....	65
5.5.14 Toggle Work Visibility	66
5.5.16 Get Hidden Works.....	66
Chapter 6 : Front-End Pages and Components.....	68
6.1 Front-End Infrastructure.....	68
6.1.1 Hooks and State Management	68
6.1.2 Event System.....	70
6.1.3 API System.....	71
6.2 Application pages	71
6.2.1 Home page.....	71
6.2.2 Author page	76
6.2.3 Work page	79
6.2.4 Groups page.....	81
6.2.5 Successful login page	86
6.2.6 VerifyAuthorIdentifiers.....	86
6.3 Shared core components.....	88
6.3.1 Navigation:	88
6.3.2 Off-Canvas:	89
6.3.3 Author Item	90
6.3.4 Work Item.....	90
6.3.4 Search	92
6.3.5 Paginated List.....	93
6.4 Conclusion.....	94
Chapter 7 : Conclusion and Future Expansion	95
Challenges Faced.....	95
Future Expansion Plans	96
Conclusion.....	97
References	98

Table of Figures

Figure 2.1: OpenAlex logo.....	23
Figure 2.2: ORCID logo.....	24
Figure 2.3: Scopus logo.....	25
Figure 2.4: Crossref / DOI logo.....	26
Figure 3.1: ER (Entity-relationship) diagram of my application's database.....	28
Figure 6.1: Authors overview card in the home page.....	72
Figure 6.2: Works overview card in the home page.....	72
Figure 6.3: Top users by productivity list in the home page.....	72
Figure 6.4: Top users by citations received list in the home page.....	73
Figure 6.5: Top works by citations received list in the home page.....	73
Figure 6.6: The first step of the sign-up guide.....	73
Figure 6.7: The second step of the sign-up guide.....	74
Figure 6.8: The final step of the sign up guide.....	74
Figure 6.9: The application's banner and search bar in the home page.....	74
Figure 6.10: The search modal.....	75
Figure 6.11: Search results.....	75
Figure 6.12: Author profile overview.....	76
Figure 6.13: Author's works list overview.....	77
Figure 6.14: Citations bar chart.....	78
Figure 6.15: Works bar chart.....	78
Figure 6.16: Work sources doughnut chart.....	78
Figure 6.17: Work page title and authors overview.....	79
Figure 6.18: Work page properties section overview.....	80
Figure 6.19: References metrics section overview.....	80
Figure 6.20: References metrics unavailable message overview.....	80
Figure 6.21: Work versions section overview.....	81
Figure 6.22: Groups page groups tree view overview.....	82
Figure 6.23: New group creation modal.....	82
Figure 6.24: Active group details overview.....	83
Figure 6.25: Active group statistic bar charts.....	83
Figure 6.26: Active group work sources doughnut chart.....	84
Figure 6.27: Active group works list.....	84
Figure 6.28: Active group OMEA authors stats.....	84
Figure 6.29: Active group OMEA works stats.....	85
Figure 6.30: New group landing page overview.....	85
Figure 6.31: Search component for adding new group members.....	85
Figure 6.32: Verify Author Identifiers sub-page overview.....	86
Figure 6.33: Authors result list overview.....	87
Figure 6.34: Author connection verification modal.....	87
Figure 6.35: Navigation component preview.....	88
Figure 6.36: Off-canvas component preview.....	89
Figure 6.37: Author Item component preview.....	90
Figure 6.38: Work Item component preview.....	90
Figure 6.39: Work Item versions modal preview.....	91

Figure 6.40: Search component modal preview.	92
Figure 6.41: Paginated list component preview.	93
Figure 6.42: Paginated list loading state preview.....	94

List of Tables

Table 3.1: User properties	28
Table 3.2: Author properties	30
Table 3.3: Work properties.....	30
Table 3.4: Group properties.....	31
Table 3.5: Statistic properties	31
Table 3.6: Type properties.....	32
Table 3.7: Concept properties	33
Table 4.1: OpenAlex API Controller Methods.....	36
Table 4.2: ORCID API Controller Methods.....	37
Table 4.3: DOI API Controller Methods.....	38
Table 5.1: General Search API parameters.	56
Table 5.2: General Search API parameters.	56
Table 5.3: Search Authors API parameters.	57
Table 5.4: Search Authors successful response data.	57
Table 5.5: Search Where Not In Group API parameters.	58
Table 5.6: Search Where Not In Group successful response data.	58
Table 5.7: Work Filter API parameters.	59
Table 5.8: Work filter successful response data.	59
Table 5.9: Get All Groups Successful Response data.	61
Table 5.10: Get Group parameters.	61
Table 5.11: Get Group API successful response.	61
Table 5.12: Create Group parameters.....	62
Table 5.13: Create Group API successful response.	62
Table 5.14: Delete Group parameters.....	62
Table 5.15: Add Group Member parameters.....	63
Table 5.16: Remove Group Member parameters.	63
Table 5.17: Get Groups Min Info Successful response.	64
Table 5.18: Get OMEA Author Stats parameters.....	64
Table 5.19: Get OMEA Authors Stats Successful response.....	64
Table 5.20: Get OMEA Type Stats parameters.....	65
Table 5.21: Get OMEA Type Stats Successful response.	65
Table 5.22: Get Works Metadata Successful response.	66
Table 5.23: Toggle Work Visibility parameters.....	66
Table 5.24: Get Hidden Works Successful response.....	67

Chapter 1 : Introduction

1.1 Organizing Research Publications

In the landscape of academia, the effective organization and management of research publications within an educational institution is of outmost importance. The analysis of various statistics including the quantity, citations, and the thematic scope of the publications can greatly enhance the institution's reputation and spotlight its contribution to the research community. Tracking metrics such as citation counts, and thematic scope showcases its impact on a national and global scale and underscores its commitment to advancing knowledge.

1.2 Utilizing a local Research Publication Database

The existence of a correctly structured and organized local database provides lots of advantages to academic staff and students.

1.2.1 Utilization by students

Students can benefit significantly from accessing a local database for their research needs. One major advantage is the time saved from searching through numerous databases, many of which may contain publications outside the students' specific thematic interests. By providing a centralized repository of relevant publications since most of them are authored or co-authored by the institution's academics, students can efficiently direct their focus towards their research objectives, rather than spending excessive time on searching for sources.

1.2.2 Utilization by academics

Academics can also derive significant benefits from accessing a local publication database, as it provides opportunities to enrich both teaching and research endeavors. By incorporating various relevant publications into their classes, academic staff can illustrate the evolution of their field and present practical use-cases of emerging technologies. Showcasing real-world examples can inspire students and academics to further contribute to research through their own unique and innovative projects, develop new technologies or improve existing ones. Having explored the practical uses for both students and academics, we can now delve into the broader institutional applications of the database.

1.3 Wider Impacts and Benefits of the Local Database

Having a local database for research publications within the institution yields broader impacts beyond mere facilitation of access to research materials.

Support for Research and Education

As previously mentioned, the presence of the database aids in promoting research and education within the institution itself by providing a rich and organized dataset that supports the teaching and research process.

Enhancement of Professional Development

The abundant research material available in the database provides opportunities for the professional development of members of the academic community, contributing to the deepening of their knowledge

and the expansion of their skills. Moreover, it fosters research advancement and the development of new technologies, or the enhancement of existing ones.

Support for Collaborations and Networking

Access to common research materials facilitates collaboration and networking among academic staff and students within the institution, across institutions, and fosters connections with the broader scientific community.

1.4 Conclusions

In this chapter, we have examined the role and benefits of implementing a local database for scientific publications within an educational institution. We have outlined the advantages it offers to students and academic staff, as well as its potential for enhancing the institution's recognition across various fields at national and international levels. Subsequently, we will provide a summary of the key findings from each section.

Organizing the Publications

The importance of proper organization and management of research publications at the institutional level was emphasized. Additionally, the benefits of collecting statistics were highlighted, demonstrating how their use can contribute to the recognition of the institution, its academic staff, and attract more students.

Benefits for Students and Academic Staff

The practical applications of having a database and its advantages for students were examined, including its usefulness in the educational process and in completing assignments. Additionally, reference was made to how the database can promote the improvement of existing technologies and contribute to research and the development of new ones.

Wider Benefits

We discussed the broader perks for the academic community, going beyond the direct advantages of having and using the database. These include helping with career growth and research promotion, aiding networking among staff and students, and building connections between institutions.

To sum up, local databases and similar tools are vital for supporting institutional academic missions. Therefore, continuous investment in their development is crucial to fully tap into their potential and achieve high academic standards. Next, we'll delve into the details of implementing such a database, looking at the tech used, and decisions made based on requirements analysis.

Chapter 2 : Technological Stack

In this chapter, we will delve into the technical framework that powers the application's functionality. Understanding these technological components is essential for grasping how the application operates.

The technological stack used comprises several key elements, such as the database, front end (user interface), back end (server-side operations), as well as additional tools and libraries. Each of these components plays a pivotal role in shaping the user experience and ensuring smooth functionality.

The selection of technologies was driven by practical considerations, such as compatibility with project requirements, scalability for future expansion, and my personal expertise. Throughout this chapter, I'll provide insights into the rationale behind each of my choices and discuss how they contributed to the development process. I will also provide detailed explanations of each component of the technological stack used including functions, implementation strategies and any challenges I have encountered while building the application. By the end of this chapter, you will have a comprehensive understanding of the technical infrastructure that powers the application.

2.1 Database Selection

A database is a structured collection of data organized in a way that allows for efficient storage, retrieval, and manipulation. It acts as a central hub for storing information, facilitating access and management by users and applications. Databases are essential components of software systems, enabling organizations to organize, analyze, and utilize data effectively.

For the database component, I opted to utilize MariaDB[1], a widely used open-source relational database management system. The decision to leverage MariaDB was influenced by several factors, including familiarity with MySQL, its open-source nature, my expertise from previous works.

Having worked extensively with MySQL[2] in previous projects, I was already well-versed in its functionalities and capabilities. MariaDB, being a fork of MySQL, offered a seamless transition while providing additional features. This familiarity with the MySQL ecosystem streamlined my development process and allowed me to leverage my existing knowledge and expertise effectively.

Furthermore, the open-source nature of MariaDB aligns with my project's principles of accessibility and collaboration. By opting for an open-source database, not only do I benefit from a robust and community-driven development model but also contribute to open-source software.

In summary, the decision to utilize MariaDB as my database component was based on a previous expertise, its open-source nature, and its compatibility with the project's requirements.

2.2 MariaDB vs MySQL

In this section, I will delve into some historical facts about each one of the ecosystems, delve into key differences and explain the rational behind the decision to use one over the other.

MySQL was initially developed by MySQL AB, which was later acquired by Sun Microsystems and then by Oracle Corporation. In response to concerns about MySQL's acquisition by Oracle, the original developers founded the MariaDB Foundation, leading to the creation of MariaDB.

Both ecosystems are licensed under the GNU General Public License (GPL), with MariaDB specifically licensed under GPL version 2. Additionally, both databases include components that are licensed under a variety of different licenses.

In terms of security, MariaDB offers various enhancements compared to MySQL, including roles-based access control (RBAC) and improved password validation plugins.

MariaDB also tends to be slightly more performant than MySQL, mostly due to its more community-driven development and its more rapid release cycle.

When choosing between MariaDB and MySQL for my project, factors such as these security enhancements and performance optimizations were key considerations. The transparent development model and community involvement associated with MariaDB aligned perfectly well with my project's principles and requirements. Having examined the database selection process and the reasons behind my choice, let's delve into the 'core' of my project: the back end.

2.3 Back End

In this chapter, we will explore the backend development aspect of the project, which forms the foundation for the functionality and operation of the application. We will discuss the technologies, frameworks, and tools that were used to build and manage the server-side components that power the application's logic and data management.

The backend of the application handles critical tasks such as data storage, retrieval, and manipulation, authentication, and business logic execution. It serves as the backbone of the application, ensuring seamless connection between the front end and the database.

Throughout the chapter, I will provide insights into the architecture and design decisions that influenced the backend development.

2.3.1 Laravel

Laravel[3] is the framework that powers the back end of my application. We explore the framework's nature, its extensive capabilities, the rationale behind choosing it, and its alignment with the project's requirements.

Laravel is a PHP[4]-based web application framework. It's like a toolbox filled with tools commonly needed for building modern web apps. By providing these tools out of the box, Laravel allows developers to focus on building the specific features of their application without worrying about the technical details of infrastructure. Laravel simplifies common tasks such as routing, authentication, and database management, that would otherwise take a lot of effort to build and optimize from scratch. It is built in a way that developers of any skill level can use it, its syntax is expressive and intuitive, with naming conventions that clearly speak their use. This makes utilizing Laravel's functionality a straightforward process, even for those who are new to the framework.

Laravel proves to be an excellent choice for applications with advanced requirements, offering built-in support for features such as dependency injection, unit testing, and real-time events. Its scalability is noteworthy, capable of seamlessly accommodating varying levels of traffic, from low usage to enterprise-scale workloads. Laravel seamlessly integrates with the latest technologies in web development, such as AWS (Amazon Web Services) and others. Its extensible architecture and robust ecosystem provide support for incorporating these latest advancements into your application effortlessly.

Making use of the best packages the PHP community has to offer, Laravel combines all the functionality in one single framework, in whose development, thousands have contributed from all around the world.

Now that we've explored the numerous advantages Laravel brings to the development experience, let's delve into why it was the best choice for my project. Matching Laravel's features with my project's specific helps us understand how it enhances the development process.

With a lot of experience using Laravel in various projects and after carefully researching its features, it is my go-to back-end framework. It simplifies tasks like setting up the database infrastructure and managing relationships, which is crucial for my project, considering it has to do with a well-organized database. Additionally, Laravel's maintained packages allowed me to seamlessly integrate features like authentication through my university's provider and effortlessly connect the back and front end. I'll delve into more detail about the specific packages used and how certain features were implemented in the following chapters.

2.3.2 Socialite

One of the main features in my application is authenticating users through my university's OAuth provider. Laravel maintains a package called Socialite[5] that provides seamless integration with OAuth providers. It supports very popular providers out of the box, like Google, Facebook, Twitter, LinkedIn, and Git Repository platforms like GitHub, GitLab and Bitbucket. In addition to these providers, there are hundreds of community managed drivers that can be used to authenticate using other providers like more social media platforms, gaming platforms and more. For my projects needs though, I had to create my own driver to support authentication through my university's provider. Laravel's and Socialite's documentation makes it trivial to implement your own drivers to accommodate your project's needs.

2.3.3 Eloquent

As previously discussed, the database plays a vital role in my project. Laravel comes with a feature called Eloquent, an Object Relational Mapper (ORM) that equips developers with the necessary tools for building and managing databases. Despite the complexity of the relationships between database entities in my project's requirements, Eloquent simplified the whole process [6].

2.3.4 Routing

Routing in Laravel is straightforward. It allows for grouping routes using route groups and prefixes, applying middleware to route groups, and binding eloquent models to them. This feature helped me organize my API routes based on various factors such as relevant entities and actions, as well as user permissions through middleware. Additionally, Laravel provides support for request-specific validation, a feature that we will discuss in the next chapter.

2.3.5 Form Requests

Some requests made to the backend require complex data validation. Laravel's Form Requests streamline this process by encapsulating all validation logic within a single class. By type-hinting the class's name on the handler, Laravel automatically validates the request's data based on rules provided by the developer before either passing the request to the handler or rejecting it. Using Form Request classes significantly simplified the validation of my API request inputs, especially when dealing with various data types and complex validation logic.

2.3.6 Conclusion

To sum up, Laravel's useful features like its easy routing system, strong database tools, and simple data validation have been crucial for meeting my project's needs. Because these features fit so well with what my project requires, I'm confident in my choice to use Laravel as the framework. Its tools make development easier and set a solid foundation for my project's success. In the next chapters, we'll dig deeper into how I've used Laravel to build my project.

2.4 Front End

Now that we've covered the foundational aspects of the backend development with Laravel, it's time to turn our attention to the front-end technologies that bring my project to life for its users. Just as Laravel provides the backbone for my application's functionality, my choice of front-end technologies plays a crucial role in delivering a seamless and engaging user experience. In this chapter I'll delve into the tools and frameworks I've used to create the responsive interface that powers my application.

For the front-end, I'm relying on React, a popular choice for building user interfaces that feel smooth and interactive. React's way of breaking down the interface into reusable components makes it efficient and flexible.

2.4.1 React

React[7] is a JavaScript library used for building user interfaces. It was developed by Facebook, and it is now among the most popular choices for front-end development mostly due to its flexibility and performance. React allows developers to break down the user interface into smaller re-usable components, encapsulating their own logic and state, making it easier to work in large-scale complex applications.

Like most popular front-end libraries, React leverages a concept known as the Virtual DOM. By maintaining a lightweight copy of the browser's DOM and efficiently updating it based on changes, React ensures faster rendering and ultimately a smoother user experience.

Because of its popularity, and its growing community, there are numerous community-supported libraries and other tools that simplify common tasks, provide ready-to-use components, or help you manage application-wide state. These are the key reasons why I chose React as the front-end technology for my project, as it provides a powerful toolkit for building modern and responsive user interfaces for web applications.

Now that we've examined the foundational aspects of the front-end development with React, let's dive into the tools and techniques that shape the appearance and responsiveness of my application, beginning with Tailwind CSS.

2.4.2 Tailwind CSS

Tailwind CSS[8] is a CSS framework that offers pre-designed utility classes for creating responsive and customizable user interfaces. Unlike other frameworks that provide predefined components and styles, Tailwind CSS emphasizes low-level utility classes, allowing developers to easily customize designs without writing custom CSS. In addition to the numerous styling classes that Tailwind has to offer, the developer can also extend or even define their own classes to be used in conjunction with the existing ones, making it easier to meet project specific needs.

Tailwind CSS promotes consistency by encouraging a single source of truth for styling. With all styles defined directly in the HTML markup using predefined classes, developers can easily understand and modify the styling of individual components without the need to search through separate CSS files. This approach greatly simplifies the maintenance of large-scale projects that utilize Tailwind.

It has gained popularity among developers mostly due to its simplicity, its naming conventions, and the vast number of utility classes it has to offer. These features significantly expedite the development process, making it an excellent choice for many developers, including myself.

Finally, due to its popularity, the Tailwind ecosystem includes a wide variety of third-party libraries that offer ready-to-use components, and in my case React components, styled with Tailwind. These libraries provide developers with commonly used components ready to be used out-of-the-box allowing them to focus their effort on building project-specific components maximizing their productivity.

2.4.3 Flowbite React

Flowbite[9] is an open-source UI component library built on top of Tailwind CSS. It offers a collection of responsive and customizable UI components, layouts, and utilities to help developers build modern and visually appealing user interfaces. With its advanced theming infrastructure, Flowbite provides developers with the flexibility to customize and extend its components along with their own to maintain consistency throughout their projects. I selected Flowbite for its collection of components and their alignment with the specific requirements of my project.

2.4.4 Material UI

Like Flowbite React, Material UI[10] is a popular React component library that implements Google's Material Design. It provides a comprehensive set of pre-designed React components, styled according to the Material Design guidelines. Material UI's components are highly customizable and can be easily integrated into React applications. While Flowbite offered a great collection of components, Material UI provided additional components that were essential for meeting specific project requirements. This versatility and range of components were key factors in my decision to use Material UI alongside Flowbite in my project.

2.4.4 Chart.js

Given that my project revolves around extracting valuable statistics from an organized research publication database, it became crucial to find an effective way to visualize and present this data to the end user. This is where Chart.js comes into play. Chart.js[11] is a JavaScript library that allows developers to create interactive and visually appealing charts and graphs for displaying data. Its simplicity, flexibility, and wide range of chart types make it an ideal choice for projects like mine. I chose Chart.js, because of its extensive documentation, its simplicity, and the ability to create responsive and customizable charts that enhance the presentation of statistical data.

2.4.5 Inertia.js

In addition to the libraries mentioned earlier, I also explored the use of Inertia.js [12] for handling page hydration within my project. Inertia.js is a framework that allows developers to build modern, reactive web interfaces using server-side routing and client-side components. It offers excellent support for Laravel-React projects, seamlessly integrating with both Laravel on the backend and React on the frontend. This makes it a compelling option for projects with this specific technology stack. Moreover,

Inertia.js is not limited to Laravel and React as it provides support for various other front-end and back-end stacks, offering flexibility and compatibility for a wide range of development environments.

While I have previous experience with Inertia.js and initially considered it for my project's architecture, I ultimately limited its use to specific cases where it provided clear benefits, such as page hydration, due to architectural considerations. Despite these limitations, Inertia.js remains a valuable tool in my toolkit, a versatile framework suitable for different project requirements and technology stacks.

2.4.6 Lodash

Lodash [13] is a JavaScript utility library that provides a wide range of functions for simplifying common tasks such as data manipulation, array, and object manipulation. Lodash is widely recognized for its efficiency, performance, and extensive documentation, making it a popular choice among developers for handling complex data operations in JavaScript applications.

In my project, I used Lodash mainly for its data structure manipulation utility functions. These functions helped me implement utility components used extensively throughout my application. By integrating Lodash, I improved my code's maintainability, and overall development efficiency, which played a key role in the project's development process.

2.4.7 Prop-types

Prop-types [14] is a library used in React applications to specify the types of properties passed to components. It helps developers catch bugs early by validating the types of props during development. This library offers a wide range of validators, such as string, number, array, shape (object), and more, allowing developers to define the expected types for props and ensure their components receive the correct data. This approach to type validation improved the development experience, made debugging easier and reduced the chance of runtime errors occurring.

While I initially explored the possibility of incorporating TypeScript into my project for static typing, I found that it slowed down my workflow significantly. Instead, I opted to just use prop-types, which provided sufficient type validation without impacting development speed.

Overall, prop-types played a crucial role in ensuring the stability and robustness of my React components, making it an essential tool in my development toolkit.

2.4.8 CLSX

Clsx [15] is a utility library commonly used in React applications for conditionally joining classNames together. It simplifies the process of dynamically applying classNames based on conditions within components. Clsx provides a readable way to manage classNames, particularly in scenarios where multiple classNames need to be applied based on different conditions. By integrating clsx into my project, managing classNames was effortless, it improved code readability and reduced duplicated code. This library proved to be a valuable tool during the development experience, allowing for cleaner code in my components.

2.4.9 useHooks

React hooks are a powerful feature introduced in React 16.8 that allow developers to add state and other features to functional components. They provide a way to reuse stateful logic across different

components without using class components. Hooks are used for a variety of purposes, including managing component state, performing side effects, and integrating with external data sources.

In React development, hooks have become essential for building reusable and composable components. While React provides built-in hooks, there are also libraries that offer additional custom hooks to further simplify common tasks.

One such library is uidotdev/usehooks [16], which provides a collection of custom React hooks designed to address common use cases and simplify complex tasks. These hooks cover a wide range of functionality, from managing local storage to implementing debouncing and throttling functions. By leveraging the hooks provided by uidotdev/usehooks, developers can reduce duplicated code, improve code readability, and enhance development efficiency.

I chose to integrate uidotdev/usehooks into my project due to its extensive collection of hooks and their alignment with the specific needs of my project. The library's variety of hooks offered a convenient solution for implementing various features and functionalities.

2.4.9 React Icons

React Icons [17] is a library offering a wide range of icon components for React applications. It includes popular icon sets like Font Awesome and Material Design Icons, making it easy to add visually appealing icons to components. By using React Icons in my project, I effortlessly integrated high-quality responsive icons, enhancing the application's visual appeal and user experience.

2.5 Data Sources

My application uses several data sources to gather information about authors and their works. OpenAlex is the main source, while ORCID, Crossref, and Scopus are used to fill in missing data or compare information between different sources.

2.5.1 OpenAlex

OpenAlex [18] is a free, open dataset that offers extensive information on scholarly papers, authors, institutions, and more. Managed by OurResearch, a non-profit organization, OpenAlex aims to provide a comprehensive and accessible resource for academic research and analysis. The idea to develop it, came after a generous donation to OurResearch by Arcadia [19] and after the shutting down of Microsoft Academic Graph (MAG)[20] was announced both in 2021. A beta version was launched on January 3rd, 2022, and since October 2023, its database and web application are fully operational and live for everyone to explore.



Figure 2.1: OpenAlex logo. [21]

Why OpenAlex?

OpenAlex serves as the main source of information for my application due to its inclusivity, affordability (free), and availability. The API provided is fast, modern, and well-documented, making it easy to integrate into my application and retrieve the needed data. Additionally, the complete dataset is free

under the CC0 license, which allows for transparency and reuse. This openness aligns with the goal of making research accessible to everyone.[22]

Data retrieved

Firstly, it collects profiles of authors, which include their publication records, offering detailed insights into their academic contributions.

For works, the application retrieves details about academic works such as titles and publication dates, ensuring users have access to essential information about each piece.

Additionally, it gathers information on work concepts, which highlight the main topics of works, a feature intended for use in a future version.

Finally, the application collects citation and works count/year statistics, providing yearly data on citations and works to help users track academic impact over time.

2.5.2 ORCID

ORCID [23] (Open Researcher and Contributor ID) is a global, non-profit organization that provides unique identifiers for researchers. Supported by member fees, ORCID is governed by a Board of Directors with wide stakeholder representation and sustained by a dedicated professional staff.



Figure 2.2: ORCID logo. [24]

Why ORCID?

ORCID was chosen as a data source due to its unique vision, global recognition, and its professional support system. ORCID which was launched in 2012, aims to provide transparent and trustworthy connections between researchers and their contributions, making it an essential tool for maintaining accurate academic records. [25]

- **Unique Vision:** ORCID envisions a world where all participants in research, scholarship, and innovation are uniquely identified and connected to their contributions across disciplines, borders, and time. Unique identifiers for each of the assets in my database are crucial to maintaining transparency.
- **Professional Support:** ORCID is supported by a knowledgeable professional staff and governed by a board representative of its diverse membership.
- **Global Recognition:** Widely recognized and used by researchers and organizations globally, ORCID ensures reliable attribution and integration with various academic systems.

Data retrieved

Detailed profiles for authors, including their biographies and other personal information. This allows users to access comprehensive information about each author. For works, the application retrieves information about each work associated with the authors. This includes titles, publication dates, and details about co-authors, providing users with a thorough understanding of each piece.

Interesting statistical data about ORCID:

- ORCID has over 5,692 yearly active integrations.
- Average monthly "Read" requests amount to 166.81 million.
- Active researchers include 814,866 from the US, 499,728 from China, and significant numbers from other countries.
- ORCID is used by organizations in 58 countries and has 1,393 member organizations. [26]

2.5.3 Scopus

Scopus [27] is a large database of academic articles, managed by Elsevier. It started in 2004 and covers many subjects, providing important metrics to evaluate research.



Figure 2.3: Scopus logo.[28]

Why Scopus?

Scopus was chosen because it offers detailed and reliable data, combining information from many sources and covering a wide range of topics, making it a trusted resource for academic research. It has a vast database of abstracts and citations from academic works, with data reviewed by experts to ensure reliability and accuracy. Scopus also provides useful tools and analytics for detailed author profiles and citation results. Additionally, it offers a variety of research metrics and includes many peer-reviewed publications.[29]

Data retrieved

Author profiles with metrics like citation counts and works.

Interesting statistical data about Scopus:

- Scopus includes over 94 million records, including articles, conference papers, and book chapters.
- It covers content from more than 29,200 active serial titles and 330,000 books.

2.5.4 Crossref / DOI

Crossref [30] is a non-profit organization that provides Digital Object Identifiers (DOIs) for academic content. Founded in 2000, Crossref works to improve the discoverability of research outputs by offering accurate and comprehensive metadata.



Figure 2.4: Crossref / DOI logo.[31]

Why Crossref?

Crossref was chosen for its data integrity, and accessibility along with its commitment to making research easy to find, cite, link, and reuse. [32]

Data retrieved

Detailed information about academic works, including DOIs, titles, publication dates, publishers, and other relevant information. Additionally, it collects statistical data related to the works, such as citation counts and references, providing users with a comprehensive view of each work's academic impact and context.

Interesting statistical data about Crossref:

- Has over 130 million records in its database.
- It serves more than 17,000 members from 146 countries.
- Handles over 600 million monthly queries.

2.6 Conclusion

In this chapter, we have explored the technologies that power my application, including both front-end and back-end aspects. From Laravel for robust back-end development to React for dynamic front-end interfaces, each technology was carefully chosen to meet the specific needs of the project. Additionally, libraries such as Chart.js, Lodash, and useHooks provided valuable functionality and enhanced the user experience.

We also examined the data sources used in my application, such as OpenAlex, ORCID, Crossref, and Scopus, detailing the information retrieved from each one, providing an overview of their features, and presenting interesting statistics. With a solid foundation set, the following chapters will delve into the development process, providing insights into how these technologies were implemented and integrated to bring the project to life.

Chapter 3 : Database Design

3.1 Introduction

In this chapter, we will delve into the design process of the database. How the project requirements were analyzed, how the required entities for the application's functionality were extracted, and how each entity is related with each other. I will elaborate on how the technologies mentioned in the previous chapter were used during the process.

3.2 Defining the entities

As the database serves as a foundational element of my project, it was sensible to start the implementation process by designing the database infrastructure. This involved analyzing the project's requirements to determine the necessary entities, defining the properties of each entity, and establishing relationships between them. Addressing these tasks was essential in laying the groundwork for the project.

After carefully analyzing my project's requirements, I identified several key entities that were essential for meeting the project's objectives:

- User
- Author
- Work
- Group
- Statistic
- Type
- Concept

These entities were designed to encapsulate specific parts of the application's functionality. Additionally, relationships between these entities were established. Laravel's Eloquent ORM proved to be invaluable in managing these complex relationships, providing a seamless way to define the associations between entities. Its expressive syntax and powerful features made it a breeze to work with, while ensuring the database's integrity and efficiency.

Now that the database infrastructure has been designed and the key entities have been identified, I will proceed with the implementation details of each model in the following sub-chapters. These chapters will provide insights into how each model was implemented, explain their properties, and the relationships between any other entities they might be related to.

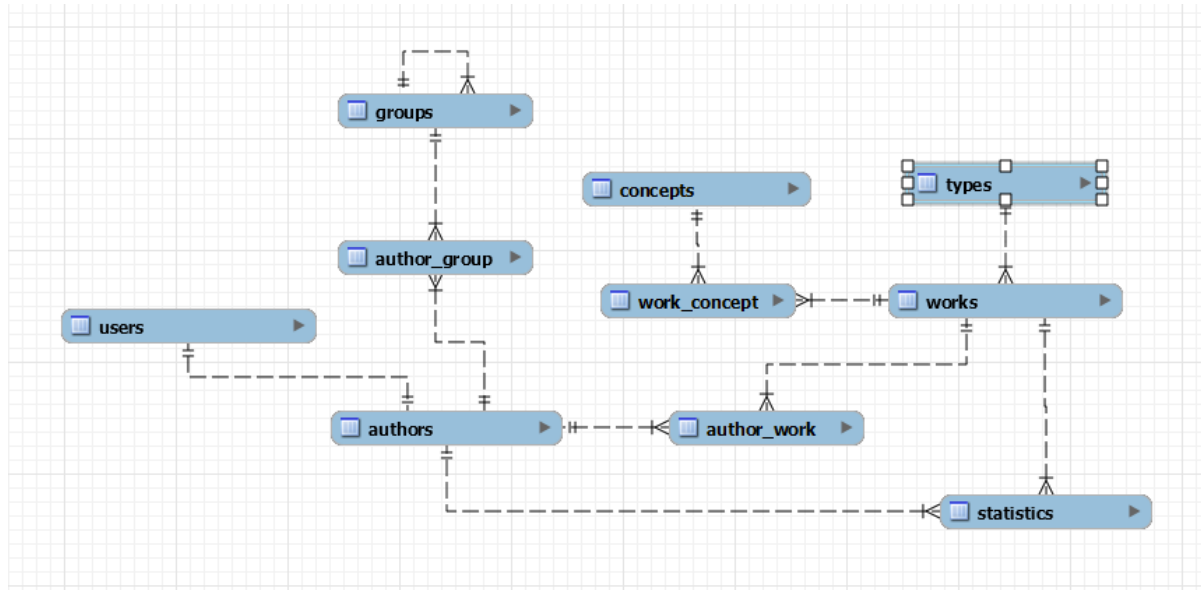


Figure 3.1: ER (Entity-relationship) diagram of my application's database

3.3 User

The User entity refers to the users of the application. Initially, it was necessary to identify the end users of the application, the user categories, access permissions for each user category, the actions that each user can perform in the application, as well as the impact of their actions on the database and the application.

Table 3.1: User properties

Property	Type	Description
id	bigint	Unique identifier for each user. Automatically generated.
first_name	varchar	The first name of the user.
last_name	varchar	The last name of the user.
Email	varchar	The email address of the user. Unique identifier.
display_name	varchar	The display name of the user.
is_admin	tinyint	Indicates whether the user is an admin (1) or not (0).
orc_id	varchar	The ORCID (Open Researcher and Contributor ID) of the user.
scopus_id	varchar	The Scopus ID of the user.
open_alex_id	varchar	The OpenAlex ID of the user.
ids_verified_at	timestamp	Timestamp indicating when the user's identifiers were verified.
is_staff	tinyint	Indicates whether the user is staff (1) or not (0).
external_id	varchar	External identifier for the user.
author_id	bigint	The claimed author's id (foreign key)

Relationships

The user entity has a single direct one-to-one relationship with the Author entity. Each user can only correspond to one author and each author can only be claimed by a single user. Their relationship is constrained using the author's id as a foreign key in the users table.

3.3.1 User Categories

The user entity can be categorized into two main groups: academic staff, including professors and researchers, and students, primarily as guests. In the application's context, academics are also referred to as Authors. Upon their initial login, academics are prompted to provide at least one unique identifier such as their ORCID, OpenAlex, or Scopus ID. Using these identifiers, the application seamlessly fetches their information from the mentioned sources and builds their profile. Finally, there is one last user category, which is the administrators.

3.3.2 Permissions

- **Academics:** Academics have access to their own separate public profiles, with editing permissions. They also have permission to view work pages and other author profiles, as well as permission to view groups available in the application.
- **Students:** Students do not have a public profile. However, they have the ability to view work pages, author profiles, and groups, similar to academic users.
- **Administrators:** Administrators possess the highest level of access within the application. While they may not necessarily have a public profile, as they might not be authors, they have permission to view any part of the application. Administrators can create, delete, and modify groups within the application, and they can add or remove members from each group.

3.3.3 User activities

- **Academics:** Academics have several capabilities within the application. They can toggle the visibility of any work in their public profile, allowing them to control which works are publicly accessible. They can manually add works, though this feature is slated for a later version. Academics also have the ability to edit their personal information in their public profile, ensuring that their details are always up-to-date. Additionally, they can view work pages, author profiles, and groups, providing them with comprehensive access to the platform's content and collaborative features.
- **Students:** Students can view work pages, author profiles, and groups within the application. This access allows them to explore and learn from the available works and profiles, fostering an environment of academic growth and collaboration.
- **Administrators:** Administrators possess the highest level of access within the application. They have all the capabilities granted to academics and students, which includes viewing work pages, author profiles, and groups. Additionally, administrators can create, edit, and delete groups, giving them control over group management. They also have the authority to add or remove members from each group, ensuring proper membership and administrative oversight.

3.3.4 How user actions affect the database

The actions that users can perform have minimal impact on the database. Specifically, academics can only edit their personal information and adjust the visibility of publications on their public profile. Additionally, using unique identifiers such as the DOI (Digital Object Identifier), as mentioned earlier, they can search for and add publications to it. In contrast, the actions of students have no impact on the database or the display of any information within the application. Administrators can manage groups, so they do have more authority and can impact the database slightly more than academics can. Generally, though, the user actions do not have much impact on the database since it's primarily used for reading.

3.4 Author

The author entity refers to the academics whose publications live in the database. When an author's profile is built, the publications that come with it are most likely co-authored by authors that are not users of my application and thus an author is not necessarily a user.

Table 3.2: Author properties

Property	Type	Description
id	bigint	Unique identifier for each author. Automatically generated.
display_name	varchar	The display name of the author.
is_user	tinyint	Indicates whether an author has been claimed by a user (1) or not (0).
orc_id	varchar	The ORCID ID of the author.
scopus_id	varchar	The Scopus ID of the author.
open_alex_id	varchar	The OpenAlex ID of the author.
biography	Text	The author's biography (sourced from ORCID).
works_url	varchar	The url used to fetch the author's publications (OpenAlex)
cited_by_count	Int	A number representing how many times the author's works were cited.

Relationships

Compared to the user entity, authors are related with multiple other entities in various ways.

Authors have a one-to-one relationship with users, with the author's ID used as a foreign key in the users table. They have a many-to-many relationship with works, constrained using an intermediate table called `author_work`. Similarly, authors have a many-to-many relationship with groups, constrained using an intermediate table called `author_group`. Additionally, authors are involved in a many-to-many polymorphic relationship with statistics, managed using an intermediate table called `statistics`.

3.5 Work

The work entity represents the research publications of the authors. It is one of the most important aspects of the application as most of its features revolve around this entity.

Table 3.3: Work properties

Property	Type	Description
id	bigint	Unique identifier for each author. Automatically generated.
doi	varchar	The Digital Object Identifier of the publication.
source_title	text	The title of the work's source (usually the article's title)
abstract	text	The publication's abstract.
publication_year	smallint	The year the work was published.
is_referenced_by_count	int	The number of times this work has been cited.
source_url	varchar	The source url of the publication (external source)
external_id	varchar	The work's id from its source.
language	varchar	The language that the work is written in.
sub_type	varchar	The publication's detailed type.
type	varchar	The publication's type (article, book, etc..).

is_oa	tinyint	Whether the publication has open access.
authors_string	text	A string containing the names of the work's authors.
source	enum	The work's source (ORCID, OpenAlex, Crossref, Aggregate)
last_updated_date	datetime	The date this work was last updated on its source.
event	varchar	The name of the event that the work was published on.
type_id	bigint	The OMEA type's external id (foreign key)

Relationships

Like the author entity, the work entity is a core aspect of the application and has multiple relationships with various other entities.

Works have a many-to-one relationship with types, with the type's ID used as a foreign key in the works table. They have a many-to-many relationship with authors, managed using an intermediate table called `author_work`. Additionally, works have a many-to-many relationship with concepts, constrained by an intermediate table called `work_concept`. Furthermore, works are involved in a many-to-many polymorphic relationship with statistics, managed using an intermediate table called `statistics`.

3.6 Group

In the context of our university and thus, my application, groups represent collections of authors who share common affiliations, interests, or roles within the academic community. For instance, a group might consist of all laboratory instructors or research teams focused on specific areas of study. Essentially, groups serve as organizational units that allow authors to connect with peers who share similar expertise, or objectives.

Table 3.4: Group properties

Property	Type	Description
id	bigint	Unique identifier for each group. Automatically generated.
name	varchar	The display name of the group.
description	tinytext	A brief description of the group.
parent_id	bigint	The id of the parent group (optional)

Relationships

The group entity establishes a direct relationship with the author entity, indicating membership of authors within specific groups. This direct association enables efficient management and organization of authors based on shared affiliations or roles within the academic community. Furthermore, through the author entity, indirect relationships are established with the works of group members, facilitating access to collaborative group projects and publications. This interconnected structure allows for seamless navigation and retrieval of group-related works within the application.

3.7 Statistic

The Statistic entity is used to describe various statistics both for the Work entity, as well as the Author entity. It is a means to a polymorphic relationship since it is the same model that's used for both other entities.

Table 3.5: Statistic properties

Property	Type	Description
id	bigint	Unique identifier for each statistic. Automatically generated.
year	tinyint	The year that the statistic is about.
works_count	int	The amount of works for the specified year (Author only)
cited_count	int	The number of citations for the specified year (Author & Work)
asset_id	bigint	The id of the associated model
asset_type	varchar	The classname of the associated model

When compared to the properties table of the rest of the entities so far, table 3.5 presents 2 additional properties. Asset_id and asset_type is used to identify the associated model since the Statistic entity is used in a polymorphic relationship. The asset_type field is used to identify which model the statistic is referring to, either a Work or an Author. Then the asset_id is used to connect the statistic to the row in the corresponding table.

Relationships

As mentioned in the previous chapter, the statistic entity is involved in a polymorphic relationship. The statistic can either be for a Work entity or an Author entity.

For authors, there is a one-to-many polymorphic relationship. Both works_count and cited_count can be filled for this relationship. In this case, the asset_type takes the value of the Author class name.

For works, there is a one-to-many polymorphic relationship. Only cited_count may be filled for this relationship. In this case, the asset_type takes the value of the Work class name.

3.8 Type

Type is an entity used to describe various “special” categories of work as needed for OMEA classification. It is a pretty simple entity primarily used for sorting works based on a specific type, and to extract statistics based on the amount of works that are of the same type.

Table 3.6: Type properties

Property	Type	Description
id	Bigint	Unique identifier for each type. Automatically generated.
name	Varchar	The name of the type.

Relationships

Provided that the Type entity is described as a simple entity, it makes sense that its relationships with other entities, would also be simple. Type can only be related to one other entity, Work. The relationship between the two is one-to-many relationship and is defined using the Type’s id as a foreign key to the Works table.

3.9 Concept

Concept is an entity that is currently providing no extra functionality in the current version of the application. It was added and configured to support easier functionality expansion in the upcoming versions of the application. It is used to describe the thematic scope of each Work entity and provide insights regarding the topics that each work is about. The Concept property is only used for Work assets that are sourced from the OpenAlex database.

Table 3.7: Concept properties

Property	Type	Description
id	bigint	Unique identifier for each concept. Automatically generated.
name	varchar	The name of the concept.
external_id	varchar	The external (OpenAlex) id of the associated Work entity.

Relationships

The Concept entity can only be related to the Work entity. More specifically to works that are sourced from the OpenAlex database. The relationship between these two entities is a many-to-many relationship, using an intermediate table called `work_concept`.

With the database structure and relationships now defined, I will now talk about the implementation of migrating and populating the database. In the next chapter, we will explore the process of database migration and delve into the details of seeding, fetching information from various sources such as OpenAlex, ORCID, and Crossref. This crucial step lays the foundation for the application's functionality by ensuring that it is populated with accurate and comprehensive data on authors and their works.

3.10 Migrating the database

In this section, we delve into the process of structuring and defining the database schema for my application using Laravel's built-in migration system. Migrations are as a crucial part of database management, allowing for version control and effortless deployment of database changes across development, and production environments. We will explore how each entity's properties are translated into database migrations, and how the database is seeded with author information and their publications.

Creating the migration files

A migration in Laravel is a file that defines and manages changes to the database schema. It encapsulates operations such as creating, modifying, or dropping tables, columns, and constraints. Migrations provide a version-controlled approach to managing database changes, allowing for easy tracking and application of schema modifications.

The Laravel migration system simplifies the creation and management of database schemas by allowing developers to define and manipulate database structures using PHP code. The first step to creating a migration, is to utilize the artisan command-line tool provided by Laravel, which generates a new migration file. Each migration file contains 2 methods, the `up` and `down` methods, defining the changes to be applied and rolled back, respectively.

For example, let's consider the creation of a migration for the `users` table in my application. We can use the artisan `make:migration` command to generate a the migration file. Within the migration file, I define the schema blueprint using Laravel's syntax, specifying the table's columns, indexes, and constraints.

```

public function up(): void {
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('first_name');
        $table->string('last_name');
        $table->string('display_name');
        $table->string('external_id');
        $table->string('email')->unique();
        $table->string('is_admin')->default(false);
        $table->string('is_staff')->default(false);
        $table->string(Ids::ORC_ID_ID)->nullable()->unique();
        $table->string(Ids::SCOPUS_ID)->nullable()->unique();
        $table->string(Ids::OPEN_ALEX_ID)->nullable()->unique();
        $table->foreignId('author_id')->nullable()->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->rememberToken();
        $table->timestamps();
    });
}

```

Listing 3.1: Users Migration overview

As shown in Figure 3.1, the schema for the users table is defined using chaining methods on the `$table` object. The first step to creating a column is to define the type of data that will be stored in it by using the corresponding method, which is usually named after the datatype itself. The developer can then make the field nullable by chaining the `nullable()` method, set a default values with the `default()` method and by passing the default value as an argument, and define whether the values of that column should be unique by chaining the `unique()` method. Setting constraints is also easy since most of the logic is handled by the framework the developer follows the naming conventions required by Laravel. For example, by chaining the `constrained()` method, Laravel understands which table needs the constraint corresponds to by the name of the column (e.g `user_id` means that the column is a foreign key to for the users table). Following the `constrained()` method, the developer can then set the behaviour on delete and update actions by chaining the `onDelete()` and `onUpdate()` methods. The migration files include the `id()` method and `timestamps()` by default, setting an auto-incremented integer id column, and the `created_at` and `updated_at` columns respectively.

Chapter 4 : Database Management

Database seeding is the process of populating a database with some initial data. This initial data often includes data that is necessary for the application to function properly or for testing purposes. In the context of my application, the initial data were all the authors from my university's academic staff, their works, and all the required relationships between these entities.

The seeding is managed by a dedicated module in my application, responsible for both the initial seeding and periodic updates. This module consists of 2 Laravel jobs, `InitializeDatabaseJob`, executed during application setup, and `UpdateDatabaseJob`, scheduled to run biweekly via a cron job. In the upcoming chapters I will explain the implementation and functionality of these services.

Laravel jobs are tasks that can run in the background, separate from the main part of the application. They're great for tasks that take up a lot of resources or time or need to happen later or be repeated at a regular basis. Laravel's jobs were the perfect match for the requirements of the seeding and updating services. They both demand substantial resource allocation, they should both be executed asynchronously. Additionally, the updating job should run in a biweekly basis, a task which is made easy by the functionality provided by Laravel jobs.

In the process of retrieving data from various external sources, a robust mechanism for making HTTP requests and handling responses is crucial. To address this requirement, I developed the `Requests` utility class within the application. This class is responsible for managing HTTP requests and handling their responses.

4.1 Request Handling Infrastructure

The `Requests` class offers a simple yet flexible method for sending HTTP GET requests to specified URLs. It provides a reliable means of communicating with external APIs or endpoints. This approach ensures efficient data retrieval from external sources, contributing to the overall performance and functionality of my application. The `get` method ensures that requests are sent with the necessary headers and options. Additionally, the class includes methods for parsing response bodies, extracting the relevant information efficiently. This way, the `Requests` class ensures consistency and reliability across all requests, allowing the development of more generic, reusable code.

4.1.1 Source-Specific Controller Classes and Configuration

In this section, we delve into the controller classes dedicated to interacting with each data source, OpenAlex, ORCID, and Crossref that also come with their corresponding configuration files. Each controller encapsulates source-specific logic, including request URLs, parameters, and field selection. Additionally, I will also elaborate on the purpose and structure of the configuration files, which store essential information such as base URLs, API keys, and required fields for data retrieval.

All API controller classes follow a similar pattern. Each has an `init()` function that sets up a single instance of the class, ensuring only one instance is ever created during the application's lifecycle. This function grabs all the necessary config details from the corresponding source-specific config file and stores them as constants within the class. This way, all the internal methods have quick access to the resources they need to do their job effectively. Utility functions to build the endpoint urls and add any necessary parameters based on the request are also part of these classes, again encapsulating all this logic making the code cleaner, easier to understand, expand and maintain.

4.1.2 OpenAlex API Controller

The OpenAlex API controller class is responsible for handling all the requests associated with the OpenAlex repository. Being the primary source for my project, OpenAlex is used to fetch information for both the authors and their works. This controller class encapsulates the logic for making author requests, author works requests, work requests and all the utility classes to build the urls to the necessary endpoints. The key methods of this class are:

Table 4.1: OpenAlex API Controller Methods

Method	Parameters	Description
authorRequest	id (required), ignore_field_selection = false (optional)	Retrieves author information based on a unique identifier. Dynamically selects the appropriate endpoint based on the identifier type. The method accepts the author's ID as a required parameter and an optional boolean flag to specify whether only essential fields should be included in the response.
authorFilterRequest	id (required), ignore_field_selection = false (optional), single_entity = true (optional)	Retrieves authors based on specific identifiers. Includes an additional parameter, single_entity, which determines whether the response should contain only the first item. Useful when the filter query may return multiple results. The required parameter is the author's ID.
authorUpdateRequest	id (required)	Retrieves specific fields for the author identified by the provided id. Used to update author data during the biweekly database update process. Accepts only the OpenAlex id of the author, obtained during the initial seeding of the database.
workUpdateRequest	id (required)	Updates data related to a work asset retrieved from OpenAlex. Accepts only one parameter: the OpenAlex id of the work to be updated.

authorWorksRequest	id (required), page = 1 (optional), ignore_field_selection = false (optional), additional_filters = [] (optional)	Retrieves all works associated with an author. Manages resource allocation efficiently while ensuring all works are fetched. Processes works in a staggered manner if their quantity exceeds a specified limit. May recursively call itself to retrieve and process all works. Required parameter is the author's ID. Optionally, accepts an array of additional filters for the request.
--------------------	---	---

4.1.3 ORCID API Controller

The ORCID API controller works much like its OpenAlex counterpart. It includes methods used to retrieve author and work data from ORCID, just as it is done for OpenAlex. ORCID is another key source for my project, providing valuable information about authors and their works. This controller class manages requests to fetch author profiles, works, and important details from the ORCID database. The key methods of this class are:

Table 4.2: ORCID API Controller Methods

Method	Parameters	Description
authorRequest	id (required)	Retrieves author information using the author's ORCID unique identifier. The method accepts the author's id as a required parameter.
workRequest	path(required)	Retrieves work information using the work's unique ORCID path. It accepts the path as its own and required parameter and internally builds the url using the path provided.

4.1.4 DOI API Controller

The DOI API controller focuses solely on retrieving work information from the DOI repository. Its structure is simpler than the previous two and straightforward, containing only one method:

Table 4.3: DOI API Controller Methods

Method	Parameters	Description
workRequest	doi (required)	Retrieves work information using the work's Digital Object Identifier (DOI). It accepts the identifier as its own and required parameter.

Having discussed the API controllers dedicated to interacting with data sources such as OpenAlex, ORCID, and DOI, we've gained insight into the structure and functionality of each controller. These controllers play a crucial role in fetching author and work data from their respective repositories, ensuring smooth integration with my application. In the next section, we'll delve deeper into the initial database seeding process, exploring how the database is populated with essential data from these sources.

4.2 Initialize Database Job

The InitializeDatabaseJob is responsible for gathering the information from the predefined sources (OpenAlex, Crossref, ORCID), processing them, and saving them in our local database when the application is first being set up.

The initial phase involved retrieving identification data from the academic staff at the university. OpenAlex can utilize various identifiers such as ORCID, or Scopus ID, or if it is already known, the OpenAlex id to fetch detailed information about each author from its database. Given its promising potential, I decided to start the retrieval process with OpenAlex. This database appeared to offer a comprehensive repository, potentially from multiple sources including ORCID and Scopus.

After gathering the required identifiers from the academic staff, I organized this information into an array. Each entry in the array included the identifier provided by the staff, along with their first and last name, and their email address. Next, the service traversed through this array, making individual requests to the OpenAlex repository for each staff member's information. Since not all staff members necessarily provided the same type of identifier, it was essential to implement a mechanism to distinguish between them and ensure the correct requests were made to retrieve the relevant information.

4.2.1 Managing Different Identifier Types

When processing the identifiers provided by the academic staff, the system needed to be able to distinguish between different types of identifiers and make the appropriate requests to the OpenAlex api. For instance, when using the OpenAlex id, a specific API endpoint was utilized for the search. However, in the case of Scopus or ORCID ids, a different approach was necessary. In these cases, the system utilized the filter API and applied specific filters to search for authors using these identifiers. This mechanism ensured that the correct requests were made to retrieve the relevant information from the OpenAlex repository, regardless of the type of identifier provided.

```
public static function matchIdentifierToEndPoint($id): mixed {
    // Check what type of id is used to fetch the author's info from
    OpenAlex api.
    $id_type = Ids::getIdType($id);

    return match ($id_type) {
        Ids::ORC_ID, Ids::OPEN_ALEX =>
        OpenAlexAPI::authorRequest($id),
        // Using a filter request since OpenAlex can only find
        Authors by scopus using filters.
        Ids::SCOPUS => OpenAlexAPI::authorFilterRequest($id, false,
        true)
    };
}
```

Listing 4.1: Dynamic Author Data Retrieval Mechanism

This listing shows an important function that's part of the system that fetches data from OpenAlex. It takes an identifier and figures out the correct way to get author information from OpenAlex. The function starts by identifying what type of id it is, ORCID, OpenAlex, or Scopus using the getIdType method.

- **ORCID and OpenAlex IDs:** For these types of ids, the function makes a straightforward request using authorRequest. This approach directly retrieves detailed information about the author associated with the ID.
- **Scopus IDs:** Because the OpenAlex API cannot directly search authors using Scopus ids, the function makes an authorFilterRequest instead. This request uses filters to search for the author effectively, overcoming this limitation.

This process ensures that the process can handle different types of ids from staff members, using OpenAlex's api to collect author and work information.

```
public static function getIdType(string $id): string {
    // If the id contains '-', then OrcId id is used.
    if (str_contains($id, '-'))
        return self::ORC_ID;
    // If the id starts/contains 'A' or 'W', then OpenAlex id is
    used.
    elseif (str_starts_with($id, 'A') || str_starts_with($id, 'W'))
        return self::OPEN_ALEX;
    // If the id is just numbers, then scopus is used.
    else return self::SCOPUS;
}
```

Listing 4.2: Identifier Type Determination Function

This figure presents the `getIdType` function, essential for identifying the type of author identifier provided. The function checks the format of the identifier and categorizes it as ORCID, OpenAlex, or Scopus ID based on specific characteristics:

- **ORCID ID:** The function checks if the identifier contains a hyphen ('-'). If it does, the identifier is recognized as an ORCID ID.
- **OpenAlex ID:** The function looks for identifiers that start with 'A' or contain 'W'. If either condition is met, it classifies the identifier as an OpenAlex ID.
- **Scopus ID:** If the identifier is made up of only numbers, it is identified as a Scopus ID.

This step is crucial for ensuring that subsequent data retrieval requests are directed to the correct API endpoint, facilitating efficient and accurate data gathering.

4.2.2 Processing the OpenAlex response

In this section, I detail how I process the responses from OpenAlex after fetching author identifiers. This involves two main steps: retrieving author information and their associated works.

1. Retrieve Authors:

- **Match Identifier to Endpoint:** I match each author's identifier to the correct OpenAlex API endpoint.
- **Request Author Information:** A request is made to OpenAlex to fetch comprehensive author data.
- **Process Response:** I process the incoming data to extract important details such as the author's names, citation metrics, and unique identifiers.
- **Create Author Entities:** New author entities are created in the database, storing the extracted information.
- **Extract Works URL:** The URL linking to the author's publications is extracted from the data for further processing.

2. Retrieve Works for Each Author:

- **Request Works Data:** Using the URLs extracted earlier, I request detailed information about each author's works.
- **Retrieve Work Details:** I process responses to gather data on each work, including titles, publication sources, citation metrics, language, and document type.
- **Create Work Entities:** New work entities are created and saved in the database, each containing the details of the publications.
- **Extract Citation Statistics:** Yearly citation statistics are extracted from each work's data and stored as separate statistic entities.
- **Link Authors to Works:** New author entities are created for each co-author mentioned in the works, and relationships are established linking authors to their respective publications.

4.2.3 ORCID and Crossref Integration

ORCID enriches author profiles by providing additional information. However, since much of the information overlaps with what I've got from the other sources, such as names, the focus is primarily

on extracting biographies where available. ORCID lacks statistical data, limiting the scope of useful data.

Then, the process proceeds to retrieve all works associated with the author from the ORCID repository, again using the works URL provided in the ORCID author's response. For each work processed, a new work entity is created. As previously detailed, this involves segregating entities by data source to ensure better data organization.

Moreover, the process creates an aggregated version of each work, omitting statistical data, to monitor unique works across different sources. After handling each work, the procedure makes a request to Crossref using the work's DOI, resulting to the creation of an additional work entity. This method results in multiple versions of the same work, in line with the strategy of maintaining discrete records from each source.

4.3 Update Database Job

This function starts the biweekly update process, ensuring that the database remains current with the latest statistics. It begins by checking for any changes in the author's data retrieved from OpenAlex. If updates are detected, such as changes in the number of works or citations, the function automatically fetches and integrates the new information. Additionally, it synchronizes the works data from all the available sources. The function also compares the last update date locally stored with the latest update date from the API response. If changes are found, indicating new data, it updates relevant fields such as `cited_by_count`, `works_count`, and yearly statistics. This process ensures that our database stays up to date, providing accurate insights into author activity over time.

To enhance efficiency, the update routine has been optimized to focus on fetching and updating the latest statistics for the current year. By targeting only the most relevant data that is likely to change, the process remains streamlined and resource efficient. This optimization ensures that unnecessary processing of unchanged data from previous years is kept to a minimum, allowing for faster and more targeted updates.

To facilitate the update process, I've leveraged Laravel's job scheduling features and a cron job. These tools automate the biweekly update routine, ensuring the database remains current without manual intervention.

Chapter 5 : Back-End Infrastructure and API

In this chapter, I dive into the back-end infrastructure that powers my application, focusing on elements such as routing, APIs and API resources, and other server-side operations. After covering the database seeding and update processes, this section explains how the application processes requests and manages data flow. I'll break down the technical architecture, focusing on the interaction between the front end and back end via the API. I will explain how data flows from user actions on the front end through to the server-side controllers that handle these requests, explaining how each page interfaces with back-end services to retrieve, process, and display data.

The application uses a customized API to link the front end to the back end, handle use requests and process data to show to the end user. More specifically into the functionality of the API:

- **Academic Data Retrieval:** Endpoints to retrieve author profiles and publication records. These endpoints handle requests from the front end and send back the processed data ready to be shown to the user.
- **Role-Based Access Control:** Custom middleware that works with the university's authentication system. This way users get to see and perform actions based on their role as students or academics.
- **Dynamic Search:** Search functionality, allowing for sorting and searching based on various criteria such as publication dates, source, authors or thematic scope.

5.1 Requests Utility Class

As mentioned in a previous chapter, I introduced the Requests utility class and discussed some of its functionality related to handling requests to external APIs. This section focuses on the part that handles responses for internal API communications between my backend and front end. This dual use of the Requests utility class ensures a uniform approach to managing both internal and external API's, enhancing the consistency and maintainability of response handling throughout the application. I will now delve into each function, explaining their specific purposes and providing practical use cases to demonstrate how they are used throughout the application.

5.1.1 Success Response Function

Purpose: This function is used to return a successful HTTP response to the client.

Parameters:

- **\$message:** A message describing the success.
- **\$data:** Additional data to be returned with the response.
- **\$code:** HTTP status code, defaulting to 200.

Example Use Case: Returning user data upon successful login.

```
public static function success($message, $data = [], $code = 200):
    JsonResponse {
        return response()->json(['ok' => true, 'success' => $message,
            'code' => $code, 'data' => $data], $code);
    }
```

Listing 5.1: Function to handle successfully fulfilled requests.

5.1.2 Client Error Response Function

Purpose: Handles responses for client-side errors.

Parameters:

- **\$message:** Error message describing what went wrong.
- **\$code:** HTTP status code, typically 400 for client errors.
- **\$data:** Additional data relevant to the error.

Example Use Case: Informing the user that there's something wrong with their request.

```
public static function clientError($message, $code = 400, $data =
[]): JsonResponse {
    return response()->json(['ok' => false, 'error' => $message,
'code' => $code, 'data' => $data], $code);
}
```

Listing 5.2: Function to handle client-side errors.

5.1.3 Server Error Response Function

Purpose: Handles responses for server-side issues, such as exceptions or processing errors that prevent the server from fulfilling the request.

Parameters:

- **\$message:** Error message describing what went wrong.
- **\$code:** HTTP status code, typically 500 for internal server errors.
- **\$data:** Additional data about the error, if applicable.

Example Use Case: Informing the user that something went wrong while the server was processing their request.

```
public static function serverError($message, $code = 500, $data =
[]): JsonResponse {
    return response()->json(['ok' => false, 'error' => $message,
'code' => $code, 'data' => $data], $code);
}
```

Listing 5.3: Function to handle server-side errors.

5.1.4 Missing Parameter Error Function

Purpose: Specifically designed to handle scenarios where required parameters are missing from a request.

Parameters:

- **\$parameter:** The name of the missing parameter.

Example Use Case: Inform the user about a required parameter missing from the request, preventing the server from fulfilling it.

```
public static function missingParameterError($parameter):
    JsonResponse {
        return response()->json(['ok' => false, 'error' =>
            "Parameter $parameter is marked as required", 'code' => 401,
            'data' => []], 401);
    }
```

Listing 5.4: Function to handle missing request required parameter errors.

5.1.5 Authentication Error Function

Purpose: Handles authentication failures, such as when a user is not logged in but tries to access restricted resources.

Example Use Case: Informing the user that the action they're trying to perform requires them to be authenticated first (logged in).

```
public static function authenticationError(): JsonResponse {
    return response()->json(['ok' => false, 'error' => 'You need to
        be logged in to perform this action.', 'code' => 401, 'data' =>
        []], 401);
}
```

Listing 5.5: Function to handle authentication errors.

5.1.6 Authorization Error Function

Purpose: Handles authorization failures, returning an error when a user attempts to perform actions they do not have permissions for.

Example Use Case: Informing the user that they are not authorized to perform specific actions.

```
public static function authorizationError(): JsonResponse {
    return response()->json(['ok' => false, 'error' => 'You are not
        authorized to perform this action.', 'code' => 403, 'data' =>
        []], 403);
}
```

Listing 5.6: Function to handle authorization errors.

Now that we've covered the different functions in the Requests utility class and how they help manage the application's API responses, we will look at the key benefits this class offers.

5.2 API Resources

When building a robust and scalable application, it is crucial to manage how data flows and is presented to the end-users. Laravel provides a powerful feature known as API Resources that allows developers to transform any models into JSON structures. This feature is crucial to my application as it ensures that the data delivered through the APIs is consistent and also tailored to meet the needs of the front-end. API Resources act as a layer between Eloquent models and the responses that are returned by the different APIs. They help in customizing the API response without changing the model structure, making it easier to make changes to the API and maintain it by having the logic centralized to one class and avoid duplicated code.

In this section, I will go through all the API resources used by my application to ensure data consistency and help manage the exact data to be exposed to the front end. Each API resource class contains the `toArray` function which takes in the resource and returns an array with the desired properties of this resource. This array is then stringified by Laravel and returned as JSON from any API. To avoid repetition and maintain focus on the transformations themselves, I will not explicitly name the `toArray` function each time it is discussed in the context of different resources. Instead, I will focus on describing the data processing and transformation that occurs in this method. Any other significant resource-specific methods will be specifically named and detailed as needed.

5.2.1 Author Resource

Conditional Loading and Validation of Statistics

The statistics relationship is conditionally loaded, ensuring that this data is only processed if it is included in the initial query to avoid unnecessary database queries. The loaded statistics are then checked to ensure they are valid, meaning they are not instances of `MissingValue` (like undefined) and contain actual data.

Handling Missing Years in Statistics

A range of years from the minimum to the maximum year found in the statistics is calculated. This range is then iterated through to ensure every year is represented in the statistics. If any year is missing, a default statistic with zero counts is added. This ensures that the client receives a continuous timeline, enhancing the visual representation of the data and making it easier for users to understand.

Constructing the Response

Basic author information is included, such as the author's ID, name, biography, various identifiers like OpenAlex, ORCID, and Scopus IDs, and whether they are a user or not. It also includes the total number of works and citations. A `WorkResource` collection is used to format the author's works when the relationship is loaded, and a `StatisticResource` collection is used for statistics if they are valid. Additionally, a URL to the author's page is provided.

5.2.2 Work Resource

Source Handling: **This resource class includes the `getSources` method which** is used to determine the source of the work. If the work is an aggregated source, it combines sources from all versions of the work into a single string. This is useful for works from multiple sources, giving a complete view of the sources for each entry.

Conditional Loading of Versions: Versions of a work are conditionally loaded based on a **Boolean passed when the collection is initialized**. This allows the API to flexibly include or exclude detailed version information based on the request, optimizing performance.

Dynamic Data

The function checks if a user is authenticated and whether they are an author of the work to determine if they can edit the work. Statistics and authors are conditionally included when loaded, ensuring they are only processed if explicitly requested, which enhances performance.

Data Representation

The resource returns a wide range of work attributes, including identifiers (DOI and title), abstract, publication year, and language. Work concepts are out of scope for this version of the application and are expected to be utilized in the next major release. It also includes URLs for direct access (`local_url`) and flags such as `is_oa` to indicate if the work is open access.

5.2.3 PaginatedWorkCollection Resource

This resource collection class is designed to handle collections of Work models that are paginated. It allows inclusion of version details for each work, based on a boolean condition. This makes it flexible and highly useful for scenarios where detailed information about each work's versions may or may not be needed.

Conditional Version Loading

In the constructor, a flag is set based on the passed parameter, determining whether version details of each work will be included in the response. The method maps each work in the paginated collection to a WorkResource, passing the flag to control the inclusion of versions.

Structure of the Paginated Response

Each work is transformed using a WorkResource to ensure consistent formatting. The response includes navigational links (`first`, `last`, `prev`, `next`) to help front-end applications understand the pagination context and provide appropriate navigation controls. Additionally, it contains metadata about the pagination, such as the current page, total pages, links, and items per page, which is crucial for front-end components that render paginated data.

Customization and Flexibility

The class provides a way to handle collections of works, allowing for easy adjustments in how detailed the data needs to be, depending on the use-case. By providing pagination metadata and navigational links, the class makes it easy to navigate through large datasets.

5.2.4 WorkCollection Resource

The WorkCollection class, just like PaginatedWorkCollection class, is built on top of Laravel's ResourceCollection compared to the previous resources classes to manage a collection of Work resources. This was necessary to address specific requirements in our application that are not supported by the default Resource:collection function.

Purpose and Need for Customization

The need to conditionally load and display versions of each work based on specific scenarios led me to create a separate new resource collection class. The standard collection method of a resource class does not provide built-in support for such conditional logic directly within the collection since it has to be passed to its constructor. Therefore, creating a custom collection class allowed me to implement this necessary functionality.

Functionality

The constructor accepts a work collection and a Boolean flag. This flag determines whether the versions associated with each work should be loaded. By allowing this to be dynamically set, the class provides flexibility in how data is presented based on the context of the request. This class simply iterates over each item in the collection, wrapping it in a `WorkResource` and passing the Boolean flag to its constructor.

5.2.5 User Resource

The `UserResource` class is particularly important, ensuring that user data is presented consistently and securely across the application. It selectively exposes user attributes and related data, ensuring that sensitive information is appropriately hidden while still providing the necessary information for the front end to function as expected.

User Information: Only basic information is exposed to the front-end, guarding sensitive information to only be processed by that back-end.

- `id`: The unique identifier for the user.
- `name`: The user's full name.
- `admin`: A Boolean flag indicating whether the user is an admin.
- `staff`: A Boolean flag whether the user is academic staff.
- `email`: The user's email address, essential for communication or identification purposes.
- `academic identifiers`: OpenAlex, ORCID and Scopus ids, used to link the user to external sources.
- `associated author`: Conditionally includes the author associated with the user if one exists. An author resource is used to ensure the data is consistent through the application.

5.2.6 Group Resource

This resource class is structured to manage the complex nested relationships of a group, including its parent and children. It utilizes recursion to ensure that both parent groups and their subgroups are handled correctly, allowing for a comprehensive representation of group hierarchy. This recursive and adaptable design makes it ideal for representing complicated group structures in a clear and consistent manner, enhancing the group handling and presentation within the application.

Basic Group Information: `id`, `name`, `description` are the standard attributes that provide basic information about the group.

Nested and Conditional Resources

The resource handles nested and conditional resources to ensure data consistency and efficient presentation. It uses an author resource collection to format and include a list of group members when they are loaded, maintaining consistency with individual author presentations. If present, the parent group's data is included using the same class. Works associated with the group are included using a `PaginatedWorkCollection`, efficiently managing potentially large sets of work data. Nested groups

(children) are conditionally loaded and represented as a collection of `GroupResource`, useful for displaying group structures or sub-groups. Additionally, the application conditionally adds counts of works sourced from different databases (like OpenAlex, ORCID, and CrossRef) if these counts are provided in the additional parameters.

5.2.6 Statistic Resource

The `StatisticResource` class is used to standardize and streamline the presentation of statistical data across the application. This class is particularly vital for ensuring that statistics related to authors and works are consistently formatted so they are easily manageable by the front-end components of my application.

Asset Type

A match expression is used to determine the type of asset (Author or Work) associated with the statistics. This allows the client-side to adapt dynamically based on the type of data being handled.

Statistic Attributes

- works count: An attribute that is conditionally included only when the statistic is related to an author.
- cited count: Represents the number of times the asset has been cited.
- year: The year to which the statistics apply.

Consistency and flexibility

Ensures that statistical data across different parts of the application follows a consistent format, which simplifies data handling on the front end. The type checking within the resource allows for dynamic responses for the specific needs of different asset types.

5.3 Form Requests

In Laravel, `FormRequest` classes are classes that encapsulate API validation logic with the aim of keeping controllers clean and focusing them more on handling request data rather than verifying its integrity and format. These classes are useful because they allowed me to define specific rules and authorization logic that apply to the data coming into the application from any API. By using these classes, I ensure that only valid and authorized data is processed, enhancing the security and reliability of the application. In the context of my application, I've created 5 request classes, for specific APIs that require complex data validation. In this section, I will go over these 5 request classes, explaining their purpose and the validation they apply to the incoming data.

All 5 classes, come with their own ***authorize*** method, which determines whether a user is authorized to perform this action, a ***rules*** method, which has the validation rules for the incoming data, and the ***messages*** method, which returns customized ***messages*** in case any validation rule fails. Some of them also come with the ***after*** method, which performs additional validation after the initial rules have successfully been checked, and before the data is passed for further processing to the corresponding controller action.

5.3.1 Add Group Member Request Validation

This request class is responsible for validating the data related to adding group members. It confirms that the information provided is correct and follows the necessary format, preventing errors during the addition of new members to groups. Multiple additions may be performed in a single request.

Validations Performed:

- **User Authorization:** Ensures the user is an admin, since only admins can manage group memberships. In any other case, the request is automatically rejected with a 403 Forbidden status.
- **Group id Verification:** Checks if the **group_id** provided is present and correctly formatted as an integer. This ensures that the group to which members are being added actually exists within the database.
- **Authors Validation:** Verifies that the **authors** field is provided, is an array, and contains valid integers. Each integer represents an author ID, confirming that only legitimate author identifiers are submitted.
- **Individual Author Validation:** Ensures that each author ID contained in the **authors** field corresponds to an actual author.

Additional Checks After Validation:

- After the initial validation rules are applied, further checks ensure that each author to be added is not already a member of the specified group. This prevents duplicate group memberships.
- It also verifies that each author being added is a registered user, since only registered users can be members of groups.

Validates: AddGroupMember Request

5.3.2 Remove Group Member Request

This request class is responsible for validating the data related to removing group members. It confirms that the information provided is correct and follows the necessary format, preventing errors during the removal of new members to groups. Only 1 removal per request is allowed.

Validations Performed:

- **User Authorization:** Ensures the user is an admin, since only admins can manage group memberships. In any other case, the request is automatically rejected with a 403 Forbidden status.
- **Group id Verification:** Checks if the **group_id** provided is present and an integer. This ensures that the group to which members are being added exists within the database.
- **Authors Validation:** Similarly, checks that the **author_id** is present and correctly formatted as an integer, ensuring the author exists.

Additional Checks After Validation: Additional validation ensures that the specified author is currently a member of the specified group before proceeding with removal. If the author is not a member, an error is added to the validation errors, preventing incorrect removal operations.

5.3.3 Create Group Request Validation

This request class is responsible for validating the data related to creating a new group. It confirms that the information provided is correct and follows the necessary format, preventing errors during the creation of a group. Only 1 group may be created per request.

Validations Performed:

- **User Authorization:** Ensures the user is an admin, since only admins can manage groups. In any other case, the request is automatically rejected with a 403 Forbidden status..
- **Name Validation:** Ensures that a name is provided for the group, preventing anonymous group creation. The uniqueness and appropriate length of the name provided are also validated.
- **Description Validation:** Like the name, the description's existence is also validated, ensuring that all groups have some kind of contextual information. The description is also validated not to go over the 255 character limit.
- **Parent Group Validation:** If a parent id is provided, ensures it is an integer, and that it corresponds to an actual group in the database to ensure a clean group hierarchy.

Validates: CreateGroup Request

5.3.4 Delete Group Request Validation

This request class is responsible for validating the data related to deleting a new group. It confirms that the information provided is correct and follows the necessary format, preventing errors during the deletion of a group. Only 1 group may be deleted per request.

Validations Performed:

- **User Authorization:** Ensures the user is an admin, since only admins can manage groups. In any other case, the request is automatically rejected with a 403 Forbidden status.
- **Group id Validation:** The id of the group to be deleted must be provided in the request. This ensures that the delete operation is targeted and prevents unintentional deletions.

Validates: DeleteGroup Request

5.3.5 Work Filter Request Validation

The WorkFilterRequest class is a sophisticated request class that handles the validation of searching, filtering, and sorting works. This class plays a crucial role in ensuring that search parameters are valid, and that data integrity is maintained during complex query operations.

Validations Performed:

- **Query Parameters:**
 - **Pagination Control:** Users can specify how many results per page they prefer, which must be a number, and more specifically an integer.

- **Author Filtering:** Users can filter works by specific author IDs.
- **Source Filtering:** Allows filtering by the source of the works, such as OpenAlex, ORCID, or CrossRef.
- **Publication Year Range:** Users can specify a range of publication years (from and to).
- **Citation Count Range:** Enables filtering based on the minimum and maximum citations.
- **Work Type Filtering:** Users can filter based on specific types of work by providing the type of the types of works they need as an array of strings.
- **Sorting Capabilities:** Users can sort the results by various attributes like DOI, title, publication year, and citation count, among others. Sorting can be done in ascending or descending order, providing a highly customizable way to view results.
- **Inclusion of Related Data:** Users can specify if they want to include any work relationships, such as authors, versions, and statistics with the returned works, by providing an array of the requested relationship names.

Validates: WorkFilter Request.

5.4 Page-Specific Controller Actions and APIS

In this section, we explore the functionality of the controllers corresponding to different pages of the application. Each sub-section details the controller actions, data handling, and security measures tailored to the requirements of each page.

5.4.1 Home page

The Home Page is the initial interface encountered by users upon accessing the application. Designed for simplicity and ease of navigation, this page provides an overview of the application's purpose and core functionalities. It features key insights and statistics about authors and publications, which are dynamically generated from the database to engage users with relevant and interesting information. Additionally, the Home Page displays a login button, offering users a means to authenticate through the university's authentication system and access more features of the application.

The index function of the HomeController is responsible for fetching and preparing data to be displayed on the Home Page.

```

Cpublic function index(): Response {
    $most_cites_users = AuthorResource::collection(Author::mostCitations(5)->user()->get());
    $most_works_users = AuthorResource::collection(Author::mostWorks(5)->user()->get());
    $most_cites_works = new WorkCollection(Work::with('authors')-
>source(Work::$openAlexSource)->mostCitations(5));
    return Inertia::render('Routes/Home/HomePage', ['mostCitationsUsers' => $most_cites_users,
    'mostWorksUsers' => $most_works_users, 'mostCitationsWorks' => $most_cites_works]);
}

```

Figure 5.7: Home Controller index function.

In more detail:

- **Retrieve Most Cited Authors:** Retrieves the top five authors with the most citations using the a custom scope method defined in the Author model. The data is then processed using an AuthorResource collection to standardize and format the output data sent to the front end.
- **Retrieve Authors with Most Works:** Retrieves the top five authors with the highest number of works using the a custom scope method in the Author model. The results are then encapsulated in an AuthorResource collection to ensure consistency in the output data.
- **Retrieve Most Cited Works:** Retrieves the top five works with the most citations, filtering to include only those sourced from OpenAlex since that's the primary source of statistical data. The results are then encapsulated in a WorkCollection to ensure consistency in the output data format.
- **Rendering the View:** Renders the HomePage component in React while also passing the processed data to that component as props.

5.4.2 Author page

The Author Page serves as a profile for an individual academic, showcasing detailed information including their publications, statistical data, and biographical details. This page is designed to provide a deep dive into the academic contributions of faculty members and researchers at the university.

In more detail:

- **Fetch Author information and statistical data:** Retrieves the specific author's data along with associated statistics using their unique OpenAlex ID to ensure that the page displays up-to-date information. The author's data is then wrapped in an AuthorResource to ensure that the front end receives consistent output.
- **Count Works by Source:** Counts the works associated with the author sourced from ORCID, OpenAlex, and Crossref, performing similar counting for each source
- **Rendering the View:** Renders the AuthorPage component in React and passing the processed data as props.

5.4.3 Work page

The Work Page is a component used to present detailed information about a specific publication. This page is essential for showcasing the work's content, its authors, and relevant statistical data.

In more detail:

- **Fetch Work Details:** Retrieves the work information based on the provided ID, while lazy loading the associated authors and any statistical data to reduce the number of database queries. The retrieved work data is then wrapped in a WorkResource to standardize the output format.
- **Retrieve Work Versions:** Retrieves the versions of the work, allowing users to see different versions of the same work from different sources. These versions are then wrapped in a WorkResource collection, formatting each version consistently with the base work.
- **Rendering the View:** Renders the Work Page component in React.

5.4.4 Groups page

The Groups Page serves as a hub for users to explore and interact with various academic groups within the university. It is designed to facilitate community engagement and collaboration among students and faculty.

The index function of the Group Controller is a simple function used to simply render the GroupsPage React component. All the hydration of the page is done by various API calls that I will be explaining in the next section.

```
public function getAllGroups (Request $request) :
AnonymousResourceCollection {
    return GroupResource::collection(Group::noParent()->noMembers()-
>with(['childrenRecursive'])->get());
}
```

Figure 5.8: Group Controller function to retrieve all groups.

Figure 5.11 illustrates a core GroupController function that fetches and organizes group data for display in a tree view on the Groups Page. This processes group data in a way to ensure users can effectively navigate through the group hierarchy, which is structured according to group relationships and the role of the authenticated user.

In more detail:

- **Scoped Queries:** The noParent scope is applied to retrieve only groups without a parent, establishing the root nodes of the tree view for clear hierarchical navigation. The noMembers scope filters the groups based on the authenticated user's role: Admin users can view all groups, including those without members, for full visibility and management purposes, while non-admin users and guests see only groups with active members, ensuring relevant content is displayed.
- **Eager loading relationships:** Recursively loads all child groups linked to each root group. This is primarily done to enhance the performance by reducing database queries necessary for dynamically rendering the group hierarchy.
- **Formatting the query result:** The resulting groups are then wrapped in a GroupResource collection, allowing for consistent output on the frontend.

After discussing how all groups are fetched, it is essential to delve into the specifics of retrieving detailed information about a specific group. This occurs through multiple API calls initiated when a user clicks on a group.

The getGroup function is a key component of the Groups Controller that handles the retrieval of detailed information about a specific group identified by its ID. This API call is critical for providing users with a comprehensive view of the group's structure, its members, and their associated works across different sources.

In more detail:

- **ID Validation:** The function first checks if the id parameter is provided in the request. If the parameter is missing, it returns a missingParameterError, informing the user that the 'id' parameter is required.
- **Group Retrieval:** The function attempts to find the group by its ID, including details about its members, the parent group, and statistics related to each member. If no group is found matching the provided ID, the function returns a client error indicating that no group exists with that ID.
- **Success and Data Handling:** If the group exists, the function calculates the number of works associated with the group's members from different sources.

- **Response Construction:** If the group data is successfully compiled, a success response is returned, including the data calculated in the previous steps. If there is any failure in processing, a `serverError` response is returned instead.

Following the retrieval of group details, another API call retrieves the works associated with the group's members. This call uses the `filterWorks` API, specifically using Laravel's relationships to query works through author ids. In this case, the API focuses on filtering works where the authors are members of the selected group. A full description of this API's capabilities, which include various other filters, will be provided in a later section dedicated to APIs used in multiple components.

5.4.5 Successful login page

In this section, we explore the backend functionality for the last page of the application. This part ensures the correct page component is rendered based on who has just logged in. We'll look at how the user roles checks are performed, and the actions performed in each case.

This controller includes a function that is triggered after a user successfully logs in through the OAuth2 provider. It processes the user's data received from the OAuth provider, checks specific conditions related to the user's account, and then decides the appropriate redirection within the application. Before we examine the detailed steps and implementation of this function, it is beneficial to first go over the custom Socialite driver. This driver plays a crucial role in connecting with the university's OAuth2 provider, used for retrieval and handling of user data.

Authentication via Socialite and user data retrieval

Laravel Socialite is a package that provides a simple way to handle OAuth authentication with various popular social media platforms like Google, Facebook, GitHub, and GitLab. By handling much of the boilerplate code required for OAuth authentication, Socialite makes it easier to integrate social logins in their applications.

While Socialite supports a wide range of social platforms out of the box, it does not cover every OAuth provider, like those used by some universities or organizations. In my case, the university's OAuth provider was not supported by any of Socialite's default or community-maintained drivers. These community drivers often extend Socialite's capabilities to other platforms, but none were suitable for my project's specific needs.

To integrate my university's OAuth provider, I needed to implement my own custom Socialite driver. Socialite is designed with flexibility in mind, allowing developers to override its methods to support additional OAuth providers. By overriding certain methods in Socialite, I managed to make the authentication process to fit the requirements of my university's provider. The process involved defining the required redirection URL's to and back from my provider, retrieving the user data using the access token returned after a successful authentication, and mapping these details to a user object in my application. The rest of the OAuth flow continues to be managed internally by Socialite, preserving the ease of use offered by the package.

Constants and Overridden Protected Properties

- **STAFF_AFFILIATION_NAME Constant:** This constant defines a key affiliation name used to check if the authenticated user has a specific role within the university, in this case, 'staff'.

- **protected scopes Class Property:** The scopes array defines what access permissions the application requests. For this provider, it requests access to the user's profile information. This class property is overridden and used internally by Socialite.

Key Methods

1. **getAuthUrl():** This method constructs the authorization URL to which users are redirected when initiating an OAuth login. It uses `buildAuthUrlFromBase`, a method provided by Socialite, to assemble the full URL with the base path and state parameter. The state parameter helps prevent CSRF attacks during the OAuth process. The authorization URL is safely stored in the `.env` file and retrieved during runtime.
2. **getTokenUrl():** Returns the URL used to exchange an authorization code for an access token. After users authorize the application, the OAuth provider calls back with an authorization code that is exchanged for an access token using this URL. The base token URL is safely stored in the `.env` file and retrieved during runtime.
3. **getUserByToken(token):** The method retrieves the user's profile information from the OAuth provider using the access token by making an HTTP request to the provider's profile endpoint, utilizing the Requests utility class to fire the request and process its response. The profile base URL is securely stored in the `.env` file and retrieved from there. After making the request, the method processes the response to extract the user details.
4. **mapUserToObject(array user):** The method converts the array of user information into a User entity, standardizing the data according to the application's needs for consistent processing regardless of the OAuth provider's data format. It maps attributes such as id, first name, last name, email, and display name from the provider's data and checks the user's role.

Security and Usability

- The provider ensures that all communication with the university's OAuth provider is secure, and that user data is handled safely and responsibly.
- All sensitive information like redirect and base URL's used by this class are safely stored in the `.env` file and are retrieved at runtime.
- It simplifies the process for the developer by handling the complexities of OAuth token management and other similar tasks internally.

Having explained the details about user authentication through the university's OAuth provider using Socialite, we can now dive into the implementation of the function that handles the redirect back to my application when a user is successfully authenticated. We will see how this function handles different use cases based on the status and role of the authenticated user.

The detailed steps are:

1. **User Data Retrieval and Authentication:** Using the `user()` function of the Socialite driver, the function retrieves the user's information and attempts to fetch and update the user in the database if they exist or create a new user entry. The `createOrUpdate` function is used to either find an existing user by unique identifiers retrieved from the OAuth provider, or create a new user with the provided attributes.
2. **Logging in the User:** The `Auth::login` method of the Auth Facade securely logs the user into the application using Laravel's built-in authentication system. The second parameter (`true`) indicates that the session should remember the user.
3. **Role and status check:** The function performs a condition check to decide which page the authenticated user should be redirected to. It checks whether the user is Staff (aka Academic) and if so, whether the user lacks certain necessary identifiers, which are crucial for accessing most features of the application. It also checks if the user is not an admin (in which case they're not

required to provide any identifiers). `missingAllIdentifiers()` is a method on the User model that checks for the presence of required user identifiers, such as OpenAlex, ORCID or Scopus ids.

4. **Redirection Logic:** If the condition is true (the user is missing identifiers and is not an admin), the user is redirected to a route prompting them to provide the missing identifiers. If the condition is false, the user is directed to a route indicating successful authentication, confirming they have all necessary permissions and identifiers. This page welcomes the authenticated user and informs them that they will be redirected back to the Home Page after a short period of time (5 seconds).

Now that we have explored the backend functionality specific to each page of the application, we can move to the shared APIs section which includes APIs that provide services across multiple components.

5.5 Common APIs: Shared across multiple components

In this next section, we delve into the rest of the APIs. These APIs are not limited to single page interactions but are instead critical to the operation of multiple components. By centralizing core functionalities such as search, asset filtering, shared APIs reduce redundancy, and maintain consistency across different parts of the application. This approach not only simplifies maintenance but also enhances the scalability of the system, making it easier to modify existing features or even introduce new ones. We will examine how these APIs are structured, their functionality, and how they are used across multiple components.

Starting with search APIs, I will now explain how search is implemented and used in my application, specifying the different search methods, generic or built for specific asset types.

5.5.1 General Search

Purpose: This endpoint is used as the basic search tool within my application, capable of searching for multiple asset types, including authors and works based on the provided query.

Data Validation: All validation is performed by the controller action itself.

Method: GET

Parameters:

Table 5.1: General Search API parameters.

Parameter	Type	Description
query	string	The query to search for in the database.

URL: /search

Response:

Success: If the operation is successful a `Requests:success()` (see 5.1.1) response is returned containing following data:

Table 5.2: General Search API parameters.

Name	Type	Description
query	string	The query provided.
authors	AuthorResource::collection	The search's authors results.
works	WorkResource::collection	The search's works results.

Error: If no query is provided, an empty array of results is returned instead. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned.

Functionality:

- **Multi-asset type search:** Simultaneously searches for authors and works based on DOI title, or OpenAlex id for works and the academic identifiers, and name for the author.
- **Result Limitation:** Limits the results to the top five most relevant authors and works.
- **Results Combination:** Aggregates search results into a unified response.
- **Error Handling:** Includes error logging and handling, ensuring that any errors are captured and peacefully handled.

5.5.2 Search Authors by Identifiers

Purpose: Specifically built to search only for authors, this endpoint is optimized for scenarios where a targeted author search is necessary.

Data Validation: All validation is performed by the controller action itself.

Method: GET

Parameters:

Table 5.3: Search Authors API parameters.

Parameter	Type	Description
query	string	The query to search for in the database.

URL: `search/authors/identifiers`

Response:

Success: If the operation is successful a `Requests::success()` (see 5.1.1) response is returned containing following data:

Table 5.4: Search Authors successful response data.

Name	Type	Description
orc_id	string	The ORCID identifier to be queried.
open_alex	string	The OpenAlex identifier to be queried.
scopus	string	The Scopus identifier to be queried.
authors	<code>AuthorResource::collection</code>	The search's authors results.

Error: If no identifiers are provided, an empty array is returned instead. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned.

Functionality:

- **Focused Search:** Searches for authors, based on their name, OpenAlex, ORCID, and Scopus ids using a query.

- **Result Limitation:** Like the general search, this function also limits the results to the top five authors to maintain relevance and manageability.
- **Error Handling:** Includes error logging and handling, ensuring that any errors are captured and peacefully handled.

5.5.3 Search Where Not In Group

Purpose: This search endpoint is designed to find authors who are not members of a specified group, useful for searching which authors can be added to a group.

Data Validation: All validation is performed by the controller action itself.

Method: GET

Parameters:

Table 5.5: Search Where Not In Group API parameters.

Parameter	Type	Description
query	string	The query to search for in the database.
group_id	integer	The group id to associated authors results with.

URL: search/authors/not-in-group

Response:

Success: If the operation is successful a Requests::success() (see 5.1.1) response is returned containing following data:

Table 5.6: Search Where Not In Group successful response data.

Name	Type	Description
query	string	The query to further filter authors.
authors	AuthorResource::collection	The search's authors results.
group	int	The group's id to exclude authors from.

Error: If no group id is provided, then a Requests::missingParameterError() (see 5.1.4) response is returned instead. If an error occurs during the database transaction, a Requests::serverError() (see 5.1.3) response is returned instead.

Functionality:

- **Exclusion Criteria:** Excludes all authors who are already members of the specified group, which is critical for adding new members while avoid duplicated entries.
- **Conditional Execution:** The search executes only if a valid group ID is provided, ensuring integrity, and preventing unnecessary processing and database queries.
- **No query required:** Can operate with or without a query. If a query is present, it performs a filtered search; if not, it simply returns all the authors who are not group members.
- **Result Formatting:** Results are formatted and returned as a collection, using the same structured approach as other search functions, ensuring consistency across the application.

5.5.4 Work Filter

Purpose: This endpoint is used to search, filter, and sort works, based on various criteria making it highly adaptable for various cases within my application.

Data Validation: WorkFilterRequest Form Request (see 5.3.5).

Method: GET

Parameters:

Table 5.7: Work Filter API parameters.

Parameter	Type	Description
author_ids	array	An array of author IDs to filter works by authorship.
sources	array	The name of the concept.
from_pub_year	int	The external (OpenAlex) id of the associated Work entity.
to_pub_year	int	The ending publication year to filter works by.
min_citations	int	The minimum number of citations to filter works by.
max_citations	int	The maximum number of citations to filter works by.
work_types	array	The types of works to be filtered (journals, articles).
type_filter	array	An array of custom type IDs to filter works by their custom type.
with	array	An array of work relationships names to load for each work.
with_versions	bool	Indicates whether to include work versions in the results.
sort_by	string	The attribute to sort the results by.
sort_direction	string	The direction to sort the results in ('asc' or 'desc').
filter_visibility	bool	Filter works based on their visibility status.

URL: works/filter

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing following data:

Table 5.8: Work filter successful response data.

Name	Type	Description
data	array	The collection of works or data being returned.
links	string	Contains navigation links for pagination.
links.first	string	URL of the first page.
links.last	string	URL of the last page.
links.prev	string	URL of the previous page.
links.next	string	URL of the next page.
meta	array	Contains metadata information about the pagination.
meta.current_page	int	The current page number.
meta.from	int	The index of the first item on the current page.
meta.last_page	int	The last page number.
meta.path	string	The base path of the pagination.
meta.per_page	int	The number of items per page.
meta.to	int	The index of the last item on the current page.

meta.total	int	The total number of items.
meta.links	array	Collection of additional pagination links.

Error: If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

Functionality:

- **Dynamic Parameter Handling:** The function parses a set of parameters from the request, enabling users to fine-tune their queries based on:
 - **Pagination:** Users can specify the number of results per page.
 - **Author Filtering:** Allows filtering of works associated with specific authors.
 - **Source Filtering:** Works can be filtered by their source.
 - **Type and Publication Year Filtering:** Allows filtering based on specific types of works and publication years.
 - **Citation Count Filtering:** Allows searching for works within a specific citation count range.
 - **Sorting:** Results can be sorted by various attributes like title, publication year, etc., in ascending or descending order.
- **Conditional Loading:** Based on the `with` parameter, relationships such as authors, versions, and statistics can be loaded with the works, providing a rich data set in a single request.

Implementation: Flexible and Comprehensive Query Building: The function constructs a query based on the provided parameters, utilizing Laravel's query builder to efficiently manage database interactions. This includes:

- Conditional joins and filters based on author IDs, types, and publication years.
- Dynamic inclusion of relationships based on user requests, optimizing data retrieval for specific needs.
- Since the query result can be quite large, the response is always a paginated response, allowing for efficient and flexible handling of both small and large datasets.

After thoroughly exploring the back-end infrastructure and APIs that drive my application, we now transition to exploring the front-end. I will delve into how the application presents and handles this information and how users engage with it through various pages and components.

5.5.5 Get All Groups

Purpose: This endpoint returns all the groups that the user can view based on their role.

Data Validation: No validation required.

Method: GET

Parameters: This endpoint takes no parameters.

URL: groups/all

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing following data:

Table 5.9: Get All Groups Successful Response data.

Name	Type	Description
groups	GroupResource::collection	An array containing the groups retrieved.

Error: If an error occurs during the database transaction, a Requests::serverError() (see 5.1.3) response is returned instead.

Functionality:

- **Exclusion Criteria:** Depending on the role of the logged in user, the function returns the groups that are accessible by the user. If the user is an administrator they have full access to all the groups so they can edit them, but in any other case, only groups that have at least one member are retrieved instead, since there is no actual group data the user can benefit from.

5.5.6 Get Group

Purpose: This endpoint returns detailed information for a specified group, including its members, and its parent group (if present).

Data Validation: All validation is performed by the controller action itself.

Method: GET

Parameters:

Table 5.10: Get Group parameters.

Parameter	Type	Description
Id	int	The id of the group to be retrieved.

URL: groups/{id}

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing following data:

Table 5.11: Get Group API successful response.

Name	Type	Description
group	GroupResource	The details of the group retrieved.

Error: If no group id is provided, then a `Requests::missingParameter()` (see 5.1.4) response is returned instead. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

5.5.7 Create Group

Purpose: This endpoint is used to create a new group asset in the database.

Data Validation: `CreateGroupRequest` Form Request (see 5.3.3).

Method: POST

Parameters:

Table 5.12: Create Group parameters.

Parameter	Type	Description
name	int	The name of the group.
parent_id	int	An optional parent_id to associate with a parent group.
description	string	A short description for the group.

URL: `groups/create`

Response:

Success: If the operation is successful a `Requests::success()` (see 5.1.1) response is returned containing following data:

Table 5.13: Create Group API successful response.

Name	Type	Description
group	GroupResource	The details of the newly created group.

Error: If the name or description are not provided, the form request automatically rejects the request with a 400 error code. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

5.5.8 Delete Group

Purpose: This endpoint is used to delete a group asset from the database.

Data Validation: `DeleteGroupRequest` Form Request (see 5.3.4).

Method: POST

Parameters:

Table 5.14: Delete Group parameters.

Parameter	Type	Description
id	int	The name of the group.

URL: `groups/create`

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing a message that the operation was successful.

Error: If id parameter is not provided, the form request automatically rejects the request with a 400 error code. If an error occurs during the database transaction, a Requests::serverError() (see 5.1.3) response is returned instead.

5.5.9 Add Group Member

Purpose: This endpoint is used to add members to group asset.

Data Validation: AddGroupMemberRequest Form Request (see 5.3.1).

Method: POST

Parameters:

Table 5.15: Add Group Member parameters.

Parameter	Type	Description
group_id	int	The id of the group to add members to.
authors	array	The ids of the authors to be added as members of the group.

URL: groups/members/add

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing a message that the operation was successful.

Error: If the group_id parameter is not provided, the form request automatically rejects the request with a 400 error code. If an error occurs during the database transaction, a Requests::serverError() (see 5.1.3) response is returned instead.

5.5.10 Remove Group Member

Purpose: This endpoint is used to remove members from a group asset.

Data Validation: RemoveGroupMemberRequest Form Request (see 5.3.2).

Method: POST

Parameters:

Table 5.16: Remove Group Member parameters.

Parameter	Type	Description
group_id	int	The id of the group to remove the member from.
author_id	int	The id of the member to be removed.

URL: groups/members/remove

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing a message that the operation was successful.

Error: If the `group_id` or `author_id` parameter is not provided, the form request automatically rejects the request with a 400 error code. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

5.5.11 Get Groups Min Info

Purpose: This endpoint returns all the groups but only with minimal information like their ids and names.

Data Validation: No validation required.

Method: GET

Parameters: This endpoint takes no parameters.

URL: `groups/all/minimal-info`

Response:

Success: If the operation is successful a `Requests::success()` (see 5.1.1) response is returned containing following data:

Table 5.17: Get Groups Min Info Successful response.

Name	Type	Description
groups	array	An array containing names and ids of the groups retrieved.

Error: If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

5.5.12 Get OMEA Authors Stats

Purpose: This endpoint returns OMEA-related statistical data about the members of a group.

Data Validation: All validation is performed inside the controller action itself.

Method: GET

Parameters:

Table 5.18: Get OMEA Author Stats parameters.

Parameter	Type	Description
id	int	The group's id to fetch the statistics for its members.

URL: `groups/omea/stats/authors/{id}`

Response:

Success: If the operation is successful a `Requests::success()` (see 5.1.1) response is returned containing following data:

Table 5.19: Get OMEA Authors Stats Successful response.

Name	Type	Description
countsPerAuthor	array	An array containing the statistical data about each group member.

Error: If the id parameter is not provided, then a `Requests::missingParameterError()` (see 5.1.4) response is returned instead. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

5.5.13 Get OMEA Type Stats

Purpose: This endpoint returns OMEA-related statistical data about the types of works associated with a group.

Data Validation: All validation is performed inside the controller action itself.

Method: GET

Parameters:

Table 5.20: Get OMEA Type Stats parameters.

Parameter	Type	Description
id	int	The group's id to fetch the statistics for its members.
min	int	The earliest year to calculate statistics.
max	int	The latest year to calculate statistics.

URL: `groups/omea/stats/types/{id}/{min?}/{max?}`

Response:

Success: If the operation is successful a `Requests::success()` (see 5.1.1) response is returned containing following data:

Table 5.21: Get OMEA Type Stats Successful response.

Name	Type	Description
typeStatistics	array	An array containing the statistical data about the group's works.
minAllowedYear	int	The earliest year a group's work was published.
maxAllowedYear	int	The latest year a group's work was published.
requestedMin	int	The earliest year provided.
requestedMax	int	The latest year provided.

Error: If the id parameter is not provided, then a `Requests::missingParameterError()` (see 5.1.4) response is returned instead. If the group with the provided id is not found, a `Requests::clientError()` (see 5.1.2) response is returned instead. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

5.5.14 Get Works Metadata

Purpose: This endpoint returns metadata about all the works in the database, like earliest and latest publication years, minimum and maximum amount of citations, all the OMEA custom types and all the work types.

Data Validation: No validation required.

Method: GET

Parameters: This endpoint takes no parameters.

URL: works/metadata

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing following data:

Table 5.22: Get Works Metadata Successful response.

Name	Type	Description
minYear	int	The earliest year a group's work was published.
maxYear	int	The latest year a group's work was published.
minCitations	int	The lowest number of citations a work has.
maxCitations	int	The highest number of citations a work has.
customTypes	array	An array containing all the OMEA work types.
workTypes	array	An array containing all the distinct work types in the database.

Error: If an error occurs during the database transaction, a Requests::serverError() (see 5.1.3) response is returned instead.

5.5.14 Toggle Work Visibility

Purpose: This endpoint is used to toggle a work's visibility from an author's public profile.

Data Validation: All validation is performed inside the controller action itself.

Method: POST

Parameters:

Table 5.23: Toggle Work Visibility parameters.

Parameter	Type	Description
work_id	int	The id of the work, whose visibility is to be changed.
visibility	boolean	A boolean to indicate whether the work should be hidden or shown.

URL: works/visibility/toggle

Response:

Success: If the operation is successful a Requests:success() (see 5.1.1) response is returned containing a message that the operation was successful.

Error: If there is no logged in user, a Requests::authenticationError() (see 5.1.5) response is returned. If the user is not an author of the work to be edited, a Requests::authorizationError() (see 5.1.6) response is returned instead. If the work_id or visibility parameters are not provided, then a Requests::missingParameterError() (see 5.1.4) response is returned. If an error occurs during the database transaction, a Requests::serverError() (see 5.1.3) response is returned instead.

5.5.16 Get Hidden Works

Purpose: This endpoint returns all the hidden works associated with the logged in author.

Data Validation: All validation is performed inside the controller action itself.

Method: GET

Parameters: This endpoint takes no parameters.

URL: works/hidden

Response:

Success: If the operation is successful a `Requests::success()` (see 5.1.1) response is returned containing the following data:

Table 5.24: Get Hidden Works Successful response.

Name	Type	Description
works	PaginatedWorkCollection	An array with all the author's hidden works.

Error: If there is no logged in user, a `Requests::authenticationError()` (see 5.1.5) response is returned. If the user is not an author, a `Requests::authorizationError()` (see 5.1.6) response is returned instead. If an error occurs during the database transaction, a `Requests::serverError()` (see 5.1.3) response is returned instead.

Chapter 6 : Front-End Pages and Components

In this chapter, we'll take a close look at the part of the application that users see and interact with. This chapter will walk you through the different pages and components that make up the front-end, showing how each part is structured, and how it presents the data to the end user. I'll explain how these elements work together to create a smooth user experience and how they connect back to the back end.

6.1 Front-End Infrastructure

In the initial section of this chapter, we will explore the technologies and systems that structure the front-end of my application. We'll start by examining the custom hooks and the event system that state across the user interface. Next, we'll talk about the mechanisms of the API class, which is crucial for providing communication between the front-end and the back-end. This includes sending requests, handling parameters, and processing responses, including managing errors.

6.1.1 Hooks and State Management

Hooks are a powerful feature in React that allow developers to manage state and side effects in functional components. They encapsulate complex logic into reusable functions, making the components cleaner, and easier to extend, and maintain. By using hooks, I can easily manipulate the state of my application, and handle data updates efficiently. In this section, I'll be explaining the functionality of each hook used in my application and provide the benefits that come with their use.

useAPI

The useAPI hook is designed to provide a straightforward method for accessing API functionalities throughout the application. It initializes and returns an instance of the API class, which is used for all HTTP communications with the backend. By encapsulating the API instance creation within this hook, the application ensures that components can easily and consistently interact with backend services. This approach simplifies the process of making API calls by providing a reusable, centralized access point..

useAsync

The useAsync hook is used to manage asynchronous operations. It allows the application to handle these operations smoothly by tracking three key elements: the data being fetched, whether the operation is still loading, and if any errors have occurred.

Benefits of using the useAsync hook: The main benefit of this hook is that it provides the loading flag. This flag is particularly useful, allowing the user to know if the request is still pending by showing a loading indicator based on it. This enhances the user experience by keeping them informed about the operation's progress. Additionally, it simplifies the handling of errors by centralizing error management within the hook, making it easier to peacefully handle any errors when something goes wrong.

useAuth

The useAuth hook is one of the most important hooks in my application, as it encapsulates the authentication logic for the entire app. It starts by checking whether the user is authenticated when the application launches, and then performs multiple operations based on that response. Specifically, it manages several pieces of state, including the current user object, login status, whether the authentication status is still being checked, and whether the user has administrative rights.

Upon application launch, `useAuth` performs an initial check to determine if the user is logged in by making a call to the backend. If the user is authenticated, additional details such as admin status are fetched and stored. This is essential for the user interface to reflect the user's status, role, and permissions. The hook also integrates functions for login and logout. The login function updates the user state and other relevant authentication states upon successful login, while the logout function clears the user state and updates the application to reflect that the user has logged out.

The benefits of using the `useAuth` hook include centralized authentication management, reducing duplicated code, and minimizing the likelihood of authentication errors. It ensures that authentication logic and state are consistently maintained from a single source of truth, and it is easily accessible from any component that might need it.

usePagination

This hook is crucial for managing how pages of data are displayed in the user interface. It calculates which page numbers to show in the pagination menu based on the current page, the total number of items, and the desired page size.

Firstly, it calculates the total number of pages required to display all items, based on the total item count and the number of items per page. Next, it determines which page numbers to show, including the current page, nearby pages, and always the first and last page so users can quickly jump to them. The hook also decides when to show ellipses ('...') to keep the navigation bar neat while indicating there are more pages.

For special cases, if the total number of pages can fit in the navigation bar, it shows all the pages in order. If the current page is near the start, it displays pages from the beginning without a left ellipsis. If the current page is near the end, it shows pages up to the last one without a right ellipsis. When the current page is in the middle, both left and right ellipses are shown to indicate skipped pages.

The benefits of using the `usePagination` hook include enhanced navigation by managing visible page numbers, making it user-friendly especially for many pages. The hook recalculates pagination only when necessary, optimizing performance by avoiding unnecessary recalculations. Additionally, it can be customized to suit different use cases.

useRecentSearchQueries

This hook utilizes browser storage to remember the last few searches a user has made, allowing them to easily repeat common searches without retyping. It uses local storage to save recent search queries, ensuring they persist even after the page is refreshed. The hook implements a debouncing mechanism to delay storing the search query, waiting for a specified time (2000 milliseconds) after the user stops typing before saving the query. This delay reduces the frequency of updates, minimizing the performance impact from frequent writes to local storage.

Before saving, the hook checks if the current debounced search query is empty, too short (less than three characters), or already exists in the stored queries. If any of these conditions are met, it does nothing. If the query is new and meets the length requirement, it is added to the list of recent queries, keeping only the latest three. This prevents the storage from becoming cluttered with old or irrelevant queries. The hook returns the saved recent queries, allowing any component that uses it to access them.

The benefits of using the `useRecentSearchQueries` hook include allowing users to quickly access previous queries and save time. Debouncing optimizes performance and avoids unnecessary operations, which is crucial for maintaining minimal resource allocation. Additionally, this makes the recent queries logic reusable across different parts of the application where search functionality is implemented.

useWorkFilters

This hook is a sophisticated and crucial part of my application, designed to manage the complex task of filtering and sorting works. It offers a variety of filtering and sorting options, allowing users to target different types or specific pieces of work.

The hook starts with a default state that includes various filters set to initial predefined values. It manages state by handling a wide range of actions that update the state, from adding or removing filters like authors and sources to setting complex filters for publication years and citation counts. Each action is designed to modify specific pieces of the state related to user inputs, ensuring that the interface accurately responds to user actions. It also provides options for setting or resetting multiple filters at once.

The benefits of using the useWorkFilters hook include centralized management of all filter settings, making it easier to add new features. It enables users to filter works efficiently, helping them quickly find what they're looking for. The hook's flexible design allows it to be used in several parts of the application, saving time and effort by eliminating the need to write or maintain different logic for the same functionality.

6.1.2 Event System

In web development, events help websites react to what users do, like clicking a button or typing text. JavaScript not only handles such events but also lets developers create their own types of events. These custom events can be set up to do specific things that aren't covered by standard events. This is helpful especially on complex websites where parts of the page need to communicate with each other without being directly linked to one another. [33]

The event system in the application is primarily used for dynamic communication between different components of the application. By using custom events, components can interact and respond to actions happening elsewhere without directly being linked to the source of the action. This system allows for independent components yet responsive to global application states or changes.

The primary purpose of implementing an event system was to avoid the complexity and limitations of prop drilling across multiple components. By using events, the application can broadcast important changes or updates that any part of the application can listen to and react accordingly.

Functionality:

- **Event Dispatching:** The system can dispatch custom events that carry data related to specific actions. This data is then accessible to any event listener set up anywhere within the application.
- **Global Accessibility:** Events are dispatched globally and can be listened to by any component within the application. This means any part of the application can be designed to respond to these events without needing direct access to the state or source of the event.
- **Event Listening Hooks:** Custom hooks are exported, to enable components to subscribe to specific events and define callback functions that execute when those events occur. This hook-based approach integrates smoothly with React's functional component model.

Benefits:

- Components can operate independently and only interact with global events when necessary. This reduces dependencies among components.

- As the application grows, new components can easily integrate with existing events without modifying the existing component structure.
- Components become more dynamic and interactive as they can react in real-time to changes and updates occurring in other parts of the application, improving the overall user experience.

6.1.3 API System

In my application, the API class plays a crucial role in managing how the front-end communicates with the back end. This class is designed as a singleton, which means it's created only once and reused throughout the application, ensuring all parts of our app communicate with the back end in a consistent way.

The API calls are organized by functionality, grouping related operations together. For instance, operations related to group management are all handled by one component of the API class. Similarly, other components manage different aspects like works, authentication, and search. This organization not only streamlines the management of API calls but also makes the code more maintainable and efficient.

By centralizing API calls in this way, I keep my application organized. Each sub-component of the API class interacts with its corresponding part of the back end. This method simplifies handling errors and parsing data, as all this functionality follows a standard procedure defined in the API class.

Following this architecture, each specific API class extends from a base abstract API class. This abstract class provides common functionality, essential for all types of API calls, such as sending requests, parsing responses, and handling errors. By inheriting from this base class, each specific API component gains access to these shared methods, ensuring consistency for error handling and data parsing. This design pattern helps keep the API interactions consistent and reduces the amount of duplicate code, making the entire application more robust and easier to manage.

6.2 Application pages

In this section, we'll explore the pages of the application, focusing on the functionality and key features of each. We'll examine how each page serves a distinct purpose, highlighting its features, the services it provides and how each one interacts with one another. We will also focus on how they retrieve and display data, providing a comprehensive understanding of how the application's front-end is structured and how it was designed to meet the needs of its users.

6.2.1 Home page

The Home Page component serves as the main page of the application. It's the first page a user sees, it provides a comprehensive overview of the application, including authors and their works, along with a step-by-step guide to help users get started on the platform.

Purpose and Functionality

- **Introduction to Authors and Works:** The component features sections for both authors and works. The Authors section highlights renowned authors, allowing users to explore insights and statistical data about them. The Works section showcases an extensive catalog of works, providing detailed information and statistical analysis for each piece.

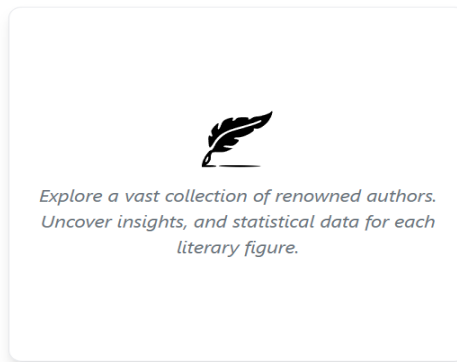


Figure 6.1: Authors overview card in the home page.



Figure 6.2: Works overview card in the home page.

- **Highlighting Top Contributors:** The component also includes sections that highlight top users by different metrics. The Top Users by Productivity section displays a list of authors who are most prolific in terms of the number of works they have published. The Top Users by Citations section showcases authors who have received the most citations for their works, indicating their influence in the academic community.

Top Users by Productivity (Works)			
1	Registered User	Citations: 3.841	Works: 267 Periklis Chatzimisios
2	Registered User	Citations: 2.608	Works: 182 Konstantinos I. Diamantaras
3	Registered User	Citations: 668	Works: 112 Christos Ilioudis
4	Registered User	Citations: 909	Works: 105 Charalampos Bratsas
5	Registered User	Citations: 888	Works: 94 Maria S. Papadopoulou

Figure 6.3: Top users by productivity list in the home page



Figure 6.4: Top users by citations received list in the home page.

- **Most Cited Works:** Displays the works that have been cited the most, indicating their impact and significance.

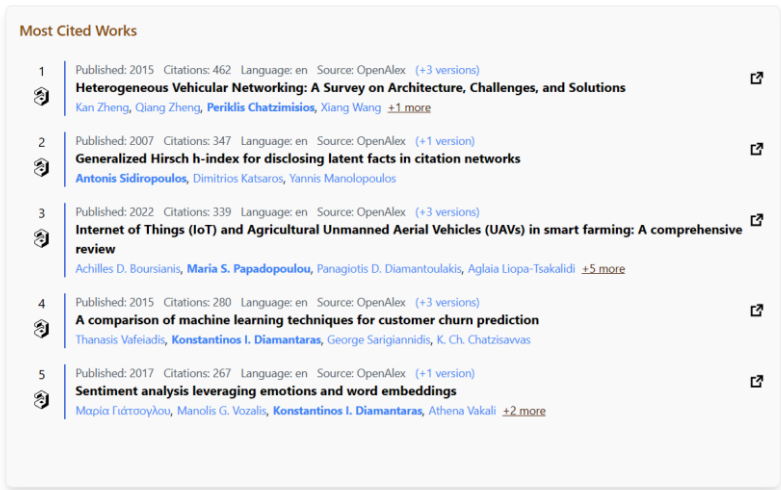


Figure 6.5: Top works by citations received list in the home page

- **Step-by-Step User Guide:**
 - **Step 1: Logging In:** The first step guides users to log in to the platform to start their journey.

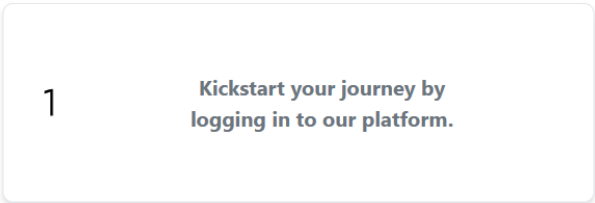


Figure 6.6: The first step of the sign-up guide.

- **Step 2: Providing Unique Identifiers:** In the second step, users are prompted to share their unique identifiers, such as ORCID, OpenAlex, or Scopus IDs, to help the platform recognize and fetch their works.

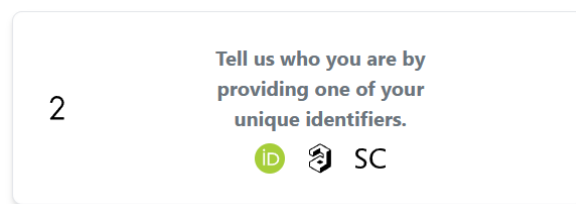


Figure 6.7: The second step of the sign-up guide

- **Step 3: Seameless Profile Creation:** The final step reassures users that the system will handle the rest by seamlessly fetching their works and building their profiles based on the provided information.

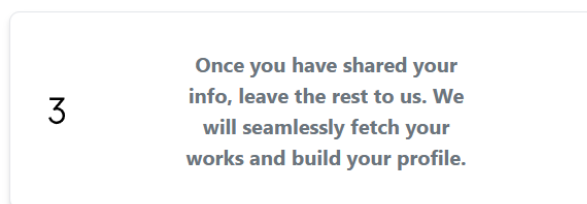


Figure 6.8: The final step of the sign up guide

- **Searching:** The home page features a search bar to help users explore the catalog of authors and their works. When users click on the search bar, a modal window appears, offering additional search options and their recently searched items.



Figure 6.9: The application's banner and search bar in the home page

Search Modal

Recently Searched Items: At the top of the modal, there is a section displaying recently searched items. This helps users quickly access their previous searches.

Search Options: A user has two main search options:

1. **Search for Authors:** Search for authors using their name, Scopus id, ORCID id, or OpenAlex id.
2. **Search for Works:** Search for works using titles, DOI (Digital Object Identifier), or OpenAlex ids.

Instructions: Below the search options, there are brief instructions on how to perform searches using external IDs or DOIs. For example, when using a DOI, users should only provide the part that comes after <https://doi.org>.

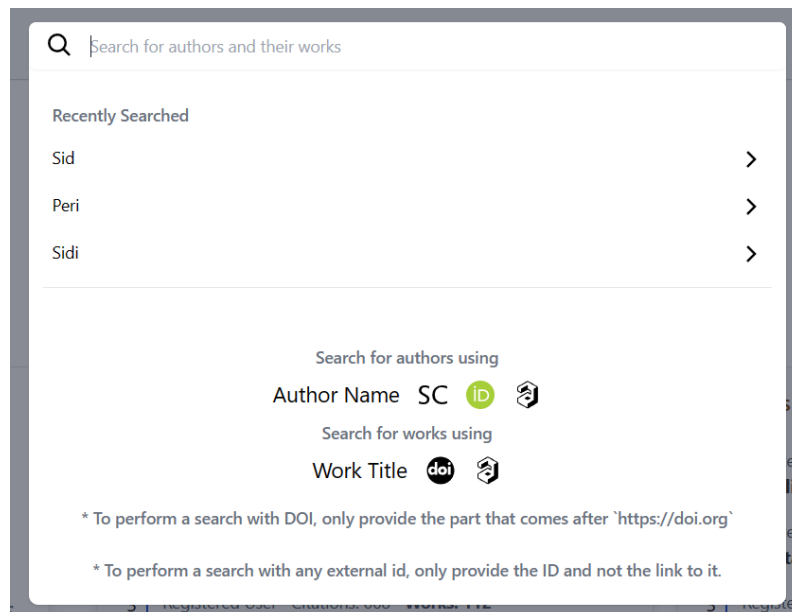


Figure 6.10: The search modal

- Search Results: After performing a search, the results are categorized and displayed to the user:
 - Authors: Lists authors that match the search criteria, showing their name along with identifiers from OpenAlex, Scopus, and ORCID.
 - Works: Lists works that match the search criteria, showing titles and associated icons.

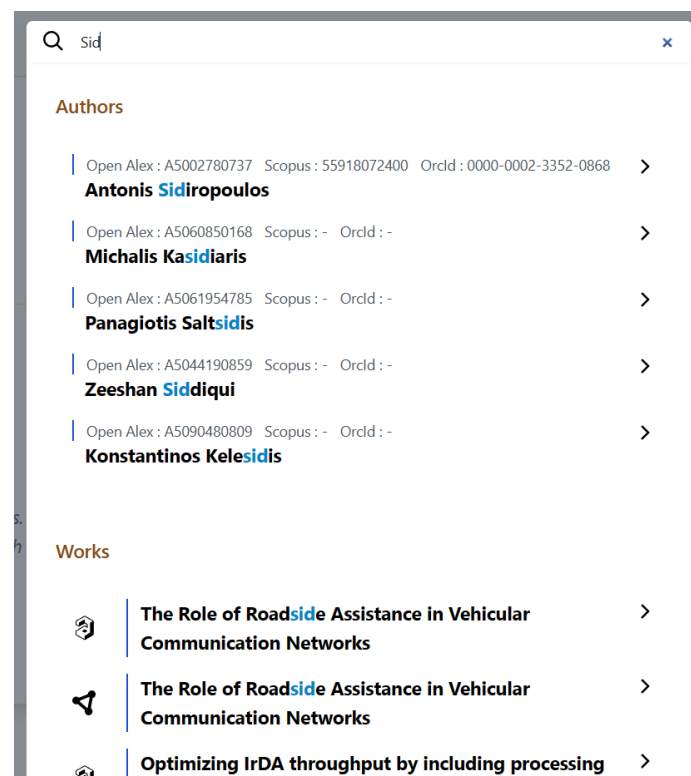


Figure 6.11: Search results.

Design and Layout

- **Cards and Lists:**

- **Cards:** Each major section (Authors and Works) is contained within a card that shows an icon and its description.
 - **Lists:** The lists of top contributors and most cited works in these sections, provide users with detailed, data that is easy to understand.
- **Icons:** The component uses different SVG icons (from ReactIcons) to show various things like authors, works, and identifiers. These icons help users quickly understand what each section is about
- **Styling and Responsiveness:** The component is styles using tailwind css and vanilla css when needed, for very specific parts that tailwind css cannot handle. The layout is designed to be responsive, adapting to different screen sizes and orientations.

6.2.2 Author page

The Author Page component is used to display detailed information about an author, including their works, biography and different kind of statistics about them. It features interactive elements like filters, charts, and expandable sections to make it visually appealing. Users can view and filter the author's works, see statistical charts on citations and publication statistical data, and read a detailed biography if available.

Purpose and Functionality

- **Author Information:**
 - **Profile Overview:** Displays the author's name, registration status, and the date of the last update. It also includes a message indicating if the author's profile is incomplete or regularly updated (based on their registration status).
 - **Biography:** If available, the biography is displayed with an option to expand and read the full text. If not, a message indicates that the biography is unavailable.

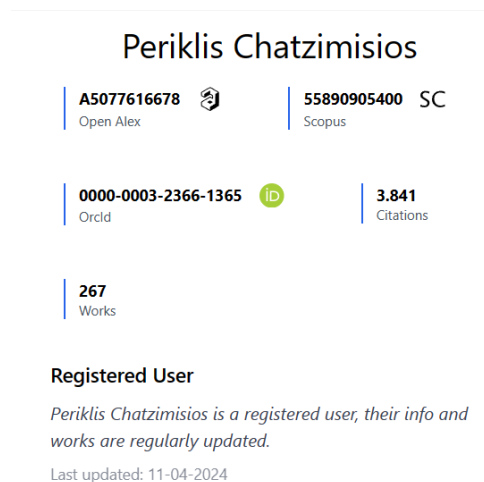


Figure 6.12: Author profile overview.

- **Works and Filters:**

- **List of Works:** Displays a list of the author's works, which can be filtered and paginated. Users can see the author's works and have the option to show hidden works if they are logged in as the author.
- **Filters:** Various filters allow users to narrow down the list of works, and sort them based on different criteria.

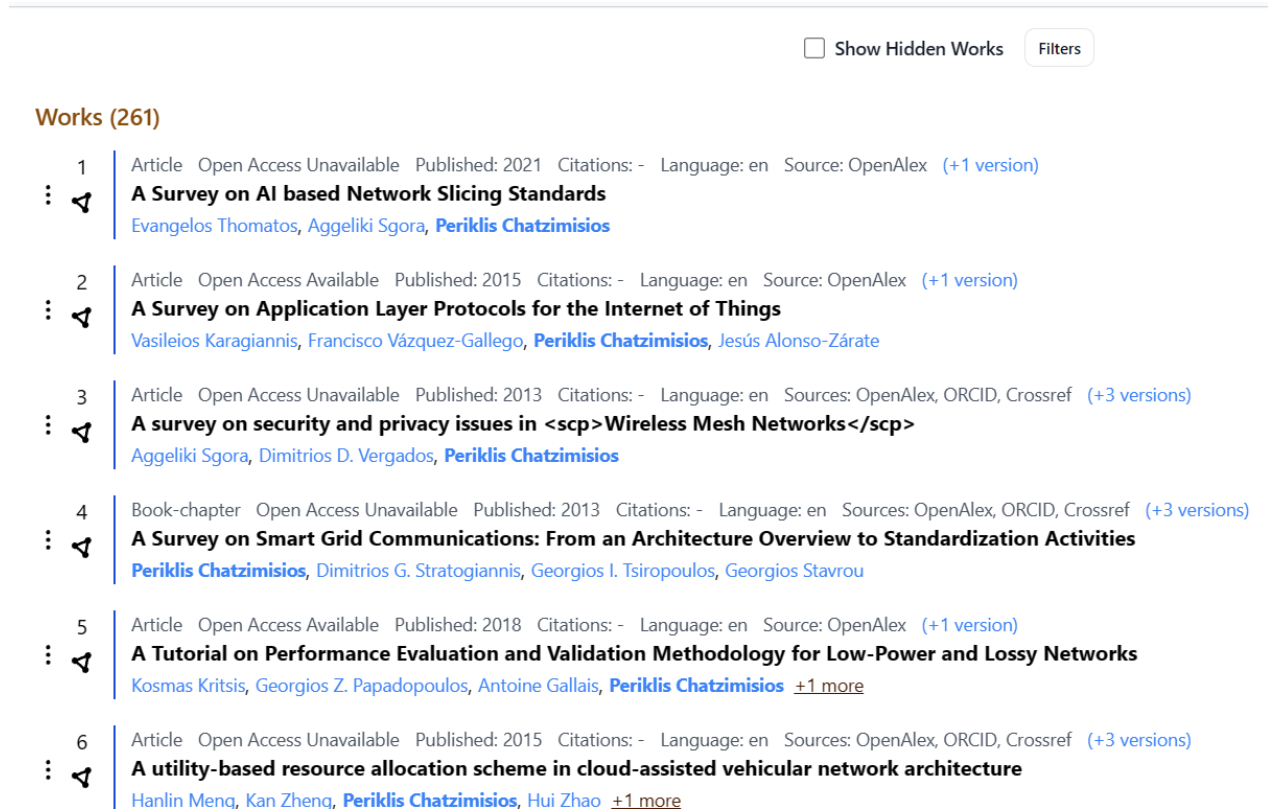
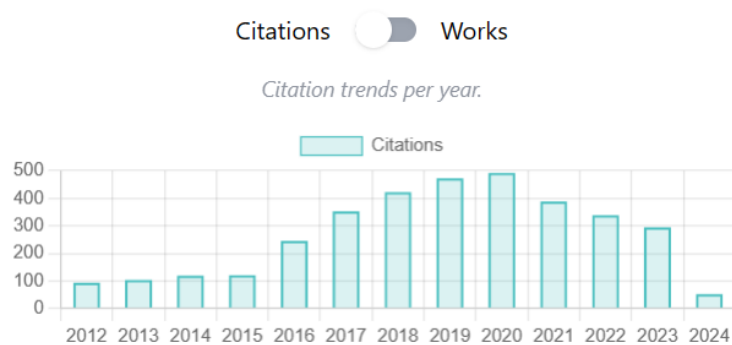


Figure 6.13: Author's works list overview.

- **Statistics and Charts:**

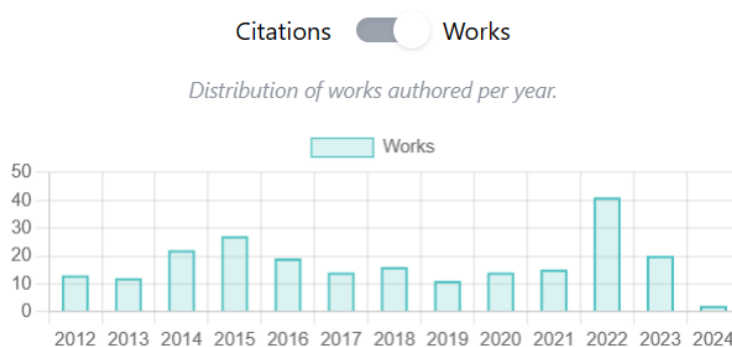
- **Citations Bar Chart:** Displays the citations received by the author per year. (This information is only retrieved by Open Alex).
- **Works Bar Chart:** Displays the number of works authored by the author per year. (This information is only retrieved by Open Alex).
- **Work Sources Doughnut Chart:** Shows the distribution of the author's works sourced from different platforms like OpenAlex, ORCID, and Crossref.



Source : Open Alex

The data presented in this chart may not capture the complete set of citations for this author. The statistics gathered might not cover every year, potentially leading to gaps in the information.

Figure 6.14: Citations bar chart.



Source : Open Alex

The data presented in this chart may not capture the complete set of works for this author. The statistics gathered might not cover every year, potentially leading to gaps in the information.

Figure 6.15: Works bar chart.

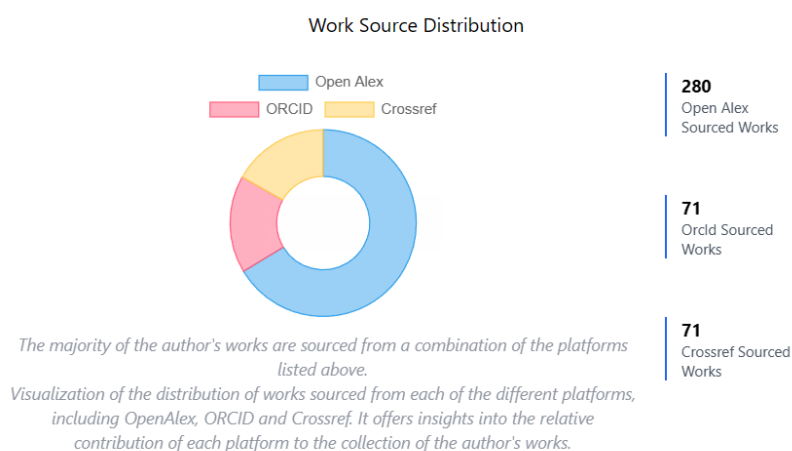


Figure 6.16: Work sources doughnut chart

- **Interactivity:**
 - **Chart Switching:** Allows users to toggle between the 2 bar charts.
 - **Pagination:** Users can navigate through multiple pages of the author's works.
 - **Hidden Works:** Logged-in authors who are viewing their own profile can choose to show or hide works from their profile.
 - **Charts:** Each chart is presented with a title, description and any disclaimers that need to be shown to the user. They are all built using the Chart.js library.
 - **Lists:** The works are displayed in a list that supports pagination, making it easy to navigate through large numbers of entries.

6.2.3 Work page

The WorkPage component provides a detailed view of a specific work, including its title, authors, abstract (if present), various statistics, and versions from other sources.

Purpose and Functionality

- Title and Authors:
 - **Title:** The title of the work is prominently displayed at the top of the page.
 - **Authors:** The names of the authors are listed below the title. Each author's name is clickable, linking to their profile page. Registered users have their names displayed in bold.

A Survey on Smart Grid Communications: From an Architecture Overview to Standardization Activities

Periklis Chatzimisios, Dimitrios G. Stratogiannis, Georgios I. Tsiropoulos, Georgios Stavrou

Figure 6.17: Work page title and authors overview.

- Properties and Abstract:
 - **Properties:** A section that lists key properties of the work, such as publication year, type, and access information.
 - **Abstract:** If the work has an abstract, it is shown with an option to expand or collapse if it is too long. Users can click on the abstract to read the full text if it is truncated.

<https://doi.org/10.1016/b978-0-12-415844-3.00026-7>
Doi

Book-chapter
Type

2013
Published

Unavailable
Open Access

4
Versions

OpenAlex, ORCID, Crossref
Source

Figure 6.18: Work page properties section overview.

1. Statistics and Charts:

- **References Chart:** Displays a bar chart showing the number of references the work has received over the years. This chart includes a description, source information, and a disclaimer about the completeness of the data.
- **References Metrics:** If reference metrics are unavailable, a message is displayed instead of the chart.

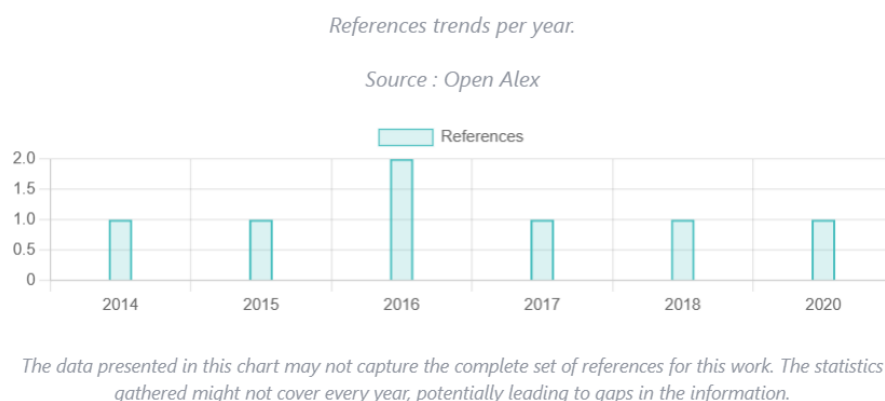


Figure 6.19: References metrics section overview.

References Metrics are not available for this work!

Figure 6.20: References metrics unavailable message overview.

2. Versions of the Work:

- **Other Versions:** If there are multiple versions of the work, they are listed at the bottom of the page. Each version is displayed with its title, type, publication year, and other details.

3 more versions of this work

1	Book-chapter	Open Access Unavailable	Published: 2013	Citations: 7	Language: en	Source: OpenAlex	
	A Survey on Smart Grid Communications: From an Architecture Overview to Standardization Activities						
	Periklis Chatzimisios , Dimitrios G. Stratogiannis , Georgios I. Tsiropoulos , Georgios Stavrou						
2	Book	Open Access Available	Published: 2013	Citations: -	Language: -	Source: ORCID	
	A survey on smart grid communications: From an architecture overview to standardization activities						
	Periklis Chatzimisios , Dimitrios G. Stratogiannis , Georgios I. Tsiropoulos , Georgios Stavrou						
3	Book-chapter	Open Access Unavailable	Published: -	Citations: 8	Language: -	Source: Crossref	
	A Survey on Smart Grid Communications: From an Architecture Overview to Standardization Activities						
	Periklis Chatzimisios , Dimitrios G. Stratogiannis , Georgios I. Tsiropoulos , Georgios Stavrou						

Figure 6.21: Work versions section overview.

Design and Layout

- **Title and Author List:** The title is located at the top of the page so it is easy to locate, with the authors listed below it. The authors' names (if information is available for them in the database), can be clicked to navigate to that author's profile .
- **Abstract Section:** If there is an abstract available for this work, and specifically for this version (at the moment, we can only retrieve abstracts from ORCID), it is presented in a collapsible section since it is usually really long and can take a lot of space, in which case only the first part is shown, with an option to expand.
- **Statistics and Charts:**
 - The references chart provides visual data on the work's references over the years, with labels and descriptions.
 - If no reference data is available, a message informs the user that the metrics are not available.
- **Versions:** Additional versions of the work are listed with relevant details, helping users see the different versions of the work from different sources.

6.2.4 Groups page

The Groups Page component allows users to view and manage groups, including their details and members. It provides functionality for administrators to create new groups and for all users to view group information.

Purpose and Functionality

1. Group Selection and Details:

- **Groups List:** On the left side of the page, a list of groups is displayed. Users can select a group from this list to view more details. A tree view is used from UIMaterial to display the hierarchy of the groups, since groups support having sub-groups for a more detailed hierarchy.
- **Group Details:** Once a group is selected, its details, including members and activities, are displayed on the right side of the page.

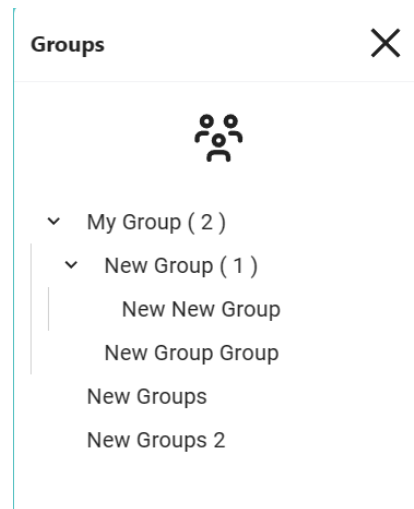


Figure 6.22: Groups page groups tree view overview.

2. Group Management:

- **Create New Groups:** Administrators have the option to create new groups using a modal window. This functionality is restricted to users with administrative privileges.
- **Update and Delete Groups:** The component listens for events related to group updates and deletions. When a group is updated or deleted, the component responds by refreshing the list of groups and displaying appropriate notifications.

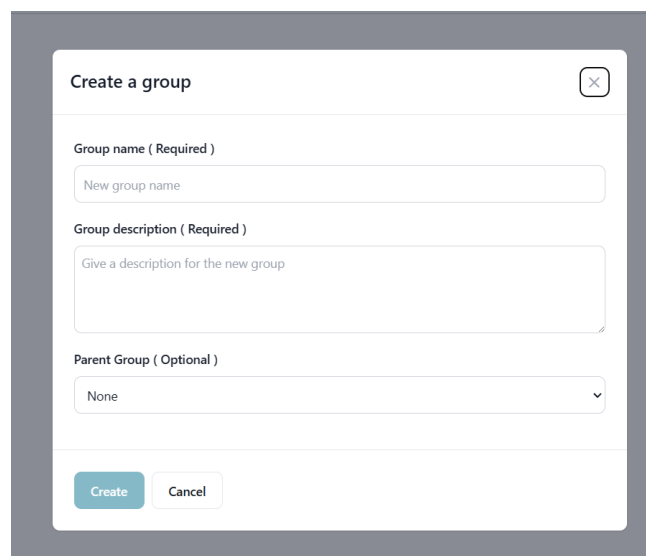


Figure 6.23: New group creation modal.

3. Interactive Features:

- **Event Listeners:** The component uses event listeners to handle updates, creations, and deletions of groups. These listeners ensure that the component remains up-to-date with any changes made to the groups.
- **Notifications:** Users receive notifications for actions like group creation, updates, and deletions using a toast message, providing immediate feedback on their actions.

Key Elements

- **GroupsList Component:** Displays the list of available groups. Users can click on a group to select it and view its details.
- **NewGroupModal Component:** A modal window that allows administrators to create new groups.
- **ActiveGroup Component:** Shows detailed information about the selected group, including its members, all the works associated with them and OMEA[34] stats.

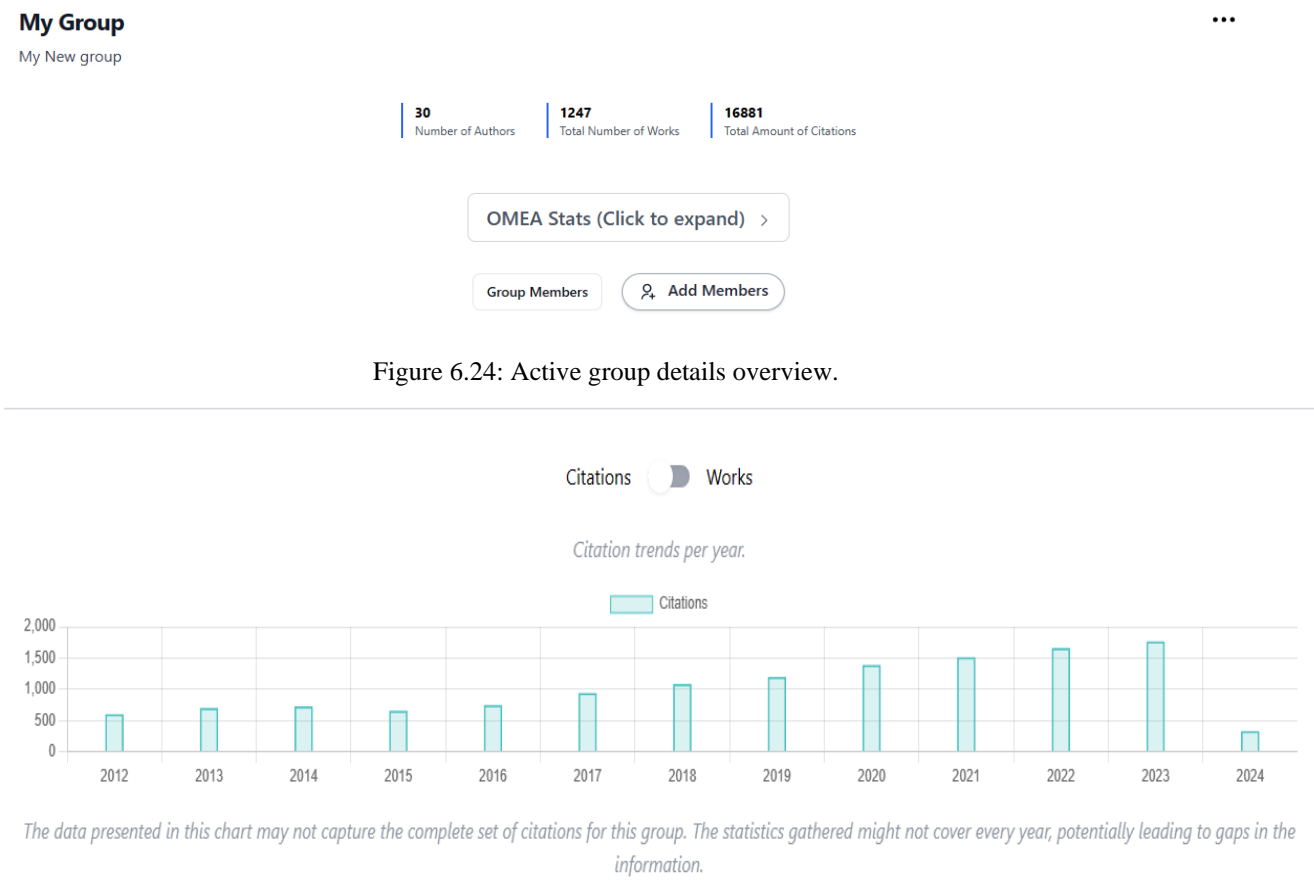


Figure 6.25: Active group statistic bar charts.

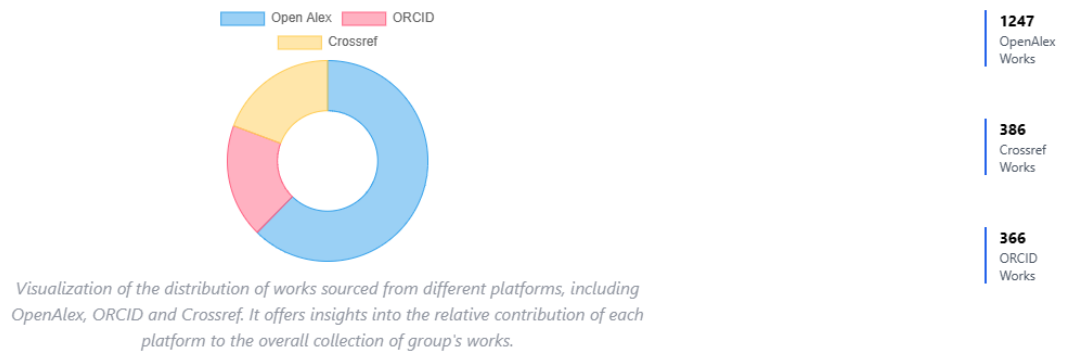


Figure 6.26: Active group work sources doughnut chart.

Filters	
Group Works (1247) (1247)	
1	Paratext Open Access Available Published: 2018 Citations: - Language: en Source: OpenAlex (+1 version) Title not available Christodoulos Asiminidis, George Kokkonis , Sotirios Kontogiannis
2	Paratext Open Access Available Published: 2018 Citations: - Language: en Source: OpenAlex (+1 version) Title not available Christodoulos Asiminidis, George Kokkonis , Sotirios Kontogiannis
3	Paratext Open Access Available Published: 2018 Citations: - Language: en Source: OpenAlex (+1 version) Title not available Christodoulos Asiminidis, George Kokkonis , Sotirios Kontogiannis
4	Paratext Open Access Available Published: 2018 Citations: - Language: en Source: OpenAlex (+1 version) Title not available Christodoulos Asiminidis, George Kokkonis , Sotirios Kontogiannis
5	Article Open Access Unavailable Published: 1999 Citations: - Language: en Source: OpenAlex (+1 version) Title not available Athanassios C. Iossifides , Fotini-Niovi Pavlidou

Figure 6.27: Active group works list.

Works per Author		Works per OMEA Type		
AUTHOR	OPENALEX	ORCID	CROSSREF	
Periklis Chatzimisios	280	71	71	
Dimitrios Amanatiadis	2	-	-	
Zafiris D. Ampatzis	1	-	-	
Efstathios N. Antoniou	45	-	-	
Katerina Asdre	9	-	-	

Figure 6.28: Active group OMEA authors stats.

Works per Author								Works per OMEA Type								
Showing data for the last 15 years.																
TYPE	'9	'10	'11	'12	'13	'14	'15	'16	'17	'18	'19	'20	'21	'22	'23	'24
Papers in conference proceedings with reviewers	24	32	35	44	40	48	61	59	42	52	67	60	76	90	74	8
Papers in conference proceedings without reviewers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Books/Monographs	9	6	3	11	6	10	4	7	4	7	9	7	9	15	9	0

Figure 6.29: Active group OMEA works stats.

- New Group Landing Page:** When a new group is created, it prompts the administrator to set it up by adding members to it. The administrator can then search for registered users to add to this group. Groups that have no members are not visible for non-administrator users, since there is no useful information to be displayed for them.

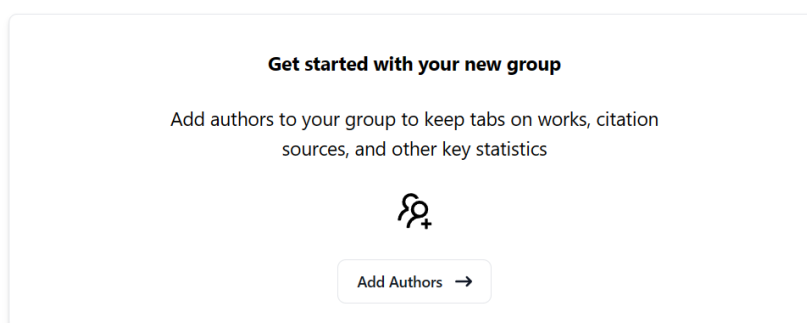


Figure 6.30: New group landing page overview.

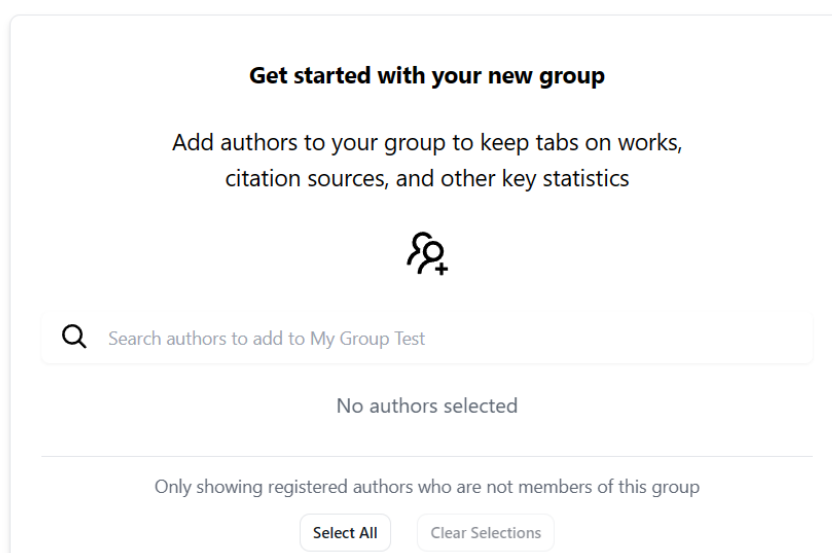


Figure 6.31: Search component for adding new group members.

6.2.5 Successful login page

The SuccessfulLogin component is displayed after a user has been successfully authenticated. It informs the user about their successful login and redirects them to the homepage after a brief delay.

Purpose and Functionality

1. **Authentication Confirmation:** A message is displayed confirming that the user has been successfully authenticated, also informing them that they will be redirected back to the homepage in 5 seconds.
2. **Automatic Redirection:** A timeout is used to redirect the user to the homepage after 5 seconds, with this redirection handled by calling an API method to navigate to the homepage.
3. **User Login Handling:** Logs in the user by saving the user's data to be accessible by all the other pages.


6.2.6 VerifyAuthorIdentifiers

This sub-page is displayed for staff members who have successfully logged in but need to provide their unique identifiers to complete their profile. It prompts users to enter one of several identifiers and verifies them.

Welcome to MyPubsV2


To ensure seamless access to our platform's resources and optimize your experience, it is essential that you provide one of the following unique identifiers to build your profile.

OpenAlex

 OpenAlex


e.g A0123456789

Scopus

 Scopus

e.g 012345678901

ORCID

 ORCID

e.g 0123-0123-0123-0123

Submit

Figure 6.32: Verify Author Identifiers sub-page overview.

Purpose and Functionality

1. **Identifier Input:** Input fields are provided for three types of identifiers: OpenAlex, Scopus, and ORCID. Users can only enter one identifier at a time, as entering an identifier in one field disables the other fields.

2. **Form Submission:** When the user submits the form, the component calls an API to search for authors based on the provided identifiers. If no authors are found, an error message is displayed. If authors are found, the results are displayed in a list.
3. **Results Display:** The component shows a list of authors that match the provided identifiers, and users can go back to the input form to enter different identifiers if needed.

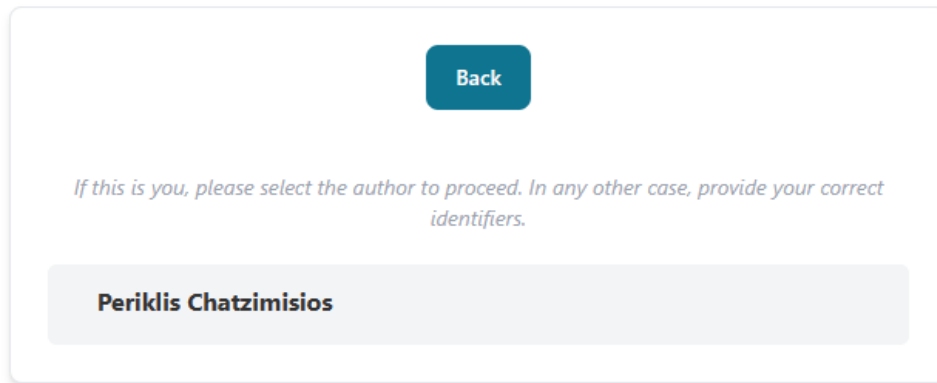


Figure 6.33: Authors result list overview.

After clicking on a search result, the user is shown a modal displaying detailed information about the identified author, including their name and various identifiers (such as ORCID, OpenAlex, and Scopus IDs). This is to ensure the user can verify they have selected the correct author profile to connect with. The modal prompts the user to acknowledge their rightful ownership of the profile and warns that this action is irreversible. The user can either accept to claim the profile or decline to recheck their details.

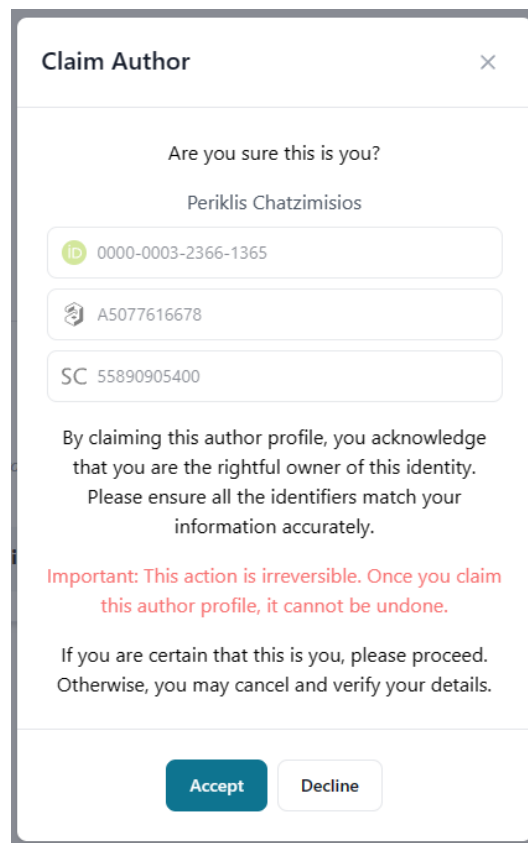


Figure 6.34: Author connection verification modal.

Access Restriction

Staff members who need to provide their identifiers will be redirected to the `VerifyAuthorIdentifiers` sub-page upon successful login. They will not be allowed to access any other page until they have provided their identifier. This ensures that all necessary information is collected to build their profile before they can fully use the platform.

6.3 Shared core components

In this chapter, I will present and explain some core components of my application that are used by multiple other components and pages. These components are primarily, but not exclusively, utility components or components that are used to present asset data (e.g., `Author`, `Work`) to the user. They are crucial for my application's functionality because they help streamline functionality and ensure consistency across the application, making it easier to manage and maintain.

I will focus on the components I created myself or those I thoroughly customized, such as the `Off-Canvas` component, where I developed a wrapper component to provide consistency and additional functionality. However, I won't be elaborating on components taken from external libraries, such as the `toast-message` and the `modal`, as they are used as-is without significant modification.

6.3.1 Navigation:

The `Navigation` component is a component that provides the main navigation menu for the application. This component ensures that users can easily access different sections of the app and find the information they need.

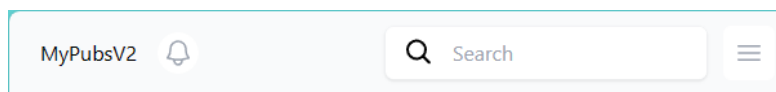


Figure 6.35: Navigation component preview.

Key Features

1. **Logo:** Displays the application's logo, which links to the homepage. This helps users quickly return to the main page from any other page.
2. **Menu Items:** Includes links to different sections such as `Groups`. These links are dynamically shown or hidden based on the user's login status and permissions. For example, the `Groups` link is only visible to logged-in users.
3. **Search Functionality:** Includes a search bar that allows users to search for specific authors or works directly from the navigation menu, making it easier to find relevant information quickly.
4. **User Authentication:** Displays different options based on whether the user is logged in. Logged-in users can see their display name and access options like `"My Profile"` and `"Logout"`. If the user is not logged in, a `"Login"` link is shown instead.

Usage

The navigation bar is present in every page of the application.

6.3.2 Off-Canvas:

The Off-Canvas component is a versatile and reusable component used to display a sidebar or drawer that slides in from different positions on the screen. This component enhances the user interface by providing additional space for menus, filters, or other important content without disrupting the main view.

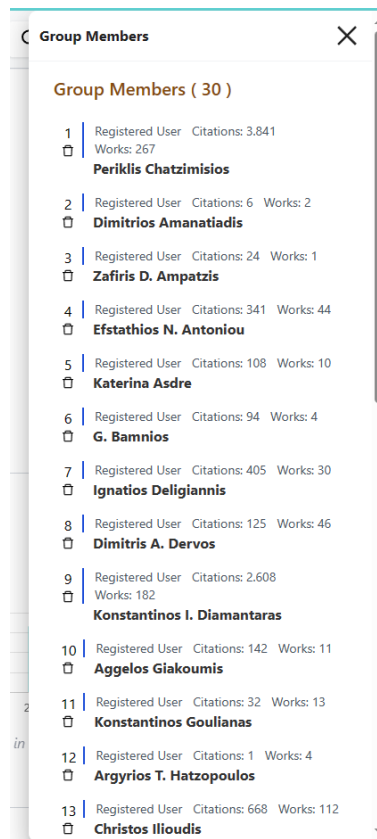


Figure 6.36: Off-canvas component preview.

Key Features

1. **Flexible Positioning:** The Off-Canvas component can slide in from the top, bottom, left, or right side of the screen. This allows it to be used in various scenarios, depending on the needs of the specific use-case.
2. **Header and Content:** The component includes an optional header area for displaying a title or other important information. Below the header, it can display any content passed as children, making it highly customizable.
3. **Click Away to Close:** An optional feature allows the drawer to close when clicking outside of it, enhancing user convenience, and improving interaction.

Usage

- **Groups Page:** One instance of it is used to display the groups tree view, where a user can select one to view more details, and another instance is used to show a list of the group's members.

6.3.3 Author Item

The Author Item component is a reusable component designed to display detailed information about an author. This component is used across various parts of the application to ensure a consistent presentation of author data.

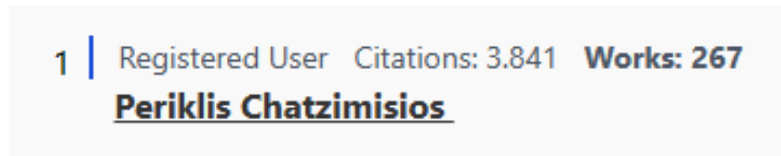


Figure 6.37: Author Item component preview.

Key Features

1. **Display Author Information:** Displays key details about an author, such as their name, citation count, works count, and user status (registered or guest).
2. **Highlighting:** It includes options to highlight certain properties, such as citation count and works count, to draw attention to important metrics.
3. **Indexing:** The component can display an index number next to each author, which is useful for when used in a list. This index can be hidden if not needed.
4. **Extra Properties:** Additional properties can be passed to the component and displayed, allowing for flexibility in the information presented.
5. **URL Handling:** The author's name can be displayed as a clickable link that navigates to the author's detailed page. This can be disabled depending on the use-case.
6. **Hide Properties:** Provides an easy way to hide any of the properties when not needed.

Usage

- **Home page:** Used to display lists of renowned authors with their details.
- **Search results:** Used to display the author results after a search query.
- **Groups page:** Used to display the members of a group.

6.3.4 Work Item

The Work Item component is a reusable component designed to display detailed information about a specific work. This component ensures that works are consistently presented throughout the application.



Figure 6.38: Work Item component preview.

Key Features

1. **Display Work Information:** Displays key details about a work, such as its title, authors, type, open access status, publication year, citation count, language, and sources.
2. **Author List:** It displays a list of authors associated with the work. If the author list is long, it provides an option to show more or fewer authors, ensuring a clean and manageable display.
3. **Source Icon:** The component displays an icon indicating the source of the work's information (e.g., OpenAlex, ORCID, Crossref), helping users quickly identify where the data comes from.
4. **Multiple Versions:** If a work has multiple versions, the component provides a way to view all versions in a modal. This allows users to see the complete history of the work.

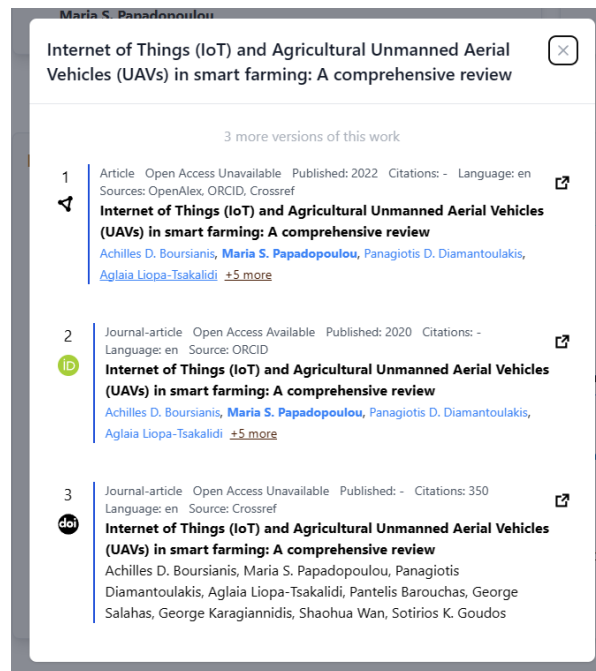


Figure 6.39: Work Item versions modal preview.

5. **Indexing:** The component can display an index number next to each work, which is useful for ordered lists. This index can be hidden if not needed.
6. **URL Handling:** The title of the work can be a clickable link that navigates to a detailed page about the work. Additionally, an external link icon is provided to navigate to the source of the work, such as its DOI page.
7. **User Options:** For editable works, user options such as hiding or showing the work in the profile can be accessed through a dropdown menu.
8. **Conditional Display:** Any property of the work, like type, open access status, publication date, citation count, language, and versions, can be conditionally displayed or hidden based on the specific use-case.

Usage

- **Home page:** Used to highlight the most cited works in the application.
- **Search results:** Used to display the work results after a search query.
- **Author page:** Used to display the works of an author in their profile.

6.3.4 Search

The Search component is a versatile and reusable component designed to search for authors and works, within the application. This component provides an easy and efficient way for users to find specific information.

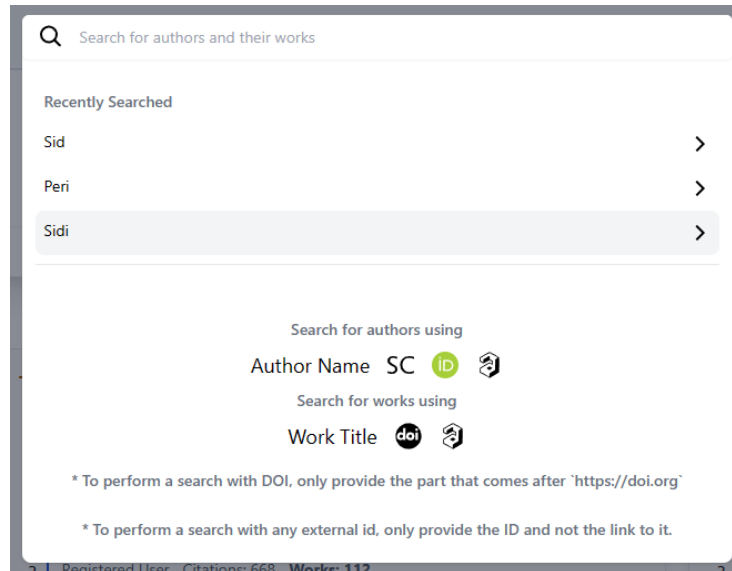


Figure 6.40: Search component modal preview.

Key Features

1. **Search Functionality:** Allows users to search for authors and works. It can be customized to search for only specific types of data, such as only authors or only works.
2. **Search Results Modal:** When a search is initiated, a modal window opens to display the search results. This modal includes:
 - A search input field to provide the search query.
 - Search tips and recent searches to assist users.
 - A list of search results categorized into authors and works.
3. **Interactive Input:** The search input field is interactive and provides real-time validation as the user types. It indicates if the query is too short and prompts the user to enter more characters.
4. **Recent Searches:** Shows recent searches, helping users quickly access their previous queries.

Usage

- **Home page:** The Search component on the home page allows users to search for authors and their works right away once they launch the application.
- **Navigation bar:** When the user navigates to any other page except the Home page, the search bar moves to the navigation, providing easy access to the search functionality no matter in which page the user is at any time.

- **Groups page:** When a new group is created, the search bar is used to search for new authors to add to the group.

6.3.5 Paginated List

The Paginated List component is a reusable component designed to display a list of items with pagination. This component is essential for handling large datasets by dividing them into manageable pages, enhancing the user experience and providing easy navigation through long lists.

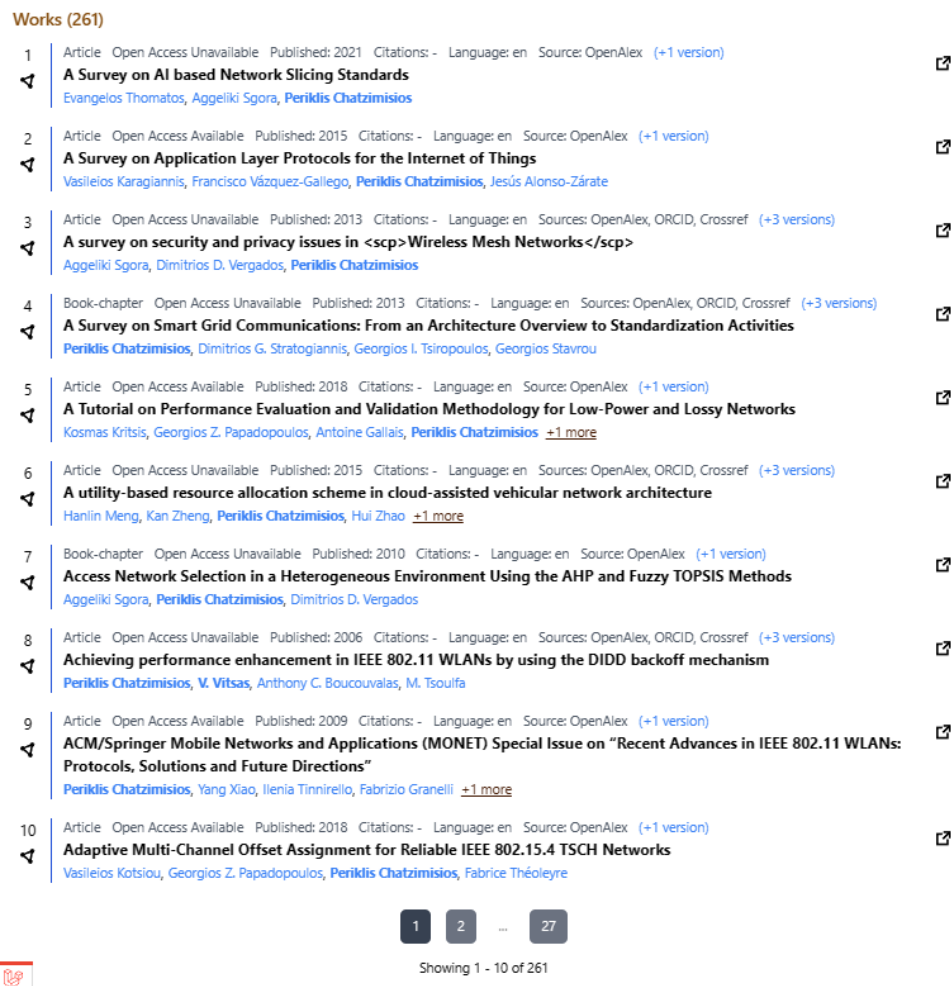


Figure 6.41: Paginated list component preview.

Key Features

1. **Pagination:** Handles paginated data, allowing users to navigate through different pages of items. This is useful for displaying long lists without overwhelming the user.
2. **Optional Parsing:** If the data needs to be processed or transformed before rendering, a parser function can be provided. This function processes each item in the response data.
3. **Collapsible List:** The list can be made collapsible, meaning it can be expanded or collapsed to show or hide the content. This feature helps in saving space and keeping the interface clean.
4. **Custom Headers and Titles:** Optional headers and titles can be added to the list, providing additional context or information about the items being displayed.

5. **Loading State:** The component handles loading states by displaying skeletons (placeholders) while the data is being fetched. This improves the user experience by providing visual feedback while the data is loading.



Figure 6.42: Paginated list loading state preview.

Usage

- **Author page:** Since in most cases the list of an author's works is long, a paginated list is used to display them and provide easy navigation through a potentially long list.
- **Groups page:** Same with groups, the list of a group's works can be quite long, so a paginated list is again used to display these works.

6.4 Conclusion

In this chapter, we have thoroughly explored the front-end aspects of my application, including the infrastructure, key application pages, and shared core components. By delving into the specifics of each page and component, we have highlighted how they interact and contribute to the overall functionality and user experience. The careful selection and customization of these components ensures a consistent and user-friendly interface. This overview provides a clear understanding of how the front-end elements work together to support the application, setting the stage for the final discussion on general conclusions and potential future improvements and expansions.

Chapter 7 : Conclusion and Future Expansion

This thesis has presented the development and implementation of an automated system for organizing research publications for my university department. The system's primary objective was to streamline the collection, organization, and statistical analysis of research publications within the academic community of the Department of Informatics at our university. Using modern web technologies, a robust database structure, and integrating with various research databases (OpenAlex, ORCID, Scopus, and Crossref), the project has successfully delivered a functional and scalable application.

The system has shown great potential in making research publications easier to access and manage. By gathering all publications in one place and providing a user-friendly web interface for entering and retrieving information, the system helps researchers work more efficiently and supports the university's goal of advancing knowledge and encouraging collaboration among researchers.

Challenges Faced

Developing this application was quite challenging. To begin with, the size of the project posed a challenge. It was a big project that required extensive planning and organization to manage all the components and automated processes effectively.

Structuring the database was another big task. Defining the relationships between all the different models, such as authors, works, and groups, was complex and required a lot of planning and precision. For some works, detailed lists of authors were not available, but only names without unique identifiers. This made it impossible to accurately associate them with all their authors. To address this, I opted to "associate" these works only with the author from whom the work originated and included the rest of the authors as a string. Ensuring the information shown to the end users, like statistics and author profiles, was accurate also posed some challenges.

Merging data from different sources into one unified system was particularly tough. To maintain clarity and accuracy, I created a new "version" of each work for each data source, plus an additional "aggregated" version that combined information from all sources, excluding stats. For example, fetching a work's info from ORCID, OpenAlex, and Crossref resulted in four versions of the work: one for each source and an aggregated version. By keeping information from each source in separate work assets in the database, it was easier to extract source-specific statistics and ensure more accurate data management and presentation.

Integrating with various APIs was also challenging. Each API had its own limitations, and making sure my application could communicate smoothly with these external databases took a lot of effort and careful implementation. Many API responses were inconsistent, with specific properties not always present. This inconsistency made development much harder because I had to triple-check and thoroughly test each API, always checking for the existence of specific properties before using them. This process was made even more challenging during the automated seeding, as it wasn't actively observed by a human. The only way to know if it worked correctly was after it was completed or if it failed.

Managing user authentication and authorization was another significant challenge. Integrating with the university's OAuth provider itself wasn't too hard but debugging an error after obtaining the user's data and trying to log them into the application itself proved to be very time consuming. The issue was a dump that was printed before the headers in the response request during the redirection back to my

application. This caused the session not to be preserved since the set-cookie header wasn't sent (headers need to be the first thing printed in a response request).

Finally, working with the automated seeding and update jobs presented its own difficulties. Making any significant database changes resulted in almost an hour of downtime due to the length of the seeding process. This was because of the large number of requests needed to fetch works and authors from the external databases. Ensuring the system could handle these tasks without prolonged downtime was a significant concern.

Despite these challenges, I managed to overcome them through persistent problem-solving, through testing, and continuous improvements. Each challenge taught me valuable lessons and helped make the application more robust and efficient. Through this process, I have learned a lot about optimizing performance and handling similar errors in the future. This experience has equipped me with the skills and knowledge to tackle future challenges more effectively.

Future Expansion Plans

Looking ahead, there are several directions for expanding this project:

1. **Real-Time Profile Fetching:** In a future version, I aim to support real-time fetching of an author's profile. Currently, the system relies on an initial dataset to build profiles, but future updates will allow users to fetch and update profile information on-demand.
2. **Manual Fetching of Works:** I plan to implement functionality that enables users to manually fetch works that might have been missed during the initial profile building process. This will ensure that the database remains comprehensive, accurate and up to date.
3. **Extended University Support:** Initially designed for use within my university department, I intend to expand the application to support other universities and research institutions. This will broaden the scope and impact of the application, allowing for greater collaboration and resource sharing across university departments.
4. **Better Data Visualization:** Incorporating more advanced data visualization tools to provide deeper insights into publication trends and author metrics. This could include more interactive charts, and graphs.
5. **User Experience Improvements:** Continuously improving the user interface to make the application interface more appealing and user-friendly. This includes optimizing the mobile experience and adding more customization options for users.
6. **Broader Database Integration:** Expanding the range of integrated databases to include other significant research repositories. This would provide a more comprehensive view of research activities and enhance the application's utility.
7. **Extraction of More Complicated Statistics:** Extracting and analyzing more complex statistics that will help the university academically. This could provide deeper insights into research impact, collaboration networks, and emerging trends.

In conclusion, this project marks a significant step forward in the effective management of research publications. It lays a solid foundation for further enhancements and sets the stage for continued innovation in supporting academic research and collaboration. The challenges encountered during the project have provided invaluable learning experiences, and the future expansion plans promise to bring even greater value to the academic community.

Conclusion

In conclusion, this project is a big step forward in managing research publications better. It creates a strong base for making the system even better and supports new ideas in academic research and teamwork. The challenges faced during the project taught me a lot. For example, now I know how to handle similar errors and improve performance. Expanding the system will make it even more useful for the academic community. This project has not only provided practical solutions but has also prepared me to tackle future problems more effectively.

References

- [1] “MariaDB Foundation - MariaDB.org.” Accessed: May 15, 2024. [Online]. Available: <https://mariadb.org/>
- [2] “MySQL.” Accessed: May 15, 2024. [Online]. Available: <https://www.mysql.com/>
- [3] “Laravel - The PHP Framework For Web Artisans.” Accessed: May 15, 2024. [Online]. Available: <https://laravel.com/>
- [4] “PHP: Hypertext Preprocessor.” Accessed: May 15, 2024. [Online]. Available: <https://www.php.net/>
- [5] “Laravel Socialite.” Accessed: May 15, 2024. [Online]. Available: <https://laravel.com/docs/11.x/socialite>
- [6] “Eloquent: Getting Started.” Accessed: May 15, 2024. [Online]. Available: <https://laravel.com/docs/11.x/eloquent>
- [7] “React.” Accessed: May 15, 2024. [Online]. Available: <https://react.dev/>
- [8] “Tailwind CSS.” Accessed: May 15, 2024. [Online]. Available: <https://tailwindcss.com/>
- [9] “Flowbite React - UI Component Library.” Accessed: May 15, 2024. [Online]. Available: <https://flowbite-react.com/>
- [10] “MUI: The React component library you always wanted.” Accessed: May 15, 2024. [Online]. Available: <https://mui.com/>
- [11] “Chart.js | Open source HTML5 Charts for your website.” Accessed: May 15, 2024. [Online]. Available: <https://www.chartjs.org/>
- [12] “Inertia.js - The Modern Monolith.” Accessed: May 15, 2024. [Online]. Available: <https://inertiajs.com/>
- [13] “Lodash.” Accessed: May 15, 2024. [Online]. Available: <https://lodash.com/>
- [14] “prop-types - npm.” Accessed: May 15, 2024. [Online]. Available: <https://www.npmjs.com/package/prop-types>
- [15] “clsx - npm.” Accessed: May 15, 2024. [Online]. Available: <https://www.npmjs.com/package/clsx>
- [16] “useHooks – The React Hooks Library.” Accessed: May 15, 2024. [Online]. Available: <https://usehooks.com/>
- [17] “React Icons.” Accessed: May 15, 2024. [Online]. Available: <https://react-icons.github.io/react-icons/>
- [18] “OpenAlex: The open catalog to the global research system.” Accessed: May 15, 2024. [Online]. Available: <https://openalex.org/>
- [19] “Open Science nonprofit OurResearch receives \$4.5M grant from Arcadia Fund - OurResearch blog.” Accessed: May 17, 2024. [Online]. Available: <https://blog.ourresearch.org/arcadia-2021-grant/>

- [20] “Microsoft Academic Graph - Microsoft Research.” Accessed: May 17, 2024. [Online]. Available: <https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/>
- [21] “OpenAlex: The Open Catalogue of the Global Research System — Researcher Connect.” Accessed: May 17, 2024. [Online]. Available: <https://blog-sc.hku.hk/openalex-the-open-catalogue-of-the-global-research-system/>
- [22] “About us | OpenAlex help center.” Accessed: May 17, 2024. [Online]. Available: <https://help.openalex.org/about-us>
- [23] “ORCID.” Accessed: May 15, 2024. [Online]. Available: <https://orcid.org/>
- [24] “ORCID iD - Wikidata.” Accessed: May 17, 2024. [Online]. Available: <https://www.wikidata.org/wiki/Q51044>
- [25] “About ORCID.” Accessed: May 17, 2024. [Online]. Available: <https://info.orcid.org/what-is-orcid/>
- [26] “ORCID Statistics.” Accessed: May 17, 2024. [Online]. Available: <https://info.orcid.org/resources/orcid-statistics/>
- [27] “Scopus.” Accessed: May 15, 2024. [Online]. Available: <https://www.scopus.com/search/form.uri?display=basic#basic>
- [28] “File:Scopus logo.svg - Wikipedia.” Accessed: May 17, 2024. [Online]. Available: https://en.m.wikipedia.org/wiki/File:Scopus_logo.svg
- [29] “Scopus | Abstract and citation database | Elsevier.” Accessed: May 17, 2024. [Online]. Available: https://www.elsevier.com/products/scopus?dgcid=RN_AGCM_Sourced_300005030
- [30] “You are Crossref - Crossref.” Accessed: May 17, 2024. [Online]. Available: <https://www.crossref.org/>
- [31] “Leveraging Crossref’s DOI System: Enhancing Publication Integrity in JMBFS – Journal of microbiology, biotechnology and food sciences.” Accessed: May 17, 2024. [Online]. Available: <https://www.jmbfs.org/leveraging-crossrefs-doi-system-enhancing-publication-integrity-in-jmbfs/>
- [32] “About us - Crossref.” Accessed: May 17, 2024. [Online]. Available: <https://www.crossref.org/about/>
- [33] “Introduction to events - Learn web development | MDN.” Accessed: May 15, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events
- [34] “AboutIEG of Internal Evaluation Group.” Accessed: May 17, 2024. [Online]. Available: https://omea.iee.i.hu/site/?locale=en_US&p=AboutIEG