MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Web application for organizing cat shows

Master's Thesis

JAKUB VANKO

Brno, Spring 2024

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Web application for organizing cat shows

Master's Thesis

JAKUB VANKO

Advisor: doc. RNDr. Eva Hladká, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2024


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jakub Vanko

Advisor: doc. RNDr. Eva Hladká, Ph.D.

Acknowledgements

I want to sincerely thank my advisor doc. RNDr. Eva Hladká, Ph.D. for allowing me to work on a topic that is particularly close to my heart. Her unwavering patience when answering my countless questions was greatly appreciated. My thanks also go to my family for their support and to all animals for the companionship they provide us with.

Abstract

This thesis describes the design and implementation of a web application dedicated to simplifying the process of organizing cat shows. The introductory chapters briefly outline the domain of cat shows while mostly focusing on the requirements and standards set by FIFe. The main body of the thesis is devoted to the selection of an appropriate database system with a short overview of different database paradigms. Other key topics, such as the ergonomics of the user interface and the templating engine, are also widely discussed.

Keywords

web application, cat show, database systems, pdf generation, cats, templating

Contents

Introduction 1

1 Background 2
1.1 ČSCH . . . 2
1.2 FIFe . . . 2
1.3 Other international organizations . . . 3
1.4 Organization of cat shows . . . 4
1.5 Existing solutions . . . 5

2 Application requirements 7
2.1 Functional requirements . . . 7
2.2 Use-case diagram . . . 11
2.3 Non-functional requirements . . . 13
2.4 System context diagram . . . 14

3 Selection of a database system 16
3.1 Relational . . . 16
3.2 Key-value . . . 17
3.3 Columnar . . . 17
3.4 Document . . . 18
3.5 Graph . . . 19
3.6 Final selection . . . 20

4 Design 22
4.1 User interface . . . 22
4.2 Architectural decisions . . . 26
4.3 Assigning cats to judges . . . 26

5 Implementation 28
5.1 Data model . . . 28
5.2 User interface . . . 31
5.3 Templating engine . . . 33

6 Conclusion 37

Bibliography 38

A Attachments 40

List of Figures

2.1 Use-case diagram of the application 12
2.2 System context diagram based on the application's requirements 15
4.1 Wireframe for a simple table of objects 23
4.2 Wireframe displaying the screen for editing object details 23
4.3 Wireframe of the login screen 24
4.4 Wireframe displaying a table of objects (cats, judges, etc.) 25
4.5 Wireframe of the modal window for editing EMS codes . 25
5.1 Entity-relationship diagram of the database model 30
5.2 Finished UI of the login screen 32
5.3 Finished UI displaying a table of cats 32
5.4 Finished UI of the modal window for editing EMS codes . 33
5.5 Word template of the show catalog written for the docx-templater engine 35
5.6 A template from Figure 5.5 processed with the actual data of the application 36

Introduction

Cat shows are events where experts, breeders, and cat lovers meet to evaluate the health and specific characteristics of various cat breeds. Apart from providing a platform for enthusiasts to showcase the beauty of their pets, they also play a crucial role in promoting the field of cat breeding to a broader audience.

The number of animals participating in a show can range from several dozen to a few hundred. Given the high number of entries, a need arises to efficiently manage the information about the participants, their owners, the judges, and the judgment results. Since it would be nearly impossible to manage all show-related information only using traditional paper-based methods, software systems can be leveraged to simplify the workflow. However, there is a significant shortage of specialized software solutions for this task, and the small number that exist are often outdated, offer incomplete functionality, or are proprietary and accessible only to a small number of show organizers.

This thesis aims to solve the outlined problem by developing a modern full-stack web application that will offer all the features required to successfully manage a cat show.

The first chapter provides insight into the domain of cat shows and describes the most important entities and processes related to cat breeding in the Czech Republic. The second chapter details the functional requirements placed on the application by the members of ČSCH and by the FIFe show rules. The third chapter examines the various database paradigms and selects the most appropriate database system for the web application. The fourth chapter centers around the application's design with a particular focus on the ergonomics of the user interface, while the fifth chapter describes the key implementational details, such as the templating engine used for generating various show documents. Lastly, the thesis concludes with a review of the resulting application and outlines the potential steps to improve the application in the future.

1 Background

This chapter briefly introduces the most important entities and concepts related to the topic of cat shows in the Czech Republic. The chapter begins by outlining the key Czech and international entities responsible for organizing these events. Subsequently, the processes related to the organization of cat shows are described, and the chapter ends with a short review of already existing systems that offer similar functionality to the application being developed as part of this thesis.

1.1 ČSCH

The Czech Association of Breeders[1], often shortened to ČSCH, is an organization registered and located in the Czech Republic. The primary purpose of this association is to unite individuals and smaller local organizations with a common interest in raising and nurturing various types of animals. The secondary goals include sharing knowledge related to animal breeding among the members in various forms, ranging from printed media to workshops and lectures for younger audiences [1].

ČSCH is made up of multiple sections, each of which is focused on a different type of animal. An example of such a section is The Association of Cat Breeders[2] which is the main organizer of cat shows in the Czech Republic [2].

ČSCH is also a member of various international organizations, and many of its activities are controlled and directed by the policies of these organizations. One such organization is *Fédération Internationale Féline*, also shortened to FIFe [1].

1.2 FIFe

FIFe is an international organization focused on uniting the processes related to cat breeding. It comprises 42 member organizations from

1. Original name in Czech is "*Český svaz chovatelů*"
2. Original name in Czech is "*Sdružení chovatelů koček*"

40 different countries, overall spanning well over 100,000 individual people [3].

The main organizational structure of FIFe consists of an executive board comprising six individuals elected by the members during a general assembly for the period of three years. Alongside this board are six specialized supporting commissions[3] tasked with implementing the decisions made by the executive board and the general assembly. The member organizations have committed to complying with the decisions of the FIFe commissions, with four out of six commissions focusing on specific areas of cat breeding. The most essential commission regarding this thesis is the Show Commission which manages a document describing the rules and concepts for organizing and running cat shows [4].

1.3 Other international organizations

While this thesis focuses on FIFe and its regulations, there are also multiple other international entities with a similar focus on cat breeding:

1. **World Cat Congress**

   The World Cat Congress is a global coordinating entity of cat associations. Its primary objective is to improve collaboration among its members (which include FIFe) in increasing the well-being of cats and sharing knowledge and resources related to cat breeding [5].

2. **Cat Fanciers' Association**

   The Cat Fanciers' Association is an organization founded in the United States that describes itself as the world's largest registry of pedigreed cats. It has over 2 million registered cats and organizes over 400 cat shows annually [6].

3. **The International Cat Association**

3. The commissions are the Judges & Standard Commission, the Breeding & Registration Commission, the Show Commission, the Health & Welfare Commission, the Disciplinary Commission, and the Commission for Public Relations

The International Cat Association describes itself as the world's largest genetic registry of pedigreed cats - it has over 70 recognized cat breeds, more than most other associations. Additionally, its members organize about 200 cat shows per year. [7].

4. **World Cat Federation**

   The World Cat Federation is an international federation of cat clubs founded in Brazil and based in Germany. It has 280 affiliated associations and organizes about 500 cat shows per year [8].

Lastly, there are many national organizations that are outside the scope of this thesis.

1.4 Organization of cat shows

In order for a cat show to comply with FIFe regulations, specific rules and procedures must be followed. These procedures and the overall organization of the show can be roughly divided into the following stages:

1. **Preparation**

   At the beginning of the first phase, a request for organizing a show is made to FIFe. If a license is given, the show is announced to the public, and applications can be collected. Once the application deadline is passed, a catalog containing details about the contestants, their owners, and the judges is printed and distributed. Only the cats appearing in the catalog[4] may participate in the show. Lastly, once all the formalities are finalized, the premises where the competition will occur are prepared and provisioned with all the necessary equipment [9].

2. **Execution**

   On the day of the show, after a veterinary inspection, the cats are assigned to individual judges. Based on this assignment,

4. Sometimes additional contestants may register after the deadline, which results in the need to print an annex of the catalog

report sheets are printed and prepared for the judges to record their assessments. There are many classes in which the cats may compete[5], and every judge can assess only the breeds they are qualified for. The cats are judged based on many characteristics, such as the shape of the ears, the tail, the nose, and others. Once all the cats have been fetched, presented, and assessed, the results from the report sheets are aggregated, and the winners for the respective categories are announced. Following that, there may be an additional assessment round for the Best in Show title. When the final results are decided, diplomas are printed and awarded to the winners [9].

3. **Conclusion**

   Following the successful execution of the show, all important documents, including the judges' reports and the catalog, must be archived for at least three years. Additionally, the results of the competition must be recorded in the relevant records of the corresponding pedigree book [9].

The processes for successfully running a cat show are complex and organizationally demanding. Therefore, it is beneficial to have a proper system that can help with organizing the data, generating documents, and managing the workflow of the show.

1.5 Existing solutions

Currently, ČSCH and their Association of Cat Breeders rely on an old system implemented using FoxPro. FoxPro is a programming language and database management system that has reached the end of life and whose official support was discontinued in 2007 [10]. However, due to a lack of other options, the association continues to use this system despite known bugs and insufficiencies that make it hard to use it effectively. It is important to note that direct access to the system could not be obtained, and all information regarding the functionality and limitations of the system is based on conversations with members of ČSCH who are familiar with its usage.

5. The cat's class is usually specified in their application form and is decided based on the cat's previous competition achievements

Apart from the aforementioned system, no other open-source or paid alternatives were found during the research. Despite rumors regarding the development of other proprietary systems by smaller local organizations, the validity of such statements could not be verified, and it is uncertain whether they are fully operational or offer significant improvements compared to the current FoxPro system. Therefore, given the lack of a publicly available and modern system for the organization of cat shows, there is a clear need to develop a new solution to fill this gap in the market.

2 Application requirements

This chapter outlines all of the functional and non-functional requirements of the application being developed as part of this thesis. These requirements were specified based on a thorough analysis of FIFe regulations and through consultations with the members of ČSCH.

The chapter begins by listing the individual functional requirements, which define the specific actions the application is expected to perform. A use-case diagram is provided for better visualization of the functional requirements. Additionally, the chapter also describes non-functional requirements, which focus on the qualities and constraints of the application. Lastly, the chapter ends by showcasing a system context diagram that provides an overview of the data flow and the external entities the application is supposed to interact with.

2.1 Functional requirements

The following functional requirements define the core capabilities that the application must have to effectively assist with organizing cat shows in compliance with the FIFe show rules:

1. **Authenticating a user**

   The application shall allow users to create their accounts and authenticate themselves. The exact way of authentication, whether it's through a magic link, password, or any other method, shall be determined by the developer. The account creation process shall involve collecting necessary user information, including at least an email address. Afterward, the application shall validate the provided information and ensure the uniqueness of email addresses across the system. Once the account is created, users shall be able to access their account using their credentials. The application shall verify the provided credentials against the stored user information and grant access to properly authenticated users.

2. **Managing a user account**

Once logged in, users shall have the ability to manage their account information. This includes updating personal details and other credentials. The application shall provide appropriate validation of the updated information.

3. **Support for multiple languages**

   The application shall be easily extensible to allow future developers to translate the application into another language. Apart from translating the user interface screen, the translation of the generated documents may also be needed, and this must be considered while designing the system's database model.

4. **Validating data**

   User-entered data shall be validated for invalid values. Dropdown menus of selected attributes (such as breed codes) shall be provided to users to simplify their workflows.

5. **Managing shows**

   Users shall be able to view a list of their shows. They shall also be able to access the show details, update them, or delete the entire show altogether.

6. **Importing data for a show**

   The application shall allow show organizers to import relevant data associated with the show contestants. The process of importing data shall parse cat information and the corresponding owner details. The application shall support importing data either in the form of a file or by calling a REST endpoint.

7. **Generating show catalog**

   Show organizers shall be able to generate a show catalog populated with the application's data. According to *FIFe Show Rules* [9], the catalog must contain at least the following information:

   - A list of all cats, including their catalog number, name, titles, designation of breed and color in the form of EMS[1]

1. A FIFe code system used to describe individual cat properties

codes, date of birth, sex, class in which they are to be judged, names and EMS codes of their parents, the name of their breeder, the name of their owner, and possibly also a group number

- A list of the judges, including the countries of their residence and the categories they are qualified to judge
- A list of all owners of the individual cats, including their name, country of residence, catalog number of their cats, and a FIFe sub-organization they are a member of

The generated catalog shall be exported in a printable format, such as PDF.

8. **Generating catalog annex**

   In some cases, additional contestants may register for the show after the catalog has already been generated. The application shall provide functionality for generating catalog annexes to accommodate these late registrations. An annex shall contain all necessary information about the additional contestants in a format similar to the one provided in the generated catalog.

9. **Managing cats**

   The application shall allow show organizers to add, update, or delete details of cats corresponding to a given show. Additionally, users shall be able to view a list of cats for the show they are organizing.

10. **Managing judges**

    Show organizers shall have the ability to manage the details of judges associated with a specific show. The application shall provide functionalities to add, update, or delete information about judges as needed. Additionally, organizers shall have easy access to view details about each judge, including the categories they are qualified to judge.

11. **Assigning cats to judges**

The application shall allow show organizers to assign cats to judges and adjust the assignments as needed. The changes shall always be reflected in the generated documents.

12. **Generating report sheets for judges**

Once cats are assigned to judges, the application shall allow show organizers to generate report sheets for every judge and the corresponding cats. According to *FIFe Show Rules* [9], the headings must be specified in English, French, and German, with additional languages being optional. Additionally, the report sheet must contain at least the following information:

- The location and the date of the show
- The catalog number of the cat to be judged
- The EMS code of the cat to be judged
- Date of birth and sex of the cat to be judged
- Class in which the cat is to be judged
- Group number (if applicable)

The generated report sheets shall be exported in a printable format, such as PDF.

13. **Managing show results**

After the judging process is complete, the application shall allow show organizers to record the results from the report sheets. Based on the provided data, the application shall provide a ranked list of the most successful contestants.

14. **Managing best-in-show assessment**

In addition to the results of individual judge assessments, some shows may include a best-in-show assessment to determine the overall winner. The application shall support this assessment and provide functionality to generate the required documents in a printable format such as PDF.

15. **Generating labels**

    The application shall allow show organizers to create labels containing the participants' data. Each label shall contain all basic information needed to identify one show participant. The labels shall be exported in a printable format, such as PDF.

16. **Generating diplomas**

    Once the show results are finalized, the application shall provide functionality to generate diplomas. The diplomas shall include relevant information about the show, the class in which the cat won, and possibly the titles the cat gained by winning. The diplomas shall be exported in a printable format, such as PDF.

17. **Exporting results for pedigree book**

    Once the show ends, the application shall provide a way to export the data to an external electronic pedigree book system.

2.2 Use-case diagram

The use-case diagram provides a visual representation of the system's actors and the use cases that the actors can perform. While the use cases were mostly defined in the previous section, it is necessary to additionally also define the actors involved in the system:

1. **Show organizer**

   A show organizer is a member of an association that organizes a show. They can view and manage all their shows as well as create all show-related entities and print the generated documents.

2. **Show application system**

   The show application system is an external system being developed by another developer. Its main responsibility shall be to collect the applications for the individual shows and export the data to the application being developed as part of this thesis.

Based on the functional requirements and the corresponding actors, the diagram in Figure 2.1 was created.

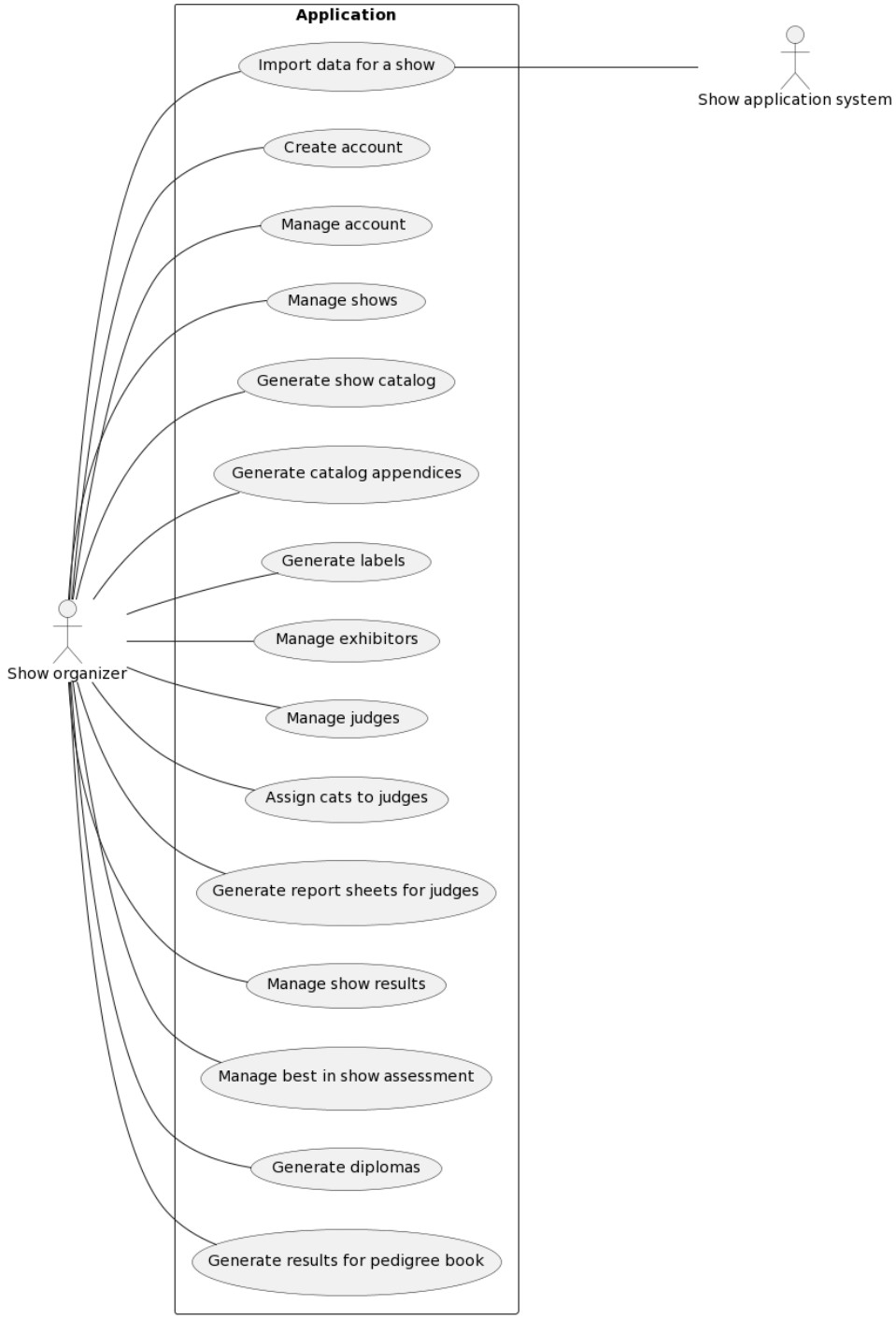


Figure 2.1: Use-case diagram of the application

2.3 Non-functional requirements

In addition to functional requirements, applications must also meet specific qualities known as non-functional requirements. These requirements encompass the desired characteristics that the application shall possess. To ensure the development of a successful application, consultations were conducted with future users, resulting in the identification of the following key non-functional requirements:

1. **Usability**

   The application shall have an intuitive and user-friendly interface. All input elements shall be clearly explained using proper icons and labels. Additionally, the user interface should be ergonomic to provide a fast and comfortable way of data manipulation.

2. **Performance**

   The application shall be capable of handling even large volumes of data without significant delays or decreases in performance.

3. **Reliability**

   The application shall run smoothly without unexpected crashes or system failures. All inputs shall be properly validated to ensure the consistency of stored data.

4. **Security**

   The application shall use modern cryptographic algorithms and ensure that only properly authorized users can access its data.

5. **Scalability**

   The application shall be prepared for future potential growth and be able to handle an increasing number of users, shows, and participants.

6. **Maintainability**

   The application shall be designed to be easily maintainable, allowing for simple bug fixes and enhancements. The codebase shall follow industry best practices with a special focus on code

quality and separation of concerns. Additionally, the code shall be modular to allow for possible future expansions into organizing shows for animals other than cats.

7. **Compatibility**

   The application shall be compatible with various devices, operating systems, and web browsers. The user interface shall be responsive and adaptable to different screen sizes and resolutions.

8. **Cost**

   The application shall have minimal deployment and operational costs. Free services, libraries, and open-source technologies shall be preferred for designing, implementing, and deploying all parts of the application.

2.4 System context diagram

A system context diagram serves to display the interactions between a system and other external entities. To further visualize all functional and non-functional requirements defined in this chapter, a system context diagram was created and can be seen in Figure 2.2.

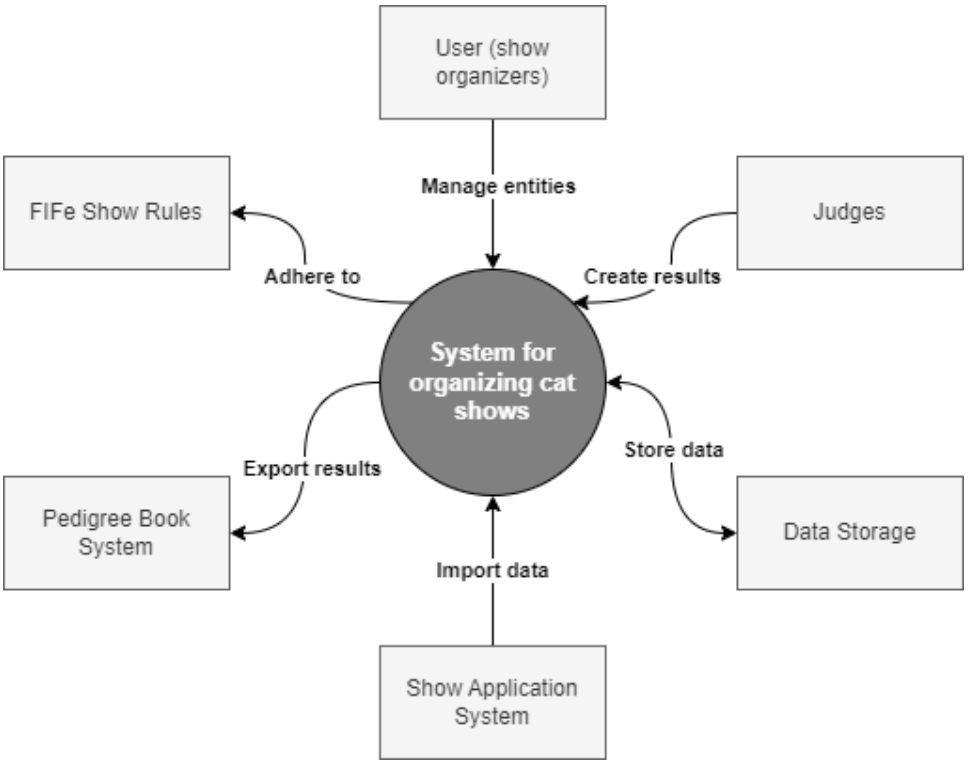


Figure 2.2: System context diagram based on the application's requirements

3 Selection of a database system

Since the application developed as part of this thesis mainly comprises data-related operations, selecting an appropriate database system is vital for its proper functionality. This chapter briefly overviews the five database paradigms described by Redmond; Wilson [11]: relational, key-value, columnar, document, and graph. Each of these paradigms is evaluated, and the most suitable is selected. Afterward, the actual database systems implementing the selected paradigm are reviewed, and the best one is picked to be used in the final application.

3.1 Relational

A relational database model is built on the foundations of relational algebra. Every collection of entities stored in the system is described as a relation of data attributes (also called schema) and is stored in its own table. Every entity is stored as a row in the corresponding table and can be clearly specified by a primary key. The connections between entities are stored using foreign keys that link to the primary key of the desired other entity. The relational databases are queried using various variants of the SQL syntax, allowing users to make complex requests easily [11].

The approach of storing entities as rows is beneficial in giving quick access to all of the given entity's attributes. However, querying based on the values of individual attribute columns may take longer. This can be improved by creating indexes for columns based on which values we expect to query. The main trade-off for improving querying speed is in slowing operations related to adding or updating the data since the individual indexes for the updated data also need to be recomputed [12].

According to a survey by DB-Engines [13], relational database systems are the most popular database solutions, with over 70% of relevant database-related mentions on the web. This popularity naturally correlates with a large knowledge base, making the relational paradigm a stable and conservative pick.

Since a schema needs to be defined for the entity upfront before any data related to it is stored, any changes in the entity attributes create

a need for the schema to be redefined, which may cause issues with the already stored data. This can be problematic when the exact data attributes of the entities are unknown or if they may quickly change [11], for example, due to the changing requirements of customers during the development process. This only signifies the conservativeness of the relational paradigm, making it ideal for stable projects and enterprise systems.

3.2 Key-value

A key-value model stores data as a collection of key-value tuples. The key is usually a simple string, whereas the corresponding value might be a more complex object - often a hash table of other key-value pairs that make up the attributes of the stored object. This simplicity guarantees a high speed of data access, making this database paradigm an ideal choice for high-traffic applications [11].

However, the unstructured nature of this database paradigm makes it difficult to store and query any complex information regarding relationships between individual objects. This feature disqualifies the key-value paradigm from being the primary data storage option of most applications which often resort to using it only as a simple cache between the actual, more complex database system and the user [11].

3.3 Columnar

Columnar database systems are very similar to relational ones as they sometimes also offer interchangeable querying functionality using the SQL language [14]. The main difference between the two paradigms is that whereas relational databases map the primary key of the entities to their attributes, columnar systems map the individual column values to the entity's key [15]. This seemingly small change drastically improves the performance of querying based on the individual column values. However, this also means that querying all of the data for the particular entity based on its primary key will be slower since the data needs to be aggregated from multiple columns [14]. Therefore, it is vital to know the application's use cases before deciding whether or not to choose a columnar database system [11].

Another benefit of using columnar database systems is flexibility which stems from the fact that the number of attributes an entity may have is optional. Since the columns can be easily split across multiple nodes, this paradigm also provides better scalability than relational database systems [11].

DB-Engines [13] ranking approximates the popularity of columnar database systems to be about 5%. This is a sharp decline compared to the relational paradigm, which directly corresponds to a lower share of technical documentation, tutorials, and community support. However, for certain domains, the benefits may still outweigh the drawbacks as columnar database systems are widely used, mainly in the domain of big data [11].

3.4 Document

Document-oriented database systems store data as documents. One object and all corresponding metadata are usually stored as one document and defined using various markup languages, such as JSON, XML, or YAML [16]. These formats allow for storing more complex objects (such as arrays) as well as for nesting such objects inside one another. The objects can also often be serialized into data objects of various programming languages [11]. Documents of similar content are grouped together in collections (for example, a collection can store all documents that specify individual users of the application), and objects are queried based on the collection they are expected to be stored in [16].

Contrary to the relational paradigm, objects of a specific type do not have to adhere to a fixed schema strictly [16], nor are they supposed to be normalized into smaller logical units [11]. This approach may lead to the duplication of some information but also to the higher efficiency of retrieving data (since all data for the queried objects can be found in one place and no join-like operations have to be performed). The flexible schemas also allow faster development and straightforward adaptation to changing requirements [16].

Document-oriented databases are NoSQL, meaning developers cannot use the SQL language and must query the data using other system-specific commands. This makes sense given the non-relational

approach of storing data and the lack of need for more advanced SQL features such as joins or secondary indexing [17]. However, sometimes the features might be lacking, and the non-standardization of query languages might cause problems when migrating to other similar systems. This downside can be, however, also solved by, for example, using an ORM framework or a repository design pattern.

According to DB-Engines [13], document database systems are the second most popular, with a popularity of over 10%. This means that documentation, tutorials, and community support are still readily available. At the same time, the simplicity of this paradigm also allows new developers to quickly understand the data model of their particular application and develop new features faster. Therefore, document-oriented database systems have quickly garnered the interest of companies adhering to agile development practices, giving rise to even more technical and financial support for this database paradigm [17].

3.5 Graph

Graph databases differ from the other paradigms by offering a novel approach to storing data that better simulates the thought patterns of data storage in human brains. Individual objects are stored as nodes in a graph, whereas the relationships between the objects are denoted as connections between the nodes. Both nodes and relationships in the graph can hold additional information called properties. Properties are simple key-value tuples, and every node and relationship can store as many of them as needed [18].

Graph databases are queried by traversing the individual nodes [11], which helps to uncover hidden relationships between the objects [18]. This makes graph databases a great option for specific use cases, such as creating knowledge graphs or storing information used by recommendation engines [11].

The main downside of graph databases is poor scaling. Since the data needs to be stored in one large graph, it may be difficult to split it without causing performance issues. That is why large graph databases often store only the basic information about relationships while other data related to the objects are stored using other, more scalable database systems [11].

Despite the recent growth in popularity, graph databases still garner only about 1.8% of the relevant database-related mentions on the web [13]. This means that there may not be enough documentation or community support. Therefore, an already skilled development team is needed to make the most use of this database paradigm.

3.6 Final selection

Based on the non-functional requirements specified in Chapter 2 - particularly the focus on stability, maintainability, and low cost - the relational database paradigm was selected as the database solution for this thesis.

When it comes to the individual implementations of the relational database paradigm, the following solutions were briefly researched:

1. **Oracle**

   Oracle is an enterprise-ready database system that supports the usage of multiple paradigms (including the relational) within a single database. It is also provided with extensive documentation and customer support. However, the free version is very limited in functionality, and the price for the professional versions can climb into thousands of dollars [19]. This high cost, along with the steep learning curve of this database system, is the reason why it was not selected for this thesis.

2. **Microsoft SQL Server**

   Similar to Oracle, Microsoft SQL Server is a relational database solution designed for enterprise customers. Apart from the functionalities that Microsoft SQL Server shares with other similar software, its main features also include support for data mining and database warehouses. Naturally, this rich feature set comes with a restrictive license and a high price tag that can reach up to 15,000 USD for a single core [19]. Consequently, Microsoft SQL Server is not a good fit for this thesis.

3. **SQLite**

   SQLite is a small-scale relational database system that stores all data and configurations in a single file that can be directly

embedded into the application. While this simplistic approach is great for small applications that do not need the extensive functionality offered by other relational systems, it also performs poorly when used concurrently by multiple users [19]. Since many people are expected to use the application developed in this thesis, SQLite was not selected for the final result.

4. **MySQL**

   MySQL is a popular relational database system with many features and a mild learning curve. Formerly an open-source software, it was acquired by Oracle Corporation and nowadays is mostly under Oracle's license. In comparison with other database systems, it performs worse in regards to scalability. Additionally, its compatibility with SQL standards is also limited, which could make it harder to switch to a different database system later [19]. Given these reasons, MySQL was not picked as the final database system for this thesis.

5. **MariaDB**

   MariaDB is a completely open-source fork of MySQL, and while it originates from the MySQL engine, a number of optimizations were done to improve its performance. Additional features worth mentioning include advanced security functionalities such as PAM and LDAP authentication, support of third-party plugins, and NoSQL options. While MariaDB sounds like a great candidate for this thesis, its community is still quite small [19], which is the reason why another option was preferred instead.

6. **PostgreSQL**

   PostgreSQL is an object-relational database system with a feature set similar to the previously discussed MariaDB. Apart from the traditional data types, it also supports storing advanced data formats such as JSON or XML. Being fully open-sourced [19] may also be one of the reasons for the popularity of PostgreSQL, with it being ranked first according to *Stack Overflow Developer Survey 2023* [20]. Given all the aforementioned benefits, PostgreSQL was selected as the final database system for this thesis.

4 Design

This chapter focuses on the major design decisions taken before the implementation of the application. The chapter starts by discussing the ergonomics of specific design approaches and showcases the wireframes of the application's final user interface. Afterward, the chapter continues by explaining the high-level architectural decisions and some notable parts of the application's technological stack. Lastly, the chapter ends by reviewing the details behind the process of assigning cats to judges and how the application can be extended to handle such workflow automatically in the future.

4.1 User interface

According to the non-functional requirements specified in Chapter 2, the application should be designed to be ergonomic and provide an efficient way of data manipulation.

Since most of the user data will be comprised of list-like data structures (list of cats, list of shows, list of judges, etc.), the application needs to provide a unified way to display these lists along with proper filtering based on the important details of the individual objects. Additionally, it is vital that the user can effortlessly navigate between the individual objects in these lists and modify their details quickly.

In the context of the web, the display and edit functionalities for list items can be easily achieved through the use of HTML tables. This method enables quick visualization of all items in a list by arranging item attributes in table columns. Additionally, the option of editing an object can be integrated directly within the table via a dedicated column containing edit buttons linked to the individual item's edit page. This concept is illustrated in the wireframes shown in Figures 4.1 and 4.2.

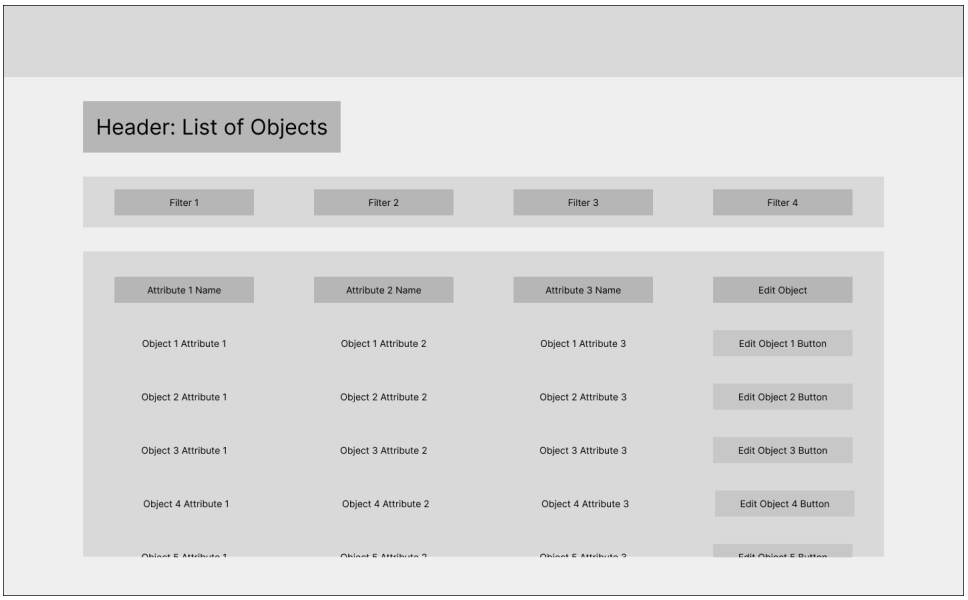


Figure 4.1: Wireframe for a simple table of objects

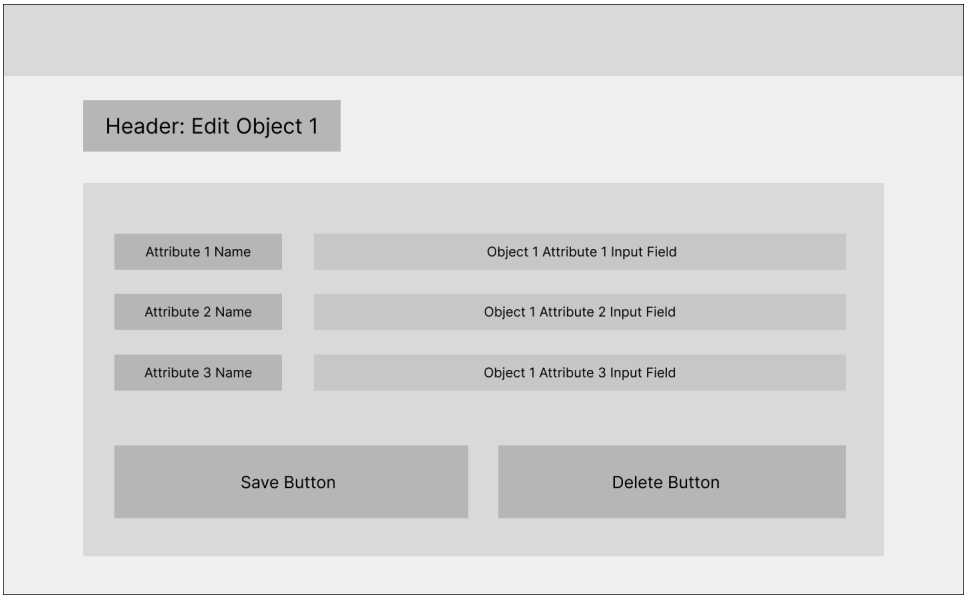


Figure 4.2: Wireframe displaying the screen for editing object details

While the described approach is straightforward for viewing and editing list items through HTML tables, it can become inconvenient for tasks requiring rapid modifications of multiple items, such as when entering judgment results. In those scenarios, navigating back and forth between the list and edit pages for each item would be inefficient and time-consuming.

To enhance the user experience, an alternative design approach was selected. Instead of having a separate edit page for each item, the fields within the HTML table can become directly editable. Changes made to these fields can be instantly synchronized with the database, eliminating the need for explicit save actions. This design allows users to quickly edit the necessary information directly within the table, significantly reducing the number of steps and time required to modify multiple items. The only exception to this workflow is setting EMS codes for cats - since it requires multiple input fields, it shall be handled by a modal window.

Based on this new approach, the final wireframes for the application were created and can be seen in Figures 4.3, 4.4, and 4.5.

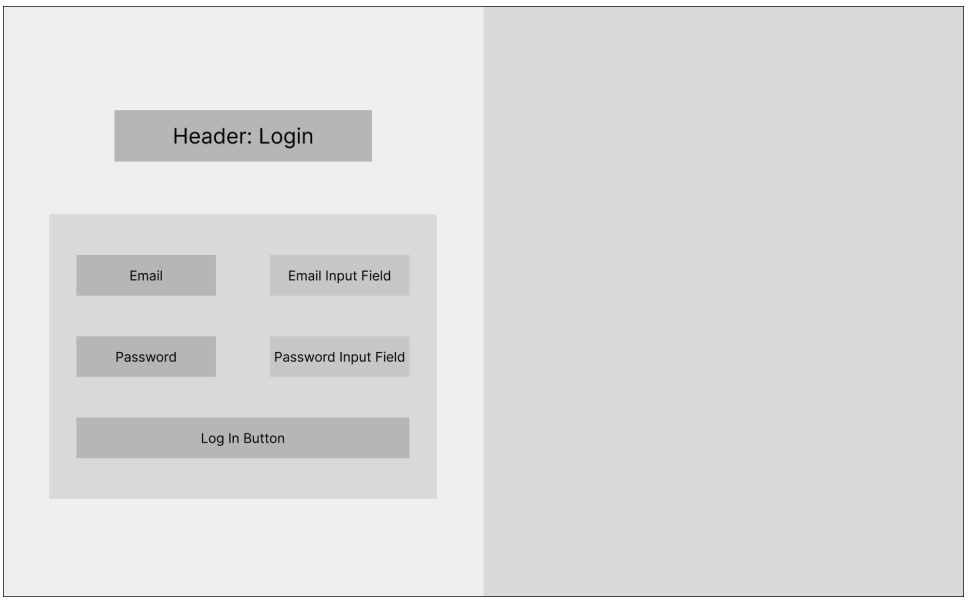


Figure 4.3: Wireframe of the login screen

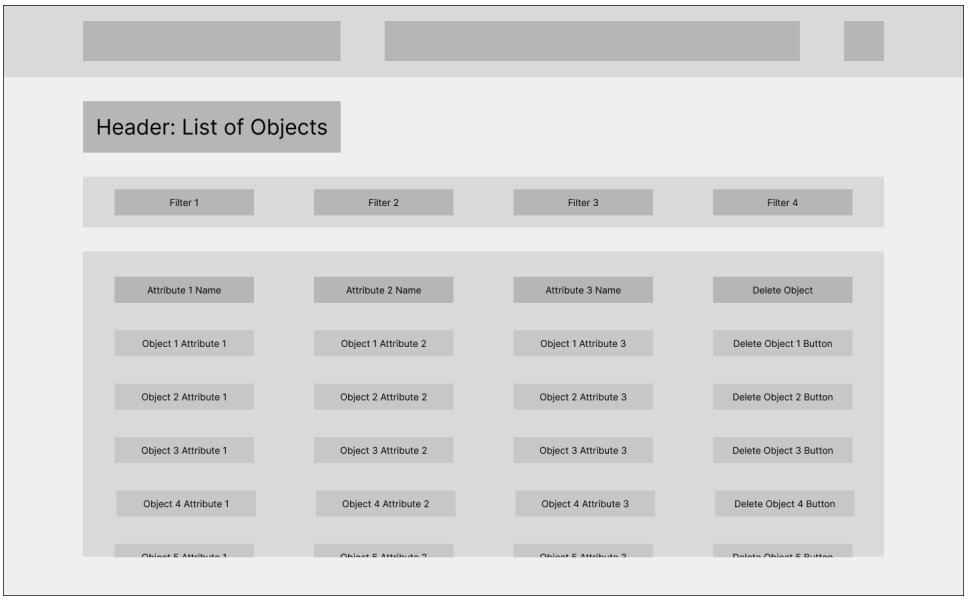


Figure 4.4: Wireframe displaying a table of objects (cats, judges, etc.)

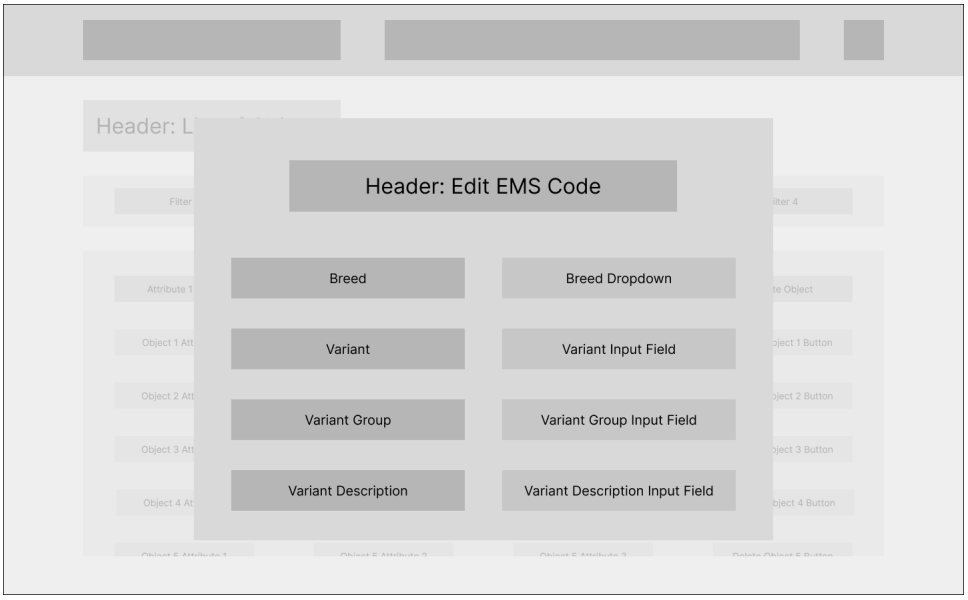


Figure 4.5: Wireframe of the modal window for editing EMS codes

4.2 Architectural decisions

In the context of the web, there are roughly two options for building a full-stack web application. The first approach consists of using a specialized full-stack framework that is able to handle both the client and the server side of the application. Examples of these frameworks include Next.js or Remix, and their code is mostly written using JavaScript or TypeScript. The other option is to split the application into completely separate front-end and back-end parts which can be developed independently. However, this freedom of development also comes with a cost since the developer needs to ensure compatibility between the front-end and the back-end layers as well as fault-tolerant communication pathways for exchanging information between them.

After brief research of the aforementioned full-stack frameworks, it was decided that they do not offer the required flexibility for the application developed as part of this thesis. Therefore, the development will proceed with independent front-end and back-end layers, allowing for easier future maintenance and updates.

To ensure reliable and robust communication between the front-end and back-end parts of the application, a front-end library called Tanstack Query is utilized. This library offers declarative fetching logic, optimistic updates, caching, and handling of network errors. Apart from that, other notable technologies that are to be used during the development include React as the main front-end framework, a headless table component called Tanstack Table, and a validation library Zod.

4.3 Assigning cats to judges

The task of assigning cats to individual judges can be very time-consuming because show organizers need to take into account the following points:

1. **Breed categories that judges can assess**

   There are four categories, and each recognized cat breed belongs to one of them. For a judge to be able to assess cats of a given breed, the judge has to pass an examination for the given cat-

egory. Therefore, when assigning cats to judges, only cats that the judge is allowed to assess can be assigned to them.

2. **Workload for individual judges**

   One of the hallmarks of a good assignment of cats to judges is that the number of cats assigned to every judge is roughly similar. One judge shouldn't have more cats assigned to them than someone else, as it can lead to exhaustion and a longer judging process.

3. **Maximizing the number of nominations for BIS per judge**

   In each specific subcategory[1] of cats, a judge is limited to nominating only one cat for the best-in-show assessment. This limitation implies that if a single judge evaluates all cats within a subcategory, only one cat will advance to the best-in-show round. This scenario not only reduces the excitement for the audience but also lessens the overall competitiveness of the show. Therefore, it is desirable to divide the cats from each subcategory to as many judges as possible.

4. **Maximizing the number of BIV titles per judge**

   Based on *FIFe Show Rules* [9], the best-in-variety is an assessment where a single cat will be chosen by a judge among all cats of a specific variant they are judging. However, the judge must have at least three cats of a specific variant for this assessment to take place. This point directly contrasts with the previous point where every judge should have as few cats of a variant as possible.

As showcased by the previous points, the process of assigning cats to judges is very complex and needs to consider multiple conditions. While automatizing this process is outside the scope of this thesis, it is important to note that it is an NP-hard problem. Given the potentially large number of participants involved in a show, this task can become computationally intensive and should not be solved using a brute-force method. Instead, in the future, the application can be extended with a solver that will be able to find a near-optimal solution quickly.

1. A subcategory is a specific breed variant further divided by sex and age

5 Implementation

This chapter outlines the implementation details of the developed application. It begins by describing the application's data model, including a detailed entity-relationship diagram. Afterward, the final implemented user interface screens are showcased with a short description of the component library called shadcn/ui that was used for their creation. Lastly, the chapter ends by detailing the templating engine that is utilized for generating the required documents.

5.1 Data model

The data model of the application is complex because the following points have to be taken into account:

1. **Multitenancy**

   The application is designed to run only as a single instance. However, it is expected that a large number of users will register into the application and manage their shows. Therefore, every user must be allowed to access only the data they own and be denied access to other users' data. This means that all entities stored in the database must have at least an indirect relation to the entity representing a registered user.

2. **Storing of document templates**

   Templates for generated documents are represented in a binary format. To allow users to upload and use their own customized templates, a way to store these files and link them to their owner would have to be provided. Although binary files can be directly stored in the database as a so-called blob, it is not considered an optimal solution. Instead, it would be better to store these templates directly in the file system while only saving their path to the database. However, since allowing users to upload their own templates is outside the scope of this thesis, there is no need for any template-related information to be stored in the database.

3. **Internalization**

   Based on the requirements specified in Chapter 2, the application shall be easily translated into other languages. While it is simple to fulfill this requirement on the client side of the application by extracting the individual strings into files for individual locales, the complexity of this problem increases when the translation of the generated documents must also be supported. In that case, it is no longer efficient to store the translated strings in separate files. Instead, some of them should be stored directly in the database, which will allow for simple and direct access by the templating engine.

4. **High number of relationships between entities**

   The entities that the application is required to store have a lot of relationships with each other. One example of this is the show entity which is related to all other entities that a user can create (cats, judges, results, etc.) because all of those entities make sense only in the context of a particular show. Therefore, it is important to carefully define those relationships in the data model so that no fixes are necessary during the application's future development.

Based on the previous points and other functional requirements, a database model was constructed, and its entity-relationship diagram can be seen in Figure 5.1.

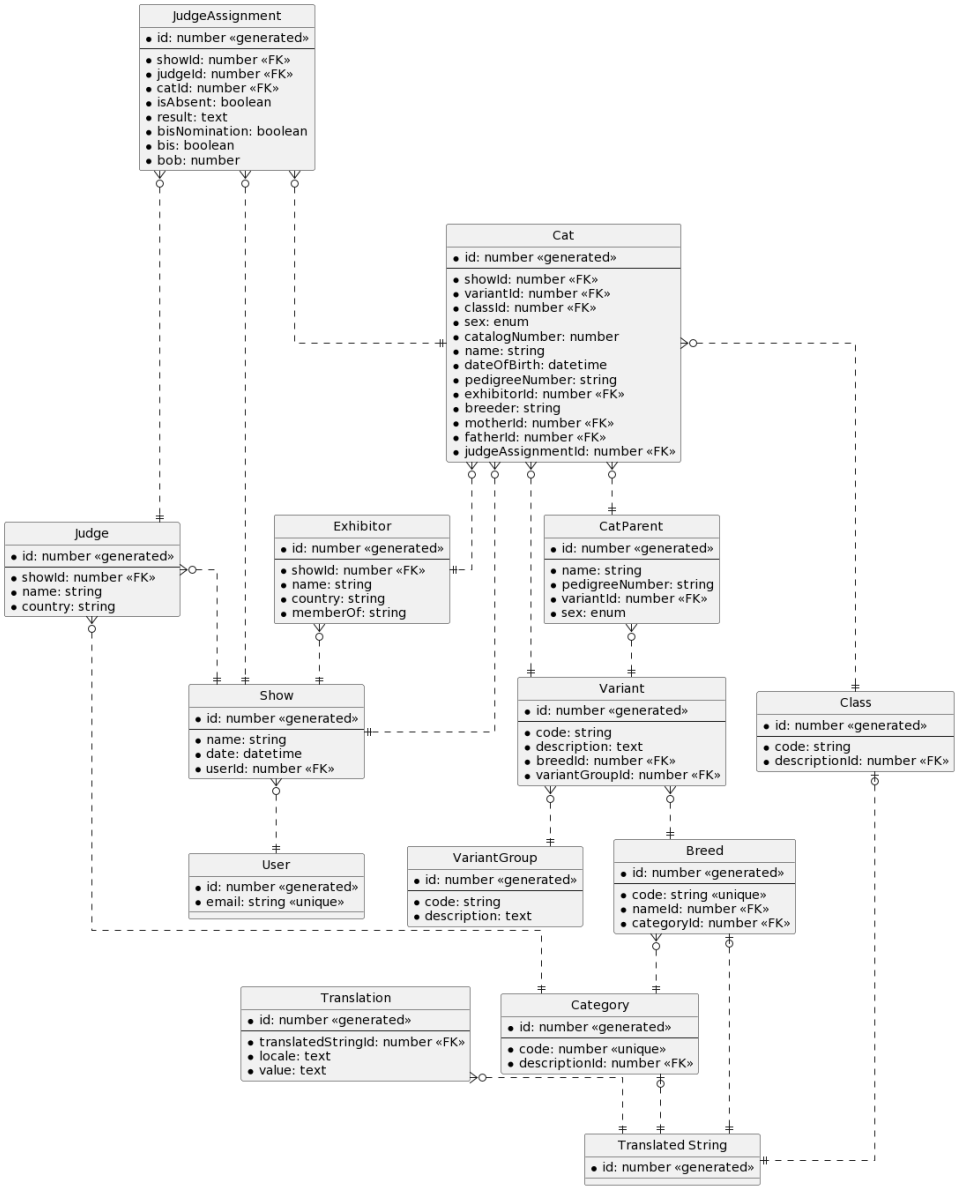


Figure 5.1: Entity-relationship diagram of the database model

5.2 User interface

As described throughout this thesis, a modern ergonomic user interface is vital for a good user experience of the application.

There are a number of premade accessible component libraries for any modern UI framework. However, a vast majority of these libraries provide their UI components as npm[1] packages. This means that only a programmatic API is provided to the developer without direct access to the underlying source code. One exception to this is a React component library called shadcn/ui which provides scripts to directly paste the components into the code of the application. This grants the ownership of the entire component's code to the developer of the application, enabling more customizability and robustness for future updates. Given these benefits, shadcn/ui was selected as the main technology for the implementation of the wireframes from Chapter 4.

Additionally, it is worth mentioning that generative artificial intelligence[2] was utilized for the creation of the illustration that is displayed on the login screen of the application and can be seen in Figure 5.2. Furthermore, screenshots of the application's user interface corresponding to the wireframes from Chapter 4 can also be seen in Figures 5.3 and 5.4.

1. Node package manager
2. The exact model of the generative artificial intelligence is called Stable Diffusion XL and was used with the following prompt: "Software for organizing cat shows, black and white background, artistic, stripes, artstation, 4K, high quality, fantasy, modern, clean, professional, business"

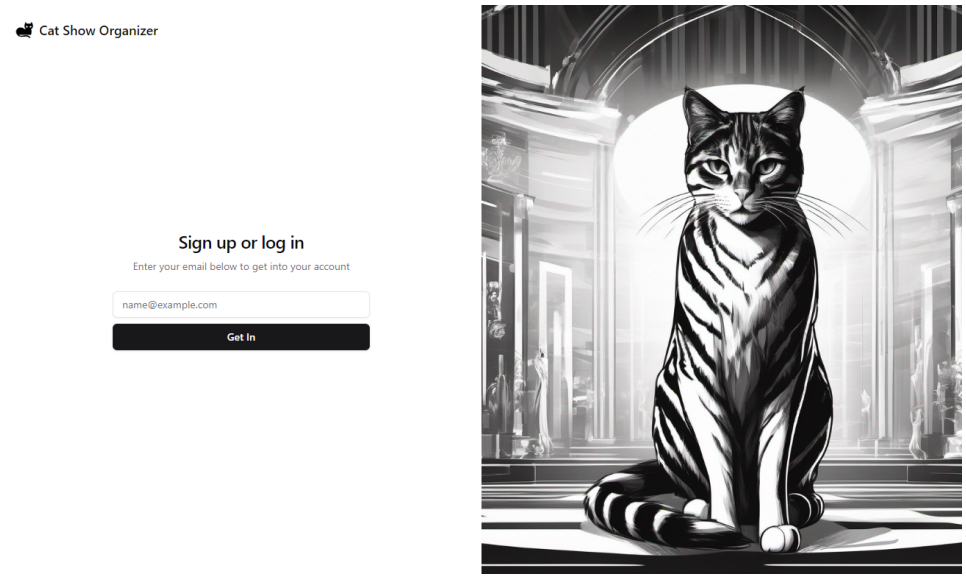


Figure 5.2: Finished UI of the login screen

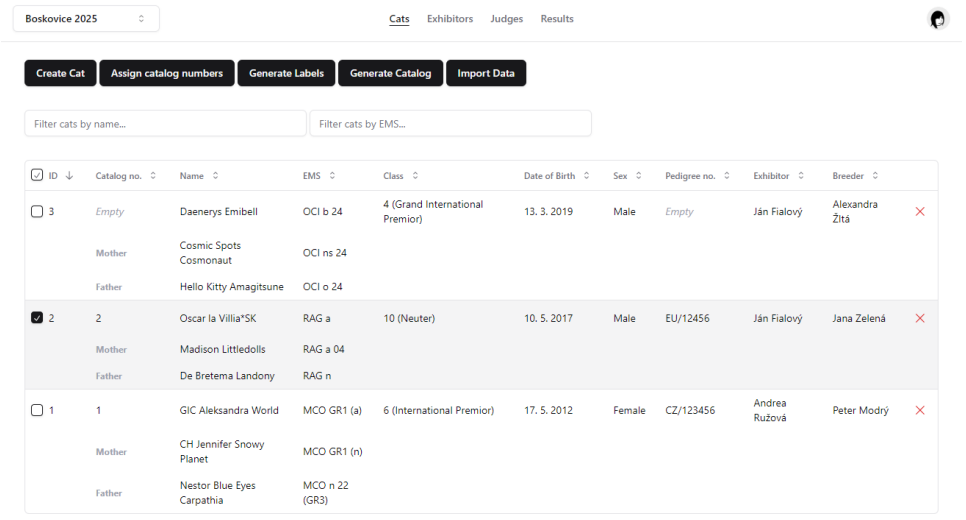


Figure 5.3: Finished UI displaying a table of cats

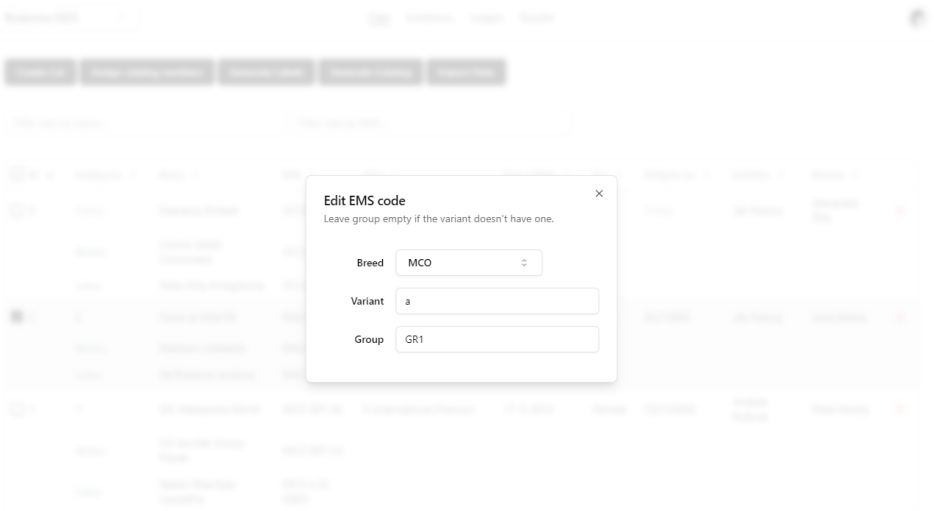


Figure 5.4: Finished UI of the modal window for editing EMS codes

5.3 Templating engine

The templating engine used to generate the required documents, as specified in Chapter 2, is a key component of the application's architecture. There are a number of options that can be employed for specifying the document templates and rendering them based on the actual data. The following approaches were researched:

1. **Generate PDF documents directly using a specialized library**

   The arguably most simple and straightforward approach for generating PDF documents is to use a dedicated code library built for creating and manipulating PDF files. The optimal place for the inclusion of the library would be on the back-end side of the application since it could have simple access to the database for filing out the PDF with the required data. The specific selection of the library depends on the programming language in which the back-end layer is programmed. However, many of these libraries use a similar declarative approach where the

programmer defines directly in the code which elements to insert in which place in the given PDF document. Although this approach is simple, the document templates would be difficult to update because their structure would be hard-coded directly in the application's code. Since allowing more flexibility in defining the templates would be beneficial, this approach was not deemed suitable for this thesis.

2. **Generate PDF documents using React**

   A great option to conserve computational resources is to render the PDF files on the application's client side. Since the frontend of the application is built using React, it could be systematic to use React also to render the PDF documents. This could be made possible by a library named react/pdf which uses a similar declarative approach to the other libraries previously discussed, however, using the syntax of React. While this would save server resources, it would be necessary to send all data from the server to fill out the templates, potentially increasing data transfer costs. Additionally, the templates would have to be hardcoded within the front-end codebase, introducing limitations similar to those observed with the previous approach. Therefore, this option was also not selected for the final implementation.

3. **Generate PDF documents by rendering HTML**

   One option to grant more flexibility in modifying the individual templates would be to define them as HTML pages that would just get populated with data and rendered into PDFs. While this approach would grant a lot of creative freedom and options in defining the templates, it would be quite time-consuming as every document would have to be created as a separate HTML page and styled to fit multiple printing paper sizes. Due to this complexity, this approach was also rejected.

4. **Generate Word documents based on Word templates**

   The final approach that was explored involves using Microsoft Word documents as templates. These documents would incorporate specific placeholder texts, which the templating engine would replace with actual data while preserving the document's

original formatting. This approach would simplify the creation of the templates while ensuring that anyone who understands basic Word functionality could modify them to their taste. A templating engine called docxtemplater supports this functionality and was therefore selected as the final templating solution for this thesis.

An example of a document template defined in the syntax of docxtemplater can be seen in Figure 5.5. Subsequently, Figure 5.6 showcases the final document once the template is processed with the actual data of the application.

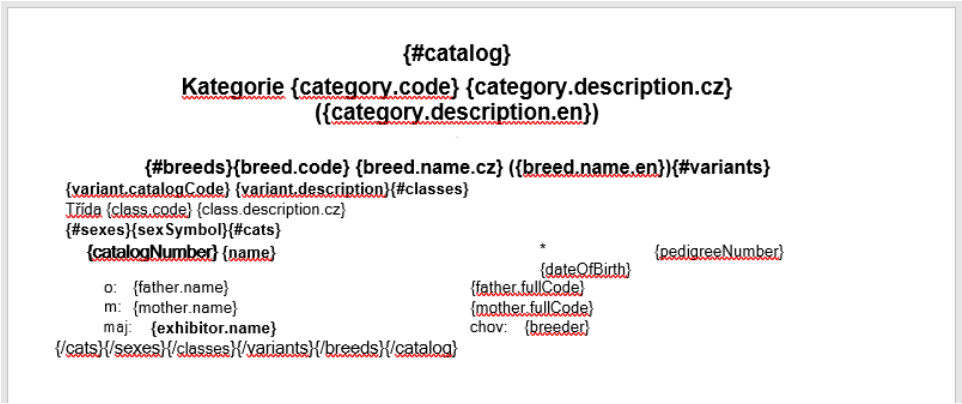
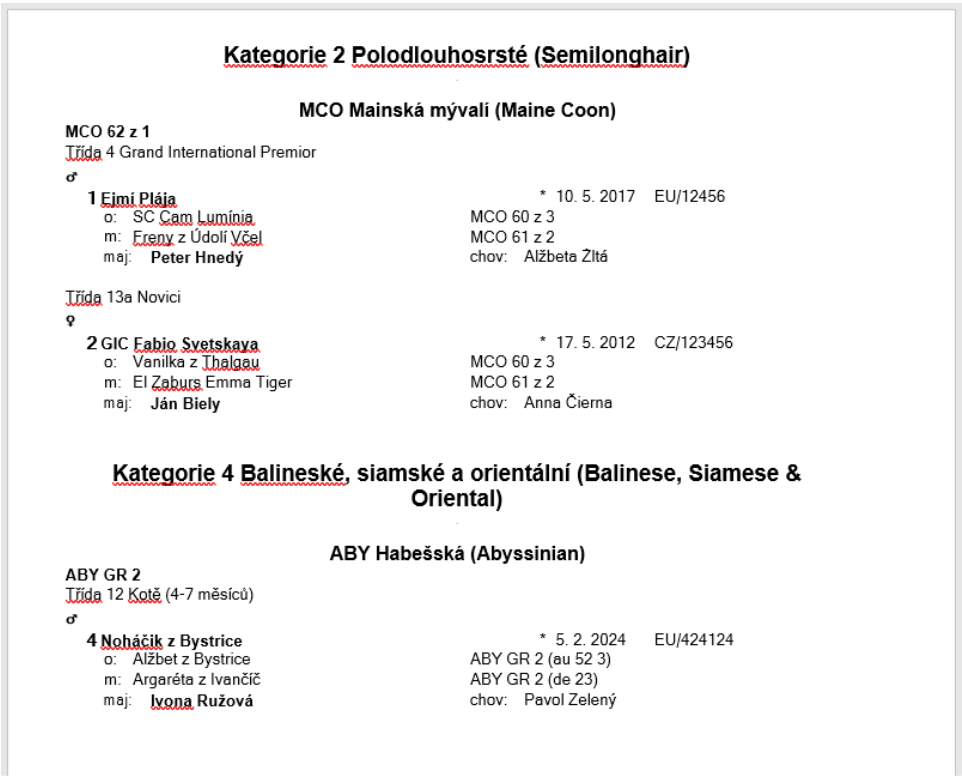


Figure 5.5: Word template of the show catalog written for the docxtemplater engine



Kategorie 2 Polodlouhosrsté (Semilonghair)

MCO Mainská mývalí (Maine Coon)

MCO 62 z 1
Třída 4 Grand International Premior
♂

1 Eimí Plája * 10. 5. 2017 EU/12456
o: SC Cam Lumínia MCO 60 z 3
m: Freny z Údolí Včel MCO 61 z 2
maj: **Peter Hnedý** chov: Alžbeta Žltá

Třída 13a Novici
♀

2 GIC Fabio Svetskaya * 17. 5. 2012 CZ/123456
o: Vanilka z Thalgau MCO 60 z 3
m: El Zaburs Emma Tiger MCO 61 z 2
maj: **Ján Biely** chov: Anna Čierna

Kategorie 4 Balineské, siamské a orientální (Balinese, Siamese & Oriental)

ABY Habešská (Abyssinian)

ABY GR 2
Třída 12 Kotě (4-7 měsíců)
♂

4 Noháčik z Bystrice * 5. 2. 2024 EU/424124
o: Alžbet z Bystrice ABY GR 2 (au 52 3)
m: Argaréta z Ivančíč ABY GR 2 (de 23)
maj: **Ivona Ružová** chov: Pavol Zelený

Figure 5.6: A template from Figure 5.5 processed with the actual data of the application

6 Conclusion

The goal of this thesis was to develop a comprehensive full-stack web application for assisting with the organizing of cat shows. The resulting application successfully meets the standards set by FIFe while enabling a user-friendly and ergonomic workflow for the show organizers. The backend of the application, powered by PostgreSQL, provides a robust and scalable foundation, while the utilization of a modern templating engine ensures efficient generation of the required documents. The application is ready to be deployed to production and tested during an actual large-scale cat show.

Naturally, there are a number of possible improvements that could enhance the functionality of the application even more. The major one that was already discussed in Chapter 4 is the utilization of a solver for assigning cats to judges. Further improvements could include expanding the application's capabilities to support mobile devices or standards of organizations other than FIFe. Lastly, providing further integration with the system for collecting applications for the show could also be beneficial.

Bibliography

1. *Stanovy ČSCH 2021* [online]. Český svaz chovatelů [visited on 2023-06-06]. Available from: https://www.cschdz.eu/svazove-predpisy.aspx.
2. *Sdružení chovatelů koček* [online]. Sdružení chovatelů koček [visited on 2023-06-06]. Available from: https://www.schk.cz/index.html.
3. *74 Years of Happy, Healthy Cats* [online]. Fédération Internationale Féline [visited on 2023-06-06]. Available from: http://fifeweb.org/wp/org/org_intro.php.
4. *FIFe Statutes* [online]. Fédération Internationale Féline, 2023-01 [visited on 2023-06-06]. Available from: http://www1.fifeweb.org/dnld/rules/statutes_en.pdf.
5. *CONSTITUTION OF THE WORLD CAT CONGRESS* [online]. World Cat Congress [visited on 2023-06-06]. Available from: https://worldcatcongress.org/wp/org_con.php.
6. *About CFA* [online]. The Cat Fanciers' Association [visited on 2023-06-06]. Available from: https://cfa.org/cfa-history/.
7. *About TICA* [online]. The International Cat Association [visited on 2023-06-06]. Available from: https://tica.org/about-us-2.
8. *THE WORLD CAT FEDERATION E.V* [online]. World Cat Federation [visited on 2023-06-06]. Available from: https://wcf.de/en/.
9. *FIFe Show Rules* [online]. Fédération Internationale Féline, 2023-01 [visited on 2023-06-06]. Available from: http://www1.fifeweb.org/dnld/rules/show_rules_en.pdf.
10. *Left Out in the Code: FoxPro and End of Life Migrations* [online]. InterSoft Associates [visited on 2023-06-06]. Available from: https://intersoftassociates.com/articles/legacy-systems/foxpro-and-end-of-life-migrations/#:~:text=FoxPro%5C%20and%5C%20Visual%5C%20FoxPro%5C%20were,and%5C%20software%5C%20designed%5C%20around%5C%20FoxPro..

11. REDMOND, Eric; WILSON, Jim R. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2012. ISBN 9781934356920.

12. GARCIA-MOLINA, Hector; ULLMAN, Jeffrey D.; WIDOM, Jennifer. *Database Systems: The Complete Book*. 2nd ed. Pearson Education, 2009. ISBN 9780131873254.

13. DB-ENGINES. *DB-Engines Ranking* [online]. Solid IT, 2023-06 [visited on 2023-06-01]. Available from: `https://db-engines.com/en/ranking_categories`.

14. ABADI, Daniel; BONCZ, Peter; HARIZOPOULOS, Stavros. *The Design and Implementation of Modern Column-Oriented Database Systems*. Hanover, MA, USA: Now Publishers Inc., 2013. ISBN 1601987544.

15. ABADI, Daniel; MADDEN, Samuel. *Debunking Another Myth: Column-Stores vs. Vertical Partitioning*. 2008-07. Available also from: `https://web.archive.org/web/20081204091753/http://www.databasecolumn.com/2008/07/debunking-another-myth-columns.html`. Archived from the original on December 4, 2008.

16. *What is a Document Database?* [online]. MongoDB [visited on 2023-06-06]. Available from: `https://www.mongodb.com/document-databases`.

17. *What is NoSQL?* [online]. MongoDB [visited on 2023-06-06]. Available from: `https://www.mongodb.com/nosql-explained`.

18. *What is a graph database?* [online]. Neo4j [visited on 2023-06-06]. Available from: `https://neo4j.com/docs/getting-started/get-started-with-neo4j/graph-database`.

19. *Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others* [online]. AltexSoft [visited on 2023-06-06]. Available from: `https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/`.

20. *Stack Overflow Developer Survey 2023* [online]. Stack Overflow [visited on 2024-02-02]. Available from: `https://survey.stackoverflow.co/2023/#overview`.

A Attachments

- source code: cat-show-organizer.zip
- documentation: README.md
- example data import file: input_data_example.json