

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies

FIIT-100241-103151

Ondrej Špánik

**Exploratory data analysis and
structuralization in database systems**

Bachelor thesis

Thesis supervisor: doc. Mgr. Monika Kováčová, PhD.

May 2024

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies

FIIT-100241-103151

Ondrej Špánik

**Exploratory data analysis and
structuralization in database systems**

Bachelor thesis

Study programme: Informatics

Study field: Computer Science

Training workplace: Institute of Informatics, Information Systems and Software
Engineering, FIIT STU, Bratislava

Thesis supervisor: doc. Mgr. Monika Kováčová, PhD.

May 2024

Annotation

Slovak University of Technology Bratislava

Faculty of Informatics and Information Technologies

Degree Course: Informatics

Author: Ondrej Špánik

Bachelor Thesis: Exploratory data analysis and structuralization in database systems

Supervisor: doc. Mgr. Monika Kováčová, PhD.

May 2024

Animal database providers like PawPeds, Omakissa or Sverak currently utilize limited software and hardware environments due to their financing options and commercial viability. These limitations force them to utilize webhosting services for operation of their evidence systems and management of datasets instead of utilizing more suitable offerings like virtual machines. Because societal and managerial interest exists in processed data, we apply explorational data analysis techniques to look for new ways to conceptualize and implement a more suitable evidence system with regards to modularity, algorithmical feasibility, user experience, data exchangeability and future-proofing.

Anotácia

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Študijný program: Informatika

Autor: Ondrej Špánik

Bakalárska práca: Exploratory data analysis and structuralization in database systems

Vedúci bakalárskeho projektu: doc. Mgr. Monika Kováčová, PhD.

Máj 2024

Poskytovatelia databáz zvierat ako PawPeds, Omakissa alebo Sverak momentálne využívajú limitované softvérové a hardvérové prostredia, kvôli ich finančným možnostiam a komerčnej životaschopnosti. Tieto limitácie ich nútia využívať webhostingové služby pre chod ich evidenčných systémov a manažment datasetov namiesto využitia vhodnejších ponúk ako sú virtuálne stroje. Pretože spoločenský a manažérsky záujem existuje v spracovaných dátach, aplikujeme techniky exploratívnej dátovej analýzy pre hľadanie nových spôsobov ako konceptualizovať a implementovať vhodnejší evidenčný systém s prihliadnutím na modularitu, realizovateľnosť algoritmov, používateľskú skúsenosť, výmennosť dát a odolnosť do budúcnosti.



BACHELOR THESIS TOPIC

Student: **Ondrej Špánik**
Student's ID: 103151
Study programme: Informatics
Study field: Computer Science
Thesis supervisor: doc. Mgr. Monika Kováčová, PhD.
Head of department: doc. Ing. Valentino Vranič, PhD.

Topic: **Exploratory data analysis and structuralization in database systems**

Language of thesis: English

Specification of Assignment:

Veľké množstvo štruktúrovaných údajov o zvieratách je uchovávané v samostatných databázových systémoch pochádzajúcich z rôznych zdrojov a uložených v rôznych (aj historických) formátoch, mnohé sú voľne dostupné na internete ako dátové sety, alebo je ich možné jednoducho získať formou web-scrapingu. Rovnako chovatelia a majitelia registrovaných zvierat počas ich života vytvárajú veľké množstvo dát, ktoré sú obvykle len evidované, ale nie sú následne žiadnym spôsobom spracovávané a analyzované a preto nie je možné na ich základe prijímať zodpovedné rozhodnutia. Zároveň ale existuje veľmi veľká spoločenská požiadavka takto spracované dáta prezentovať, prípadne používať v následných "manažérskych" rozhodnutiach. Analyzujte možnosti spracovania rôznych vstupov pochádzajúcich z animal datasetov tak, aby bola stále jasná väzba medzi viacerými zdrojmi údajov a daným zvieratkom, navrhните možné vylepšenia evidenčných systémov a naprogramujte rôzne typy exploratívnych analýz, ktoré dokážu takto evidované dáta sprehľadniť – ideálne formou webovej aplikácie. Riešenie vytvorte ako komplexný, modulárny a v budúcnosti ľahko rozširiteľný systém. Exploratívne analýzy by mali umožňovať jednak bližšie pohľady (EDA) na analyzovaný dátový set, jednak realizovať základné metódy data miningu a umožniť export údajov a analýz (xlsx, pdf) resp. iný podobný typ súboru, vhodný pre ďalšie spracovanie. Implementované riešenie overte na dodaných vzorkách údajov a navrhните ďalšie možnosti modulárneho rozšírenia systému.

Length of thesis: 40

Deadline for submission of Bachelor thesis: 22. 05. 2023

Approval of assignment of Bachelor thesis: 18. 04. 2023

Assignment of Bachelor thesis approved by: doc. Ing. Valentino Vranič, PhD. – study programme supervisor

Contents

1	Intro	1
2	Analysis	5
2.1	Initial considerations	5
2.1.1	What should be considered in EDA	5
2.1.2	Modular system expectations	6
2.2	Interested parties and their needs	8
2.2.1	Introduction to breeding stations and breeders' responsibilities	8
2.2.2	FIFe, standardization and potential in existing data	9
2.2.3	Breeders' needs and database providers' limitations	10
2.2.4	Exploring further options	11
2.3	Dataset readiness and algorithms	12
2.3.1	State and readiness of the inherited dataset	12
2.3.2	Expected data	12
2.3.3	Features of ideal database tables	13
2.3.4	Actual data and database tables	13
2.3.5	Migration considerations	14
2.3.6	Upkeep considerations	14
2.3.7	Dataset processing algorithms	16

2.3.8	Considered algorithms and their applicability	16
2.3.9	Webhosting feasibility	17
2.3.10	Future expansion and experiment potential	18
2.4	Database systems	19
2.5	Conclusion of the analysis	20
2.5.1	General project feasibility	21
2.5.2	Commercial viability	21
2.5.3	What will (not) be programmed and technology stack	22
3	Conceptualization	25
3.1	Project goals and requirements	25
3.2	Obstacles	27
3.3	Model View Controller pattern	29
3.3.1	Laravel environment	29
3.3.2	Data models	30
3.3.3	Data Modeling	31
3.3.4	Views	33
3.3.5	Controllers	34
3.4	Major changes	35
3.5	Algorithms and Data Processing	35
3.6	Laravel	36
3.6.1	Illuminate	37
3.6.2	Eloquent	37
3.6.3	Laravel Breeze	37
4	Implementation	39
4.0.1	Inherited Project	39
4.0.2	Migrations	40

4.0.3	Routing and APIs	45
4.0.4	Controllers and functionality	47
4.0.5	Cat registration	49
4.0.6	Search	49
4.0.7	Editing capabilities	49
4.0.8	Birthdays	54
4.0.9	Family trees	55
4.1	Algorithms	57
4.1.1	Inbreeding	57
4.2	Data processing	61
4.2.1	CSV import	64
4.3	Docker image	67
4.4	Testing	68
4.5	Expansion potential	69
5	Conclusion	71
6	Resumé	75
6.1	Explorácia dát v databázových systémoch	75
6.1.1	Analýza	75
6.1.2	Konceptualizácia	77
6.1.3	Implementácia	78
A	Planning	89
B	User Documentation	103
B.1	Setting up webhosting access	103
B.2	Webhosting deployment	104
B.3	Accessing common functionality	104

B.4	Creating a new user account	105
B.5	Accessing Backups and Cat Registration functionality	105
B.5.1	Setting up an administrator account on a Dockerized build .	105
C	Technical Documentation	107
C.1	Folder hierarchy	107
C.2	Dockerfile contents, build and launch order	108
C.2.1	Ports	109
C.3	APIs and Routes	109

Chapter 1

Intro

During the analysis chapter we research deeper meanings between keywords which were assigned including potential impact. [15]

In terms of this Bachelor's thesis, there is an inclusion of understanding Exploratory Data Analysis in general and also how it relates to this project in particular. Afterwards we also consider modular requirements in how this project could be conceptualized, especially because modularization was one of the major requirements while giving space to other areas for future expansion.

Highlight of the implementation is on handling datasets in Laravel, particularly in relation to cats. As such we also consider people and organizations related to handling of cats and cat related data in prior chapters.

Breeding stations, cat registries and other cat database providers currently utilize software and hardware environments limited in nature to webhosting services [45] [42] [53] and as such PHP in particular. This limitation is considered in the analysis, conceptualization, implementation and future proposals. After assuming potential viability of such a limited solution, we proceed to analyze priorities of

breeders and owners, then write down options as to what could be conceptualized and implemented within the predefined constraints.

This thesis is approached with the understanding, that societal interest in presenting processed data related to cats does indeed exist, and not only that, but potential for use of such processed data in managerial decisions exists as well. [44]

Understanding the environment we're dealing with and impact this work could have, we proceed to analyze different parts that will fit into such a system including its inputs and outputs in relation to Docker and Laravel in particular. This will include existing partially structuralized datasets related to animals, but also transformations necessary for additional structure and clarification within them.

As this thesis doesn't involve direct cooperation with cat database providers, the data would have to be obtained from freely available databases by means of web scraping. This part was thankfully already covered by a separate team project and as such, thanks to the management by our project leader, we will explore and utilize a pre-existing dataset instead. [21]

Intentions of breeders, owners and other users are also important, as they weigh heavily on which algorithms should be utilized, which parts of the system should be given a higher preference and be more optimized than others. The goal in this regard is to create a system that is efficient on resources due to already mentioned limitations, but also fast and modular. Speed will be considered mainly in terms of inbreeding, migrations, caching and other data-related operations excluding some backup functionalities.

This thesis is divided into multiple chapters, mainly analysis, conceptualization, implementation, testing and conclusion. Conceptualization and implementation

will cover creation of a system, which tries to improve over existing solutions. This part will consider implementing EDAs to better highlight the true potential behind obtained datasets. Additionally, data is expected to be exportable in a modern format ideal for exchange between different instances of the given software. Testing and conclusion will cover improvements and shortcomings of our implementation.

Chapter 2

Analysis

Before we can conceptualize a solution to the problem at hand and implement it, we need to consider analyzing not only the most important scopes from afar, but also details they entail. [15]

Once we get the initial considerations out of the way, we can proceed to delve deeper into each discussed topic and truly understand their possibilities and implications of a newer solution. Afterwards we will be able to come to a sound decision about chosen technologies, recognized obstacles and understand better how to proceed in the conceptualization phase. [15]

2.1 Initial considerations

2.1.1 What should be considered in EDA

The Exploratory Data Analysis field was brought to light by Tukey [59], which eventually resulted in development of software like R studio, preceded by the S programming language [6]. The term "Exploratory Data Analysis" was coined by

Tukey, because he believed, that there was too much focus on hypothesis testing [58]. The EDA process usually involves analyzing datasets and pointing out the main characteristics of the analyzed datasets [58], often by utilizing data visualization methodologies, e.g. displaying data from tables inside charts. The main point is to find out additional meanings behind data beyond formal modeling and as such it isn't meant to utilize traditional hypothesis testing. [58]

2.1.2 Modular system expectations

While talking about modular approaches, we have to consider how modularity impacts different parts of a web-based evidence system. On the frontend side of things, the field of modular web design [37] describes visual characteristics that allow for modular workflow inside of a website, e.g. displaying data inside visual blocks that can be moved and adjusted to user's liking. More importantly for us however is to modularize the system underneath, that means not necessarily the frontend, but rather the backend.

A truly modular approach can do dynamic module loading and unloading while in runtime [49]. Such implementation shows, that when a certain part of backend crashes. management software will restart it. Such approach helps the system to be more robust as each module's failure won't cause others to fail [49], which can be achieved by utilizing multiple services or Docker containers. Additionally the system can become more performant if utilizing the service approach as services can then be distributed over different nodes, creating a kind of distributed system [4], commonly referred to nowadays as a cluster.

On top of that, if we're talking about a cluster of services, then at the same time a reverse proxy will likely exist to move the traffic generated by users to a separate instance of a certain service (or module) which hasn't crashed. This

however is a backend modularization approach on a more complex level than the one we're aiming for, but serves to demonstrate what is quickly becoming an industry standard [10].

Our interpretation of a modular system is built around commonly used practices related to modularity, a design principle allowing for optimization of modules independently of others [49]. This doesn't necessarily mean that conceptualizing or implementing such a system is a difficult task. Nowadays it's found to be such a common practice, that many developers may not even realize they're working with or developing it [26].

Websites as we know them today are meant to be split into a backend managed by servers and a frontend displayed on users' side. In the traditional sense these sites used to be programmed, and in many cases still are, using languages and frameworks that bound the frontend and backend together into a single codebase or a single deployment, e.g. Wordpress [38] [43]. Nowadays a more modern approach would be to create endpoints on the backend side and connections to them on the frontend side, in other words decouple the two. By utilizing this we could help introduce more modularization into the system, however this is not necessary in the case of Laravel. Such decoupling could allow for multiple different backends or frontends to work concurrently and can be achieved by utilizing APIs, or in case of more complex systems, which need to synchronize tasks between each other, yet have to operate asynchronously, we would consider utilizing message queuing services (e.g. ActiveMQ or RabbitMQ) [3]. An implementation that only considers APIs should suffice, especially due to limited resources on the side of cat database providers.

Our goal is not to create a distributed system alongside its backend API endpoints and connections between them, but rather a unified system underneath the banner

of Laravel, which will manage these routes for us and allow for the application of a MVC pattern for easier management of a simpler system. This system could still be decoupled in the future by introducing Docker in a more deeper way than we intend to, including its docker-compose feature.

2.2 Interested parties and their needs

We previously mentioned breeding stations, breeders and owners of animals. We also mentioned societal and managerial interest in the outputs of a system that this thesis could provide. These are considered to be the interested parties in this chapter, with main focus being put on breeding stations' and owners' direct needs.

Because there are multiple parties involved and also because the bureaucratic processes [39] around animals are present in most of developed world, we are presented with international standardization. This chapter will firstly introduce the responsibilities of breeders, followed up with standardization, breeders' needs, limitations of existing cat database providers or databases themselves and exploration of further options.

2.2.1 Introduction to breeding stations and breeders' responsibilities

A breeding station, more commonly referred to as a breeding cattery is a building or a reserved space within a building in which cats are bred. These breeding stations can range from reputable establishments that keep proper track of data to less reputable establishments. In many cases we're simply talking about sheds next to family houses. One of the main purposes of a breeding station is often

to sell pet cats and make profit. Because cat breeding or any form of breeding is a process that could get out of hand, licensed breeding stations exist. These licensed stations are heavily regulated by government legislation and it is expected that they comply with the Code of Ethics [5] alongside other guidelines related to given breeds. The Code of Ethics refers to breeding healthy and happy kittens in a healthy environment with veterinary care, being honest during sale, disclosing disease during sale, guaranteeing contact with the eventual owner and other criteria. [5]

2.2.2 FIFe, standardization and potential in existing data

With the data for implementation being centered around cats, relevant standards specifically for cats do indeed exist. Cat-related standards are overseen by FIFe. [30] FIFe is an association present within the biggest international confederation called the World Cat Congress. [50]

FIFe is an international federation of cat registries that, alongside other interests, aims to cross boundaries between individual cat registries and thus help in cross-registry activity. As of right now there are 41 cat registries within FIFe, mostly one per country. [30]

Each cat database provider offers its own database of cats [45] [42] [53] alongside relevant data such as the usual identifiable information like name or breed, but also other valuable information pointing to who the parents are, which can then be easily linked with previous generations thanks to relational databases. [11] This way we can generate a family tree that allows us to properly apply techniques of data analysis, such as finding out how much inbreeding is present for a given cat. [7] [dbrelatonalanalysis]

2.2.3 Breeders' needs and database providers' limitations

The data published by cat database providers is used by individuals - breeders and owners of cats - in order to find out more information relevant to the health and breeding possibilities [52] for a given cat, but also has potential for statistical research, e.g. figuring out characteristics for population of cats in a given country or understanding overall amount of cats per breed out there.

We have identified that cat database providers are limited in terms of what they can achieve with the data they have at hand. Currently cat database providers tend to utilize simple webhosting providers and simple data management solutions that fall short when it comes to any additional data analytics and statistical research. Often we simply talk about a simple PHP based solution with a corresponding MySQL database, apparently without modularity. A system similar to this can be observed at Pawpeds [45], Omakissa [42], Sverak [53] and other cat registries. That however doesn't mean that there aren't registries that consider better approaches, as is the case with the Pedigree online database [46], which provides reports and additional features such as filters optimized for specific kinds of attributes.

Because these systems have been operating for years, they are lacking in features which internet users would appreciate nowadays, such as a proper interactive search (e.g. Elasticsearch [65]) and chart visualization subsystems (e.g. Kibana [66], Grafana [24]). Such features could not only improve user experience, but also realize certain possibilities in terms of data analytics and research. Deploying such systems seems to be unfeasible due to limitations of webhosting providers or cat registries in general. [1] [2] As such we will attempt an implementation relying on existing technologies provided by webhosting providers (PHP and MySQL or MariaDB) [67] [63] while taking resource usage limitations into account. Additionally, we will look for alternative approaches that became possible in recent

years, e.g. utilizing WebAssembly [61] and running parts of the code on client side on-demand with Javascript.

The goal of our implementation is not necessarily to replace the existing implementations offered by cat registries, although that would be the ideal scenario. Our implementation is meant to allow usage of a scraped dataset for independent data analysis and similar purposes.

2.2.4 Exploring further options

From our understanding and analysis there are two options we can think of, that could be feasible when it comes to data handling between individual cat registries or database providers.

First proposition would be an International Research Database under FIFe or World Cat Congress, which could unify databases under a single organization by pulling data from individual providers or having data pushed onto the database from them, alternatively completely merging and discarding them. Options other than a merge would likely result in a bureaucratic or communication burden, however merging the databases would remove sovereignty over data and cause monopolization. It would be possible to provide paid access to this database and tooling for data analytics and statistical research. Such option could potentially become financially self-sufficient through paid services, but would require initial funding. [1] [2]

On-demand dataset handling for individual researchers and API standardization proposition is easier in theory, but harder to actually implement in real world. Because this is a non-standard way of operating, it would be classifiable as an experiment, requiring someone to create necessary standards for it to work. Such a system, with each part operated by a different database provider would contain

constraints causing a worse overall performance than the first option proposed above.

Further exploration is outside of this thesis' scope, however it would be interesting to see at least one of these ideas explored more thoroughly.

2.3 Dataset readiness and algorithms

2.3.1 State and readiness of the inherited dataset

In order to commence, test and implement a proper solution that can be utilized by researchers, breeders, cat owners or cat database providers, we first need to have an access to a dataset that the solution will be tested on and built around.

Understanding the individual attributes found within the dataset can help us establish which data can be used for data analysis, which has to be derived and which is missing in general. [15] Such transformations would be applied ahead of time to the dataset and lessen the burden on actual code implementation.

The dataset used in scope of this project was obtained from the project leader at FIIT STU and contains cat records of a Slovak cat registry. [21] The data will be manually processed beforehand using well-known office tools, because alongside well-kept records, there is a need to transform the data into a proper format for easier implementation.

2.3.2 Expected data

A proper dataset from a cat database provider should contain records of individual cats including their identifiable characteristics alongside links to the breeding station, breeder, owners and parents. The identifiable characteristics should comply

with well-known standards [20] like the EMS code [57] or color coding [57].

2.3.3 Features of ideal database tables

Ideally a history management solution could be seen in the dataset as well, e.g. an additional table solely for monitoring changes in other tables. [55] On top of that dynamic structures (e.g. JSON field) could be used to extend an otherwise static database model. These dynamic structures wouldn't provide efficient access for algorithms [47], however we do happen to believe, that they would make the system future-proof and reusable for use cases that may not even be considered nowadays. Administrators of such a system could simply add a new field next to existing ones, revert changes or browse through them.

Other than user experience related to data management, we also have to consider impact on algorithm execution. Ideal database table structure promotes low query difficulty behind sought-after algorithms [25]. This optimization however takes a lot of manual work e.g. migrations and testing, depends heavily on chosen algorithms, which depend on actual use cases of users.

Additionally we can talk about special features of databases for better data management such as indexing. These features might be useful to consider in certain parts of implementation, such as in relation to algorithms in order to significantly boost performance where applicable.

2.3.4 Actual data and database tables

The table from the inherited dataset [21] contains approximately 40 000 records of individual cats. Previously a dataset of 5 tables was considered, however a more useful solution consists solely of the Slovak dataset importable after first run using the appropriate interface.

2.3.5 Migration considerations

We will establish the following naming for the kinds of migrations that will be considered: generative migrations, optimization migrations and conversion migrations. In practice we will only convert CSV on import and not as a part of these migrations. Migrations are done ahead of runtime operation and import as a part of runtime.

A conversion based migration should only involve obtaining a database dump in the form of SQL language queries, then adjusting the queries so that they will be compatible with a different RDBMS instead of the source RDBMS. [12] This migration can not be done natively in Laravel.

A refactoring or optimization based migration will basically be a standard migration, where structure of tables and columns is adjusted for better performance [35]. This migration can be done natively in Laravel. It may need to be applied repetitively for optimization, because of algorithm tuning or similar tasks.

A generative migration will be considered as creation of tables necessary for the implemented solution to function at all. These will be core database table structures and can be done natively in Laravel.

2.3.6 Upkeep considerations

In order for the resulting implementation to be usable long-term, a precondition was set, that it has to support one or more update channels, which will allow it to enroll new records from external sources.

When considering database dumps, we have to point out, that certain hosting providers don't provide access to database dump/restoration [56], and thus the dataset would have to be non-binary (consisting of SQL queries and no binary

data). Alternatively an API [28] can be utilized as well, but in this case on the side of external database providers, whose databases are obtained. If such API doesn't exist, or rather the database provider isn't capable or willing to implement it, then we don't have much of a choice. In other words, as databases from different providers register new cats, these changes have to be captured, ideally without re-scraping the entire dataset.

Running a web scraper time and time again wouldn't be efficient in regards to resources and time. We can avoid this by having an access to the external database's list containing only links to the existing records alongside their modification time. This list would ideally be obtainable using an API [28] for proper automation, however its very likely that database owners don't have resources or reasons to implement said API. Alternartively, an export using a special SQL SELECT could be performed by the administrators of the given database. If this is not possible, e.g. we're trying to obtain the updates for the given dataset without interacting with dataset's owners, then we can once again utilize scraping. In such case we can achieve higher efficiency by scraping only a list of records on the database's website, which must contain and ideally also be sorted by dates of modification.

This way we would only need to update the records with modification time lower on our side than the side of the external database.

During an update of our database a situation could occur, when newer data from external sources has lower date of modification than our database in case users of our evidence system updated the record with their data, but haven't fetched updates from said external sources beforehand. To handle such issues we propose to save two dates of modification for each record, the original date of modification and the local date of modification. In cases where table layouts between datasets differ, we would also have to utilize multiple databases, one per dataset. The extent

to which this will be implemented is questionable due to the depth of this problem we have identified so far and possible issues this may additionally incur.

2.3.7 Dataset processing algorithms

In this section we explore the applicability of considered algorithms, dividing them into possible and unlikely to be possible in a useful context. These considerations will be made in regard to the webhosting's "Fair usage policy" [56] and similar rules. Afterwards we look at webhosting feasibility overall and then propose options for future expansion alongside unexplored experiments allowed by modern technologies.

Any implementation or modification of processing algorithms could require database restructuring and therefore a migration. [35] Ranking of algorithms by priority is not considered, because this thesis doesn't take into account consulting interested parties (e.g. breeding stations, breeders, owners) directly regarding which algorithms should be considered or preferred.

2.3.8 Considered algorithms and their applicability

We have identified that possible algorithms within the scope of this project can be certain graph and text-processing algorithms. [16] [13] One such graph algorithm is one useful for calculating inbreeding. [8] It aggregates data that is already present in the live database and should only require simple expressions and SQL queries, as such we currently consider its implementation possible. Additionally, other graph algorithms can be implemented in a limited scope such as TSP [22], which can find whether two cats are related among other use cases.

Another possible algorithm, or rather a series of algorithms is related to text processing [13]. A simple solution would involve recognizing segments of a flexible

text content and assigning them as attributes to the given record. Such recognition can be as simple as utilizing regular expressions [23] to localize standardized codes. It could also be used to recognize foreign words by comparing them against a phrasebook, which would prove to be a bit difficult due to how human languages work. Lastly, the most difficult algorithm that we considered would be involved in approximating whether any two records are actually meant to be the same record, but misspellings or utilization of different languages or writing scripts had caused these two to not match using simpler techniques like regular expressions. Such algorithm can be implemented using multiple different approaches. [36]

Algorithms that are unlikely to be possible in a useful extent mostly involve high difficulty or platform incompatibilities such as machine learning algorithms. These would have to be bound to webhosting related technical limitations such as processor limits or fair resource usage policies. Machine learning could help in clustering individual records (cats) together based on certain characteristics. As for changes to the database structure, when we consider more difficult algorithms such as the last two text processing algorithms or machine learning, we also have to consider adding one or more fields to each related database record to remember output for the given algorithm, e.g. matching precision within a given threshold.

2.3.9 Webhosting feasibility

Webhosting providers offer cheap hosting options to customers with a few preconditions. One such precondition tends to be, that the customers will abide by limitations like maximum CPU execution time, bandwidth limits, memory usage limits and other limits, which are usually covered in a document called "Fair usage policy" [56]. Another precondition could be, that even if within resource limits, its likely that webhosting providers wouldn't be particularly keen on hosting a social

network or a complex application.

A potential solution for resource limits could be found in client-side experiments with more modern technologies, which we cover in a later section. Not utilizing a webhosting would mean incurring higher costs, but also much better data processing capabilities. [60]

2.3.10 Future expansion and experiment potential

We have noticed that over the last few years multiple technologies have sprung up, that can be run on the client-side and avoid overusing server resources, or in this case webhosting resources. We will introduce these below.

WebAssembly [61] allows for execution of sandboxed binary code inside of a web browser and can be utilized not only thanks to its performance characteristics, but also because it makes execution of code outside the capabilities of Java Script possible.

Other than WebAssembly, technologies for communication and fast communication between external components exist as well. In the slower department we have the usual AJAX and fetch requests. In the faster department we have WebSockets interfaces. [18] Combining WebSockets with a modern approach towards high-performance memory-safe computing utilizing Rust programming language [54] could enable us to create a client-side data processing unit.

The idea is that the datasets processed by the hosting services could be further processed locally after being retrieved without passing through the hosting's servers again. An alternative of sending and retrieving the data using an API is also possible, but would strain resources on the server side if saving the processed data is not useful, such as when the processed data is rarely used.

Furthermore, if we consider our analysis of modular system requirements, especially the part related to distributed systems, then we could consider an experiment where multiple webhostings are clustered together, creating a pool of resources [4].

Conducting these experiments is outside of the project's scope, so neither conceptualization nor implementation will refer to them, however we still included this chapter as they likely have potential for organizations and individuals with limited funding that have to rely on webhosting services.

2.4 Database systems

Offerings like MariaDB or MySQL provide us with a multitude of data types, with keys and foreign keys alongside other options, thanks to which we can define a proper database structure. It is up to us to consider which models will be implemented and how they will interact with each other. In order to satisfy this requirement we will once again look at the MVC pattern and also how models interact with other parts of the system. Migrations will be a necessary tool to achieve the intended database structure and will act as a gateway for models to link to real data stored in the database. Certain actions on the frontend side such as search queries will also require a database connection and a SQL query to be present within respective controller functions, although in such circumstances a good system will utilize ORM to safeguard SQL against attacks such as injection bypass. Once again Laravel comes at the top of the list providing a good MVC experience with tooling necessary to establish database relationships (e.g. one to many and many to one), models with migrations, separate database queries and much more. Laravel supports both MySQL and Postgres databases out of the box.

Conceptualization We have considered the use of MySQL database as a primary way for the backend models to have a way of storing information. This decision was made by comparing existing offerings of webhosting providers and coming to a conclusion that using MySQL or MariaDB has wider support than simply using Postgres. While it is true that Postgres would provide us with efficiency unparalleled in the opensource world [x], MySQL remains dominant in the webhosting space. There are solutions available for conversion between different database kinds such as DBConvert [<https://dbconvert.com/postgresql/mysql/>]. Using MySQL will provide us with a decent user experience.

2.5 Conclusion of the analysis

As a part of the analysis we have gone over scopes of the project and the details they entail, with importance to obtain understanding as to what the possibilities and implications for this project could be. We came to an understanding as to what EDA means, handled analysis of datasets, considered limited financing options, societal interest and managerial decisions. We have analyzed different parts of the system including their inputs and clarified relevant details in regards to combining database systems, utilizing web scraping and an inherited dataset. We delved into what algorithms should be considered and preferred based on users' intentions. We have described our approach to a modular system, which relates to decoupling of frontend and backend by utilizing APIs. Furthermore, responsibilities, needs and limitations of interested parties were introduced alongside standardization and potential present in the dataset. Analysis of dataset proceeded in three stages - expectations, ideal state and actual data, then concluded with migration and upkeep considerations. After we understood what data we're dealing with and who the users are alongside various constraints, we were able to offer solutions in

the form of algorithms, rechecked feasibility of the project as a whole and proposed experiments that seem to have certain potential.

In this section we will summarize feasibility of the project and finally decide which technologies will be used during the conceptualization and implementation chapters.

2.5.1 General project feasibility

As described in the "Webhosting feasibility" section, we do indeed see the project as feasible and therefore will proceed with conceptualization and implementation.

2.5.2 Commercial viability

A question of comparison between usual VM approach and webhosting approach arose, especially considering that any commercial solutions would likely consider a VM approach instead of utilizing anything webhosting related. Here we have to consider the reality, that interested parties or rather cat database providers don't have access to required funding and that their operations aren't commercialized to such extent. Additionally, key to this thesis was to consider the current environments that cat database providers operate with, e.g. webhostings and not necessarily explore other options. Therefore we purposefully haven't explored alternatives further, as we considered exploring the possibilities of utilizing limited resources presented by webhosting providers to be an interesting and useful challenge.

But to answer the actual question as to whether this project is commercially viable, we have to look at the reality of how related institutions are currently ran and understand that their interests don't lie in commercialization. If we don't

consider this important finding, then indeed there isn't much viability as we could just utilize virtual machines and proper dedicated resources, otherwise however the viability in question doesn't really have a commercial motive to begin with and as such the question can't really be answered as of now within the scope of this thesis.

2.5.3 What will (not) be programmed and technology stack

Finally, now that the analysis is coming to a conclusion, we can make sound decisions about which technologies to include or exclude. These technologies will be explored further, combined, partially tested and sometimes possibly replaced as a part of conceptualization, while this section simply serves as a starting point.

Technologies will be split based on MVC, RDBMS and separately any utilities that may need to be programmed for easier development and maintenance. When talking about MVC pattern, we will consider frontend (view) and backend (model and controller) to be a part of the same package.

Backend will be mainly programmed in PHP and SQL, while frontend will see HTML intertwined with PHP backend and CSS. While we did consider a headless API approach, which is more modern, we ended up utilizing the MVC pattern instead, which provides us with a more stable environment immune to "spaghettification" of the codebase. Relevant frameworks alongside their feasibility within a webhosting environment will be described during the conceptualization chapter.

Connection between backend and frontend will be done using APIs and controllers defined in the MVC pattern as a part of a semi-standardized system, which will enable us to deliver a complete package without the need to deploy multiple different parts separately. We will not be programming a GraphQL API nor a separate

or headless REST API and will solely focus on maintaining a singular project codebase.

As for frontend, we already have experience in utilizing the fully released SvelteKit, which happens to be much simpler to deal with and more performant than alternatives like React or Vue.js, however we will consider a simpler Bootstrap solution instead and focus more on delivering a stable product which is easier for upkeep than a combined solution. Our intention is to stay next to the MVC pattern as a golden standard of patterns when it comes to implementing a combined solution of frontend and backend as a part of the same package.

There are multiple PHP frameworks which fit the situation, starting with Laravel, the most well-known and supported of these.

With initial considerations for the technology stack out of the way, we can now properly proceed to the conceptualization chapter.

Chapter 3

Conceptualization

3.1 Project goals and requirements

Number	Goal	Feasible technologies	Potential obstacles
G01	Create a modular system for managing cat data	Docker, Git, multitude of frameworks	- Stability of Docker- Selection of a good PHP framework (Laravel, Symphony, CodeIgnite) or alternative solution
G02	Analyze a dataset of cat information	Postgres, SQL, CSV	- Dataset integrity and cleanliness- Size of the dataset
G03	Consider the feasibility of using web hosting resources	PHP + Apache + SQL, WebSupport.sk, Wedos.cz	- Webhosting compatibility with given technologies- Database size limit

G04	Propose potential experiments for future expansion	None, just a proposal	None
G05	Develop a backend and implement an API for frontend communication	PHP, REST API	- Modularization- API endpoint integration with frontend including adjustments
G06	Integrate the frontend (the rest of the frontend part of the project is done by a 3rd party in a separate Bachelor's thesis)	Docker	- Changes in environment variables- Version synchronization between backend API and frontend integration of it
G07	Implement data processing algorithms for dataset management	PHP, MySQL, SQL	- Processing power offered by webhosting services - Postgres capabilities- Limits of available solutions
G08	Allow for the import of data in the form suitable for further processing	SQL, CSV, MySQL	- Modularity- Integrity before and after import or export
G09	Verify the implemented solution on provided data samples	PHP + Apache + SQL, WebSupport.sk, Wedos.cz	- Depth, reproducibility and provability of tests
G10	Propose further possibilities for modular extension of the system	None, just a proposal	None
G11	Allow for a comprehensive, modular, and easily extensible solution for future development	Markdown, Code documentation	- Interest in writing documentation- Comprehensive view

3.2 Obstacles

PHP framework selection: Selecting the right PHP framework is important for streamlining the development process and ensuring maintainability. Even though there are simpler and more modular solutions available, such as the *phpcrudapi* script[48]. These solutions are also easier to run, but not necessarily expand upon[17] compared to more complex frameworks like Laravel, Symfony or CodeIgnite, which are preferred. The learning curve, community support, performance, and available components were also looked at when selecting a solution.

Dataset integrity and cleanliness: This will also be vital for obtaining accurate results. The provided dataset has undergone data preprocessing techniques, such as data validation, removing duplicates, handling missing values, and outlier detection, before conducting exploratory data analysis.[14] This way we also ensured the data is consistent, accurate, and free of errors that could skew the results of the analysis. It is recommended to follow up with the 3rd party, as they worked on the scraper to address certain issues with the dataset. [62]

The size of the dataset will influence our choice of data storage, processing methods, and web hosting capabilities.

Web hosting compatibility with our chosen technology stack should be verified. We will need to verify that our chosen web hosting service also supports these technologies: Apache (otherwise in case of Nginx, *.htaccess* would have to be translated), PHP, and MySQL or MariaDB (in case of Postgres certain functionalities would have to be translated). Alongside that, also ensure that the hosting service provides the necessary resources for our application, such as storage, bandwidth, and processing power, check if it imposes limits on the database size and provides sufficient processing power to handle our data processing needs.

MySQL will be a suitable choice for data analysis applications due to its widespread features [40] such as its availability on major webhosting providers and efficiency when it comes to Laravel.

Considering the **limitations of our chosen technologies** and tools will help avoid potential bottlenecks in the development process. For example, PHP may not be the best choice for computationally intensive tasks, and alternative solutions, such as Python, may be more suitable for advanced data analysis, however those are not made available by webhosting providers and thus can only be planned to be run locally during build process. [9]

Developing modular code will ensure that our application is maintainable, scalable, and easily testable. We will need to use design patterns and principles, such as SOLID and DRY [32], to create a modular architecture.[19] Modularity will help ensure that the code is easy to maintain and update, making it easier to scale the application as needed.

Testing must also be done as well. We will need to ensure that our tests cover various scenarios, are reproducible, and provide provable results. Main testing scenarios should cover the overall integrity of the system, reproducibility and multiple deployments on a webhosting provider.

- **Stability of Docker:** While Docker containers offer many benefits, such as consistent deployment and reproducibility of results, there tend to be stability issues - Docker crashing, containers not starting, changes in package versioning from external repositories.[68] We need to keep these potential issues in mind when working with Docker, and to have a plan in place for dealing with any issues that may arise. As modern build systems rely on external repositories, there is no guarantee that the software will always run, however clear documentation helps with maintenance of such issues in the future.

Proper documentation will also be required for maintaining and scaling our data analysis solution. It immensely helps developers such as my colleague to understand the project, its structure and also serves as a reference for future improvements.

3.3 Model View Controller pattern

MVC is a popular design choice when it comes to designing web applications [64]. The goal is for Controller to be used by the user while seeing the View and for Model to be manipulated by the Controller while updating said View. This is the approach taken by the popular Laravel PHP framework which will be used during the implementation of this project.

Other than the application of SOLID and DRY principles[32], we will also look at how we can implement MVC pattern [34] and work with it in order to ensure a system that is prone to "spaghetification". We can do this by using a framework that has MVC [34] at its core such as Laravel.

This framework separates the application into three parts - the model, the view, and the controller. This separation helps in organizing code, making it easier to maintain and modify, especially in large projects with unstructured code [34] [29]

3.3.1 Laravel environment

In case of Laravel MVC is distributed across its file hierarchy [**laravel-mvc**] as follows: - Models can be found at */app* - Views can be found at */resources/views* - Controllers can be found at */app/Http/Controllers* with their respective routes available at */routes/web.php* This way we can easily pinpoint exact approach

taken during development and extend on it without having to resort to alternative approaches, which makes the use of a pattern such as MVC [34] the ideal approach.

The chapter "**Components and Folder Hierarchy**" in this thesis provides a comprehensive overview of the project's directory structure and its various components.

The project's **components table** then proceeds to provide information on the current state of each component, including whether it is complete, in progress, or cancelled. This will serve for a clear understanding of the state of each component and helps to ensure that development resources have been allocated efficiently. Cancelled components, for example, aren't necessary to the project and development resources have been moved elsewhere.

There is also an outline of potential **future expansion** components. These components are not currently part of the project but may be added in the future to support additional functionality or new use cases. This forward-looking approach helps ensure that the project can easily adapt to changing requirements or new opportunities.

3.3.2 Data models

We will need to implement the following models: - Cat - Breed - Ems - User These three models will be sufficient to contain all major information across the web application developed using Laravel framework.

Additionally, we can consider implementing these models as well: - CatMetadata - Couple - Upload trait

3.3.3 Data Modeling

When we embarked on the journey to represent cat populations and their relationships, several data models were considered, including relational, graph, and hierarchical databases. After a thorough evaluation, we found that the relational database model provided the best combination of efficient querying and data manipulation, especially for analyzing complex relationships within the cat populations.

To transition our raw data into the selected relational model, we meticulously followed several steps, starting with data cleaning, normalization, and finally, schema design. Docker, Apache, PHP, MySQL were our chosen technological allies, providing an easy-to-deploy environment that smoothly handled the data.

Intention	Description
Ensure dataset consistency, accuracy, and reliability	A comprehensive dataset management strategy will be adopted to ensure that the dataset remains consistent, accurate, and reliable. A meticulous version control system will be used to track changes and data storage norms that emphasize secure and easy access will be implemented. Access controls should be put in place to maintain data integrity and prevent unauthorized data access.
Handle data in a modular system	A modular approach will be adopted in handling the data, which will enhance maintenance, scalability, and collaboration. Each module will be encapsulated, allowing only necessary interfaces to interact and keeping the system's inner workings secure. Docker, Apache, PHP, and MySQL will be instrumental in supporting the modular system, each contributing its unique capabilities to form a cohesive, efficient whole.
Select appropriate data processing algorithms	Relevant algorithms will be carefully selected based on their relevance to the specific task, and will be subjected to thorough pre-processing steps to ensure optimal results. The selected algorithms will be instrumental in uncovering patterns, relationships, and trends within the cat population data.
Manage dataset changes effectively	In the face of changing datasets, a proactive and multifaceted approach will be taken. Data validation, cleaning, and transformation steps will be implemented to manage dataset changes effectively. The Docker, Apache, PHP, MySQL, and Laravel solution will play a crucial role in maintaining data consistency.

Bridge data and algorithms with views	Efficient use of views will be made to link the cat population data with the chosen processing algorithms. Views will significantly simplify data processing tasks, enabling more efficient use of resources.
---------------------------------------	---

3.3.4 Views

We will consider the following views depending on routes and individual pages: -

backups

– forms

– index

- cats

– partials

— parent

— edit

— profile

– search

– edit

– show

– register

– tree

– test

- layouts

– app

Related routes to views feature API design which only includes necessary routes for POST submission and doesn't reflect a fully dynamic API as there was not sufficient need to implement dynamic API routes for each Controller function or each CRUD capability.

The *show* view is divided into *partials* for easier management.

3.3.5 Controllers

We will consider the following controllers to be the most important during the development of the application: - Backups - Cats - Home - Register

The *Backups* controller should mostly rely on *import* and *export* functionalities provided by the respective functions, namely: - *import*, *export* for the *cats* table - *import_ems*, *export_ems* for the *ems* table - *import_breeds*, *export_breeds* for the *breeds* table Other than that the controller should provide functionality to show respective *views* such as *index* and *help*.

The *Cats* controller should mostly contain *edit*, *search* and *show* functionalities. Other than that functions that display the respective *tree* and *test* for test mating should exist.

The *Home* controller should exist as an example to display a view for *home* and *profile* otherwise not used in the application, only acting as a demonstration piece. The homepage is handled by the *search* functionality in the *Cats* controller.

The *Register* controllers should exist to delineate functionality related to registering new users or cats.

There is also the default *Controller* merely serving as a backbone for the controller

functionality. It is here that *Auth* should be declared over other controllers.

These controllers should be enough to allow for majority of the functionality provided by the web application. Other than that the registration of users also has to be considered alongside login functionality. This functionality should be provided by Laravel Breeze to optimize development time and resources. Laravel Breeze offers backbone for user authentication, enabling us to simply declare *Auth* over controllers and then utilize respective functionality in relevant views.

3.4 Major changes

One of the major changes is the diversion from CRUD-PHP-API idea to Laravel for MVC integration. This helps us in achieving a more performant and complete solution that addresses issues directly instead of a dynamic Dockerized approach with custom build tools.

Another major change is the dataset handling, which now consists of CSV import and export functionality alongside a more robust backup capability due to the restoration of a full dump being considered as not needed by the project leadership. The more robust backup functionality relies on *mysqldump* being available by the given webhosting provider or server.

3.5 Algorithms and Data Processing

In terms of algorithms and data processing, our plan will include developing an algorithm for calculating the inbreeding coefficient of a given cat. This algorithm will take into account the cat's family tree to determine how closely related the cat's parents are. In addition, we will design an algorithm to calculate the inbreeding coefficient for all cats in the database.

The algorithm will also be capable of creating an associative array to represent the family tree of cats. We will then fill in the ancestor arrays and calculate the inbreeding coefficient for each cat in the tree.

We will first define the algorithm for calculating the inbreeding coefficient. This is a measurement of how closely related two individuals are within a certain population or group. If a cat has less than two parents, it is not inbred. If this is not the case, we will calculate the inbreeding coefficient based on common ancestors and their path lengths.

The solution will be implemented as a front-end addition on top of the tree displaying data for individual cat's ancestors. This way we will utilize the processing power of the guest's computer on top of the processing capabilities provided by the Laravel-based backend solution.

3.6 Laravel

Laravel, a PHP-based web framework, is designed around the MVC architecture. It simplifies the setup, architecture, and dependencies of a project, allowing developers to focus more on the core aspects of development rather than intricate technical details [34] [**mvc-intro**]

By utilizing MVC in Laravel, developers benefit from increased security, scalability, and high performance. The framework offers features like Eloquent ORM for database interactions and Blade template engine for dynamic web pages[51][69]

The reason behind the usage of this framework is not just that it utilizes model-view-controller (MVC) pattern at its core and is based on the popular Symfony framework, it also ensures compatibility across major webhosting providers as tested in latter chapter.

Laravel also has a wide community of developers globally, providing support and resources for developers working with the framework. This community contributes to the flexibility and advancement of the open-source framework.[**mvc-intro**][69]

3.6.1 Illuminate

"is the namespace Laravel choose to put their code in. The word Illuminate means to light-up something. By using Laravel you are illuminating PHP development experience in their terms, hence the name." [27]

3.6.2 Eloquent

Laravel includes Eloquent, an object-relational mapper (ORM) that makes it enjoyable to interact with your database. When using Eloquent, each database table has a corresponding "Model" that is used to interact with that table. In addition to retrieving records from the database table, Eloquent models allow you to insert, update, and delete records from the table as well. [33]

3.6.3 Laravel Breeze

Breeze provides a minimal and simple starting point for building a Laravel application with authentication. Styled with Tailwind, Breeze publishes authentication controllers and views to your application that can be easily customized based on your own application's needs. [31] In our project Laravel Breeze was utilized to generate the backend code for authentication, later adjusted to meet our needs.

Chapter 4

Implementation

4.0.1 Inherited Project

We have chosen to build on top of an existing system provided on GitHub under a MIT license to highlight issues related to cats and cat data processing instead of handling a complete solution all on our own. [41] This has lowered the development time necessary for a fully fledged solution and lowered the bar to entry for development by us. We can now decide to spend our time and effort on cat-related functionality instead.

This system has provided us with the necessary information to handle population records (later customized to be the *cat* model) and display useful information about them within a profile page and in trees (later completely rewritten to display ancestor *cat* data).[41] Significant work has been done on reworking the system in order to get the necessary information customized to our own models and for them to be exportable in our format.

4.0.2 Migrations

Laravel migrations play a crucial role in managing the database schema. Within the `/database/migrations` folder, each migration file represents a set of instructions for modifying the database structure. These migrations ensure that changes to the database schema are version controlled and can be easily applied or rolled back.

As our application evolved, additional migrations have been created to accommodate new features or changes in the data model. For instance, during the consideration for tracking of vaccination records for cats, we considered creating a new migration specifically for managing the associated database tables and relationships.

In our application, the migrations for *Cats*, *Breeds*, and *Ems* serve as building blocks for our data model. The *Cats* migration defines the table structure for storing information about individual cats, including their names, ages, and other attributes. Similarly, the *Breeds* migration establishes the schema for recording different cat breeds and their characteristics, while the *Ems* migration handles data related to codes of cats.

The following migrations have been developed in total:

- 2024_01_12_000000_create_cats_table.php
- 2024_01_27_151536_create_couples_table.php
- 2024_01_30_215601_create_cat_metadata_table.php
- 2024_02_06_000000_create_breeds_table.php
- 2024_02_06_010000_create_ems_table.php
- 2024_03_23_010000_create_users_table.php

- 2024_03_23_020000_create_password_resets_table.php

Each migration file typically contains methods for defining changes to the database schema, such as creating or altering tables, adding or modifying columns, or defining indexes and foreign key constraints. Laravel's migration system ensures that these changes are applied in a consistent and controlled manner, making it easier to manage the database structure over time. Each file has an *up* method that allows for the migrations to take place and a *down* method that allows for resetting of the migrations to a prior state.

As of the project's final version, the tables contained the following columns:

cats:

- id (uuid)
- full_name
- gender_id
- sire_id
- dam_id
- dob
- titles_before_name
- titles_after_name
- ems_color
- breed
- chip_number
- genetic_tests

- breeding_station
- country_code
- alternative_name
- print_name_r1
- print_name_r2
- dod
- original_reg_num
- last_reg_num
- reg_num_2
- reg_num_3
- notes
- breeder
- current_owner
- country_of_origin
- country
- ownership_notes
- personal_info
- genetic_tests_file
- photo
- vet_confirmation

- doo

- Majority of the columns uses the *string* format, there are no relations to other database tables, only to itself (sire_id, dam_id).

couples:

- id (uuid)

- husband_id

- wife_id

- marriage_date

- divorce_date

- This serves as an example migration without additional usecases.

cat_metadata:

- id (uuid)

- cat_id

- key

- value

- This serves to store individual values for unique keys that user may wish to save. Currently there is no dashboard implemented to retrieve or store this data.

breeds:

- id (uuid)

- breed

- name

- This is linked to the *breed* of a given cat by comparing the string value of the *breed*. There is no direct relationship between the tables.

ems:

- id (uuid)
- breed_id
- ems
- english
- This is linked to the *ems_color* of a given cat by comparing the string value of the *ems*. There is no direct relationship between the tables.

users:

- id (int)
- name
- email
- email_verified_at
- is_admin
- password
- The *is_admin* functionality has to be manually toggled for the respective user in the database to allow for access to the Backup subsystem and cat registration.
- Currently any visitor to the site can register an account and afterwards will show up in this table.

password_resets:

- email
- token
- Exists only to allow for password reset links to contain a unique token that will refer to the user's account.

4.0.3 Routing and APIs

Routes have been defined in the *web.php* file found in the *routes* directory. These routes are defined in the following order: a HTTP method (GET, POST, DELETE) followed by the URI pattern and finally a callback function or a Controller method that handles the request. On the returning side of the method there is a View which the user sees underneath the given route.

The following major routes have been implemented in order to guarantee proper functioning of each View and its relation to a related Controller:

HomeController

- GET home
- GET profile

RegisterController

- GET register-cat
- POST register-cat

CatsController

- GET cats/cat
- GET cats/cat/edit

- POST cats/cat
- GET cats/cat/tree
- GET cats/cat/tree/generations
- GET test/cat/cat2
- DELETE cats/cat

BirthdayController

- GET birthdays
- GET birthdays/month/day

BackupsController

- POST backups/export
- POST backups/import
- POST backups/export_breeds
- POST backups/import_breeds
- POST backups/export_ems
- POST backups/import_ems
- POST backups/upload

POST language/change

Other routes have been added or left untouched as considered during development.

4.0.4 Controllers and functionality

The following Controllers have been developed as a part of the core functionality of our Laravel-based application:

- BackupsController.php
- BirthdayController.php
- CatMarriagesController.php
- CatsController.php
- Controller.php
- CouplesController.php
- FamilyActionsController.php
- HomeController.php
- RegisterCatController.php

Some functionality which we can highlight in the *CatsController* is as follows:

- *search* gets the user's request and obtains the most important parameters from it, more specifically the individual search parameters which will get passed into the database query (relevant model *Cat* is used in order to access its query functionality with *Cat::with*) and subsequently the returned query results get passed into the view using *compact* before the view is returned. Interesting note about this is the *paginate* functionality in the query, which implies pagination, however no additional adjustments are necessary as the functionality is built-in to Laravel's Eloquent ORM.
- *show* is the main functionality which shows individual cat data including related cat population.

- *chart* is a hidden functionality which enables the family tree to be displayed in a chart, however due to its unstable nature (certain pages took longer to load) we have decided to not keep this as a part of main functionality and leave it on the backend as a future proposal.
- *tree* has been adjusted to allow for generations on input. It copies cat information into the related view for the user to see.
- *edit* and *update* allows for the editing of cat data. Related map functionality has been kept inside of the codebase as a part of a future proposal where it can be utilized once again. *update* contains all the necessary information for validation of individual requests.
- *updateCatMetadata* is leftover functionality as a part of a future proposal to allow for individual record's attributes to be kept dynamically. This could be helpful in case there is leftover information which can't be saved within existing predefined fields.

Similarly we can highlight functionality in the *RegisterCatController*:

- *create* handles the majority behind cat registration by adding each field from the registration request.

And in the *BackupsController*:

- *help* displays the help information
- *store* creates a new backup
- *destroy* removes the existing backup
- *download* allows the user to download the backup
- *restore* allows for backup restoration

- *upload* allows adding a backup
- *import* and *export* allow work with CSV for the table *cats*
- *import_breeds* and *export_breeds* allow work with CSV for the table *breeds*
- *import_ems* and *export_ems* allow work with CSV for the table *ems*

4.0.5 Cat registration

Each cat can be registered using the respective *register* route underneath the */routes/web.php* file. In other words once the user accesses the register View using a simple GET request, he or she will be presented with a form that links to the POST route for registration and therefore implements the necessary functionality to register a new cat.

All of the relevant cat data can be added during registration except files and images. These have to be added separately by editing the given cat record.

Afterwards the profile of a registered cat can be viewed using the search functionality.

4.0.6 Search

Search capability is provided by the *CatsController* in a way which envelops ORM usage in order to get an exact, approximate or substring (default) search for a given number of cat generations (1-5) inside the cat tree. This way the user will be able to find exactly what they're looking for.

4.0.7 Editing capabilities

Editing of existing cats is also possible, rather the user has to manipulate the View on a *cats/cat* route or when it comes to details *cats/cat/edit* route. Following

The screenshot displays a web application interface with a navigation bar at the top containing links: Search, Birthdays, Test Mating, Find A Cat, Register Cat, Backups, and a user profile for 'admin' with a Logout option and a language dropdown set to 'EN'. The main content area is a 'Register' form for cat registration. The form is titled 'Register' and contains the following fields:

- Titles before Name
- Full Name *
- Titles after Name
- EMS Color
- Breed
- Gender * ☐ Male ☐ Female
- Date of Birth
- EMS Color
- Breed
- Genetic Tests
- Breeding Station
- Country Code
- Alternative Name
- Print Name Line 1
- Print Name Line 2
- Date of Death
- Original Reg No
- Last Reg No
- Reg No 2
- Reg No 3
- Notes
- Breeder
- Current Owner
- Country of Origin
- Country of Current Residence
- Ownership Notes
- Personal Info
- Date of Last Ownership Change

A yellow 'Register' button is located at the bottom of the form.

Figure 4.1: Registration

Chapter 4. Implementation

The screenshot shows the 'Eulalie Olivia Profile' page. At the top, there is a navigation bar with links for Search, Test Mating, Find A Cat, Register Cat, Backups, and admin - Logout. The profile page has three main sections: Profile, Family, and Children (6). The Profile section contains a table with fields: Full Name (Eulalie Olivia), Gender (F), Date of Birth (2011-07-20), Breed (RAG = Ragdoll), EMS Color (n 03 = seal bicolour), Original Reg No (SK) FFS LO 1116/2018/RAG, Reg No 2 (US) TICA SBT 072011 026, and Breeder (x, x). The Family section contains a table with fields: Sire (Kizzykat's Skipper RAG a 03 2006-06-04), Dam (Vanilla Sky of Eulalie RAG n 04 2007-04-06), and Test Mating (Test Mating). The Children (6) section lists six children with their names, breeds, and dates of birth.

Profile	
Full Name	Eulalie Olivia
Gender	F
Date of Birth	2011-07-20
Breed	RAG = Ragdoll
EMS Color	n 03 = seal bicolour
Original Reg No	(SK) FFS LO 1116/2018/RAG
Reg No 2	(US) TICA SBT 072011 026
Breeder	x, x

Family	
Sire	Kizzykat's Skipper RAG a 03 2006-06-04
Dam	Vanilla Sky of Eulalie RAG n 04 2007-04-06
Test Mating	Test Mating

Children (6)

- with (M) PL* Ragissa EL CAPITAN RAG a 03 2014-09-19
- (M) Xavie Olivier Ragdoll's Song RAG n 03 2016-08-16
- (M) Xavie Olivier Ragdoll's Song RAG n 03 2016-08-16
- (M) Hugo Ragdoll's Song RAG a 03 2018-07-12
- (M) Happy Ragdoll's Song RAG a 03 2018-07-12
- (F) Hannah Ragdoll's Song RAG a 03 2018-07-12
- (F) Harietta Ragdoll's Song RAG a 03 2018-07-12

Figure 4.2: Profile details

The screenshot shows the 'Find A Cat' search page. At the top, there is a navigation bar with links for Search, Birthdays, Test Mating, Register Cat, Backups, and admin - Logout. The main content area has a large background image of a cat's face. The title 'Find A Cat' is centered. Below the title is a search input field labeled 'Full Name'. Below the input field is a link for 'Advanced Search'. At the bottom is a yellow 'Search' button.

Find A Cat

Full Name

Advanced Search

Search

Figure 4.3: Search

The screenshot displays the 'Find A Cat' web application interface. At the top, there is a navigation bar with links for Search, Test Mating, Register Cat, Backups, and admin - Logout. A language dropdown is set to EN. The main heading is 'Find A Cat'. Below it, a search bar contains the text 'olivia'. An 'Advanced Search' button is visible. The search results are presented in a grid of 20 cards, each representing a cat profile. Each card includes the cat's name, sex, breed, registration number, date, sire, and dam. Buttons for 'Show Profile' and 'Show Family Tree' are provided for each entry.

Sex	Name	Breed	Reg. No.	Date	Sire	Dam
(F)	Bolivia	de Broeckloni	BRI ns 22 64		Sire : Beethoven	Dam : Billy Jean
(F)	Eulalie Olivia		RAG n 03	2011-07-20	Sire : Kizzykat's Skipper	Dam : Vanilla Sky of Eulalie
(F)	Lady Olivia Higgins	Vens Passion	MCO fs		Sire : Sir Aleksandar Higgins	Dam : Alwaro
(F)	Marlcreek Only Olivia		RAG n 03	2014-09-02	Sire : Marlcreek Mattias, RW	Dam : Absolutedolls Cammilicious
(F)	Olivia	Medicat, SK	SBI c	2010-04-01	Sire : Qeedo	Dam : Charlotte
(F)	Olivia	Charmant Angel, SK	PER ny 11	2010-04-06	Sire : Aldo	Dam : Midnight Tango
(F) CH	Olivia	vom Koberland	BRI c 33		Sire : Will	Dam : Jenny
(F) CH	Olivia	Ann Gatta, PL	NFO n 09 23		Sire : Golden Delicious	Dam : Jennifer Lopez
(F) CH	Olivia	Brilliant Star, SK	MCO n 09 22	2011-08-20	Sire : Darius	Dam : Marlon
(F) GIC	Olivia	Pretty Laureen Chérie, CZ	BRI ns 22 64		Sire : Jimmy	Dam : Whiskey
(F)	Olivia	Ruby Dolls, SK	RAG a 03 (RAG)	2012-04-16	Sire : Galileo of	Dam : Elza of
(F) SC	Olivia	S* Abayomis	RAG b 03 21		Sire : Zebastian	Dam : Emilia
(F)	Olivia	von Hunady, SK	BRI n 21 33	2014-08-09	Sire : Sanny Lynn	Dam : Leontynka
(F)	Olivia	Setasa, SK	BSH a(BRI)	2016-05-06	Sire : Casper	Dam : Cleopatra
(F)	Olivia	z Útan, SK	BSH c 33	2017-03-11	Sire : Fester	Dam : Carita
(F)	Olivia	LG Dolls, CZ	RAG n 03	2017-09-01	Sire : Cooki Lord Blue Anfe	Dam : Ohermgee Cinderella Ontario
(F)	Olivia	Katzen von Vanda, SK	BSH c	2018-03-02	Sire : Udyn	Dam : Bela
(F) GIC	Olivia	Jamren*PL	PER b 33		Sire : původ neuvedený	Dam : původ neuvedený
(F)	Olivia	SK* Cassovia Dolls	RAG a 04	2019-04-06	Sire : Joy my Soul Conan	Dam : Azmina of
(F)	Olivia	Lady Orchidea, SK	BSH ay 11	2020-05-03	Sire : Rocco Moon Sissi *CZ	Dam : Timea Bety
(F)	Olivia	Agostino, CZ	MCO f 02 21 64	2021-04-02	Sire : Pink Panther Oxymoron	Dam : Naomi Noblesse A Lyns Star, CZ
(F)	Olivia	z Ružinova, SK	BSH ns 11	2021-07-31	Sire : Nicolas Flox	Dam : Samanta Airin
(F)	Olivia	de Venorti *SK	BSH ns 03 22 64	2021-11-11	Sire : Hiroko v. Pftonwelt	Dam : Nala
(F)	Olivia	Werra pearl, CZ	MCO es 25	2021-12-20	Sire : Yonny Blue Agostino, CZ	Dam : Tri-D Jolie

Figure 4.4: Search Details

attributes have been made available as a part of the editing functionality:

- Full Name
- Gender
- Date of Birth
- Titles before name
- Titles after name
- EMS color
- Breed
- Genetic Tests file and text
- Chip number
- Breeding station
- Country code
- Alternative name
- Print name R1
- Print name R2
- Date of Death
- Original Registration number
- Last Registration number
- Second Registration number
- Third Registration number

- Notes
- Breeder
- Current owner
- Country of Origin
- Country
- Ownership notes
- Personal information
- Photo
- Veterinary confirmation

Format in which this information has to be provided is defined within the respective migration and in *CatsController* underneath the *update* functionality. Similarly it is also defined for registration.

4.0.8 Birthdays

Displaying the birthdays of individual cats has been considered as an addition on top of the existing routes as follows:

- Controller: BirthdayController
- Model: Cat
- Route: GET birthdays
- View: birthdays/index.blade.php

This view actively approaches the cats table using the *BirthdayController* by obtaining the *dob* attribute and using it to generate an ordered table of birthdays.

Search Birthdays Test Mating Find A Cat Register Cat Backups admin - Logout EN

Edit Eulalie Olivia

Show Profile Eulalie Olivia

Edit

Details

Delete

Edit

Titles before Name

Full Name

Eulalie Olivia

Titles after Name

Breed

RAG

EMS Color

n 03

Gender ☐ M ☒ F

Update Cancel

Figure 4.5: Edit Profile 1

This functionality has been removed from the menu on the webhosting due to compatibility issues.

4.0.9 Family trees

Displaying the ancestor trees of individual cats has been considered and implemented as follows:

- Controller: CatsController
- Model: Cat
- Route: GET cats/cat/tree
- View: cats/tree.blade.php

This view actively approaches the cats table using the *CatsController* while at the same time generating a necessary JSON interpretation using its respective view

Search Birthdays Test Mating Find A Cat Register Cat Backups admin - Logout EN

Edit Eulalie Olivia

Show Profile Eulalie Olivia

Edit

Details

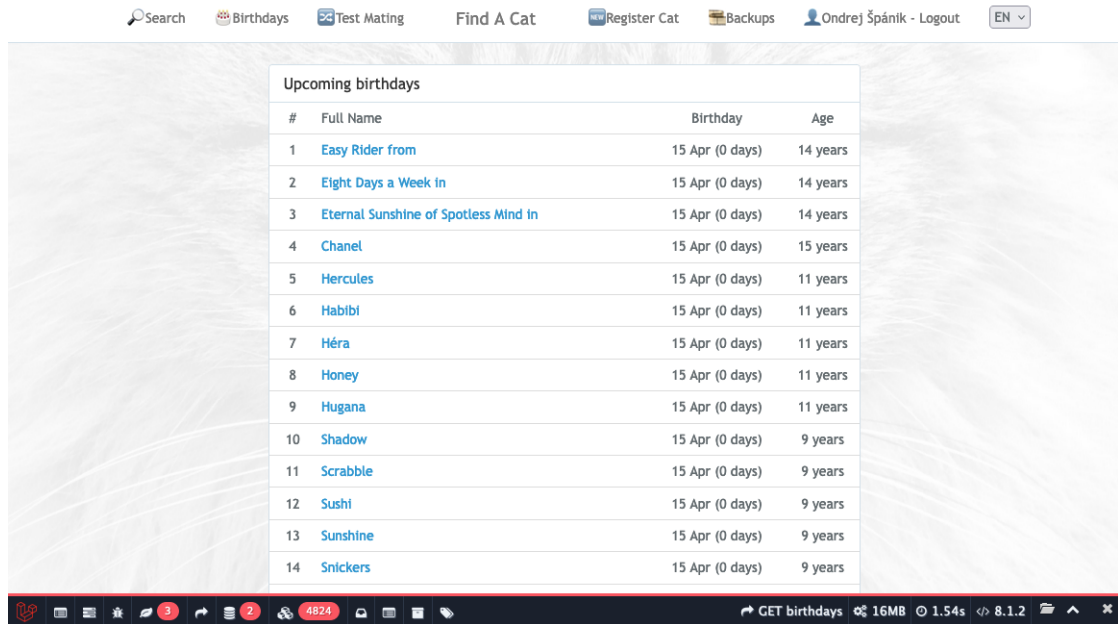
Delete

Details

Alternative Name	Date of Birth
<input type="text"/>	<input type="text" value="2011-07-20"/>
Print Name Line 1	Date of Death
<input type="text"/>	<input type="text" value="Eg. 1959-07-20"/>
Print Name Line 2	Date of Last Ownership Change
<input type="text"/>	<input type="text" value="Eg. 1959-07-20"/>
Chip Number	Original Reg No
<input type="text" value="96700009322174"/>	<input type="text" value="(SK) FFS LO 1116/2018/RAG"/>
Breeding Station	Last Reg No
<input type="text"/>	<input type="text"/>
Notes	Reg No 2
<input type="text"/>	<input type="text" value="(US) TICA SBT 072011 026"/>
Breeder	Reg No 3
<input type="text" value="x, x"/>	<input type="text"/>
Current Owner	Country Code
<input type="text"/>	<input type="text"/>
Ownership Notes	Country of Origin
<input type="text"/>	<input type="text"/>
Personal Info	Country of Current Residence
<input type="text"/>	<input type="text"/>
Photo	
<input type="text" value="Prehľadávať... Nie je zvolený súbor."/>	
Genetic Tests	
<input type="text"/>	
Genetic Tests	
<input type="text" value="Prehľadávať... Nie je zvolený súbor."/>	
Veterinary Confirmation	
<input type="text" value="Prehľadávať... Nie je zvolený súbor."/>	

Update Cancel

Figure 4.6: Edit Profile 2



Upcoming birthdays			
#	Full Name	Birthday	Age
1	Easy Rider from	15 Apr (0 days)	14 years
2	Eight Days a Week in	15 Apr (0 days)	14 years
3	Eternal Sunshine of Spotless Mind In	15 Apr (0 days)	14 years
4	Chanel	15 Apr (0 days)	15 years
5	Hercules	15 Apr (0 days)	11 years
6	Habibi	15 Apr (0 days)	11 years
7	Héra	15 Apr (0 days)	11 years
8	Honey	15 Apr (0 days)	11 years
9	Hugana	15 Apr (0 days)	11 years
10	Shadow	15 Apr (0 days)	9 years
11	Scrabble	15 Apr (0 days)	9 years
12	Sushi	15 Apr (0 days)	9 years
13	Sunshine	15 Apr (0 days)	9 years
14	Snickers	15 Apr (0 days)	9 years

Figure 4.7: Birthdays

at *views/cats/tree.blade.php* for the frontend processing of the tree in order to determine the inbreeding coefficient using an external library.

The relevant test mating functionality is available under the *views/cats/test.blade.php* and utilizes select2 to display searchable inputs.

The code behind family trees consists of functionality which enables the frontend JS library to access the tree hierarchy in a JSON format.

4.1 Algorithms

4.1.1 Inbreeding

The *views/cats/tree.blade.php* creates the necessary JSON interpretation for the frontend processing of the tree in order to determine the inbreeding coefficient using an external library called *inbreeding.js*. This way we can propagate backend

Chapter 4. Implementation

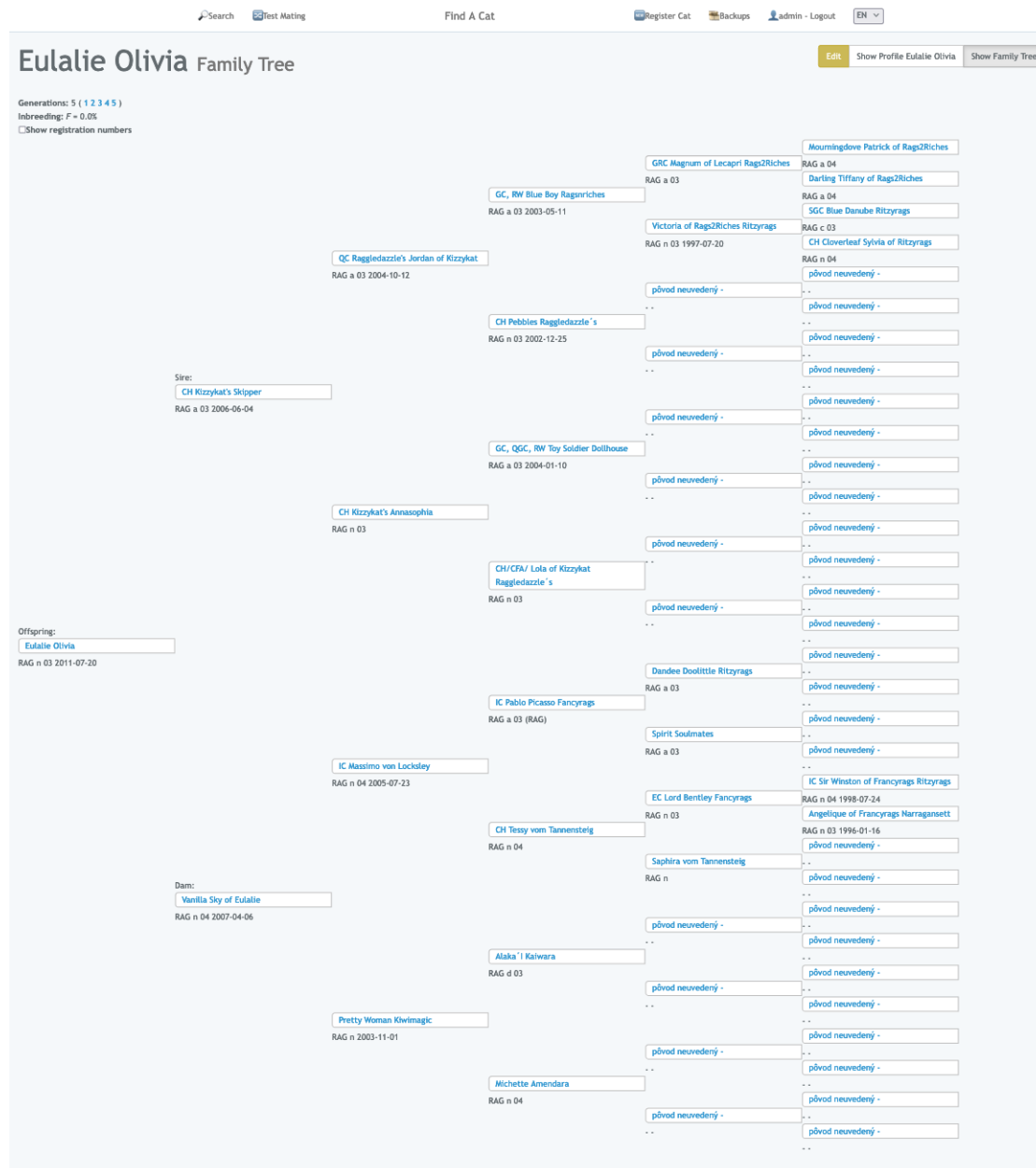


Figure 4.8: Family Tree

Chapter 4. Implementation

Search

Test Mating

Find A Cat

Register Cat

Backups

admin - Logout

EN

Test Mating

Inbreeding: $F = 1.56\%$
☒ Only show same breed for the opposite gender in select options
☐ Show registration numbers

(RAG) CH Benjamin Czech Star, CZ

(RAG) GIC Imagination Si*Luxus Beyond

Update

Update

		RW QGC Dillon Pitterpaw's	přvod neuvedený -
		RAG a 03	přvod neuvedený -
	
		Shae D Lyn of Farandoll Pitterpaw's	přvod neuvedený -
		RAG n 04 (RAG)	přvod neuvedený -
	
		Sydney Ragmeister's	přvod neuvedený -
		RAG a 03	přvod neuvedený -
	
		Anastasia Ragmeister's	přvod neuvedený -
		RAG n	přvod neuvedený -
	
		RW.DGC Lonerock Vinter Knight	
		Lonerock Issey Miyaki	RAG c
		RAG c 04	Caramel Nut Lonerock
		..	RAG h 04
		Excelsiis Showgirl & Chardani'DK	přvod neuvedený -
		RAG b 03	..
		..	přvod neuvedený -
		Alaka'I Kaiwara	..
		RAG d 03	přvod neuvedený -
	
		Michelle Amendara	přvod neuvedený -
		RAG n 04	přvod neuvedený -
	
		GRC Magnum of Lecapri Rags2Riches	
		GC, RW Blue Boy Rags2Riches	RAG a 03
		RAG a 03 2003-05-11	Victoria of Rags2Riches Ritzyrags
		QC Raggledazzle's Jordan of Kizzykat	RAG n 03 1997-07-20
		RAG a 03 2004-10-12	přvod neuvedený -
		CH Pebbles Raggledazzle's	..
		RAG n 03 2002-12-25	přvod neuvedený -
	
		GC, QGC, RW Toy Soldier Dollhouse	přvod neuvedený -
		RAG a 03 2004-01-10	přvod neuvedený -
	
		CH/CFA/ Lola of Kizzykat	přvod neuvedený -
		Raggledazzle's	..
		RAG n 03	přvod neuvedený -
	
		Dandee Doolittle Ritzyrags	
		IC Pablo Picasso Fancyrags	RAG a 03
		RAG a 03 (RAG)	Spirit Soulmates
		..	RAG a 03
		CH Tessy vom Tannensteig	EC Lord Bentley Fancyrags
		RAG n 04	RAG n 03
		..	Saphira vom Tannensteig
		Alaka'I Kaiwara	RAG n
		RAG d 03	přvod neuvedený -
	
		Michelle Amendara	přvod neuvedený -
		RAG n 04	přvod neuvedený -
	

Sire:

CH Benjamin Czech Star, CZ

RAG n 21

Dam:

Eulalie Olivia

RAG n 03 2011-07-20

IC All Tuckered out of Crystalcatzz / TICA/

RAG n 04 2003-05-23

North Star of Unidolls Pitterpaw's

RAG n

Crystal Dreamer Ragmeister's

RAG n

Sir Lancetti Royale & Chardani'DK

RAG b 04 21

CH Orinka Royal Kiwi of Eulalie, CZ

RAG n 04 21

Pretty Woman /NZCFU/ Kiwimagic

RAG f

QC Raggledazzle's Skipper

RAG a 03 2006-06-04

CH Kizzykat's Annasophia

RAG n 03

IC Massimo von Locksley

RAG n 04 2005-07-23

Vanilla Sky of Eulalie

RAG n 04 2007-04-06

Figure 4.9: Test Mating

changes to the frontend library for up to 5 generations while leaving room for future improvement.

The function *doCalculation* and *calculateForCode* does the handling:

- Variables: List of common ancestors, Consolidated list of common ancestors, Ancestor breakdown, Ancestor cache.
- *calculateForCode* resursively computes inbreeding paths.
- Total inbreeding is calculated by summing up products of inbreeding coefficients and the number of paths, this coefficient is then displayed on the page in percentage form.

The way the inbreeding is computed in *calculateForCode* works as follows:

- Variables are initialized: List of common ancestors, Copy of ancestors, Set to store intermediate ancestors, other possible variables for iteration and computation.
- If *baseCode* is provided (indicates offspring's common ancestor), then the function trims the ancestors accordingly by removing any codes that start with *baseCode* from the ancestors' list.
- The function then iterates over ancestors to find pairs of different codes leading to the same ancestor, these pairs represent potential inbreeding paths.
- Afterwards the function checks if there are any intermediate ancestors common to both paths for each pair of codes leading to the same ancestor. If not, it considers it as an inbreeding path.
- Calculation proceeds with the inbreeding coefficient based on the length of the codes and the inbreeding coefficient of the common ancestor: $2^{-(length1 + length2 - 1)} * (1 - FA)$ where *FA* is the inbreeding coefficient of the common ances-

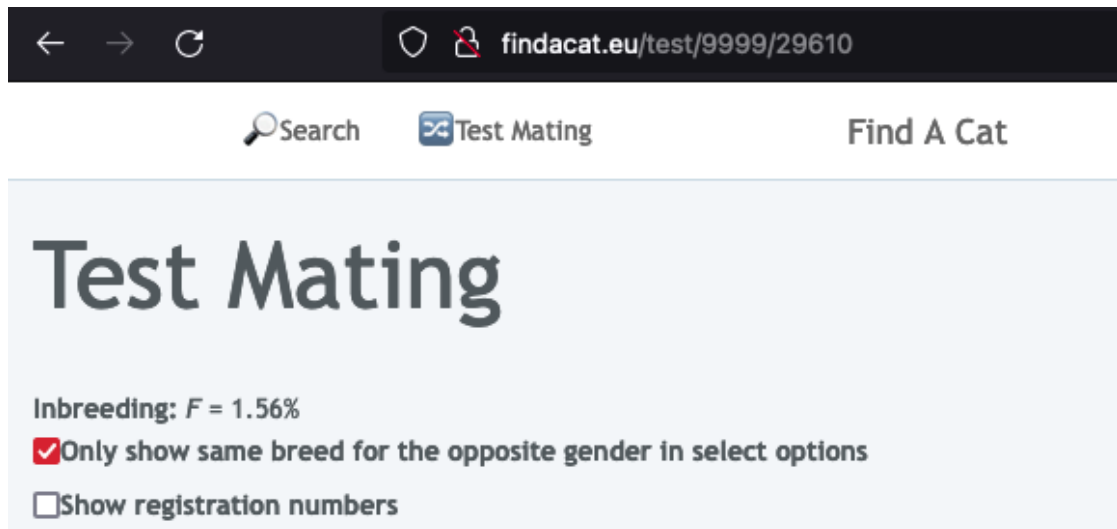


Figure 4.10: Inbreeding

tor.

- The function caches the calculated inbreeding coefficient for each common ancestor to avoid redundant calculations.
- The function either returns an array of objects containing name and inbreeding coefficient for each common ancestor or a total inbreeding coefficient calculated by summing up the inbreeding coefficients of all common ancestors.

4.2 Data processing

On top of inbreeding there is additional data processing available underneath the *birthdays* route and for each cat underneath its *show* route or in other words underneath its profile. This data processing is usually done using queries with the Eloquent ORM capability of Laravel.

Search:

```

$cats = Cat::with('sire', 'dam')->where(function ($query) use (
  ↪ $kind, $full_name, $ems_color, $dob, $breed, $reg_num) {
    $query->where(array_filter([
      $full_name ? ['full_name', ($kind == "exact")
        ↪ ? '=' : (($kind == "substring") ? 'like'
        ↪ ' : 'like'), ($kind == "exact") ?
        ↪ $full_name : (($kind == "substring") ?
        ↪ '%'.$full_name.'%' : '%'.$full_name.'%'
        ↪ )] : '',
      $ems_color ? ['ems_color', ($kind == "exact")
        ↪ ? '=' : (($kind == "substring") ? 'like'
        ↪ ' : 'like'), ($kind == "exact") ?
        ↪ $ems_color : (($kind == "substring") ?
        ↪ '%'.$ems_color.'%' : '%'.$ems_color.'%'
        ↪ )] : '',
      $dob ? ['dob', ($kind == "exact") ? 'like' :
        ↪ (($kind == "substring") ? '=' : 'like')
        ↪ , ($kind == "exact") ? $dob : (($kind
        ↪ == "substring") ? '%'.$dob.'%' : '%'.
        ↪ $dob.'%')] : '',
      $breed ? ['breed', ($kind == "exact") ? 'like'
        ↪ : (($kind == "substring") ? '=' : '
        ↪ like'), ($kind == "exact") ? $breed :
        ↪ (($kind == "substring") ? '%'.$breed.'%'
        ↪ ' : '%'.$breed.'%')] : '',
      $reg_num ? ['original_reg_num',
        ↪ ($kind == "exact") ? '=' : (($kind

```

```
        ↪ == "substring") ? 'like' : '
        ↪ like'),
    ($kind == "exact") ? $reg_num : ((
        ↪ $kind == "substring") ? '%'.
        ↪ $reg_num.'%' : '%'.$reg_num.'
        ↪ %'), "or",
    'last_reg_num',
    ($kind == "exact") ? '=' : (($kind
        ↪ == "substring") ? 'like' : '
        ↪ like'),
    ($kind == "exact") ? $reg_num : ((
        ↪ $kind == "substring") ? '%'.
        ↪ $reg_num.'%' : '%'.$reg_num.'
        ↪ %')] : '',
    ]));
    })
    ->orderBy('full_name', 'asc')
    ->paginate(24);
```

Birthdays:

```
$birthdayDateRaw = "concat(YEAR(CURDATE()), '-', RIGHT(dob, 5)) as
    ↪ birthday_date"
$catBirthdayQuery = Cat::whereNotNull('dob')
    ->select('cats.full_name', 'cats.dob', 'cats.id as
        ↪ cat_id', DB::raw($birthdayDateRaw))
    ->orderBy('birthday_date', 'asc')
    ->havingBetween('birthday_date', [today()->format('Y-m-d
```

```
→ '), today()->addDays(60)->format('Y-m-d'))]);
```

Show:

```
$catsMariageList = $this->getCatMariageList($cat);  
    $allMariageList = $this->getAllMariageList();  
    $malePersonList = $this->getPersonList(1);  
    $femalePersonList = $this->getPersonList(2);  
  
Cat::where('gender_id', $genderId)->pluck('full_name', 'id');
```

4.2.1 CSV import

There has been work put into implementing a decent CSV import and export feature for the following tables: *cats*, *breeds*, *ems*. The necessary information is provided alongside import and export functionality at the *backups* view. This view enables the user to set the necessary database values during the runtime and add additional values later on. It does not support changes on the existing attributes, however those are supported within the given cat's *show* view. Registration is also possible manually using the *register* view.

The order of columns is coded in the *BackupsController* for each imported CSV as follows:

cats:

```
public function import(Request $request)  
{  
    try {  
        $file = $request->file('file');
```



```
if ($file == null) throw new FileNotFoundException("");
if ($file->getPathname() == "") throw new
    ↪ FileNotFoundException("");
$fileContents = file($file->getPathname());

foreach ($fileContents as $key=>$line) {
    if ($key == 0) {
        continue;
    }
    $data = str_getcsv($line);

    Cat::create([
        'id' => $data[0],
        'full_name' => $data[1],
        'gender_id' => $data[2],
        'sire_id' => $data[3],
        'dam_id' => $data[4],
        'dob' => $data[5],
        'titles_before_name' => $data[6],
        'titles_after_name' => $data[7],
        'ems_color' => $data[8],
        'breed' => $data[9],
        'chip_number' => $data[10],
        'genetic_tests' => $data[11],
        'breeding_station' => $data[12],
        'country_code' => $data[13],
        'alternative_name' => $data[14],
```

```
        'print_name_r1' => $data[15],
        'print_name_r2' => $data[16],
        'dod' => $data[17],
        'original_reg_num' => $data[18],
        'last_reg_num' => $data[19],
        'reg_num_2' => $data[20],
        'reg_num_3' => $data[21],
        'notes' => $data[22],
        'breeder' => $data[23],
        'current_owner' => $data[24],
        'country_of_origin' => $data[25],
        'country' => $data[26],
        'ownership_notes' => $data[27],
        'personal_info' => $data[28],
        'photo' => $data[29],
        'vet_confirmation' => $data[30]
        // Add more fields as needed
    });
}
} catch (FileNotFoundException $e) {
    return redirect()->route('backups.issue');
}

return redirect()->route('backups.index');
}
```

breeds:

```
Breed::create([
    'id' => $data[0],
    'breed' => $data[1],
    'name' => $data[2]
    // Add more fields as needed
]);
```

ems:

```
Ems::create([
    'id' => $data[0],
    'breed_id' => $data[1],
    'ems' => $data[2],
    'english' => $data[3]
    // Add more fields as needed
]);
```

4.3 Docker image

The docker image has been built using the *Dockerfile* with *ubuntu:jammy* and with the following two commands:

```
docker build -t bp:2.0 .
docker run -dp 127.0.0.1:8000:8000 bp:2.0 --name bp
```

First command builds the version *1.0* of the image using its *Dockerfile* and the second command runs the built image while exposing the necessary port for *php artisan serve* command from *entrypoint.sh*.

4.4 Testing

Extensive testing has been done on the Wedos.cz and WebSupport.sk providers in order to determine any issues related to the MySQL migrations and PHP support. Webhosting on WebSupport.sk has been provided by the project leadership while webhosting on Wedos.cz has been made available by a friend.

There are relevant test classes available underneath the *test* folder. *Feature* tests cover authentication, cat deletion and *Unit* tests cover Cats. The following test cases have been considered:

- cat_have_profile_link
- cat_can_have_many_couples
- cat_have_sire_link_method
- a_cat_has_many_metadata_relation
- cat_model_has_get_metadata_method
- cat_have_dam_link_method
- cat_has_age_attribute
- a_cat_has_birthday_attribute
- test_registration_screen_can_be_rendered
- test_new_user_can_register
- user_can_delete_a_cat
- test_login_screen_can_be_rendered
- test_users_can_authenticate_using_the_login_screen
- test_users_can_not_authenticate_with_invalid_password

- test_email_verification_screen_can_be_rendered
- test_email_can_be_verified
- test_email_is_not_verified_with_invalid_hash
- test_confirm_password

Certain functionality does not work on webhosting providers such as the birthday functionality, obtaining database GZ dumps due to missing *mysqldump* and e-mailing password reset links.

4.5 Expansion potential

In the future the project could be expanded to cover genetic testing. Currently the genetic tests are only available in string and file format without additional processing. Additional metadata could be added as well. Better user management such as a profile dashboard and better mobile experience are additional suggested proposals. Permission system could be added, which would enable the authorized administrators to set what could others access/modify.

Chapter 5

Conclusion

The thesis provides an in-depth exploration of a database system designed for the management of cat data. The system is built on Laravel and comes with a range of features that allow for extensive interaction with the data.

The primary feature is the ability to register cats within the system, with each cat's data including a variety of fields such as full name, gender, date of birth, breed, chip number, and more. This registration process is facilitated through specific routes and controllers, providing a streamlined experience for the user.

In addition to registration, the system also allows for comprehensive search functionality. This is achieved through the 'CatsController', which utilizes Laravel's Eloquent ORM to allow for exact, approximate, or substring searches for a given number of cat generations. This ensures users can easily find the specific data they are looking for.

Editing of existing cat records is another key feature. This is made possible through specific routes and controllers, allowing for a wide range of cat data fields to be edited as required.

The system also includes a CSV import and export feature, providing a convenient method for handling bulk data. This feature covers the 'cats', 'breeds', and 'ems' tables, with the order of columns for each table being specified in the BackupController.

Additional functionalities include the ability to display the birthdays of individual cats and the presentation of ancestor trees of individual cats. These features bring additional depth to the data provided by the system, offering a more complete picture of each cat's data.

The thesis also notes the potential for future expansion of the system. This could include the incorporation of genetic testing data into the database, which is currently only available in string and file format without additional processing. Other potential expansions include improved user management capabilities, enhanced mobile experience, and the addition of a permission system for better access control.

In conclusion, the thesis presents a detailed exploration of a robust and comprehensive database system for managing cat data. With its extensive range of features and potential for future expansion, the system offers a powerful tool for any organization or individual in need of such data management capabilities.

Chapter 6

Resumé

6.1 Explorácia dát v databázových systémoch

6.1.1 Analýza

V kapitole o analýze sa zaoberáme možnosťami a problémami, ktoré sa týkajú implementácie algoritmov pre spracovanie dát v databázových systémoch. Prediskutujeme rôzne algoritmy a ich uplatnenie, ako aj možné obmedzenia a problémy, ktoré môžu vzniknúť v súvislosti s implementáciou týchto algoritmov.

Následne kapitola úvodu do chovateľských staníc a zodpovednosti chovateľov sa zaoberá chovateľskými stanicami a zodpovednosťami chovateľov. Popisuje, čo je chovateľská stanica, ich účel a ako sú regulované.

Časť "FIFe, šandardizácia a potenciál v existujúcich dátach" hovorí o FIFe, medzinárodnej federácii mačacích registrov, a o tom, ako sa dáta z týchto registrov využívajú.

Kapitola "Potreby chovateľov a obmedzenia poskytovateľov databáz" diskutuje o

tom, ako sú aktuálne databázy mačiek obmedzené a aké sú potreby chovateľov mačiek.

V časti "Skúmanie ďalších možností" sa skúmajú ďalšie možnosti pre spracovanie a správu dát o mačkách, vrátane potenciálu medzinárodnej výskumnej databázy a štandardizácie API.

Kapitola "Pripravenosť súboru údajov a algoritmy" sa zameriava na to, ako je súbor údajov pripravený pre spracovanie a aké algoritmy sa môžu použiť na analýzu dát.

Časť "Očakávané dáta" popisuje, aké údaje by mali byť v ideálnom súbore údajov od poskytovateľa databázy mačiek.

V časti "Črty ideálnych databázových tabuliek" sa popisujú ideálne črty databázových tabuliek pre správu histórie a pre budúcnosť.

Kapitola "Aktuálne údaje a databázové tabuľky" hovorí o aktuálnych údajoch a databázových tabuľkách, ktoré sú dostupné pre tento projekt.

Časť "Zváženia migrácie" sa zaoberá rôznymi aspektmi migrácie databáz, vrátane konverzie, optimalizácie a generovania migrácií.

V časti "Zváženia údržby" sa diskutuje o tom, ako udržiavať implementáciu dlhodobu udržiateľnú, vrátane podpory jedného alebo viacerých kanálov aktualizácií.

Kapitola "Algoritmy spracovania súborov údajov" sa zaoberá možnosťami spracovania súborov údajov, vrátane grafových a textových algoritmov a ich aplikovateľnosti.

Časť "Možnosti webového hostingu" sa zaoberá možnosťami webového hostingu, vrátane jeho obmedzení a potenciálu pre budúce rozšírenie.

V časti "Budúce rozšírenie a potenciál experimentov" sa skúmajú možnosti budúceho rozšírenia a experimentovania s novými technológiami, ako je napríklad WebAssembly.

Časť "Spracovanie dátových sád" sa zaoberá rôznymi aspektmi spracovania dát, vrátane možných problémov a obmedzení. Diskutujeme o tom, ako by sme mohli implementovať rôzne algoritmy a aké zmeny by to mohlo vyžadovať v databázovej štruktúre.

V časti "Databázové systémy" sa zaoberáme výberom a použitím databázových systémov. Diskutujeme o výhodách a nevýhodách rôznych databázových systémov, ako aj o tom, ako by sme ich mohli využiť v našom projekte.

V záverečnej časti kapitoly analýzy sa zaoberáme zhrnutím našich zistení a rozhodovaním o tom, aké technológie budeme používať v nasledujúcich kapitolách.

6.1.2 Konceptualizácia

V tejto časti sa zaoberáme definovaním cieľov a požiadaviek projektu, hľadaním možných prekážok a rozhodovaním o technológiách, ktoré budeme používať pri implementácii našich riešení.

Kapitola "Vzor Model View Controller" popisuje, ako je vzor „Model View Controller“ použitý pri navrhovaní webových aplikácií a ako sa tento vzor uplatňuje v rámci rámca Laravel.

Kapitola "Laravel prostredie" vysvetľuje, ako je MVC rozdelený v rámci súborovej hierarchie Laravel.

Kapitola "Dátové modely" sa venuje implementácii jednotlivých dátových modelov, ktoré budú potrebné pre webovú aplikáciu.

Kapitola "Dátové modelovanie" popisuje, ako sme pristupovali k reprezentácii populácií mačiek a ich vzťahov pomocou rôznych dátových modelov.

Kapitola "Pohľady (Views)" sa zaoberá pohľadmi, ktoré sa budú zvažovať v závislosti od trás a jednotlivých stránok.

Kapitola "Ovládače (Controllers)" popisuje, ktoré ovládače budú najdôležitejšie počas vývoja aplikácie.

Kapitola "Hlavné zmeny" popisuje hlavné zmeny, ktoré nastali počas vývoja projektu.

Kapitola "Algoritmy a spracovanie dát" sa zaoberá plánovaním a vývojom algoritmov pre výpočet inbrídneho koeficientu pre danú mačku.

6.1.3 Implementácia

Kapitola "Implementácia" popisuje, ako sme využívali rámec Laravel pri vývoji nášho projektu.

Kapitola "Migrácie" sa zaoberá migráciami v Laravel, ktoré hrá dôležitú úlohu pri správe schémy databázy.

Časť "Trasovanie a API" popisuje, ako sú definované cesty v súbore *web.php* nachádzajúcim sa v priečinku *routes*. Tiež sú tu popísané hlavné trasy, ktoré boli implementované pre správne fungovanie každého zobrazenia a jeho vzťahu k príslušnému kontrolóru.

Časť "Controller funkcionality" obsahuje zoznam kontrolórov, ktoré boli vyvinuté ako súčasť základnej funkcionality nášho Laravelového systému. Kľúčové funkcie sú podrobne popísané.

V časti "Registrácia mačiek" je popísané, ako môžu byť jednotlivé mačky reg-

istované pomocou príslušnej trasy *register* v súbore */routes/web.php*.

Zdôrazňuje sa v kapitole "Vyhľadávanie" schopnosť vyhľadávania poskytnutá kontrolórom *CatsController*, ktorý využíva ORM na presné, približné alebo podreťazcové vyhľadávanie.

Časť "Upravovanie" hovorí o možnostiach úpravy existujúcich mačiek.

Kapitola "Narodeniny" - jedná sa o dodatočnú funkciu, ktorá zobrazuje narodeniny jednotlivých mačiek.

V časti "Rodinné stromy" je popísané zobrazovanie rodokmeňov jednotlivých mačiek.

Časť "Príbuzenské párenie" podrobne popisuje, ako je príbuzenské párenie vypočítané a zobrazené na webovej stránke.

Kapitola "Spracovanie údajov" popisuje dodatočné spracovanie údajov, ktoré sa vykonáva v rámci funkcionality *birthdays* a pre každú mačku v rámci jej trasy *show*.

Časť "CSV Import" podrobne popisuje, ako sú importované súbory CSV pre rôzne tabuľky.

V časti "Docker obraz" je popísané, ako bol vytvorený Docker obraz a ako ho spustiť.

Časť "Testovanie" hovorí o testoch, ktoré boli vykonané na platforách Wedos.cz a Web-Support.sk.

V časti "Rozšírenie potenciálu" sú uvedené návrhy na rozšírenie projektu do budúcnosti.

V záverečnej časti je sumarizované celé zdokumentované informácie o projekte.

Literature

- [1] *About PawPeds*. URL: <https://www.pawpeds.com/cms/index.php/en/about/donate>.
- [2] *About Sverak*. URL: <https://www.sverak.se/om-sverak/>.
- [3] *ActiveMQ Use Cases*. URL: <https://activemq.apache.org/use-cases>.
- [4] Gregory R Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison–Wesley, 2000. ISBN: 978-0-201-35752-3.
- [5] The Cat Fanciers Association. *CFA Breeder Code of Ethics*. Dec. 2022. URL: <https://cfa.org/breeder-code-of-ethics/>.
- [6] Richard A Becker. *A Brief History of S*, Murray Hill, New Jersey: ATnT Bell Laboratories. July 2015. URL: <https://web.archive.org/web/20150723044213/http://www2.research.att.com/areas/stat/doc/94.11.ps>.
- [7] Aleksandar Bulajic and Nenad Filipovic. “Implementation of the Tree Structure in the XML and Relational Database”. In: 2012, pp. 27–51.
- [8] *Calculation of the coefficient of relationship*. URL: <http://www.genetic-genealogy.co.uk/genetics/Toc115570135.html>.
- [9] Bluein Christian. “Python vs PHP: Which is better”. In: (). URL: <https://www.openxcell.com/blog/python-vs-php/>.

- [10] *Cloudflare fixes outage that knocked popular services offline*. June 2022. URL: <https://techcrunch.com/2022/06/20/cloudflare-outage-knocks-popular-services-offline/>.
- [11] Edgar F Codd. “A relational model of data for large shared data banks”. In: *Communications of the ACM* 13 (6 1970), pp. 377–387.
- [12] *Converting from other Databases to PostgreSQL*. URL: https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL.
- [13] Maxime Crochemore and Wojciech Rytter. *Text algorithms*. Maxime Crochemore, 1994.
- [14] *Dataset Integrity*. URL: https://web.archive.org/web/20111005085820/http://www.fdewb.unimaas.nl/marc/ecais_new/files/boritz.doc.
- [15] *Definitions and Descriptions of Analysis*. URL: <https://plato.stanford.edu/entries/analysis/s1.html>.
- [16] Narsingh Deo. *Graph Theory with Applications to Engineering and Computer Science*. 1974. URL: <https://www.edutechlearners.com/download/Graphtheory.pdf>.
- [17] *Difference between script and framework*. URL: <https://www.quora.com/What-is-the-difference-between-a-script-and-a-framework>.
- [18] *Difference between WebSocket and REST*. URL: <https://www.educba.com/websocket-vs-rest/>.
- [19] *Exploring modular architecture in Laravel*. URL: <https://moezmissaoui.medium.com/exploring-modular-architecture-in-laravel-c44a1e88eebf>.
- [20] *FIFe Breed Standards*. URL: http://fifeweb.org/wp/breeds/breeds_prf_stn.php.
- [21] *FIIT STU Team project dataset*. URL: <https://www.fiit.stuba.sk/>.
- [22] Merrill M Flood. “The traveling-salesman problem”. In: *Operations research* 4 (1 1956), pp. 61–75.

- [23] Jan Goyvaerts. *Regular Expression Tutorial - Learn How to Use Regular Expressions*. Nov. 2016. URL: <https://web.archive.org/web/20161101212501/http://www.regular-expressions.info/tutorial.html>.
- [24] *Grafana*. URL: <https://grafana.com/grafana/>.
- [25] Feng Hao, John Daugman, and Piotr Zielinski. “A fast search algorithm for a large fuzzy database”. In: *IEEE Transactions on Information Forensics and Security* 3 (2 2008), pp. 203–212.
- [26] *How Microservices And APIs Can Make Your Company Modular*. Feb. 2022. URL: <https://www.forbes.com/sites/forbestechcouncil/2022/02/17/how-microservices-and-apis-can-make-your-company-modular/?sh=32de0984e1cf>.
- [27] *Illuminate*. URL: <https://stackoverflow.com/questions/65246691/what-is-the-meaning-of-illuminate-in-laravel>.
- [28] *Intro to APIs: History of APIs*. URL: <https://blog.postman.com/intro-to-apis-history-of-apis/>.
- [29] *Introduction to Laravel and MVC framework*. URL: <https://www.geeksforgeeks.org/introduction-to-laravel-and-mvc-framework/>.
- [30] *Introduction to the FIFe*. URL: http://fifeweb.org/wp/org/org_intro.php.
- [31] *Laravel Breeze*. URL: <https://github.com/laravel/breeze>.
- [32] *Laravel DRY Principles*. URL: <https://medium.com/@kevinmakwana189/keeping-your-laravel-application-fresh-and-efficient-with-dry-principles-370d588619b9>.
- [33] *Laravel Eloquent ORM*. URL: <https://laravel.com/docs/11.x/eloquent>.
- [34] *Laravel MVC use*. URL: <https://pusher.com/blog/laravel-mvc-use/>.
- [35] Abdelsalam Maatuk, Akhtar Ali, and Nick Rossiter. “Relational database migration: A perspective”. In: 2008, pp. 676–683.

- [36] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Vol. 1. McGraw-hill New York, 1997.
- [37] *Modular Web Design*. URL: <https://www.elinext.com/blog/modular-web-design/>.
- [38] Rogério Moreira. *From monolithic to headless: how and why you should adapt your WordPress stack*. URL: <https://medium.com/pixelmatters/from-monolithic-to-headless-how-and-why-we-adapted-our-wordpress-stack-309f0536007e>.
- [39] *Movement of pets - EU countries' specific information*. URL: https://food.ec.europa.eu/animals/movement-pets/eu-countries-specific-information_en.
- [40] *MySQL Documentation*. URL: <https://dev.mysql.com/doc/>.
- [41] *nafiesl's silsilah Project*. URL: <https://github.com/nafiesl/silsilah>.
- [42] *Omakissa Database*. URL: <http://kissat.kissaliitto.fi/kissat>.
- [43] Igor Omelchenko. *Monolithic vs. Microservices - the choice that defines the whole development process*. URL: <https://clockwise.software/blog/monolithic-architecture-vs-microservices-comparison/>.
- [44] Omkar M Parkhi et al. "Cats and Dogs". In: 2012.
- [45] *PawPeds Database*. URL: <https://www.pawpeds.com/db/>.
- [46] *Pedigree Online Dog Pedigree Database*. URL: <https://dogs.pedigreeonline.com/>.
- [47] Dušan Petković. "JSON integration in relational database systems". In: *Int J Comput Appl* 168 (5 2017), pp. 14–19.
- [48] *PHP CRUD API*. URL: <https://github.com/mevdschee/php-crud-api>.
- [49] Federico Posted. *A Modular Approach to Web Development*. June 2008. URL: <https://blog.fedecarg.com/2008/06/28/a-modular-approach-to-web-development/>.

- [50] *Purpose of the World Cat Congress*. URL: https://worldcatcongress.org/wp/org_purp.php.
- [51] *Reasons to choose Laravel*. URL: <https://osf.io/fgq3z/download/?format=pdf>.
- [52] *Responsible cat breeding guidelines in the European Union*. Nov. 2020. URL: https://food.ec.europa.eu/system/files/2020-11/aw_platform_plat-conc_guide_cat-breeding.pdf.
- [53] *Stambok Sverak Database*. URL: <https://stambok.sverak.se/>.
- [54] Michal Sudwoj. “Rust programming language in the high-performance computing environment”. ETH Zurich, 2020.
- [55] *Temporal Tables in Relational Databases*. URL: <https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-2017>.
- [56] *Terms and Conditions of WEDOS Hosting and Cloud Services*. URL: <https://www.wedos.com/conditions/hosting>.
- [57] *The FIFe Easy Mind System (EMS)*. URL: http://fifeweb.org/wp/breeds/breeds_ems.php.
- [58] John W Tukey. *Exploratory Data Analysis*. Pearson, 1977. ISBN: 978-0201076165.
- [59] John W Tukey. *The Future of Data Analysis*. July 1961. URL: http://projecteuclid.org/download/pdf_1/euclid.aoms/1177704711.
- [60] *VPS vs. VM vs. “The Cloud” - A Helpful Guide Comparison*. URL: <https://www.dynamichosting.ca/blog/vps-vs-vm-vs-cloud-helpful-guide-comparison>.
- [61] *WebAssembly Documentation: Use-Cases*. URL: <https://webassembly.org/docs/use-cases/>.
- [62] *WebScraper.io*. URL: <https://webscraper.io>.
- [63] *WEDOS Webhosting Options*. URL: <https://www.wedos.sk/webhosting>.

- [64] *What are the benefits of MVC?* URL: <http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/>.
- [65] *What is Elasticsearch.* URL: <https://www.elastic.co/what-is/elasticsearch>.
- [66] *What is Kibana.* URL: <https://www.elastic.co/what-is/kibana>.
- [67] *What is shared hosting definition.* URL: <https://www.namecheap.com/hosting/what-is-shared-hosting-definition/>.
- [68] *When not to use Docker.* URL: <https://www.freecodecamp.org/news/7-cases-when-not-to-use-docker/>.
- [69] *Why Laravel is the best PHP framework.* URL: <https://www.aalpha.net/blog/why-laravel-is-best-php-framework-features-and-benefits/>.

