Astana IT University

Ayaganov Aitbek
Zhumagaliyev Alibek

Web application for tracking and analyzing Internet traffic

6B06102 — Software Engineering

Diploma project

Supervisor
Akhmetov Olzhas
Senior-lecturer

Kazakhstan Republic
Nur-Sultan, 2022

CONTENTS

DEFINITIONS

Following terms are used in this work:

Concurrency	the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the final outcome.
WebSocket	is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.
Unix-Like	is an operating system is said to be Unix-based or Unix-like if it's designed to function and behave similar to the Unix operating system.
HMAC	A specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. As with any MAC, it may be used to simultaneously verify both the data integrity and authenticity of a message.
Thread	The smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.

DESIGNATIONS AND ABBREVIATIONS

Following designations and abbreviations are used in this work:

API	Application Programming Interface.
HTTPS	HyperText Transfer Protocol Secure.
HTTP	HyperText Transfer Protocol.
TCP	Transmission Control Protocol.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
REST	Representational State Transfer.
RPC	Remote Procedure Call.
SOAP	Simple Object Access Protocol.
SQL	Structured Query Language.
UI	User Interface.
UX	User Experience.
IOS	User Experience.
NPM	Node Package Manager.
HTML	HyperText Markup Language.
CSS	Cascading Style Sheets.
SCSS	Syntactically Awesome Style Sheet.
URL	Uniform resource locator.
QR	Quick response.
FTP	File Transfer Protocol.
DBMS	Database Management System.
RDBMS	Relational Database Management System.

INTRODUCTION

Work relevance. Analyzing online traffic that flows in and out of resources and targets is one of the challenges of digital marketing and advertising. A web application should be designed to tackle this problem. There are a few components of the application that must be considered. The application should be split into two parts: front-end and back-end. While the front-end should be not only user friendly, but also give users with quick and simple access to the application's primary features, the back-end should be constructed to manage a high and constant demand. The web application should give users with a link that will allow them to view statistics on the individuals and devices who have followed the link. To produce such application, is to do everyone, from small business owners to big corporations a favor.

Goal of the work. The main goal of this project is to create an internet platform for tracking and analyzing internet traffic utilizing the the best technology available, avoiding the deprecated realisations at all, and providing the end-users with user-friendly web-interface and system integration.

Research object. The main research object of this project is methods of data gathering and internet traffic tracking, as well as existing systems that can provide that functionality.

Novelty. At the present time, there is a plethora of tools available for tracking and collecting data, but there are a few issues with them. For starters, they are difficult to utilize and are not ubiquitous between platforms. Most of the projects with similar functionality have aged since their build and to the modern web environment, thus utilizing old and deprecated technology stack. Second, most of them are tied to the system for which they were built. Owner of the products that required such tools left them for their private use only.

Research methodology. We conducted an external research on existing on market analogues.

Practical relevance of the work. We strive to not just provide a platform for users with this project, but to do so utilizing the best currently available technology stack to achieve and create the most secure and quality product appropriate for future and modern widespread use. With this project we intend to push forward the technology front and raise the bar for web platforms.

1 RESEARCH ANALYSIS

1.1 Description of the problem

The general problem all creators face is proper understanding of their audience. Small businesses, web services, online content creators and many other would greatly benefit from understanding their audience and even expand their line of products or providing functionality, or expand their audience. [1] Understanding your main audience, their needs and desires is very important. For example, for web applications knowing their target audience and product users is key to creating quality user experience, resulting in better results. Good UI can make all the difference in the world, it can attract new users and create a feeling of product being tailor made for them and their purposes, resulting in maximum product investment. [2] General advantages of having good knowledge on main audience are:

- Developing better UI/UX for web and mobile applications;
- Make better decisions;
- Provide insightful data that can be used to expand;

1.2 Research of the existing analogues

For our research analysis we decided to conduct a research of existing analogues with similar more or less similar functionality available on the market. We came to the conclusion that it would be useful when designing the structure, functionality in comparison and scope of the project. So we could then apply that knowledge in development of the project itself. We decided to focus on two main web analogues that were the most popular in the filed of functionality they provided.

Bitly (https://bitly.com) This web application presents itself as an online URL shortener tool with additional functionality in form of providing click-through traffic data. (Figure 1.1). Main functionality is link shortening. To get started you only need to create an account and log in.

Advantages:

- Custom shortened links;
- QR codes;
- Stable services;
- Easy to pick up;
- Good UI;
- HTTPS encryption with every link.

Disadvantages:

- Free option are scarce. Full functionality of the project is locked behind a very steep paywall;
- Slow customer service;
- Intrusive web design practices like pop-ups and forced dialogue windows;

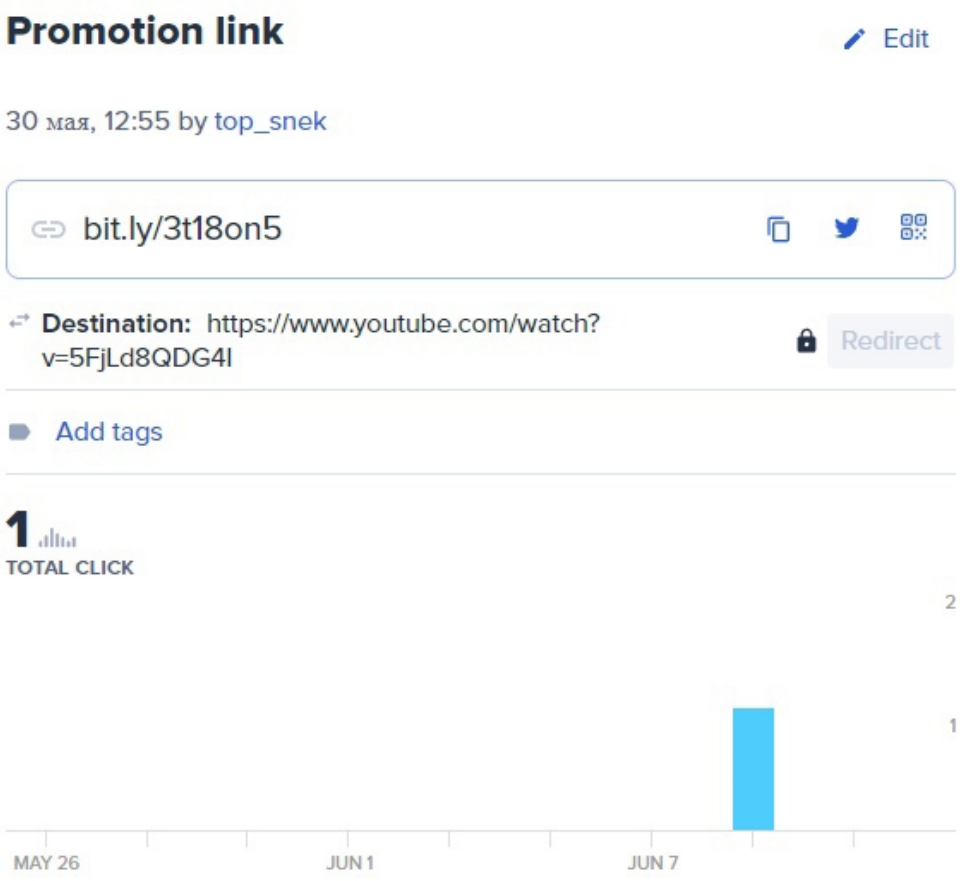

Figure 1.1 – Bitly link window example

- Traffic tracking data is very limited;
- Data export is lacking.

FingerprintJs (https://fingerprintjs.com) This web application provides visitor identification service, collecting user data and detecting fraudulent activity. (Figure 1.2). Main functionality is fraud detection. To get started you need to register an account and import their Javascript snippet in your project.

Advantages:

- GDPR and CCPA compliant;
- Unparalleled accuracy of fraud detection on the market;
- Geo location of visitors;

Disadvantages:

- The service's free plan is rather limiting in number of requests;
- Functionality provided is hard-focused and limited to fraud detection;
- Expensive;

After we conducted a throughout research and analysis of these applications, their capabilities and general usefulness, we came up with a table of main features of these projects. (Table 1.1). While having intersecting fields of functionality each application has its own pros and cons that we took into account in the process of

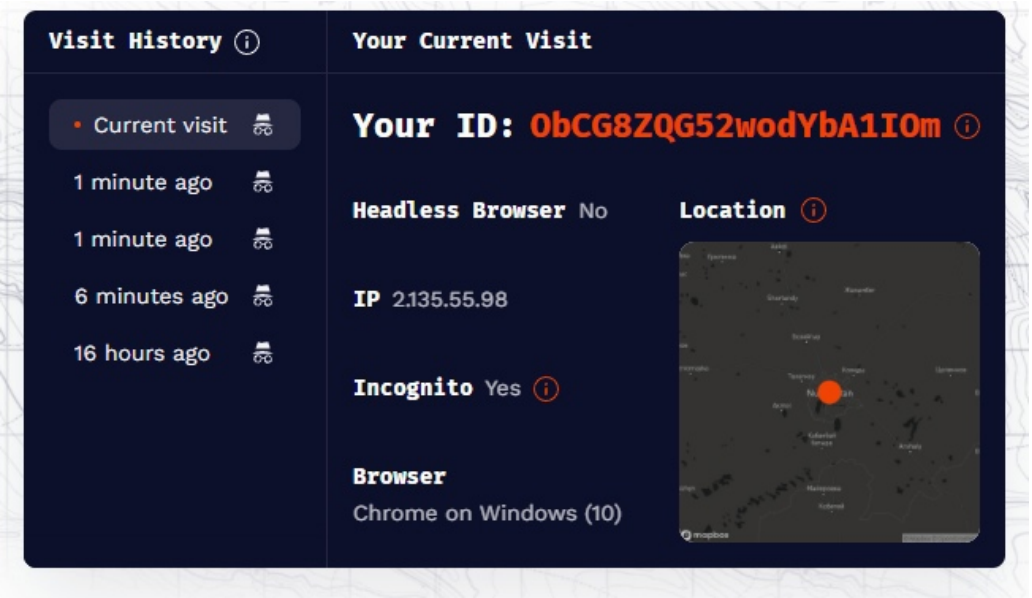

Figure 1.2 – FingerpringJS link window example

creating and development of our project .

Table 1.1 – Comparison of existing analogues

Parameters	Bitly	FingerprintJS
Bot/fraud detection	No	Yes
Geo location data	Yes	Yes
Mobile version	Yes	Yes
Link shortening	Yes	No
Device details	No	Yes
Customer support	Yes	Yes
Account sharing	No	No
Mobile application	Yes	No

1.3 Technical task

Our main task is to create a platform for a link management system. To achieve that we would need to develop an application for serving web interface that will handle metadata analysis and provide a system of backend applications that will handle redirecting and metadata retrieving.

We outlined technical tasks as follows:

1 Explore existing analogues;
2 Design web-interface for application;
3 Configure hardware for serving database, backend applications and frontend static files;

4 Set up convenient development and delivery environments;
5 Develop a platform for a link management system;
6 Develop an application for serving web interface that will handle metadata analysis;
7 Develop a system of backend applications that will handle redirecting and metadata retrieving;

1.4 Project architecture

The microservices architectural style is an approach in which the system is built as a set of independent and loosely coupled services that can be created using various programming languages and data storage technologies. The concept of microservices allows you to maintain loose coupling of services in the process of working on the system, which is determined by the Low Coupling and High Cohesion patterns [3]. Depending on the demands of each service, it may be built, launched, and maintained individually, and it can use a variety of programming languages, technologies, and software environments. Individual services may be scaled individually, and new components can be introduced without the need for system downtime or re-deployment. Services can also be spread over numerous servers, reducing the impact of more demanding components on overall performance.

With a monolithic architecture, the system usually consists of 3 blocks: the user interface, the data store, and the back end. The server part processes requests, executes business logic, works with the database, fills in HTML pages. Any change in the system results in an update of the version of the server side of the application.

For this project we decided to go with microservices architectural style and frontend application. This project architecture will help us maintain simplicity around developing process and better scalability.

1.5 API architectural style

The value of a well-chosen architectural style cannot be underestimated. Often, it is the architectural style that sets the parameters for choosing a programming language, principles and patterns for writing code, and the design of the entire system as a whole. We need to choose one of the approaches from REST, GraphQL and SOAP, so we need to make a detailed analysis and comparison of each of the architectural styles.

To begin with, it's worth saying that REST ("Representational State Transfer" is a representation state transfer) is not a protocol, but an architectural style, so there is no single standard for its use. This term was introduced by Roy Fielding in 2000 [4]. At the moment, this style is most often used to "communicate"the client with the server via the HTTP protocol.

Client requests to the server are made using HTTP methods, the most popular of which are GET, POST, PUT and DELETE. It is also common to use HTTP headers and request parameters to specify additional information. Sometimes developers limit themselves to supporting only two request methods - GET and POST.

SOAP (Simple Object Access Protocol) is a simple protocol for accessing objects. Unlike REST, it is a protocol and is standardized by the World Wide Web Consortium (W3C). Can be used with any application layer protocol, but is most commonly used in conjunction with HTTP.

GraphQL is an open source API data query language created by Facebook, which has been using it in their projects since 2012. GraphQL is standardized and has a detailed specification.

As mentioned earlier, each of the approaches has both advantages and disadvantages, which are revealed both in their study and in practical implementation [5].

Advantages of REST

1 Ease of implementation. Using the REST architecture in a web application usually does not require additional solutions because the client and server support HTTP requests by default;
2 There are a large number of tools and libraries for developers that can make development and debugging easier, but their presence is not required for normal development;
3 In a server response, as well as in a client request, data can be presented in different formats within the same application.

Disadvantages of REST

1 Complexity of support. As mentioned earlier, there is no single standard for REST, so practical implementations vary greatly;
2 The number of default HTTP methods supported by different servers and clients may vary. Therefore, the developer is often limited to using only the most popular ones.

Advantages of SOAP

1 Fully standardized for easy support;
2 Uses XML with strong data typing, which guarantees their integrity and makes automatic validation possible;
3 It is possible to automate XML generation using schemas.

Disadvantages of SOAP

1 The server response, like the client request, can only be represented in XML format;
2 The protocol is difficult to learn for developers, so since version 1.2 the abbreviation SOAP has not been deciphered;

3 The complexity of the protocol comes with a performance penalty;
4 In practice, when creating an API using SOAP, you may need to install additional libraries and tools for editing XML schemas.

Advantages of GraphQL

1 Greater query flexibility. The client requests from the server what is really needed and with as much nesting as it wants;
2 The ability to generate dynamic queries at the specification level and reuse their code;
3 Fully standardized language with detailed specification for easy support;
4 The choice of libraries and development tools is gradually increasing.

Disadvantages of GraphQL

1 Due to the flexibility of processing requests by the server, performance is reduced;
2 The language is quite new, so there is not such a large selection of libraries and tools, as in the case of REST or SOAP, without which the rapid deployment and testing of a GraphQL server is problematic;
3 Server response and client request can only use JSON for data.

Based on our priorities and the current state of the project, we have decided to choose REST as the architectural style. With the help of it we will be able to launch a stable and reliable product as quickly as possible. However, we do not deny that other architectural styles and protocols can be used in the further development of the product.

1.6 Comparison of back-end tools

Since our goal is to create not just a product, but a platform that can be easily developed and scaled in the future, it is very important for us to choose the right set of tools. It is necessary to choose the right database, programming language and web server. Let's start with the database. Let's put forward the requirements for the database

1 The database should be light and simple;
2 Strong database access protection is important;
3 The speed of operations is important.

We need a free relational database, so MySQL and PostgreSQL are our candidates. There are several situations where MySQL becomes a priority for developers. RDBMS is chosen if:

- the speed of working with data is important;
- need more features and functionality;
- security of performed operations and reliable protection of access to data are important;
- it is supposed to work with sites and applications;
- need flexibility and ease of use.

PostgreSQL is more suitable for situations where:

- we need support for foreign keys, triggers and views that allow you to hide the complexity of the database from the application to avoid complex SQL commands;
- the ability to create selects (nested subqueries) is important;
- you need the ability to create complex SQL commands (due to compliance with ANSI SQL standards);
- data integrity is important;
- MVCC support is required to provide simultaneous access to the database to a large number of users for reading and writing;
- need support for NoSQL and different data types;
- it is supposed to perform complex procedures and expand the database;
- the next move of the database to another solution is planned. [6]

Table 1.2 – Comparison of RDBMS

Feature of	MySQL	PostgreSQL
Open Source	Open source but owned by Oracle	Open source
ACID Compliance	Partial Compliance	Full Compliance
NoSQL/JSON support	Support for some features	JSON data only
Declarative partitioning	Supported	Supported
Logical replication	Supported	Supported
Window functions	Supported	Supported
Nested Selects	Yes	Yes
Transactions	Yes	Yes
Triggers	Yes	Yes
Memory-only storage capability	Yes	No

Despite the huge number of features and technical improvements of PostgreSQL (Table 1.2), we chose MySQL. The choice was made based on the set criteria, with MySQL we will be able to speed up our development and speed up all the operations that will take place in the system.

Now we need to decide on the choice of programming language. Since the speed of work and optimization of parallel processes are very important for the system, we will consider programming languages that support multithreading. Therefore, we will not have Python in the list of candidates, since Python does not have a built-in multithreading mechanism. We will consider such languages as: Node, Go, PHP and Java. In order to determine which of the programming languages works faster, it is necessary to conduct performance tests and compare

the overall performance of the HTTP server in different server environments. But keep in mind that the resulting HTTP request/response performance is influenced by many factors, and the data presented here will only provide a general idea [7]. For each environment, here is code that reads a 64K file filled with random bytes, then hashes it N times using the SHA-256 algorithm (N is specified in the URL query string, for example, .../test.php?n=100) and displays the resulting hash in hexadecimal. This is a very simple way to run the same benchmark with a consistent amount of I/O and a controlled way to increase CPU usage. First, let's look at examples with little parallelization (low concurrency). Run 2000 iterations with 300 concurrent requests and only apply one hash per request (N = 1)

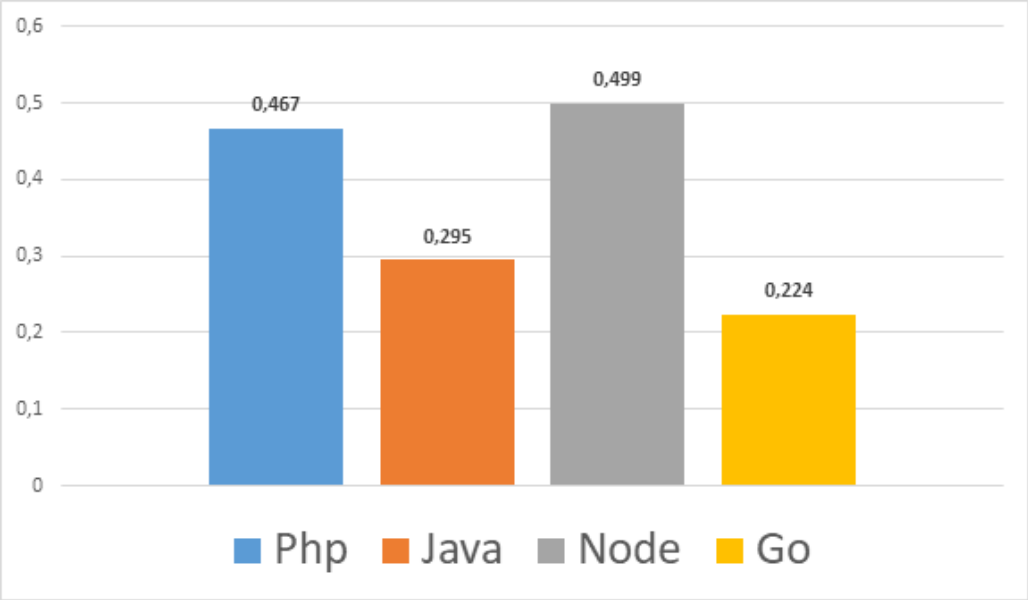

Figure 1.3 – First Benchmark Test

Based on one graph (Figure 1.3), it is difficult to draw any conclusions. But it seems that with this amount of connections and calculations, we see results that are more like the total execution time of the languages themselves, and not the duration of I / O operations. Please note that the so-called scripting languages (weak typing, dynamic interpretation) work the slowest [8].

Increase N to 1000, leaving 300 simultaneous requests - the load is the same, but you need to perform a hundred times more hashing operations (significantly increases the load on the processor)

Suddenly, Node's performance dropped significantly because operations that are actively using the processor in each request block each other. Curiously, PHP has become much better in performance (compared to others) and has overtaken Java. It should be noted that PHP's implementation of SHA-256 is written in C, and in this loop the execution path takes much longer because we now need 1000

hash iterations (Figure 1.4).

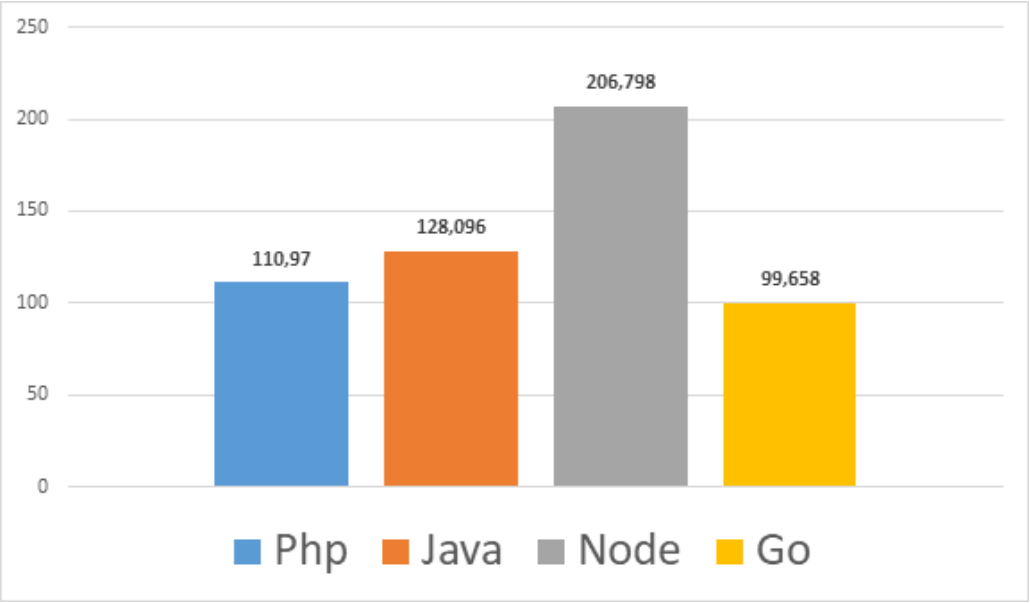

Figure 1.4 – Second Benchmark Test

Now let's make 5000 simultaneous requests (N = 1) or as close as possible to this number. Unfortunately, in most environments, the failure rate was significant. The graph shows the total number of requests per second.(Figure 1.5)

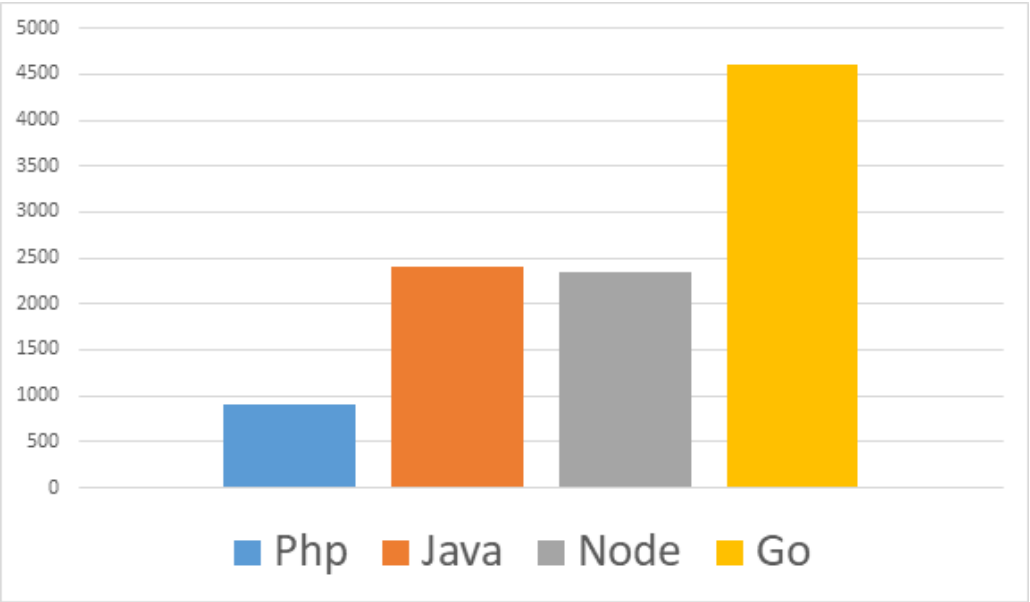

Figure 1.5 – Third Benchmark Test

Through a performance test, we found that Go is a better fit.
In addition, Go has the following advantages:

- statically typed;
- there is a garbage collector that automatically cleans up memory;
- support for network protocols;
- simplified syntax;
- a huge number of modules out of the box, there is no need to master third-party libraries and frameworks;

Now let's move on to comparing the web servers that need to be installed on the server. The main candidates here are Nginx and Apache.

Since ancient times, Apache creates a separate process (or thread, depending on the selected mpm module) for each request from the client. It looks like this - the client sends a request, the web server creates a separate process for this request, responds to the client and blocks the process until the client closes the connection. This is easy and simple to implement, but a process in any OS requires memory and resources, and when there are indecently many processes, connection processing slows down indecently, memory runs out, CPU load increases. For small projects, such an implementation of the connection processing architecture will not add complexity, but for highly loaded systems, it is necessary to look for alternative solutions or increase computing power.

Nginx consists of a master process and several child processes. The master process is usually one - it creates child processes (workers, cache loader and cache manager), reads the configuration and opens ports. There are usually several workers, Nginx developers advise to determine the number of workers equal to the number of machine cores [9]. These child processes will service all client connections in a non-blocking manner. Nginx uses an infinite loop that runs through all connections and responds to client requests. When a connection is closed, it is removed from the event loop. This solution is ideal for projects that are serving 10k+ connections at the same time. At the same time, CPU loading and memory usage are usually even, with no visible peaks.

The speed of a web server is usually measured for 2 cases of content delivery: for static and dynamic. Based on performance tests, Nginx is about 2.5 times faster to serve static than Apache. That's a pretty big advantage. If you need to serve a large amount of static content, Nginx is the best choice. When testing the return of dynamic content, Apache and Nginx show approximately the same results. In terms of memory, both servers use the same amount of resources [10].

Since the speed of the system is very important for us and it is assumed that the system will serve a large number of connections (despite the fact that the time of these connections will be short), we decided to choose Nginx as our web server.

1.7 Comparison of front-end tools

For the front-end part of our project, a web application, we had a good number of options to take from in terms of web development framework [11]. The

choice was between 4 mainstream modern web frameworks - AngularJS, ReactJS, EmberJS and VueJS. [12] We make the decision for development framework based on 2 main attributes we were looking for in our web application: good performance and flexible development with convenient documentation. All of the frameworks mentioned above were decently suited to handle the task of building the application, so we had to conduct a research and determine the best option for our product [13]. (Table 1.3). In terms of performance ReactJs and VueJs take a lead in the this aspect, ReactJS being a view library with its one-way binding makes it the fastest in the field of performance, and VueJS having a lot in resemblance with ReactJS makes it practically equal. (Figure 1.6). On the other hand, AngularJs and EmberJs have a two-way data binding system that while yes, makes it a little slower, prompting more actions from the system, the data changes that affect the model are immediately propagated to the corresponding template, and conversely, any changes made in the UI are promptly reflected in the underlying model. This is very convenient and comes in hand when data binding. While AngularJS and EmberJs have that in common, AngularJS is so much more faster that EmberJs in almost every way. And it for the flexibility of the frameworks presented we leaned towards AngularJs in the end. Being known for its flexibility and the most established framework of them all, Angular provides web developers with official support and the most elaborated official guidelines and utilitarian templates with great documentation. [14] The community is huge and amount of useful modules, libraries, blogs and is a big plus in our eyes.

Table 1.3 – Comparison of front-end tools

Parameters	AngularJS	ReactJS	EmberJS	VueJS
Performance	Excellent	Good	Slow	Excellent
Data binding	Two-way	One-way	Two-way	One-way
Flexibility	Best	Good	Fine	Good
Support	Best	Good	Bad	Fine

Name Duration for...	vue-v2.6.2-keyed	angular-v8.2.14-keyed	react-v16.8.6-keyed
create rows	160.9 ± 2.4	157.3 ± 3.0	176.2 ± 3.9
replace all rows	133.1 ± 1.2	137.5 ± 1.4	145.7 ± 1.3
partial update	223.8 ± 7.9	152.3 ± 3.4	218.7 ± 2.8
select row	305.9 ± 10.7	74.8 ± 2.7	124.6 ± 5.1
swap rows	69.9 ± 2.8	441.7 ± 2.4	445.5 ± 2.1
remove row	29.0 ± 0.5	27.7 ± 0.8	27.6 ± 2.1
create many rows	1,300.6 ± 15.1	1,406.4 ± 36.5	1,827.0 ± 75.1
append rows to large table	310.2 ± 4.5	303.3 ± 13.0	345.9 ± 6.8
clear rows	156.3 ± 1.1	248.6 ± 3.3	149.2 ± 2.1
slowdown geometric mean	1.24	1.32	1.46

Figure 1.6 – Benchmark showcase between ReactJS, VueJS and AngularJS

2 APPLICATION DEVELOPMENT

2.1 Development environment

First, we need to build a web application development and deployment environment. It is important for us to launch our product as early as possible, and to be able to update it very easily and quickly. During the analysis, we put forward these requirements for the development process:

- The development processes of backend and frontend applications should not interfere with each other, and all dependencies should be specified in advance;

- All applications must be up and running using HTTPS protocols, backend applications associated with the frontend application must use the same domain name;

- There should be a convenient way to upload updated files to servers, as well as a convenient way to restart applications.

These steps have been taken to meet these requirements:

- The entire development process is divided into weekly plans;

- Backend applications should be updated before the development of the functionality of the Frontend application that depends on it;

- Postman used for API documentation;

- Github is used to push updated backend application files. To transfer updated files to the Frontend application, FileZilla's FTP client is used using the SFTP protocol. The Systemd utility is used to start and restart Backend applications;

- NGINX is used to host the Frontend application and redirect requests by domain name. It is also used to make and process requests over the HTTPS protocol.

The process of delivering updated backend application files is carried out in accordance with this algorithm

1 Program code is written and tested on the developer's local computer. Depending on the specifics of the application, various testing methods are used;
2 Updated files are uploaded to their respective Github repositories;
3 The developer connects to a remote server using the SSH protocol;
4 Files are downloaded from the Github repository using Git utility commands;
5 A programming language compiler is used to create a binary application file. After the previously downloaded files are deleted for security purposes in case of seizing access to the connection to the server;
6 If the application has not been previously run on the server, a service is created using the Systemd utility (Figure 2.1);
7 Restarting the relevant service using Systemd utility commands;

8 If the application has not been previously launched on the server, you must create the appropriate setting in the NGINX configuration of the file serving the application system;
9 Restarting the NGINX service.

```
[Unit]
Description= Klickz backend

[Service]
Type=simple
Restart=always
RestartSec=5s
ExecStart=/root/repos/go/klickz-backend/api
WorkingDirectory=/root/repos/go/klickz-backend

[Install]
WantedBy=multi-user.target
```

Figure 2.1 – New service configuration file

After that, we needed to write an NGINX configuration that would allow us to use one domain name and an SSL certificate to redirect requests to all applications on the system (Figure 2.2).

2.2 Hardware

To deploy all applications, as well as for a convenient development process, a remote server is required. The server must be based on the Ubuntu operating system. The following utilities and services must be installed on the server:
1 Systemd is a system and service management utility.
2 Git is a distributed version control system.
3 MySQL is a free relational database management system.
4 PhpMyAdmin is a web application that is a web interface for administering the MySQL DBMS.
5 NGINX is a web server and mail proxy that runs on Unix-like operating systems.
6 Go is a compiled multi-threaded programming language.

Since for us the priority is not the computing power of the server, but the speed of development and deployment of applications, we decided to use ready-

```
root /var/www/klic.kz;
index index.html;
server_name klic.kz www.klic.kz;
location / {
    try_files $uri $uri/ /index.html;
}

location /l {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header Host $http_host;
    proxy_pass http://127.0.0.1:4003;
}

location /web {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header Host $http_host;
    proxy_pass http://127.0.0.1:4000;
}

location /api {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header Host $http_host;
    proxy_pass http://127.0.0.1:4000;
}

location /ipparser {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header Host $http_host;
    proxy_pass http://127.0.0.1:4002;
}
```

Figure 2.2 – NGINX server configuration file

made solutions (ps.kz). We ended up with this server configuration:

- Ubuntu 18.04 LTS x86
- 2Gb RAM
- 25Gb permanent memory

- Number of cores: 2

On this server, all the utilities and services we needed were already installed, with the exception of the Golang language compiler, which we later installed manually. When installing, we relied on the official guide for installing the compiler on a Linux system.

2.3 Database Architecture

After we have determined that we will use MySQL as our DBMS, we must also define

- How will access to the database be configured;
- What information will be stored in the database;
- How to build a database architecture;
- How to secure a database.

The database must meet the following requirements

- The database must have users with local and external access;
- The database must support two or more connections;
- The database will store information corresponding to the activity of backend applications, namely information about users of the service, links created by them and information about clicks on these links;
- The database should be normalized, but the architecture should not become more complicated;
- After the end of each development cycle, you need to run all backend applications using a local database connection;
- All sensitive information must be converted using hash functions.

The database was designed according to the scheme (Figure 2.3).

The database contains the following tables:

- "Users" - stores information about users, such as their input data, first name, last name, path to the file on the server responsible for their avatar, registration confirmation result and registration time;
- "Links" - stores information about links, such as their name, end address, display name, tags, link status and creation time;
- "Links-Users" - stores information about the relationship between users and links. The table is necessary to implement the possibility of transferring control over the link to other users of the service;
- "Apitokens" - stores information about the keys required to use the service by an API requests that implements the HTTPS protocol;
- "Requests" - stores information about link clicks, such as request headers and other metadata.

Foreign keys were used to create relationships between tables. With this architecture, it is possible to get information about clicks and links using only a unique key in the "Users" table, whether the user is the owner of the link or not.

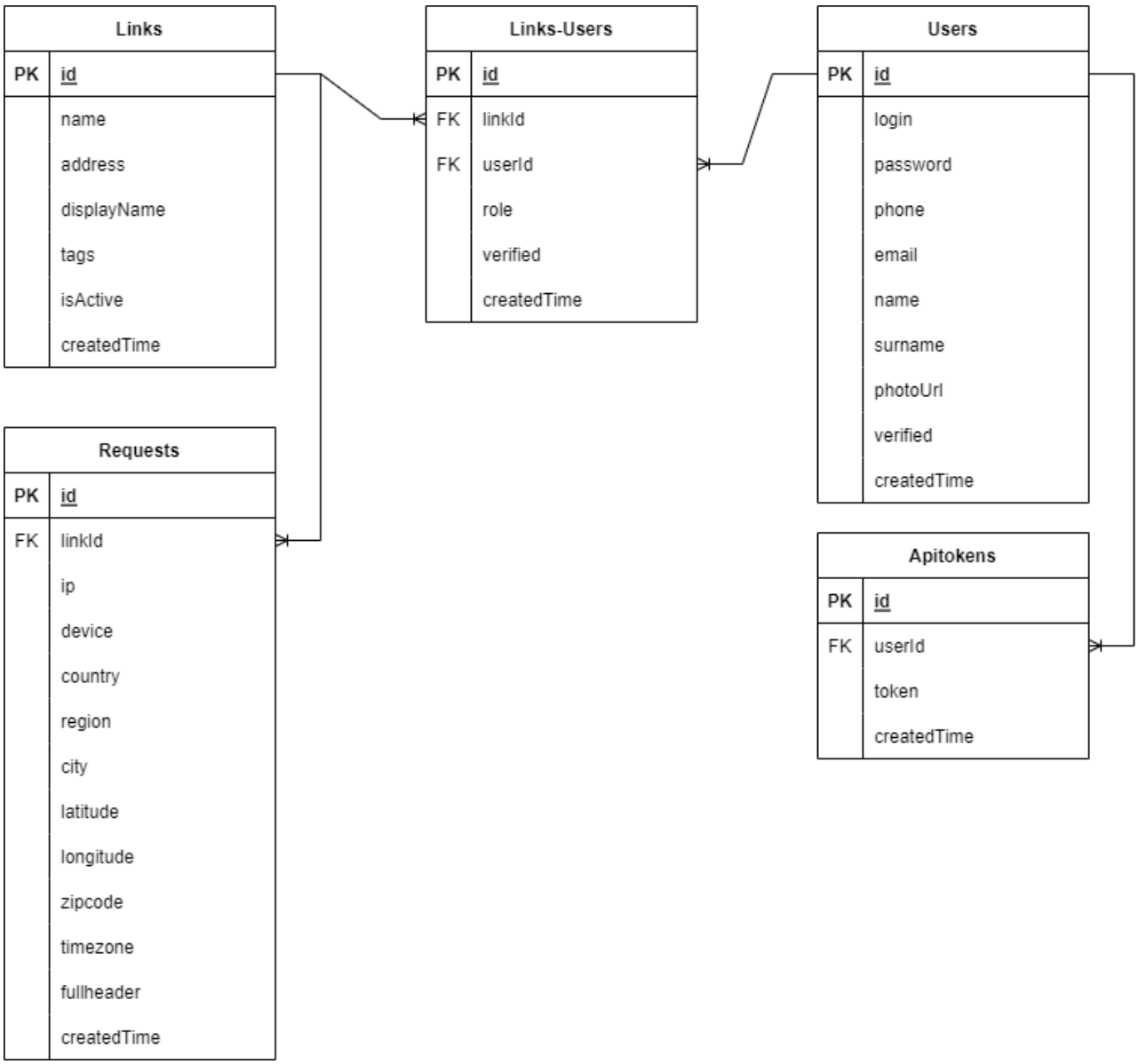

Figure 2.3 – Entity Relationship Diagram

This table architecture is not fully normalized, but it fulfills all the requirements of the system: there is no data redundancy and it is very convenient to work with it in the micro-service architecture of the system.

2.4 System Design

A system of backend and frontend applications must meet the following requirements:

- The system and its development must support the CI/CD principle. Even in case of errors, the services must work and not interfere with other system tasks;

- Since individual applications of the system can be updated and stop their work for a short period of time, it is necessary that separate and independent tasks should be performed in separate services;

- As a consequence of the previous point, it is necessary to develop a queue

algorithm for system tasks. With this algorithm, data will not be lost in the event of a short restart of individual services, but will be cached in the memory of the sending service;

It was decided to create a system of 3 backend applications and a frontend application (Figure 2.4):

- Frontend application is a web interface and platform for creating and working with links;

- Backend application that serves the frontend application and provides access to the service functionality through the API interface;

- Backend application which is responsible for redirecting requests to the original address of links;

- The backend application is responsible for processing IP addresses and getting metadata such as region, city, coordinates, and so on.

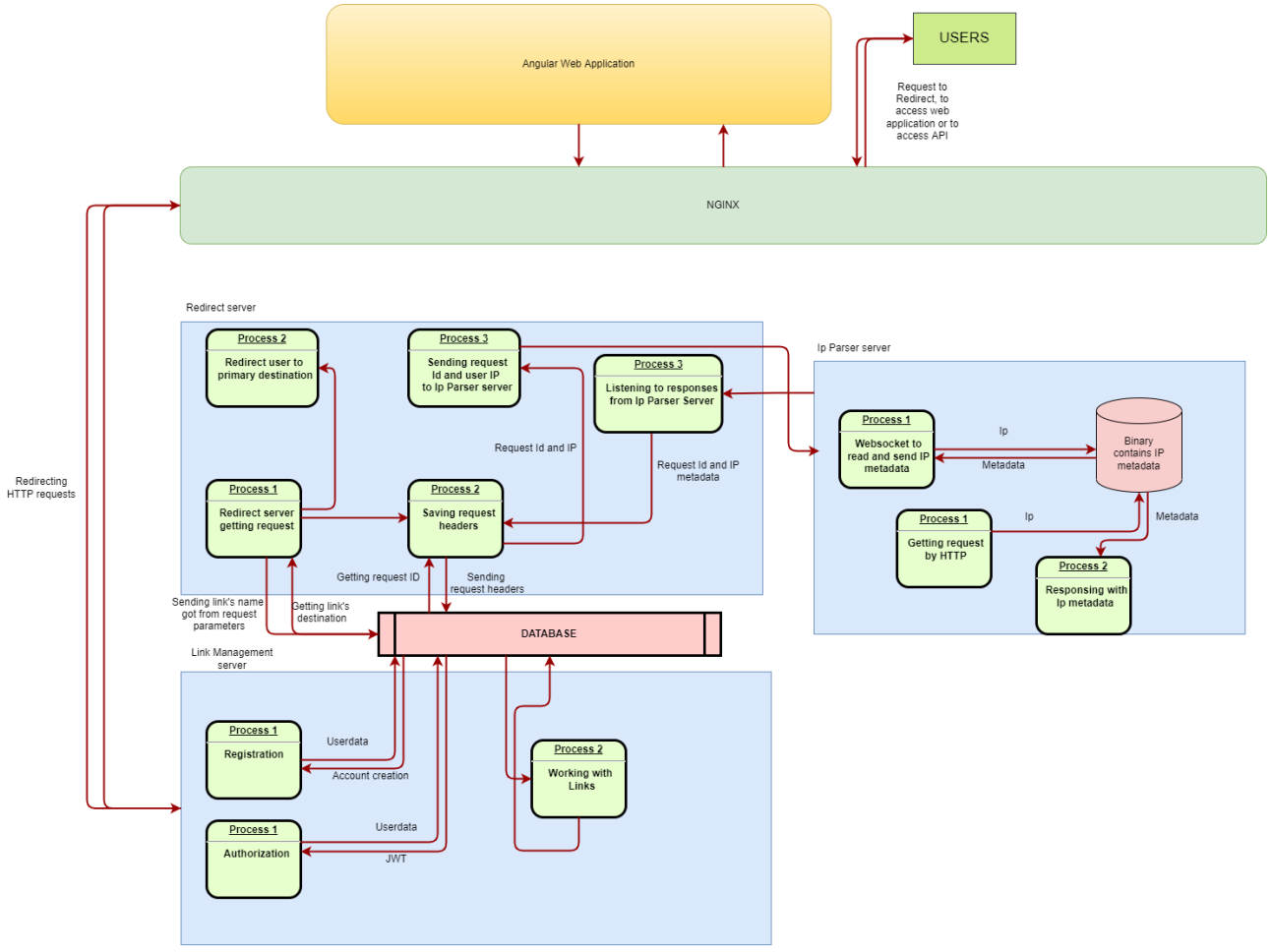

Figure 2.4 – System Design

2.5 Backend Application Development

2.5.1 Backend Application Structure

Since we need to start services as early as possible, we resorted to the universal structure of all backend applications. The application structure was based on the project structure from the book "Let's Go Further" by Alex Edwards [15]. (Figure 2.5) This material was studied by us during the 2nd and 3rd courses at the university in the Software Engineering course. Despite the similarity of the structures and development process, we had to change the structure of the application itself, as well as make changes to the authorization algorithm, routing, and others.

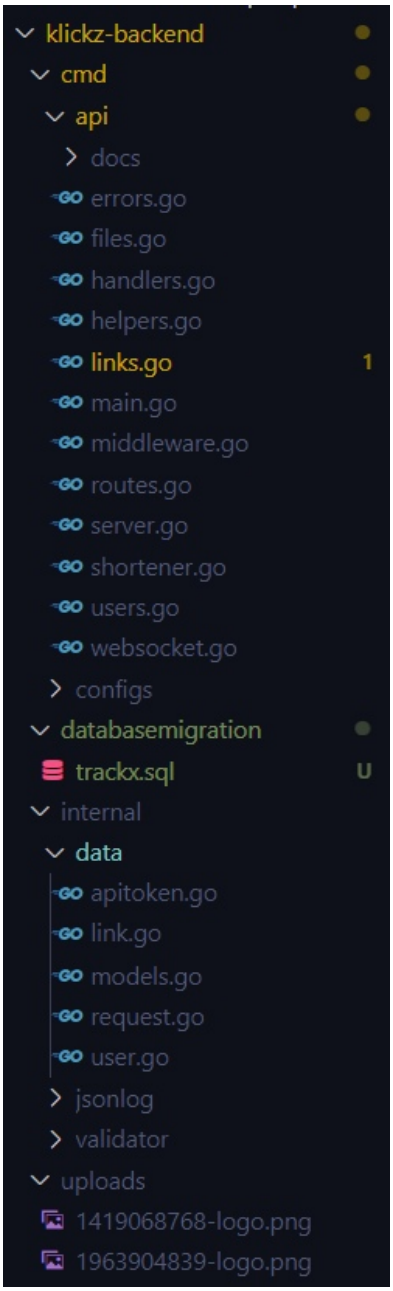

Figure 2.5 – Backend Application Files

The /cmd/api folder contains the files necessary for starting and running the server, such as connecting to the database, building the application, routing, authorization, encryption, file uploads, protection, and other functions that are involved in the execution of application tasks.

The /cmd/configs folder contains a file that is responsible for the initial configuration of the application and the data necessary for encryption and connection to the database.

The /databasemigration folder contains the database backup file.

The /internal folder contains files responsible for individual application components.

The /internal/data folder contains files of object structures and functions that implement SQL queries to the database server.

The /internal/jsonlog folder contains a file that contains functions that are responsible for displaying important messages and errors that may occur during application operation to the console.

The /internal/validator folder contains a file that contains functions responsible for object validation. The validation logic for each of the objects is stored in the file responsible for the object itself, in the /internal/data folder.

The /uploads folder contains files that have been uploaded by users. The paths to these files are stored in the database in a table that stores information about users.

2.5.2 Application routing

For routing, we used the "net/http" package imported from the "gin-gonic/gin" package. The features of the router from the imported package are:

- A ready-made set of a large number of middleware functions;
- Very convenient process of adding custom middleware functions;
- Combining handlers into groups, and applying individual middleware functions on groups;
- Simplified syntax;
- Different methods to return a response for different types of returned documents;
- Simplified serving of static files.

We created two groups in the router "web"and "api". Due to the structure of the application, both groups can use the same set of methods. Also, inside the "web" group, the "auth" group was created, which uses the Auth() middleware function. Thus, we protect a certain segment of the API interface (Figure 2.6). We also created custom middleware function to handle CORS Policy. CORS is a mechanism the browser uses to handle cases of when an application running at a particular origin wants to access resources from another application running at another origin. This middleware was added into router configuration, so that new

settings would be applied to whole API interface.

```
//this has no logger and recovery, so include it in middleware list if needed.
gin.SetMode(gin.ReleaseMode)
router := gin.New()

//list middleware that u want to include by default
router.Use(
    //enabling AllowAllOrigins = true
    //cors.Default(),
    CORSMiddleware(),

)

router.Static("/web/uploads", "./uploads/")

router.GET("/l/:name", app.Redirect)

web := router.Group("/web")
{
    web.GET("/healthcheck", app.Healthcheck)

    common := web.Group("/common")
    {
        common.POST("/login", app.Login)
        common.POST("/registration", app.Registration)
        common.POST("/startverification", app.StartVerfication)
        common.POST("/verify", app.Verify)
    }

    auth := web.Group("/auth", app.Auth())
    {...
    }
}

api := router.Group("/api")
{
    api.POST("/create", app.CreateLinkApi)
    api.GET("/get/:id/data", app.GetLinkData)
}

return router
```

Figure 2.6 – Router configuration

2.5.3 Authorization algorithm

For authorization, we implemented the creation of JWT keys that are transmitted to the client application using the server response via the HTTPS protocol. JWT token has such features as: storing data in the body of the token; the ability to be validated both on the backend application and on the

client application; no need to store in a database. However, it also has certain disadvantages. After creating a JWT token, it must be encrypted so that the client, having received a JWT token, cannot fabricate it manually. To create a JWT token, the "dgrijalva/jwt-go" package was used. Symmetric key algorithm was chosen because it is simple and fast, and in our case there is no need in validating token on client application. Our authorization algorithm uses the HMAC-SHA256 algorithm (Figure 2.8). During the authorization process, a request comes with user data. First you need to get the user object from the database. Next, we compare the password entered by the user, pre-converted using a hash function, and the password received from the database. If the passwords match, we generate a JWT token, encrypt it, and return it. If the passwords do not match, then we return a 401 error (Figure 2.7).

```go
func (app *application) Login(c *gin.Context) {
    var input data.User
    if err := c.BindJSON(&input); err != nil {
        app.serverErrorResponse(err, c)
        return
    }
    user, err := app.models.User.GetByLogin(input.Login)
    if err != nil {
        if err.Error() == "sql: no rows in result set" {
            app.InvalidCredentials(err, c)
            return
        } else {
            app.serverErrorResponse(err, c)
            return
        }
    }
    logIn, err := CompareHash(input.Password, user.Password)
    if err != nil {
        app.serverErrorResponse(err, c)
        return
    }
    if logIn {
        newJWT, err := GenerateJWTUser(app.config.Jwtkey, user)
        if err != nil {
            app.serverErrorResponse(err, c)
            return
        }
        encText, err := Encrypt(newJWT, app.config.MySecret)
        if err != nil {
            fmt.Println("error encrypting your classified text: ", err)
        }

        c.JSON(http.StatusOK, gin.H{"payload": encText})
        return

    } else {
        app.InvalidCredentials(err, c)
        return
    }
}
```

Figure 2.7 – Login Handler

```go
func GenerateJWTUser(jwtkey []byte, user *data.User) (string, error) {
    token := jwt.New(jwt.SigningMethodHS256)

    claims := token.Claims.(jwt.MapClaims)

    claims["authorized"] = true
    claims["login"] = user.Login
    claims["id"] = user.Id
    claims["exp"] = time.Now().Add(time.Minute * 60 * 2).Unix()

    tokenString, err := token.SignedString(jwtkey)

    if err != nil {
        return "", err
    }

    return tokenString, nil
}
```

Figure 2.8 – Generating JWT Token

2.5.4 Data models

Before writing functions for receiving and returning JSON, it is necessary to write the structure of objects and functions that implement SQL queries. The object structures must match the table structures in the database. In our case, it is necessary to write the structures of the following objects: User, Link, Request, Apitoken. When creating a structure, you also need to take into account the data types of the fields in the database, as well as additional options, such as whether the field is empty by default, whether the object is a unique key, and so on. Also, the name of this field must be written in the structure for conversion to JSON. An example of the structure of a user object (Figure 2.9).

Often, simple SQL queries are created that retrieve, create, update, and delete records from the database. However, sometimes you have to write complex or nested SQL queries in order to reduce the handlers and take full advantage of MySQL. Parameterization is also used to protect against SQL injections. For example, as in this query "SELECT id, name, address, displayName, tags, isActive, createdTime FROM links WHERE id IN (SELECT linkId FROM links_users WHERE userId = ? AND verified = 1) ORDER BY id;". With the help of this request, we extract all links that are available to the user with the id passed in the request.

```
type User struct {
    Id          int64     `json:"id"`
    Login       string    `json:"login"`
    Password    string    `json:"password"`
    Phone       string    `json:"phone"`
    Email       string    `json:"email"`
    Name        string    `json:"name"`
    Surname     string    `json:"surname"`
    PhotoUrl    string    `json:"photoUrl"`
    Apitoken    string    `json:"apitoken"`
    Verified    bool      `json:"verified"`
    CreatedTime time.Time `json:"createdTime"`
}
```

Figure 2.9 – User object structure

2.5.5 Microservice Architecture

There are few things that should be considered about IP-Parser Server and Redirect Server. Redirect server is working apart from other services because it performs the main purpose of the system - collecting data for system's users. This is why this backend application should always be online, and should not be restarted. However, the IP-Parser Server is completely about any time changes, because it holds the connection with a binary file which is the source of the IP metadata, which is then sent back to the database. So it is very important to make the communication between these applications not only fast, but reliable also. According to system design, there should be some sort of queue system that will hold all of the temporary data that need to be processed, in case of IP-Parser Server Failure or restart. In the beginning we developed a graceful shutdowns procedure for our applications, and put longer NGINX timeouts, so that any requests that will appear between end of graceful shutdown and restart of the server will be held by NGINX. But this algorithm is not very reliable, because we were not really sure what was happening to temporary data and were not able to manipulate it. This led to the development of a custom queue system. Initially we thought about implementing REDIS caching, but then decided to not complicate the system and implement a custom queue system using Golang channels. After redirecting the user to the primary destination address, we get the headers of the request, get IP address from the header, insert this information into the database, and then send the request id and IP address to the special channel. Simultaneously, the application is connected to the IP-Parser Server with the help of WebSocket Protocol. While this connection is open, the new thread is open. In this thread, the

application reads from the channel, and sends data over WebSocket connection. In the parent thread, the application listens for WebSocket messages, and when the message is marshaled, we use the id of the request to update its metadata (Figure 2.10). On the other side, IP-Parser Server is waiting for the WebSocket messages, containing request Id and IP. Retrieving metadata from IP is performed with the help of "ip2location/ip2location-go" package. This package uses binary files to get data relating to IP addresses.

```
func (app *application) connectToIpparser(cfg config) {
    ws, _, err := websocket.DefaultDialer.Dial(url, nil)
    if err != nil {
        app.logger.PrintError(err, nil)
    }
    defer ws.Close()
    defer app.connectToIpparser(cfg)
    //creating goroutine for sending requests to ipparser server
    go func() {
        for {
            request := <-app.requests
            fmt.Println(request)
            err = ws.WriteJSON(request)
            if err != nil {
                app.logger.PrintError(err, nil)
            }
        }
    }()
    //waiting for messages to update requests in database
    for {
        msgType, msgContent, err := ws.ReadMessage()
        if err != nil || msgType == websocket.CloseMessage {
            break                       // Close connection if client leaving
            app.connectToIpparser(cfg) // try to restart
        } else if msgType == websocket.TextMessage {
            var req Response
            err = json.Unmarshal(msgContent, &req)
            if err != nil {
                ws.WriteJSON(gin.H{"error": "can not unmarshal message"})
            } else {
                //saving metadata to database on retrieved id
                err = app.models.Request.UpdateMetadata(req.RequestId, req.CountryLong, req.Region, req.City,
                    req.Latitude, req.Longitude, req.Zipcode, req.Timezone)
                if err != nil {
                    app.logger.PrintError(err, nil)
                }
            }
        }
    }
}
```

Figure 2.10 – Sending request's information over WebSocket

2.6 Frontend application development

For this project we chose Angular as our platform and main framework. We had 5 potential options: vue.js, ember.js, react, angular and ionic. Out of 5 of these, the last one is most interesting, being originally intended for development of native

IOS/Android mobile applications since release of Ionic 4 became widely popular web developing platform. Additionally, we personally already had considerable amount of experience working with this platform and wouldn't have any trouble starting the development of the frontend part of the project strong from the get-go. Nevertheless, the choice fell on the Angular. While Ionic wan not a bad choice we opted for Angular for few but crucial reasons: Angular being older and more popular meant we would have more space for implementing specific features we had in mind for this project. Ionic just couldn't compete with upmost rich ecosystem of easily importable libraries Angular provides. Thousands upon thousands of npm packages covered every possible function or featured the project could need. Ionic being relatively young framework meant not it didn't cover a lot of things necessary things we needed or implemented them rather poorly. From personal experience, time and again we would search internet for implementation of certain feature only to hit a dead end in form of official response from ionic development team on their forum stating that the sought-after feature could not be achieved by any means currently, but it was on their list of potential additions for next version of Ionic. Suffice to say we deemed that unacceptable. To actually code frontend part of the application we used Visual Studio Code editor with just a little bit of Sublime Text. VS Code is UI friendly, easy to use and just like Angular possesses high level of customization and flexibility like Angular with its own collection of third-party libraries and plugins aimed to simplify and accelerate the coding process. Sublime text was used solely because we used it before and some code from my previous projects was needed. It is not even remotely close to VS Code in terms of flexibility and power.

2.6.1 Structure

Being an Angular project, it follows basic Angular 4+ structure, it consists of modules and components. Components are the most basic part of Angular app. It is essential to abide to single responsibility principle when working with Angular, every component is responsible for itself and has only one job. It keeps code structured and readable. Typical component consists of a corresponding html, scss, spec.ts and ts file.(Figure 2.11). Html template file holds html tags that are to be displayed on page. Scss file holds classes, ids and other selectors we use to define how objects in html should be displayed. When creating a new project Angular will provide user with choice:" scss or css?". SCSS is a CSS extension language dialect that supplements ordinary CSS with lots of features and abilities, provides coding logic and many other useful things. SCSS is not only a straight upgrade to CSS but an industry standard too. In spec.ts files we run different test if needed. They only execute while running tests and are not needed in deployment. Typescript flies hold the logic behind every component and defines its behavior. All the required imports, objects, variables, function is written and stored here.

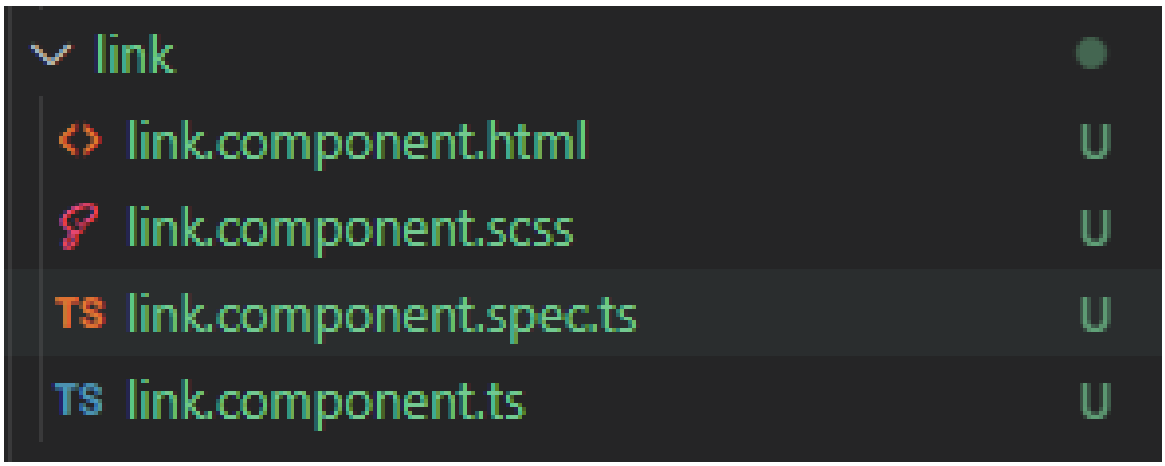

Figure 2.11 – Component structure example

2 default required and perhaps the most crucial for application work Angular modules are App-routing module and App module. It's clear from the name that App-routing module handles the transitions between components. It stores components urls and practically outsources the routing of the application for convenience and readability purposes. App module stores all the imports needed for application to function and imports it all in one file so other don't have to, again, all for the sake of convenience. Beside that all angular projects have a few other not rooted in components files like angular.json, package.json, style.scss, main.ts and sometimes more. These are not used or edited by typical developers for most of angular projects but can provide helpful utilities like style.scss providing overarching styling through whole project overriding everything else and holding global styling classes, ids and variables, which especially useful when working with third-party packages with prebuild styling. Angular.json holds current project specifications and configurations for production and development builds of the project. Package.json holds metadata of the project, its dependencies, and their respective versions. This project structure turned out to be quite complex with many different components and subcomponent that are displayed inside other components via routing module and angular material tabs. We will tackle the angular material later in text, but for now we will only explain tabs. Tabs like router display elements from different component in separate view, but they also allow for dynamic switching of said router content, for example to show different component elements. Also, modal windows aka Angular Material Dialog Module allows us to open Dialog component from the parent component. This in combination resulted in complex component rich web application.

2.6.2 Component by component

From the user perspective the web application can be divided into 4 main fundamental parts:

- Landing page. Our project's landing page is the first and only page new user sees when using our product. It uses the traditional modern web design choices to make it instantly familiar and as easy to navigate as possible. Whilst showcasing some of the 3D scenery made specifically for this project and to engage user even more to interact with website further;

- All the links. Page containing all the links and summary diagrams can easily be called our main page. Mainly because the fact that user will likely spend most of his time with our product here scrolling and examining his personal links and observing summary graphs driven by links performance. 3 Big graphs represent All time active links performance via doughnut chart, daily traffic through active links via doughnut chart and weekly performance of active links. By clicking on single link, a modal will open that will contains full display of traffic data our product provides. Daily activity via line chart, all time clicks by regions via pie chart, all time clicks by devices via pie chart and interactive world map showing all time clicks by location. Left part of modal shows link details and contains old link address, new link our service provides user for traffic monitoring, input containing in-application display name that exist purely for user convenience and that can be edited here, QR code of application provided link, pack of buttons for sharing and user inviting, link content preview. By clicking invite button a second modal window will appear prompting to input user login. The modal displays all the active users whose login fits the search input. Then invited person can see a new invite request in his profile;

- Profile page. Profile page contains 3 tabs: user account data with user profile picture that he can upload here, non-changeable login, phone number, changeable email, name, and surname; API tab contains input showing the personal API token that can be used to link our service to other projects. Just below there is a button to create or re-generate API token; Invites tabs show all other users links invite requests. Clicking on one will prompt modal window showing name and surname of inviting user if available, data of his account creation, original link address and time of this link creation. 2 actions are available to the user either to accept or reject the invite;

- Create a link page. This page only needs to elements to function. Input field for want-to-be-monitored link and button for submitting it. We only accept https secured links addresses;

2.6.3 Page design

When we just started designing the general look of the project pages integral part of the main components structure was creating hub area, dashboard component that would hold the tabs navigation bar and change the displayed components accordingly dynamically. We chose that component would slide left or right with slick animations while dashboard component in itself would have a

dynamically changing scenery or animation in the background giving the whole page a theoretically spectacular look and feel. We took some inspiration for it from UIX/UI Design of futuristic Desktops from sci-fi movie and games we know and love. Smaller components and their look we discussed together as a team and in the process of brainstorming left the good ideas and ditched less appropriate ones.

2.6.4 Data visualization

Data visualization is a crucial part of the identity and subsequently main functionality of the project. While back end handles receiving headers and data from links traffic, front end part focuses on delivering that information in best for user understanding manner. To handle the display of such data streams we used different types of charts and visual diagrams. And to display such diagrams in Angular there are to major ways to go about it. There is Chart.js and d3.js library. These two are the most popular charting and data visualization JavaScript libraries on the market. Both have their pros and cons. Main strengths of Chart.js are its set up time and built-in set of charts. (Figure 2.12). From personal experience We can tell that in time that took me to understand, implement and build all of the available charts from Chart.js chart types (Figure 2.13), with d3.js we would have a full page of code functions that we think we understand but that ultimately won't work for inexplicable reasons.(Figure 2.14). While being more powerful and flexible with different ways of data visualization d3.js feel very hard and clunky in comparison with Chart.js. (Figure 2.15). Chart.js provides limited number of charts for display but they are easy to understand, fast to build, come already animated and responsive. But to display something more special and complex than pie or line chart more time and effort would need to be spend researching and learning d3.js. In this project we use both to cover whole spectrum of data visualization. For example, to present to the user daily activity the link receives, clicking and redirecting to it in the last 24 hours span, the most elegant and frankly obvious solution would be to build a line chart, with its X-axis representing the amount of activity and its Y-axis representing the timeline of internet users clicks and re-directions to the link. While the amount of time and effort needed to fully utilize d3.js library is severe, investing in it can gain great benefits in form of unique in style and visuals complex yet understandable display of data to the user. Utilizing the svg tag allows d3.js to draw and display any visuals or data in any shape of form literally as long as the developers find a way to implement it, there are of course many online packages, sub-libraries, modules, blogs and guides to help with understanding and learning it, but rather unsettling portion of them tackle no longer viable templates and solutions for older versions of d3.js. That can result in features working in older web environment but not being viable in current workspace.

```
this.myChart24 = new Chart("doughnut24", {
  type: 'pie',
  data: {
    labels: this.label24,
    datasets: [{
      data: this.data24,
      backgroundColor: [
        'rgb(255, 99, 132)',
        'rgb(54, 162, 235)',
        'rgb(255, 205, 86)',
        'rgb(153,102,255)',
        'rgb(255,159,64)',
        'rgb(75,192,192)',
        'rgb(201,203,207)'
      ],
    }]
  },
  options: {
    plugins: {
      legend: {display: false}
    },
    maintainAspectRatio: false,
    responsive: true,
  }
});
```

Figure 2.12 – Chart.js pie chart code example

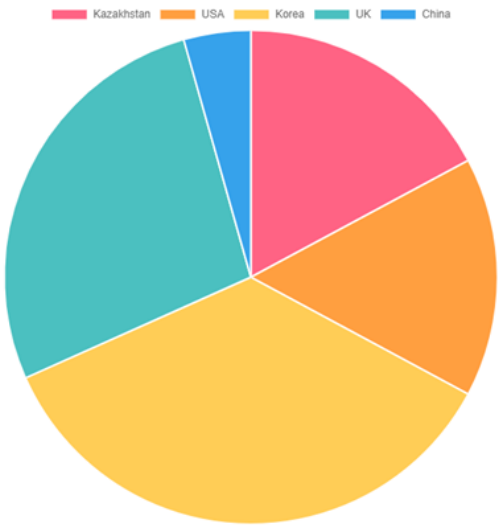

Figure 2.13 – Chart.js pie chart example

Pictures presented here show the difference and amount of effort required from the web developers to achieve the similar or functionally equal output. The leaning curve for d3.js is quite steep and can halt the development process for unpredictable amount of time.

```
graph24(id,data: any){
  this.buildSvg(id);
  this.addXandYAxis(data);
  this.drawLineAndPath(data);
}

private buildSvg(id) {
  this.svg = d3.select(id)
      .append('g')
      .attr('transform', 'translate(' + this.margin.left + ',' + this.margin.top + ')');
}
private addXandYAxis(data : any[]) {
   this.x = d3Scale.scaleLinear().range([0, this.width]);
   this.y = d3Scale.scaleLinear().range([this.height, 0]);
   this.x.domain(d3Array.extent(data, (d) => d.id ));
   this.y.domain(d3Array.extent(data, (d) => d.count ));
  this.svg.append('g')
      .attr('transform', 'translate(0,' + this.height + ')')
      .call(d3Axis.axisBottom(this.x).tickValues([]).tickSizeOuter(0)
      .tickSizeInner(0));
  this.svg.append('g')
      .attr('class', 'axis axis--y')
      .call(d3Axis.axisLeft(this.y)).call(g => g.select(".domain").remove());
}

private drawLineAndPath(data) {
  this.line = d3Shape.line()
      .x( (d: any) => this.x(d.id) )
      .y( (d: any) => this.y(d.count) );
  this.svg.append('path')
      .datum(data)
      .attr('class', 'line')
      .attr('d', this.line);
}
```

Figure 2.14 – d3.js line chart code example

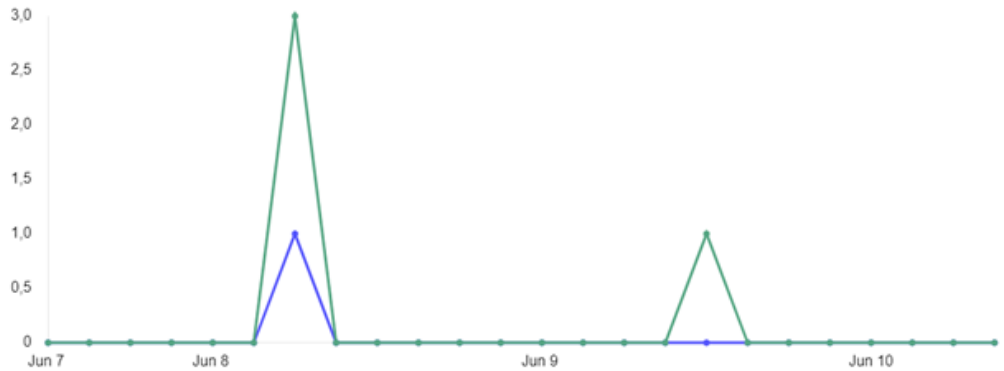

Figure 2.15 – d3.js line chart example

2.6.5 Connection with backend

For this project we created custom API-caller service that handled the requests sent from frontend to the backend part of the project and responses to these requests. (Figure 2.16). Here we store our API URL in form of variable that can

be easily changed if needed be and user JWT-token that is required all throughout the usage of our web application. (Figure 2.17).

```
sendPostRequest(data: any, url: string) {
  const httpOptions = {
    headers: new HttpHeaders({
      'Content-Type':  'application/json',
      'Accept': 'application/json',
    })
  };

  return this.http.post(this.const_url+url, data, httpOptions)

}
```

Figure 2.16 – API caller service request example

This JWT-token Is used when calling on methods that would only be accessible to authorized users only; like fetching the links tied to the user account.

```
sendPostRequestWithAuth(data: any, url: string) {
  const httpOptions = {
    headers: new HttpHeaders({
      'Content-Type':  'application/json',
      'Accept': 'application/json',
      'Akis-Jwt-Token': this.myjwt
    })
  };

  return this.http.post(this.const_url+url, data, httpOptions)
}
```

Figure 2.17 – API caller service request for authorized users example

2.6.6 Angular Material

Angular Material is a user interface component library, an implementation of Google's design specifications. [16] It helps with development of the Angular applications accelerating page code creation by providing user with a variety of elegant premade UI/UX components. It is built on modern web design principles,

supports portability, full responsiveness and provides developers with many other helpful utilities. Angular Material components library was essential in our project development, and elements from it are the foundation and core pillar of design and functionality of our frontend. MatTabsModule – is the module that allowed us to implement dashboard component how we imagined it on designing board. It sorted and organized content, in our case multiple components templates and allowed for switching between them. As the tabs component was meant to contain static content from single source as opposed to us providing it with router responses, built-in animations of transitioning did not work with our implementation. To negate that we had to create our custom router transition animations component. (Figure 2.18).

```
export const routeTransitionAnimations = trigger('triggerName', [
    transition('profile => links, links => createlink, profile => createlink', [
        style({ position: 'relative' }),
        query(':enter, :leave', [
            style({
                position: 'absolute',
                top: 0,
                right: 0,
                width: '100%'
            })
        ]),
        query(':enter', [style({ right: '-100%', opacity: 0 })]),
        query(':leave', animateChild()),
        group([
            query(':leave', [animate('0.8s ease-out', style({ right: '100%', opacity: 0 }))]),
            query(':enter', [animate('0.8s ease-out', style({ right: '0%', opacity: 1 }))])
        ]),
        query(':enter', animateChild())
    ]),
```

Figure 2.18 – Custom animation component code example

MatDialogModule – is the module that allowed us to implement opening of child components and passing data from component to component without navigating or leaving the rendered page. In fact, every link detail page is a modal component opened from links holding component. Usage of this module spares excess page reloading for user whilst allowing us to display additional information. Many more Angular Material modules were used to enhance the experience of using our product, improve visual quality and provide additional clearance like MatIconModule, MatProgressSpinnerModule, MatProgressBarModule, MatSnackBarModule, MatBadgeModule.

2.6.7 Bootstrap

Bootstrap is an open-source CSS framework for web development. It features numerous templates web development streamlining the process of web layout

coding with predefined styles classed. It is widely popular and has great support from the developers. By using Bootstrap in our Angular application we opened many possibilities from using third-party Bootstrap templates in out code to specialized bootstrap sub-libraries. Additional advantages of using Bootstrap would be its cross browser and device adaptability, allowing it to look good and not break on all kinds of devices, great responsive structure with easy to edit styling, and in my personal opinion the simplest and the best structuring grid system of any framework. [17]

2.6.8 Extra visuals

As many web developers before us came to the conclusion that animations and other kinds of eye-catching visuals grabbing users' attention just for a little longer greatly increase the chance of him spending more time on the web page, maybe even just to explore it out of interest. Users' attention instantly switches to the moving parts of the page upon loading so we can use it direct him to the points of interest like button suggesting to get started with our project or engage them with moving scenery. Even some basic animations when done right can greatly enhance user experience with web application, maybe even impress, prompting to visit again some other time. We had some knowledge and skills from previous projects working with THREE.js library. [18] It draws 3D graphics in form of scenes in browser. (Figure 2.19). Allowing for upmost impressive visuals. When combined with Blender, free open-source software for creating, texturing, and animating 3D models, the sky is the limit. Completely custom-made models playing custom made animations is a guaranteed way to grab user's attention.

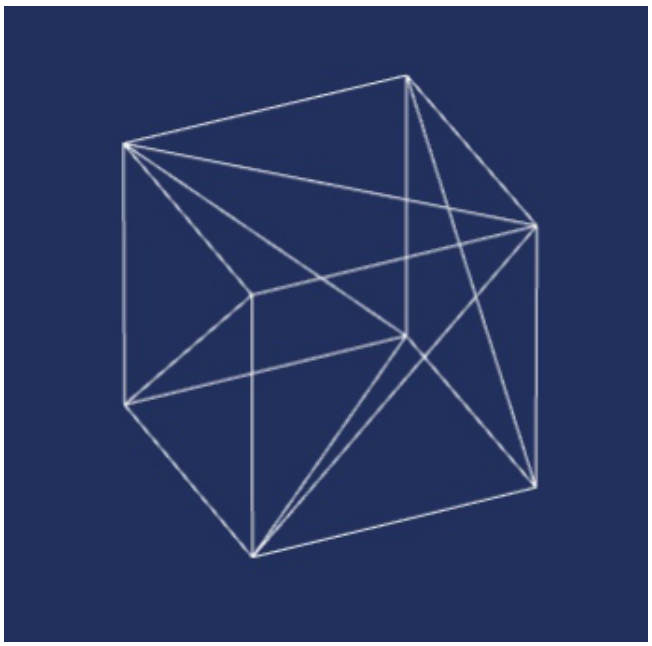

Figure 2.19 – Shape rendered in real time in browser example

2.7 Additional software used

Our team used several additional third-party software to aid with developing and deploying of the project. We used github.com and its desktop application GitHub Desktop to version control and collaboration between backend and frontend. Another major software application that helped us majorly is Postman. Postman is a free tool that allows to build and test API requests. Also, it has great Workspaces feature that allows to invite, collaborate, and share API requests between users. The utility and usefulness the Postman provided to our team and by association our project was indispensable. (Figure 2.20). Finally, we used FileZilla free FTP-client to connect to the hosting server and upload the application build, publishing the product online.

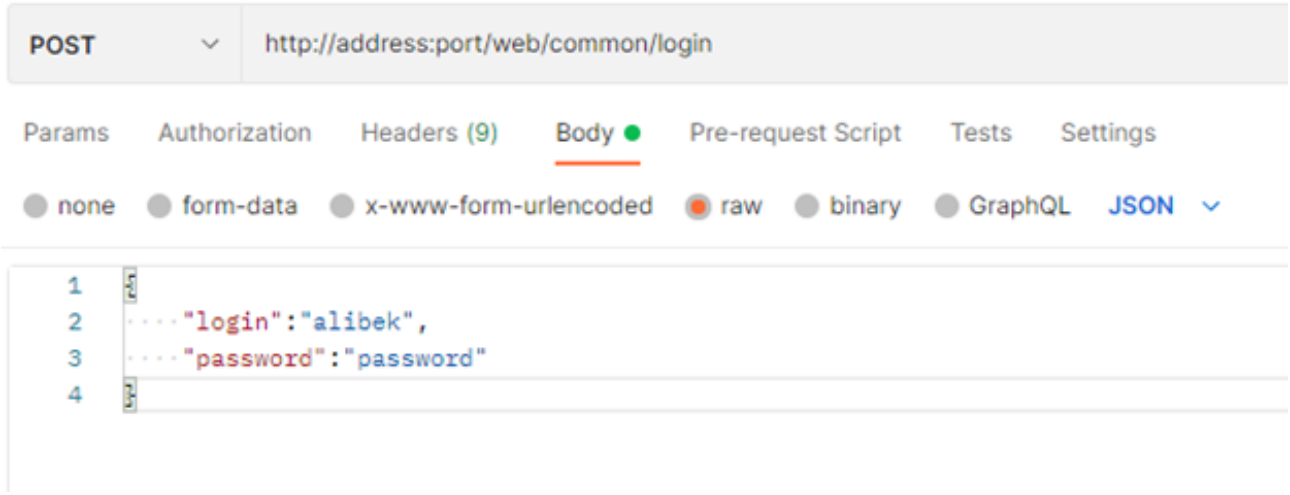

Figure 2.20 – Postman API request example

CONCLUSION

As a result of all the work done, we were able not only to launch a real product that can already help its first users, but we were also able to understand and, most importantly, apply new technologies and software development principles. As part of the development of the project, we not only improved our coding skills, but also found applications for such skills as critical thinking, team communication, market analysis, writing technical documentation, and others. The result of our work is a full-fledged web application that has gone through a full development cycle consisting of planning and requirement analysis, defining requirements, design the product architecture, building or developing the product, testing the product, deployment in the market and maintenance.

Further development. Although we have already written a huge amount of code and developed most of the applications, there is still a lot of work to be done. First, we need to improve and optimize the functionality that has already been created and launched. This includes such changes as optimization of the redirection algorithm, optimization of data caching, security improvements, web interface design improvements and addition of already written functionality to separate system segments. After that, we want to work on increasing the functionality, further developing the microservices architecture and implementing analysis through AI.

The solutions that were applied during development helped it to launch quickly and safely, but for us it is important not only how quickly and reliably the product works. One of the key requirements for the project was its scalability and the use of innovative technologies in the field of software development. Since we see a vast future for our product, it was important to design and build a product that not only attracts new ideas, but also maintains simplicity in the development process.

BIBLIOGRAPHY

1 *Herman, Carolyn.* Do you know your user? The importance of user research. — 2017.

2 *Ward, Janelle.* Why knowing your users is essential to your product's success. — 2022. https://blog.ribbonapp.com/why-knowing-your-users-is-essential-to-your-products-success/.

3 *O'Bot, Cody.* Creating a program architecture or how to design a stool. — 2016. https://habr.com/ru/post/276593/.

4 *Fielding, Roy.* Architectural Styles and the Design of Network-based Software Architectures / Roy Fielding. — USA: University of California, 2000.

5 *M., Shor Alexander.* Comparative analysis of approaches in the development of api web applications / Shor Alexander M. // *StudNet.* — 2020. — Т. 1, № 1. — С. 1–8. https://cyberleninka.ru/article/n/sravnitelnyy-analiz-podhodov-v-razrabotke-api-veb-prilozheniy.

6 *Suvorova, Marina.* PostgreSQL vs MySQL: which system is right for your business. — 2022. https://sbercloud.ru/ru/warp/blog/mysql-vs-postgresql.

7 *Alonecoder, Max.* Backend I/O performance: Node vs. PHP vs. java vs. go. — 2022. https://habr.com/ru/company/vk/blog/329258/.

8 *Peabody, Brad.* Server-side I/O Performance: Node vs. PHP vs. Java vs. Go. — 2022. https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go.

9 *Garnett, Alex.* Apache vs Nginx: Practical Considerations. — 2015. https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations.

10 *Goltsova, Ekaterina.* 10 differences between Apache and Nginx. — 2017. https://ekaterinagoltsova.github.io/posts/apache-vs-nginx/.

11 *Robbins, Jennifer.* Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics / Jennifer Robbins. — UK: O'Reilly Media, 2012.

12 *Adam, John.* The most popular JavaScript frameworks in 2022 – strengths, weaknesses and use cases. — 2022. https://kruschecompany.com/ember-jquery-angular-react-vue-what-to-choose/.

13 *Pattakos, Aris.* Angular vs React vs Vue 2022. — 2022. https://athemes.com/guides/angular-vs-react-vs-vue/.

14 *Murray, Nathan.* ng-book: The Complete Guide to Angular / Nathan Murray. — UK: CreateSpace Independent Publishing Platform, 2018.

15 *Edwards, Alex.* Let's Go Further! / Alex Edwards. — Australia: eBook, 2021.

16 *Kotaru, Venkata Keerti.* Angular for Material Design: Leverage Angular Material and TypeScript to Build a Rich User Interface for Web Apps / Venkata Keerti Kotaru. — USA: Apress, 2019.

17 *Jakobus, Benjamin.* Mastering Bootstrap 4: Master the latest version of

Bootstrap 4 to build highly customized responsive web apps / Benjamin Jakobus. — UK: Packt Publishing, 2018.

18 *Dirksen, Jos.* Learning Three.js – the JavaScript 3D Library for WebGL - Second Edition / Jos Dirksen. — UK: Packt Publishing, 2015.

Appendix A Code example Auth() middleware

```
func (app *application) Auth() gin.HandlerFunc {
return func(c *gin.Context) {
if c.Request.Header["Akis-Jwt-Token"] != nil {

// To decrypt the original StringToEncrypt
decText, err := Decrypt(c.Request.Header["Akis-Jwt-Token"][0], app.con
if err != nil {
fmt.Println("error decrypting your encrypted text: ", err)
}

token, err := jwt.Parse(decText, func(token *jwt.Token) (interface{}, 
if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
return nil, errors.New("SigningMethodHMAC checking error")
}
return app.config.Jwtkey, nil
})

if err != nil {
if err.Error() == "Token is expired" {
app.InvalidCredentials(err, c)
c.Abort()
return
} else {
app.serverErrorResponse(err, c)
c.Abort()
return
}

}

if token.Valid {
c.Next()
}

} else {
app.NotAuthorized(nil, c)
c.Abort()
return
}
```

```
}
}
```

Appendix B Code example for connecting to MySQL database

```
func openDB(cfg config) (*sql.DB, error) {
db, err := sql.Open("mysql", cfg.Db.Dsn)
if err != nil {
return nil, err
}

db.SetMaxOpenConns(cfg.Db.MaxOpenConns)
db.SetMaxIdleConns(cfg.Db.MaxIdleConns)

duration, err := time.ParseDuration(cfg.Db.MaxIdleTime)
if err != nil {
return nil, err
}
db.SetConnMaxIdleTime(duration)

ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second
defer cancel()
err = db.PingContext(ctx)
if err != nil {
return nil, err
}

return db, nil
}
```

Appendix C Functions that are used to generate short link

```
func sha256Of(input string) []byte {
algorithm := sha256.New()
algorithm.Write([]byte(input))
return algorithm.Sum(nil)
}

func base58Encoded(bytes []byte) string {
encoding := base58.BitcoinEncoding
encoded, err := encoding.Encode(bytes)
if err != nil {
fmt.Println(err.Error())
os.Exit(1)
}
return string(encoded)
}

func GenerateShortLink(initialLink string) string {
u, err := uuid.NewV4()
if err != nil {
panic(err)
}
urlHashBytes := sha256Of(initialLink + u.String())
generatedNumber := new(big.Int).SetBytes(urlHashBytes).Uint64()
finalString := base58Encoded([]byte(fmt.Sprintf("%d", generatedNumber)
return finalString[:8]
}
```