TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Oleg Berezin 175107IDDR

Architectural Solution Implementation: a Manufacturing Enterprise Case Study

Diploma thesis

Supervisor: Nadežda Furs
MBA

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Oleg Berezin 175107IDDR

Arhitektuurilise lahenduse väljatöötamine: tootmisettevõtte juhtumiuuring

Diplomitöö

Juhendaja: Nadežda Furs
MBA

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Oleg Berezin

20.04.2021

Abstract

The aim of this thesis is to design and implement an architectural IT solution for a stone manufacturing enterprise in order to enable the company undergo digital transformation, optimise internal business operations and automate manual processes. Based on analysis and case study, author elicited business and technical requirements for an architectural solution design. The implemented solution was then evaluated against the original requirements of the project.

The solution required the infrastructure design and the implementation of a web application to form the foundation for subsequent developments.

The server infrastructure required to ensure stable and efficient operation of services, using containerisation techniques in the cloud computing environment. The web application should serve as a practical service for organising and displaying stone products and other data, both as convenient decision making tool internally and when communicating with partners.

The project has significantly optimised internal company processes, automated manual labour, provided a tool for communication with partners for easy projects and materials demonstration, and helped to attract new customers.

This thesis is written in English and is 35 pages long, including 7 chapters, 10 figures and 3 tables.

Annotatsioon

Arhitektuurilise lahenduse väljatöötamine: tootmisettevõtte juhtumiuuring

Selle lõputöö eesmärk on arendada ja rakendada kivitootmisettevõttele IT-lahendus, mis võimaldaks ettevõttel üle minna digitaalsetele lahendustele, optimeerida ärilisi toiminguid ettevõtte sees ning automatiseerida manuaalseid protsesse.

Autor on analüüsile ja juhtumiuuringule tuginedes välja toonud ärilised ja tehnilised nõuded struktuurse IT-lahenduse arendamiseks. Seejäärel on rakendatud lahendust projekti esialgsete vajadustega kõrvutades hinnatud.

IT-lahenduseks oli vaja luua infrastruktuur ja veebirakendus, et rajada alus edasisteks täiustusteks.

Serveri infrastruktuur on vajalik, et tagada teenuste stabiilne ja efektiivne osutamine, kasutades konteineri tehnoloogiaid pilvandmetöötluse keskkonnas. Veebirakendus võimaldab kivitooteid ja muid andmeid sorteeritult kuvada, olles nii kasulikuks tööriistaks otsuste tegemisel ettevõtte sees kui ka partneritega suhtlemisel.

Projekt on oluliselt optimeerinud ettevõtte sisemisi protsesse, automatiseerinud manuaalseid töid, andnud vahendi projektide ja materjalide hõlpsaks tutvustamiseks partneritele ning toonud juurde uusi kliente.

Lõputöö on kirjutatud inglise keeles keeles ning sisaldab teksti 35 leheküljel, 7 peatükki, 10 joonist, 3 tabelit.

List of abbreviations and terms

API	Application Programming Interface
APM	Agile project management
Application server	Software upon which web or desktop applications run
CDN	Content Delivery Network
CLI	Command line interface
CMS	Content management system
Container Registry	Server side application for storing, managing and distributing container images
CORS	Cross-origin resource sharing
CSRF	Cross Site Request Forgery
Database schema	A structure in formal language representing design of the database
Digitalisation	The use of digital technologies to change a business model and provide new revenue and value-producing opportunities; it is the process of moving to a digital business *(Gartner Glossary)*
DoS/DDoS	(Distributed) Denial-of-Service
DSRM	Design Science Research Methodology
ERM	Enterprise Resource Management
IA	Informational architecture
IaC	Infrastructure as code
JSON	JavaScript Object Notation
LRU	Least recently used. Cache policy that discards the least recently used items first
MO file	*GNU gettext* Machine Object compiled translation file for production use
NoSQL	Non tabular databases, storing data differently than relational tables
PRNG	Pseudorandom Number Generator
S3	Simple Storage Service, data object storage with a web service interface
Schema	Database schema is a structure in formal language representing design of the database

SF	Stone factory (in the case study of the thesis)
SME	Small and medium-sized enterprise
SPA	Single Page Application
SSL	Secure Sockets Layer
Staging environment	Staging environment is a production-like replica environment for testing purposes
Transpile/transcompile	To compile (source code) by translating from one source programming language to another, producing translated source code in the other language
TTFB	Time to first byte
UI/UX	User Interface/User experience
URI	Uniform Resource Identifier
Viewport	A polygon viewing region in computer graphics
VPN	Virtual Private Network
VPS	Virtual Private Server
WSGI	Web Server Gateway Interface; a calling convention for web servers to forward requests to web application written in Python

Table of Contents

List of Figures

List of Tables

1 Introduction

With the significant growth of the information technology market segment, bringing a large number of digital goods and services to an ever wider range of consumers, traditional industrial companies are faced with the opportunities to facilitate the innovation process via digitalisation [1] .

Migration from traditional to digital business processes allows manufacturing companies to organise information about their own operations, integrate them into a single system, and significantly improve enterprise resource planning. Integration of information about production and business processes creates the prerequisites for their further automation [2] . As a result of automation, the productivity of the company increases significantly [2] . That in turn allows the company to save time, optimises efforts and resources. Thus, the company reduces the number of errors in making production and management decisions.

In addition to the production transformation, the digitalisation of enterprise services allows the company to reach and communicate more effectively [3] with current and new customers as well as partners in a digital environment, bypassing geographic, time and linguistic constraints. Digital services contribute to a better user experience and brand reputation [3] . Formalised through digitalisation, or even automated, digital interaction with partners allows the company to gather and aggregate data for a detailed analysis of user and partner requirements, as well as to analyse the strengths and weaknesses in supply chains, enabling a more efficient allocation of internal resources.

As a consequence of successful digitalisation, in the longer term enterprises compensated by efficiency gains and are able to adapt more successfully [4] to changes in their industry landscape.

For small and medium-sized industrial enterprises (SMEs), the digitalisation process can be accompanied by problems. These may include:

- the complexity of planning the enterprise transformation process;
- the lack of expertise for the effective implementation of the transformation;
- the lack of the necessary budget to design, implement and support the solution.

Successful digitalisation requires an investment in expertise and technology acquisition in order to implement an adequate and effective solution. A small and medium-sized industrial enterprises from traditional non-technology sectors often have limited resources and capabilities for technological experiments [4] , hence reluctant of acquiring expensive third-party technology support or an in-house staff of software development specialists. However, investments in digitalisation often pay off in the mid to long term [2] [4] , therefore industrial businesses need to have a long-term plan for the development of an enterprise with a focus on digital technologies as the main form of organising internal processes.

This thesis aims to develop an architectural solution for a specific factory in Estonia that processes and creates natural and artificial stone products. The proposed solution will allow the enterprise to have a reliable, affordable and easy to maintain and manage infrastructure, consisting of integrated network, server, as well as custom and third-party software solutions. The final software solution will allow an enterprise to manage a database of its product objects with a lot of static assets through a custom web application using an open source industrial-grade image processing software. This solution is used internally within the enterprise for making management decisions, as well as communicating with customers and partners.

The implementation of this case study solution covers the current real-world requirements of an operating commercial industrial enterprise. The final project expands the functionality of the company's interaction with customers and industry partners, as well as modernises business processes within the company and automates manual work. The proposed solution can meet the needs of a wide range of industrial companies with similar universal needs, and demonstrates the decision-making approach in implementing the architectural solution, potentially useful in developing other software systems.

The business stakeholders case study survey and business processes analysis were used as methodology for designing the architectural solution. After defining the requirements and constraints of the enterprise, a systematic literature overview as well as the personal experience of the thesis author was applied. The result of the analysis was a technical solution implementation based on the collected data in accordance with the initial requirements of the project. The theoretical study uses selected peer-reviewed and other publications on the subject, as well as software engineering industry best practices and frameworks.

The thesis is organised as follows: after the introduction, *Section 2* describes the problem identification and motivation. *Section 3* covers solution analysis. *Section 4* presents the implementation solution. *Section 5* assesses the implemented solution. *Section 6* discusses further developments. Finally, *Section 7* concludes the thesis by highlighting key results.

2 Problem identification and motivation

In this section, author outlines the problem overview and motivation for the case study. The company in the case study is a stone factory enterprise (hereafter will be referred as SF).

2.1 Enterprise background

SF is a natural stone fabrication company producing high quality products of natural and engineered stone. The manufacturing enterprise has established a modern stone processing factory in Tallinn, Estonia with advanced highly-specialised CNC machinery. Factory exports flat stone slabs from around the world in order to supply project-based stone products to a wide range of customers, primarily focusing on personalised solutions for hotels, villas, private apartments, restaurants and cruise ship interiors in accordance with architects' project design.

Factory was established in 2013 and employs approx. 20 employees, including stone installation specialists within several European countries. The size of the company corresponds to the SME classification by the European Union definitions[1].

2.2 Problem overview

The SF spends a large amount of manual work and resources for cataloguing stone material samples and portfolios of finished projects. It also takes a lot of time communicating with customers through face-to-face meetings and phone calls. The company was planning to significantly increase the number of sales managers as an alternative to digitalisation of the enterprise.

1 SME as defined by the European Union: https://ec.europa.eu/growth/smes/sme-definition_en

2.2.1 Current state

Prior to the implementation of the solution designed and developed as a result of this thesis, the SF used a predominantly simple static website that allowed the company's employees to add basic information about stone samples to the database through a web interface.

Previous web application allowed to make a query to the database to retrieve basic information about all stone samples, the resulting JSON object from the back-end of web application was translated into a list of readable items on the web page. The website displayed the products information as an extended list on a web page, not allowing editors updating or changing data about stone objects, as well as users to finely query data, viz. filtering categories of products, searching and sorting.

Previous web application did not allow uploading and linking photos to the items in the database. Images for specific stone samples had to be stored separately and published on the page manually without any entity relations.

Large images are the main content of the web application, both for the old version of the site and for the newly developed architectural solution. The old website was not optimised to handle a large number of heavy images. There was no functionality for automatically scaling images either to fit different resolutions of client-side screens, nor down-scaling images in the back-end of the web application to optimise web page load time.

The old website was not optimised for fast loading time with a low latency for the regions of the target audience. The website demonstrated low performance on optimisation measuring tools[1].

The shared server platform hosting website had successfully managed to serve the old web site, but occasionally got service quality slowdowns due to, presumably, the increased CPU load of neighbouring users of the shared server.

1 Google PageSpeed Insights, Gtmetrix, Pingdom and other external services were used to measure web-application performance

2.2.2 Operational issues of current state

The company was in the process of deeply changing its internal business and operational processes. The company's management has decided to take advantage of the transition period to perform the company's digitalisation.

Old processes within the company did not scale up, did not meet new functional requirements, hindered the growth of the enterprise and increasingly required more manual labour. This led to high operating expenses of a small company with limited resources, made it cumulatively complicated to work with partners, did not allow to find customers and new partners online outside Estonia, and ultimately — led to reduced abilities to effectively compete on the European market.

2.2.3 Potential conventional non-digital solutions

Partially, the potential solution of some problems facing the enterprise was the expansion of staff. According to the company's estimates, manual labour would allow to postpone some of the above-mentioned business growth problems for some time, but only to a small extent.

Another potential solution would be to reorient the company towards working with a strictly defined narrow circle of partners and clients, where all the interaction and communication work would be carried out by employees manually. But such a solution would not have solved the problems of optimising and automating the enterprise's internal processes, and would have dramatically limited the company's commercial opportunities for growth.

Both potential solutions were rejected by SF's management as inefficient and sub-optimal.

2.3 Solution objectives

There is no single universally accepted approach to architecture planning, however, there is a general consensus that the architectural solution design should begin with an understanding of the business objectives of the enterprise [5] .

A period of growth and rapid transformation of processes within the company introduces particular challenges for planning and accepting a decision on software architecture, as the solution should be reliable and optimal for the wide range of tasks and processes that the company may require in the future. As asserted by Clements et al. [5] , early stage architecture design decisions are likely to have great impact on how the system will meet its quality attributes.

Adopting Agile practices by non-software SMEs is observed to be complex, however industrial SMEs are able to successfully integrate selected properties of APM, as discussed by Žužek et al [6] . Due to the SF's inability to predict future business processes in a guaranteed manner, the company uses selected properties of Agile-like methodology as their planning and decision making strategy. The company defines a long-term development strategy, but makes short-term plans, iterating them if they do not produce the desired results.

It was decided to adapt the IT development workflow to the such of the manufacturing enterprise, hence identify the projected short- and medium-term goals, take into account the long-term ambitions of the company — and use this data to rapidly, iteratively design and continuously deliver solutions, according to the requirements of particular software services.

The company needed to design a robust network infrastructure and a set of software solutions that would provide a reliable and flexible foundation for future developments. Since the SF is undergoing the period of rapid growth and transformation, the architectural solution was required to be minimalist, versatile and modular. Designed in a such way, it should be a reliable foundation for a number of current and future software services with an unknown life-time period of service work and performance load.

2.4 Scope

This thesis provides a design and development process overview of an architectural solution for a private manufacturing enterprise. It examines only selected parts of the whole system within the framework of the architectural requirements of the current thesis. It aims to demonstrate how the development of custom software web services in conjunction with the prior developments, and use of third-party free open source solutions – allows the SME factory to ensure the operation of the software system with a limited technical support, with optimal performance for serving heavy static data (e.g. images), and with minimum operating costs.

This thesis does not provide full and/or sensitive proprietary code for custom enterprise software solutions. Does not discuss the specific commercial amounts of money and other values invested by the SF enterprise on developing, maintaining or supporting the solution.

Current thesis does not cover factory automation (e.g. IoT); does not provide the process of testing software due to a wide variety of topics; does not provide in-depth security analysis and penetration testing of the network infrastructure and software solutions; does not provide a comparative analysis for prior software solutions within the enterprise.

Some prior to current project technology stack solutions have to remain for backwards compatibility and due to the professional qualifications of the third-party contractors involved in the development, as well as external dependencies of current system outside the scope of current thesis (e.g. Python Flask framework and MongoDB).

As this thesis focuses primarily on the practical aspects of developing a software architectural solution for a project, the author focuses more on the software engineering than on the business and server infrastructure aspects.

2.5 Author's role

Author communicated and processed stakeholder’s requests. Discussed, observed and attempted to understand the aspects of requirements (survey was performed). Built use cases, diagrams, user personas, informational architecture diagrams and other models to validate objectives; documented and approved final requirements; developed highlighted in the current thesis solutions, and supervised the technical progress out of the scope of current thesis.

Architectural software solution presented in current thesis is currently operating in production environment.

The author assessed alternative technology stack and third-party solutions.

The author performed peer-reviewed resources analyses in order to explore similar case studies and related problems.

The software solution uses containerisation, follows the 12-factor app as a methodology, has no vendor lock-in and can be easily ported between different cloud providers. Each service provides brief documentation for easy software application scaffolding. The presented architectural solution serves as a foundation for further software development for the future needs of the enterprise.

Author took a lead software developer role in a software engineering consulting company. Author was contracted to design and implement the complete IT solution to meet the SF’s requirements.

3 Solution analysis

The plethora of available methodologies and techniques for architectural analysis, as shown by Dasanayake et al [7] , has not led to their widespread use among software practitioners involved in designing IT system architectures. Software developers tend to use their own structured approaches that share similar characteristics with some of the systematic techniques rather than formal [7] .

The author has partly relied on his own experience and analytical approach, while in some aspects not entirely formal, the chosen techniques and approaches were grounded on scientific and industry standards; as well as academical integrity was sought while performing analysis to get a holistic view of organization requirements for designing software architectural solution.

The process of solution analysis was divided into two subsequent stages:

- business analysis in close collaboration with the factory's stakeholders; and
- technical analysis based on the information received from the business analysis.

3.1 Business analysis

The SF had not previously conducted a formal enterprise architecture analysis of its business processes and did not have any formalised artefacts applicable to extract specific system requirements. To design a software solution architecture, the author applied DSRM approach as a well grounded in existing literature and practical methodology for information system research [8] . The author followed six steps defined in design science process: 1. problem identification and motivation, 2. definition of the objectives for a solution, 3. design and development, 4. demonstration, 5. evaluation, and 6. communication [8] .

Steps 1 and 2 were studied via stakeholders survey. Step 3 involved a systematic review of the related literature, as well as the application of the author's personal engineering experience, where the solution had to satisfy the requirements and constraints of the project. Steps 4, 5 and 6 were carried out in cooperation with stakeholders and iterated as needed. At each step, the author documented intermediate results, which in turn were validated by the stakeholders.

3.1.1 Stakeholder survey

The author used private communication and a series of surveys to identify qualitative and functional requirements for the project. The block of questions in surveys was divided into categories, where the questions covered various aspects relevant to the enterprise. In the course of this work, the author and the stakeholders iterated the surveys, modifying them so as to eliminate irrelevant and to ensure that the remaining questions covered the project requirements as fully and precise as possible.

The company profile survey consisted of two categories of questions: current company state (company profile, products and services; customers and sales; competition) and target company state (company prospective; customers and sales; competitions). Each category listed 3-15 questions specific to SF’s profile.

Another survey was focused on personas and user journey analysis [9] , which was later used to design the system requirements, but also to develop the UI/UX workflow design for the web services interfaces. This survey consisted of following categories: customer persona, customer organization, online behaviour and preferences. Each category listed 8-18 questions specific to SF’s profile.

3.1.2 Software development proposal

The individual specific requirements of the project were provided by the stakeholders of the enterprise in the course of personal communication via common business software development project proposal. On the basis of these requirements, the author drafted documentation and a diagram to agree on the final solutions.

3.2 Quality requirements and attributes

As a result of the surveys and provided individual specific requirements analysis, the author has identified the following qualitative requirements and attributes for the project:

- The solution as a whole must be reliable and productive. Reliability is characterised by stable uptime and data integrity. Performance is measured by the latency to render web pages, amount of time it takes to perform functional tasks by a software, as well as by the third-party web performance measurement tools (e.g. Google Pagespeed Insights). The components of the proposed solution should be well established in the industry.
- The solution should be modular. This should ensure the functionality of the software architecture in an environment of rapidly changing enterprise operational requirements, where it is difficult to predict in advance the demands of future software solutions and the load on these components.
- The solution must not be dependent on any vendor for its operability in the long-term, such as: the cloud infrastructure and its services or the proprietary licence of system component. Otherwise, in the case of a vendor lock-in or other reasons for not being able to continue using a service or system component, the architectural solution must be suitable for quick and budget replacement with a functional alternative.
- The system (components) must be designed and documented to be easily deployable on developer, staging and production environments, as well as capable of being easily migrated and deployed to another cloud computing environment.
- The solution (components) should be accessible for onboarding by external software development contractors.
- The solution must be reasonably budgeted for development and maintenance costs. As noted earlier, traditional manufacturing SMEs do not have a lot of financial capacity for high-tech IT resources.

- The solution must be optimised for customers of the target audience. As the web-application predominantly operates with large images, it is important that customers from different target regions, using different devices and network carriers, have swift and reliable access to the SF's product catalogue and projects list. It is also important to ensure that the system runs optimally during industrial expositions and trade shows, when the number of customers, and therefore the load on the system, increases.

- The solution is not critical to the high load caused by the excessive amount of concurrent users, as the company deals with a limited number of high-margin projects at the same time. However, this does not apply to the load concern of serving large amounts of heavy static data (images) to each user.

3.3 Functional requirements

The scope of current thesis does not allow for a complete detailed list of functional requirements for all services of the whole system operating in production. The key high-level requirements for the selected components covered in this thesis are listed below.

3.3.1 Web-application

The most important project requirement for the company is the development of a custom web application. This service is the centrepiece of an information system that could be used both by the SF’s managerial employees for work and decision-making within the company, and as the main tool for interaction with partners and customers. The architectural solution must consider future prospective developments for the project. The web service, amongst many others, in shortened and combined form, had the following functional requirements:

- The public web application serves as a product (stone sample templates) and project (recent and current works in customer’s premises) catalogues. Communicates information about the company, products and services. Accepts contact data and/or project inquiries.

- The private area of the web application serves as an administration control panel for managing the content of the application's data (stone sample profiles, projects, employees data, etc.)
- The high-performance image server is integrated into web-application, which is capable of serving large static images to the users, according to user's device screen resolution.
- Web-application is optimised in terms of latency and performance when displaying a large number of very high-resolution images.
- Web-application provides reasonably high level of security.
- (Future prospective) The private area of the web application will be expanded to a full-fledged project calculation and administration system for a range of private partners. This system will allow SF's partners to compose highly detailed architectural/interior projects with fine requirements, based on the resources and services currently available in the factory; keep a track of recent project requirements items, allow to duplicate and reuse them; export/import data (in PDF documents, etc.)
- (Future prospective) Integration of web-application resources (including partner's private system for inquiring projects) with the external ERM system via shared APIs.

3.3.2 Image server

The web application should be closely integrated with the image scaling system. This image server should have high-performance and be secure (protection against DoS attacks). The functional requirements of an image server include:

- Operate with JPEG files of large resolution and size (up to 25 megapixel or 5000×5000 pixels image resolution; up to 16 megabyte file size).
- Scale source images to specified by client's browser defined screen resolution.
- Cache the scaled images to reduce CPU load.

- Provide high performance and security.
- Do not depend on third-party hosted services.

3.3.3 Infrastructure

The infrastructure needs to be designed to meet the objectives of its current and potential services and tasks. The functional requirements include:

- Operational. Enables stable operation of the web application and other services.
- Portability. Operates in a multi-cloud environment, allows easily deploying services within different cloud environments.
- Geo-location. Cloud computing providers have to provide data centre locations in Northern and Western European regions, close to target audiences.
- Programmable. Cloud computing providers have to provide API and CLI-based management for the future potential infrastructure deployment automation, e.g. integrating IaC provisioning.
- Snapshots. Cloud computing providers have to provide server disk snapshotting service.
- Secure network. Secure interconnection between instances within and outside cloud computing providers.
- Back up strategy. System data backup automation and data integrity validation.
- Keep the operational costs for the information system as low as possible.
- Developers should be able to re-create application's functionality replica on a personal development machine and staging environment servers.
- CI/CD, container registry integrations.
- Servers performance tuning and security hardening.
- Metrics and logs aggregation.

3.4 Technical analysis: architectural approach

This section describes the decisions on technology stack and the rationale for selecting specific system components to implement the solution.

The solution must be implemented within a reasonable period of time and budgetary constrains. The subsequent technical support and further developments must also be approachable and easily understandable for a wide range of current and potential software developer contractors. Thus, the proposition of a microservice architecture was rejected as inappropriate and sub-optimal, since the individual services of current system are already arranged in separate modules, where services are grouped by domain-specific purpose (e.g. web application is coupled with application server, image server and reverse proxy of that group). The decision was made to build the web application as a modular monolith architecture.

3.5 Technical analysis: web-application decisions

3.5.1 Programming language, framework and application server

As noted in Scope (*Section 2.4*), Python and Flask micro-framework have to be used due to the current technology stack and professional qualifications of other developers involved in the project.

Flask is a Python WSGI-based lightweight framework, using blocking socket I/O. While blocking web frameworks are lately generally accepted as subpar in terms of request performance [10] , under the given requirements, this solution is adequate for the task. This is due to the expected low load of concurrent users[1] to the web application, as SF provides services for a limited number of concurrent partners. Also, the web application is being developed as an early version, a prototype to validate enterprise business tasks. The Flask micro-framework, being minimalist, is a convenient tool for an experienced development team to quickly prototype an operating solution that provides REST API and solves functional objectives. The author has considered

1 Concurrent users (CCU) is the total amount of simultaneous connections to a service in a predefined period of time

other options: Django, another Python-based blocking web framework and Go programming language without frameworks.

As researched by Ghimire [11] , “Django can be best fit for large-scale projects with the cost of the learning curve. Flask is best fit for the prototyping and small-scale projects but not limited to it. Flask can be learned and set up quickly, but when it comes to managing and maintaining, it requires more work than the former”. Django, provided with full-fledged extensions, was rejected as an option suitable for a large and long-term solutions rather than Agile-driven rapid prototyping development.

Go is a low-level compiling programming language, suitable for non-blocking socket I/O web applications, and typically used without frameworks — this solution was considered as a most prominent choice for further developments, but at the current stage of the project would require too much development resources for the implementation and integration of the lower-level functionality which is being provided in the Flask with ready-made extensions (e.g. authorisation, marshalling, i18n, forms processing and more).

Based on the comparison of available options, it was decided to choose the Flask web framework for developing web-application. Go programming language was chosen as a prospective target programming language to re-implement web service during future project iterations. The modular structure of the architecture should facilitate refactoring and re-implementation of web services as required by business stakeholders.

3.5.2 Database

As noted in Scope *(Section 2.4),* MongoDB database was used due to current dependencies on this database in other components of the system.

MongoDB is a NoSQL document-oriented database, using JSON-like documents with optional dynamic schemas. Győrödi et al. defines MongoDB as a database that “holds a set of collections [which] has no predefined schema like tables, and stores data as BSON documents (binary encoded JSON like objects)”, followed by “document is a set of fields and can be thought of as a row in a collection [which] can contain complex structures such as lists, or even an entire document”; they conclude that “[we] can choose MongoDB instead of MySQL if the application is data intensive and stores many data and queries lots of data" [14] .

While current web-application does not require data intensive stores, the document-based database is a robust and flexible solution for storing JSON-like schema-less objects, which was the primary reason for choosing this database. Another compelling alternative would be using PostgreSQL, which has support for native support for JSONB data types (since version 9.4, released in Dec 2014) [15] .

3.6 Technical analysis: image server

Image server is a web software which task is to process and serve images. An image server differs from a traditional web server that serves images by the fact that the former is performing image processing upon request, e.g. cropping, resizing, rotating, changing image's size, format, shape, colour bitmap, saturation, etc.

In current context, SF's requirement is to be able to serve very large high quality image files, taken by professional photographers, to a wide range of visitors using different types of devices. In order to optimise page load, we need to down-scale images by specific image width or height sizes, according to the generally popular on the Internet screen resolution layouts of different devices (see *Table 2*).

Table 1. Image width/height break points (in pixels) defined for the project

Viewport	mobile or tablet					notebook		desktop			
Pixels	180	360	540	720	900	1080	1296	1512	1728	2048	4472

To determine break point values, developers can analyse analytical data from a website's visitor audience, or consult from third-party services that aggregate such data. It is worth considering that as technology advances, mass audiences' screen resolutions and interface scaling values change, and these values change accordingly.

There are several options for image processing tools or libraries are available, among them are *LibGD*[1], *ImageMagick*[2] and others. The author uses *libvips*[3] image processing

1 LibGD: https://libgd.github.io/

2 ImageMagick: https://imagemagick.org/

3 Libvips: https://github.com/libvips/libvips/

library co-authored by Martinez and Cupitt [16] . This library is designed to provide high concurrency and low memory footprint [17] , as well as good security against DoS attacks. As presented on *Figure 1, libvips* library in benchmarking with competing solutions demonstrates good scalability and very high performance with low memory usage.

Thus this library is being appropriately suitable for image server requirements of current service.

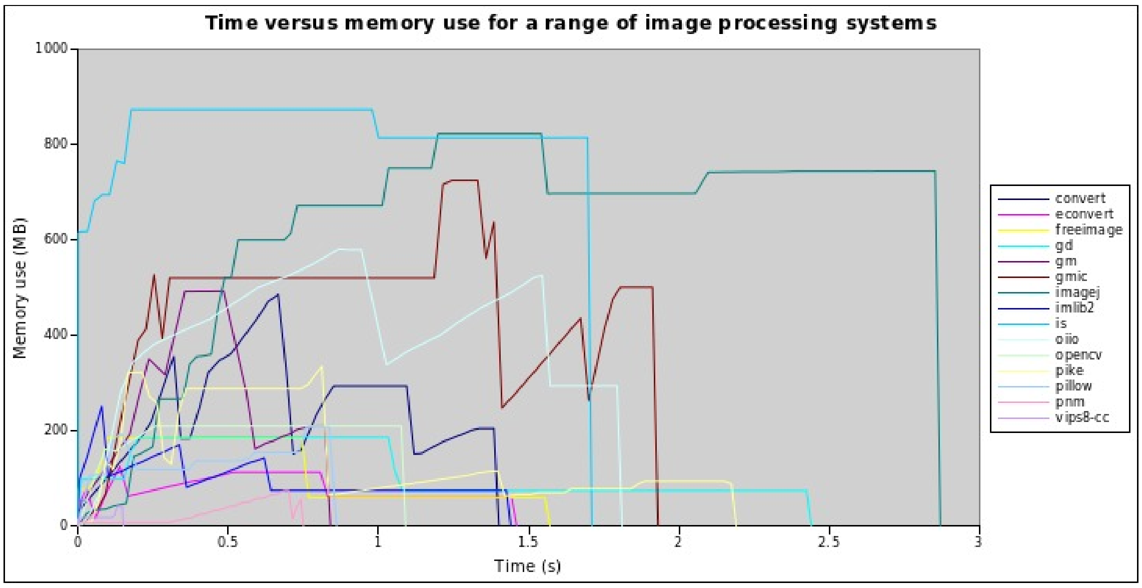


Figure 1. *libvips:* Time versus memory usage for a range of image processing systems [17]

As web-application depends on *libvips* image processing, the author used *imgproxy* — a fast and secure standalone web server, written in Go programming language and supporting Amazon S3 object storage back-end. This open-source third-party service is utilising *libvips* library, is well regarded in industry and actively supported [18] . As this web server provides web-based API to *libvips* library and is written in the non-blocking socket I/O language. This solution satisfies the functional requirement of serving a lot of heavy-weight images to the web user.

3.7 Technical analysis: infrastructure and platform decisions

3.7.1 Platform

Since all current and future components of the system are based on the Linux technology stack, the author uses the Linux operating system and tools as a base requirement for the whole project. As such, *WireGuard* used as a modern, lightweight and high performance in-kernel VPN solution to link multi-cloud servers securely. Also, *systemd* is heavily utilised to perform scheduled operational jobs, viz. backup scripts, SSL certifications updates, etc.

3.7.2 Containerisation

The current system makes full use of containerisation for each service in the system. The author utilises Docker as such solution, which provides reproducible containers with packaging code and dependencies, and allowing to run such containers on any compatible platform. Docker provides interface to supply configuration options to the application inside when running the container, to expose network ports and export logging data.

As has been researched by Fetlter et al. [19] , Docker does not bring a significant overhead in I/O performance comparing to native and KVM-based workloads, except for rare cases.

For the local development environment, Docker Compose [20] is used as a quick and easy solution to deploy a whole set of service components. This stack is supplied by *env*-configuration file with sensitive information (from local machine or *Hashicorp Vault* service in a secure manner). As for staging and production environment, corresponding Docker stack configurations are used in Swarm-mode [21] . This allows the author and other developers to utilise the 12-factor-app methodology [22] , which ensures the web application solution is portable and reproducible.

The combination of asserted solutions is considered optimal as long as the project does not yet require container orchestration[1].

1 Orchestration automates the deployment, management, scaling, and networking of containers at scale

3.7.3 CDN, load balancing and reverse proxy

Since the web application's major objective is to serve a large number of heavy images to the user, it is crucial task to plan how process of storing and serving image files will be organised. As website editor creates new object (stone sample or project), they fill in all the necessary stone properties fields in the form and attaching high-quality origin image(s) to upload. Upon sending the request, back-end processes and validates the form data, as well as validating and uploading images to the server. From now on there are two strategies are available:

1. Using image server generate a whole set of pre-scaled images in advance, as defined by viewport requirements (see *Table 2*) from each source image at the time of creating a new object. Store all the generated images on disk/S3/CDN. Link paths to the whole set of stored images in the stone/project items in the database. As user is opening the public stone/project page, only CDN/reverse-proxy is used to serve the static image files.

2. Store original images on the disk/S3/CDN. Link paths to original images in the stone/project items in the database. As user is opening the public stone/project page with their specific viewport (see Table 1), CDN/reverse-proxy checks wether the requested image is in cache: if not, request is passed to the image server in order to process the image with the correct scale size on the fly; if requested image has already been served, CDN/reverse-proxy responds with the scaled image from LRU cache.

As discussed *(Technical analysis: image server, Section 3.6)*, the *libvips* high performance image processing negates potential delays and CPU load spikes to the point where it is reasonable to use this solution for continuous image processing in the runtime. Pre-generating 11 presets for each source image (sometimes up to 25 high-quality images attached to a project) massively increases the disk capacity required for image storage, as well as the size of document entries in database — all of this complicates the data maintenance and backup processes.

Thus, the author has accepted the second strategy: saving only original images on disk, scaling requested sizes on demand and leverage caching.

Nginx [23] web-server has been chosen as a load-balancing, reverse-proxy and the static files server tool, leveraging fine tuning caching policies [24] .

3.7.4 Back-up strategy

Simple but effective 3-2-1 backup strategy [25] was chosen as easy to implement and straightforward solution:

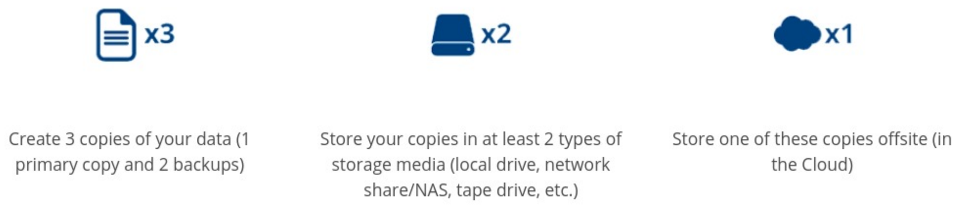

x3
x2
x1
Create 3 copies of your data (1 primary copy and 2 backups)
Store your copies in at least 2 types of storage media (local drive, network share/NAS, tape drive, etc.)
Store one of these copies offsite (in the Cloud)

Figure 2. 3-2-1 backup strategy: 3 copies, 2 types of storage, 1 offsite [26]

The solution is to backup the data to another cloud infrastructure of another provider in a different region. Also, a single copy of the backup is periodically saved locally to disk in Estonia. Backups are automated via *bash* scripts and *systemd* scheduler. Backups are periodically verified manually.

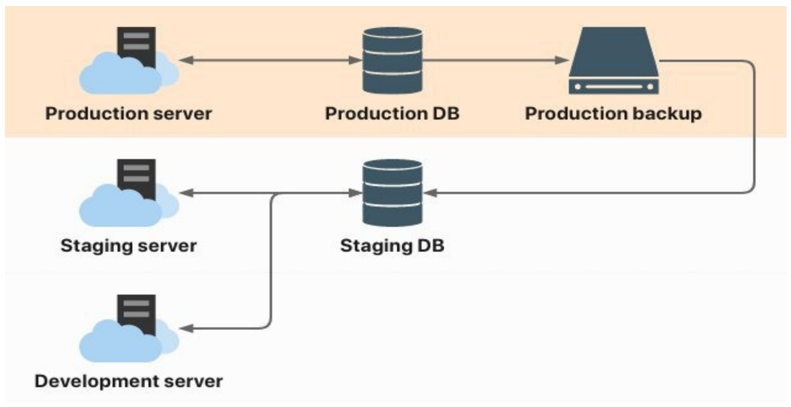


Figure 3. Production backup data verification via staging environment

However, as part of this project, author has implemented a specialised approach that is only suitable for this particular case, where one of the backups (in the second cloud) is used to populate the data for the staging environment (see Figure 3). Thus, the developers have a replica of the system for testing with a relatively fresh data, and in the meantime the integrity of the backup data is verified.

3.7.5 Cloud computing providers

Two cloud computing providers were chosen: Scaleway[1] with instances in Amsterdam and Hetzner Cloud[2] with major computing instances in Finland. Scaleway also provides Container Registry service for Docker images, and Object Storage (AWS S3 compliant) service for storing static assets. Both platforms provide CLI management tools.

Table 2. Comparison of VPS instances that meet minimal requirements per service

	Zone.ee (CloudServer PRO SSD)	**Scaleway (DEV1-XL)**	**Hetzner (CCX21)**	**Google Cloud (GCE, n1-standard-4)**
CPU (vCores)	4	4	4	4
RAM (GiB)	12	12	16	15
SSD (GiB)	125	120	160	$0.187/GB/month
IPv4 (pcs.)	1 (included)	1 (+1,00 EUR/month)	1 (included)	Egress $0.12/TB/month
Bandwidth (Mbits/s)	?	500	800	(?)
Traffic (TB)	~2.1	Unlimited	20	variable
Location	Tallinn, EE	Amsterdam, NL	Helsinki, FI	Helsinki, FI
Price/month (no VAT)	**102,20 €**	**~44,20 €**	**~41,53 €**	**$110~160**

Both clouds are inter-connected via WireGuard VPN in order to isolate back-end services from the public access. However, only Scaleway cloud service provides hosted services such as S3 object storage and Container Registry.

3.8 Technical analysis: summary

The proposed technical solution meets all the needs of the enterprise, both current and future. The use of a custom imaging server will save the company significant resources on renting or supporting alternative solutions, as well as guarantee the operability of the solution in case of unforeseen changes in third-party services.

Object Storage is a reliable and production tested way of storing and serving static assets. While MongoDB is fast and versatile database which can withhold ever growing performance demands.

1 Scaleway: https://www.scaleway.com/en/

2 Hetzner: https://www.hetzner.com/cloud/

4 Solution implementation

This section describes the implementation aspects based on the analysis of the solution. The development of each individual service of the system is separated by relevant chapters.

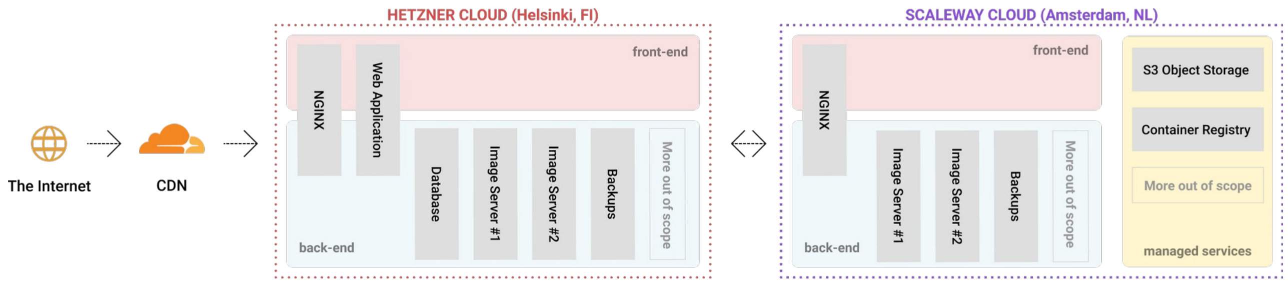


Figure 4. High-level solution architecture overview

4.1 Web application

Web application is a key part of the required solution. It consists of a front-end and back-end sides, it links the back-end to the database, manages the access to the S3 object storage, and provides users access to the image server.

4.1.1 Back-end

Flask framework can be extended by Flask-specific extensions or just by general Python packages. As such, author used *flask-babel* to implement i18n (internationalisation and localisation) support, a library for a standard *GNU gettext* format; *flask-httpauth* for HTTP authentication (RFC 7235[1]); *marshmallow* and *webargs* for the form data marshalling, *argon2-cffi* for the modern cryptography support, and many other extensions.

The author has utilised Flask Blueprints, "a concept […] for making application components and supporting common patterns within an application or across applications", their purpose is to "simplify how large applications work and provide a

1 RFC 7235: https://tools.ietf.org/html/rfc7235

central means for Flask extensions to register operations on applications” [12] . Blueprints allow to decouple separate components as if they were different Flask-based applications, which helps keep the system modular and homogeneous. As such, the author has separated general pages with language code prefix, private image server routes and API routes:

```
/app/__init__.py:
def create_app(config_name):
        app = Flask(__name__)
        app.config.from_object(config[config_name])
        config[config_name].init_app(app)
        register_extensions(app)
        register_blueprints(app)
        […]
        return app


def register_extensions(app):
    mongo.init_app(app)
    mail.init_app(app)
    babel.init_app(app)


def register_blueprints(app):
    """Register blueprints for language-prefixed URL pages"""
    app.register_blueprint(pages, url_defaults={"lang_code": "en"})
    app.register_blueprint(pages, url_prefix="/<lang_code>")
    app.register_blueprint(imgsrv)
    app.register_blueprint(api)
```

Figure 5. Instantiate application using Flask Blueprints and utilising URL-based internationalisation (i18n) via Flask-Babel extension

The Blueprint for a general page is registered with a language code prefix. When no prefix is supplied, e.g. “*https://factory.tld/*”, “*en*” key code for English language is being set by default. Valid language codes are defined in the configuration file and currently they list: “*en*” or none for English, “*ee*” for Estonian, “*se*” Swedish and “*ru*” for Russian languages. E.g. “*https://factory.tld/ee/*” would open web-page in Estonian language, loading Estonian *GNU gettext* MO-file; while invalid language code would redirect to 404 HTTP error page. A general Blueprint with a language prefix in the URI, in a simplified form, demonstrated below:

```
/app/pages/views.py:
from ..pages import pages

@pages.url_defaults
def set_language_code(endpoint, values):
    """Set default language value to the global `g` variable"""
    values.setdefault("lang_code", g.lang_code)


@pages.url_value_preprocessor
def get_lang_code(endpoint, values):
    """Set current language to global `g.lang_code` variable"""
    g.lang_code = values.pop("lang_code", None)


@pages.before_request
def ensure_lang_support():
    """Ensure that valid language is selected, 404 if unsupported"""
    lang_code = g.get("lang_code", None)
    if lang_code not in current_app.config["LANGUAGES"].keys():
        g.lang_code = current_app.config["BABEL_DEFAULT_LOCALE"]
        return abort(404)


@pages.route("/")
def homepage():
    """Homepage will be rendered in the correct language"""
    return render_template("homepage.html")
```

Figure 6. Flask Blueprints utilising URL-based internationalisation (i18n) *GNU gettext* via *flask-babel* extension

The back-end relies on storing all static long-term data in S3 object storage, provided by Scaleway cloud platform. Python *boto3*[1] package, part of AWS SDK, is used to operate with S3 service. All objects static files are being offloaded to the object storage, as such: stone sample original images, project original images, website static images, as well as some private files, logs, etc. Object storage is private by default, hence authentication is required for accessing all kind of files, including public. For this reason, all requests in S3 are proxied through the back-end middleware component, which is being integrated with the authorisation system on the website and thus performs role-base access control.

1 Boto3: https://github.com/boto/boto3

Authorisation also required for getting private fields from the REST API querying stone sample objects, e.g. some price ranges are not publicly available:

```
@auth.verify_password
def verify_password(username, password):
    if username in users:
        return argon2.verify(password, users.get(username))
    return None
```

Figure 7. Modern cryptography paired with basic and reliable HTTP authentication

To invoke callable Flask instance object was used *uWSGI* — an HTTP application server written in C programming language for WSGI-based Python applications. *uWSGI* is run with concurrent processes equal to the number of target VPS CPU cores [13] .

The crucial element of the web-application is the object creation and editing interface. As user fills in the form with product data, attaches images and clicks “Send” button — font-end *Vue.js*[1] application performs initial validation checks for obvious errors; as front-end validation succeeds, the FormData[2] object is send with POST-method via *Axios*[3] client as *multipart/form-data* to the back-end REST API (“*factory.tld/api/v1/products/*”).

As marshalling and validation checks passed on the back-end, deserialised data gets converted and expanded (see *Table 3*), e.g. “marble” object gets a “natural stone” category, or “300 EUR” unit gets “level 2” pricing category — this is automated to save web site editors’ time (see *Appendix 4* for schema example).

When marshalling is completed, we iterate over each attached image file as follows:

- check if *images* field is empty, return error if true (no stone samples and projects are allowed without at least a single image);
- check if image’s filename is empty or incorrect, check if file extension is in allowed list (only JPEG/WEBP images are allowed); getting object name from relevant form fields and construct the future file name by pattern: “*product_name.ext”;*

1 Vue.js: https://vuejs.org/

2 FormData: https://developer.mozilla.org/en-US/docs/Web/API/FormData/Using_FormData_Objects

3 Axios: https://github.com/axios/axios

- path for the file is constructed, also unique hash to file name is appended (in order to avoid same file name collisions), e.g. "products/n/quatzite/african-fusion-yB21wpFKMw.jpg", where “*n*” stands for “natural”;

- each image file gets uploaded to the S3 object storage by a specified above path, thus every single object populates storage by a predictable pattern (in case of some fatal emergency we would need to manually recover or edit images).

If every step above succeeded, image file names are rotated through the URL hashing function for every image size breakpoint, and saved to the database. For the sake of brevity, above is presented only a fraction of the checks and transformations with data and images.

Table 3. Stone object input and output auto-generated parameters

Input parameters		**Output parameters**	
str	product_name	*str*	product_id (“product_name”)
		str	product_slug (URL-friendly: “product-name”)
		str	created_at (datetime timestamp)
		str	updated_at (datetime timestamp)
str	product_type	*str*	product_type
		str	product_category
str	product_origin	*str*	product_origin
str	product_vendor	*str*	product_vendor
str	product_gallery	*str*	product_gallery
int	product_price_12mm	*decimal*	product_price_12mm
int	product_price_20mm	*decimal*	product_price_20mm
int	product_price_30mm	*decimal*	product_price_30mm
		int	product_price_category (1 to 5)
str	descr_en	*array*	product_descriptions
str	descr_et	*str*	descr_en
str	descr_sv	*str*	descr_et
str	descr_ru	*str*	descr_sv
		str	descr_ru
data	product_images (binary files)	*list*	product_images
		array	0: w180 "/<hashed-url>.jpg"
		str	w360 "/<hashed-url>.jpg"
		str	w540 "/<hashed-url>.jpg"
		str	w720 "/<hashed-url>.jpg"
		str	w900 "/<hashed-url>.jpg"
		str	w1080 "/<hashed-url>.jpg"
		str	w1296 "/<hashed-url>.jpg"
		str	w1512 "/<hashed-url>.jpg"
		str	w1728 "/<hashed-url>.jpg"
		str	w2048 "/<hashed-url>.jpg"
		str	w4472 "/<hashed-url>.jpg"
		str	n: […]

Every stone object keeps the whole list of hashed URLs for all attached images for all breakpoint sizes for each image. It allows to fetch images from the front-end, depending on the screen resolution or size, without recalculating hashed URLs on every request. Most popular fetched images are kept in cache and no longer require rescaling.

/app/api/products/views.py
```
for file in image_files:
    if file.filename == "":
        raise InvalidUsage("No image file attached", status_code=409)

    if file and allowed_file(file.filename, current_app.config["IMGPROXY_EXTENSIONS"]):
        # Construct the image filename
        filename_slug_hash = construct_image_filename(
            product_id=result["product_id"], filename_ext=Path(file.filename).suffix
        )

        # Construct the image path,
        # e.g. 'products/n/quatzite/african-fusion-yA21w7FKMw.jpg'
        file_path = Path(
            "products",
            result["product_category"][:1],
            result["product_type"],
            Path(filename_slug_hash),
        )

        # Attempt uploading image files to the S3 storage
        try:
            # Return POSIX path of uploaded object
            # e.g. "products/n/quartzite/african-fusion-yB21wpFKMw.jpg"
            object_final_uri = S3(
                _region_name=current_app.config["S3_REGION"],
                _endpoint_url=current_app.config["S3_ENDPOINT"],
                _aws_access_key_id=current_app.config["S3_ACCESS_KEY_ID"],
                _aws_secret_access_key=current_app.config["S3_SECRET_ACCESS_KEY"],
                _bucket=current_app.config["S3_BUCKET_API"],
            ).upload_object_to_s3(file, str(file_path.as_posix()))

            # Append image path
            files_list.append(str(object_final_uri))

        except IOError:
            raise InvalidUsage(
                code=406, messages={result["product_id"]: "Files upload failed"}
            )
```
Figure 8. A simplified highlight of images processing during the object creation process

4.1.2 Front-end

Front-end mostly consists of fully static HTML/CSS/JS assets, whereas some feature-rich pages utilizing *Vue.js* and *Axios* frameworks (e.g. administration panel) for a dynamic content and asynchronous data requests to the back-end REST API.

Template engine *Jinja2* as a *Flask* extension is used for pages layout at the back-end. *CSS* is being transpiled from *PostCSS*[1] via *GulpJS*[2]. This approach defined as a hybrid Single Page Application, where only selected pages are operating as SPA.

4.1.3 Security considerations

Wep application and infrastructure services follow the best industry practices in order to provide high level of security. All input forms are sanitised and CSRF token protection is used. Strong hashing algorithms with unique 32-bit hex-encoded strings for keys/salts are used (provided by secure PRNG, e.g. */dev/urandom* [29]). Reverse proxy enforces secure SSL certification usage and CORS policies.

Off-cloud VPS instances are routed via modern and secure *WireGuard* Linux in-kernel VPN. This way the network architecture does not depend on Cloud provider's proprietary Firewall solutions (such as Virtual Private Cloud and others).

4.2 Image Server

Libvips image processing library was been successfully deployed by using a thin web server *imgproxy* to provide APIs. Also, the author applies the technique of signing [27] direct URLs to the image server.

This makes following example URL to the image server:

```
factory.tld/images/products/n/quartzite/african-fusion-yB21wpFKMw.jpg?
width=1080&format=jpg&aspect=fit&crop=none
```

1 PostCSS: https://postcss.org/

2 GulpJS: https://gulpjs.com/

look like:

```
factory.tld/images/0I6xKnG31sdk3w/fit/1080/0/no/0/L3N2sjF8Bn.jpg
```

URL hashing technique prevents users or attackers from changing parameters to the image server. Hence attacker will not be able to script massive amount of queries replacing URI parameter “width=1080” (target image’s width size to scale to) with some low value, e.g. “width=10”, and then repeatedly requesting some large number, such as “width=4080”.

This scripted attack would heavily stress the image server CPU and memory load resources, since the image server would have to constantly re-scale image on demand. If no caching or rate-limiting and security measures are in place, such service may fail due to a (Distributed) Denial-of-Service attack.

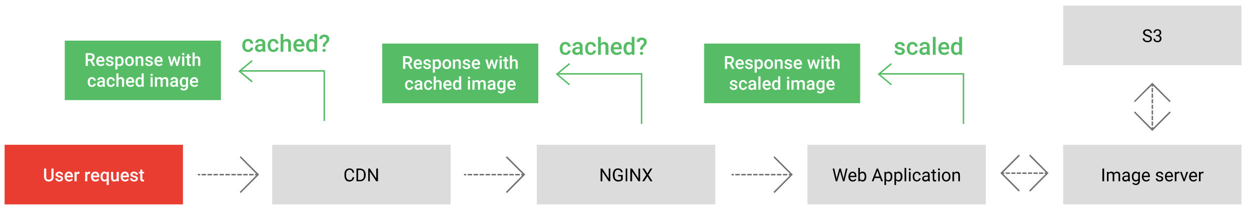

Response with cached image
cached?
Response with cached image
cached?
Response with scaled image
scaled
S3
User request
CDN
NGINX
Web Application
Image server

Figure 9. User image request diagram, e.g. requesting image with 720 px width

4.3 Server and network infrastructure

As has been discussed in analysis overview (see Section *3.7.2*), Docker and Linux environment are heavily used.

For host VPS system Debian distribution is selected as a long-term and well supported system. For Docker base images Alpine Linux is used for all images with statically compiled software (such as Go language services). For Python based applications *debian-slim* image is used, since many Python packages are dependent on cross-compiled *cffi-based* packages (e.g. cryptography and performant JSON parsers), which in their turn are usually are dependent on *glibc* C library.

As *Alpine Linux* uses alternative *musl* C library, occasional unpredictable errors and performance slow-downs occur [28] .

5 Solution assessment

This section assesses and evaluates the final solution according to the initial requirements of the project. The author also considers potential risks and weaknesses of the proposed solution.

5.1 Solution results

The implemented architectural solution fully meets the qualitative and functional needs of the industrial enterprise.

This project has been implemented and works successfully in a production environment. The enterprise uses the web application in internal processes and as a tool and demonstration catalogue of products and projects in communication with partners. Online recognition of the company has increased dramatically, potential customers readily use the web contact form to request price quotes for projects.

The web application and lean architecture allowed to achieve much higher performance in basic operations that competitors' web applications based on common CMS solutions.

The company uses the system as the main platform for attracting new clients and intends to develop a range of new tools on the basis of this architectural solution in the future. The company intends to actively invest in further development of its digital transformation.

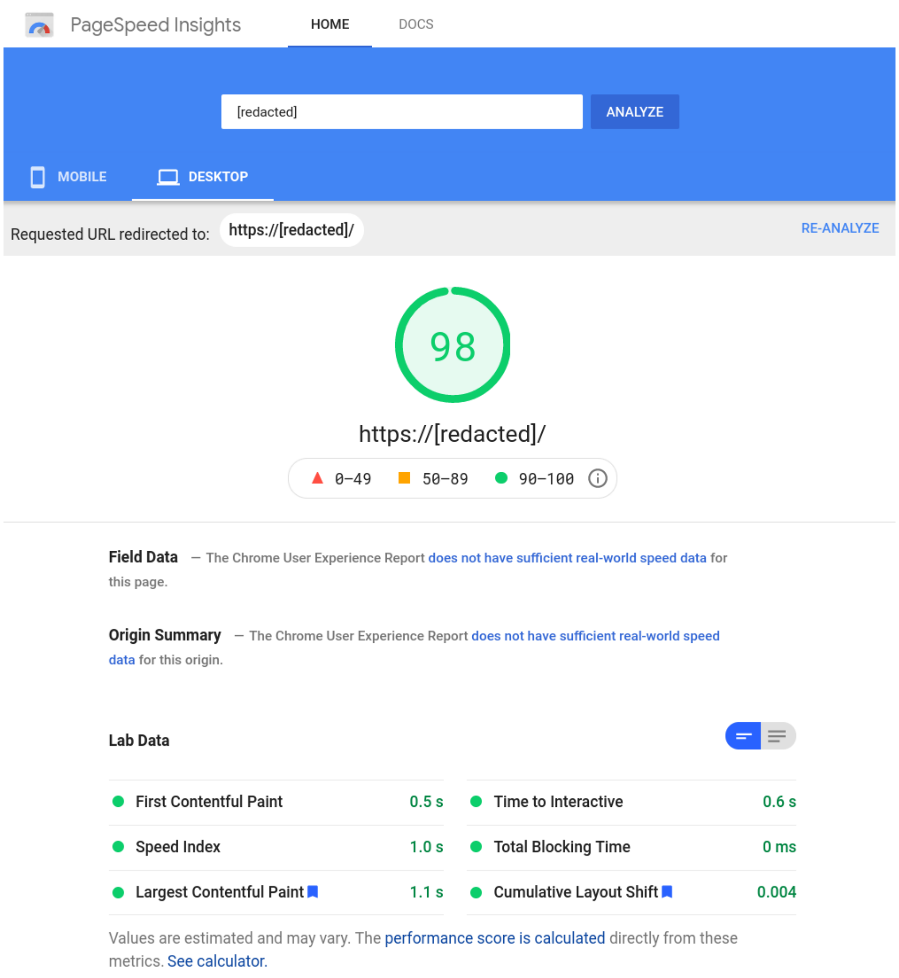
5.2 Threats to solution validity

The solution is expected to have weaknesses when or if a large number of concurrent clients will use the web application. If this risk occurs, the web application will be rewritten to a non-blocking programming language (Go or Rust).

The web application has experienced two breakdowns on the side of the cloud provider. In one case, the hypervisor that powered several key VPS instances was unavailable for two weeks. With regard to the modular architecture, the problem has been bypassed in a matter of hours.

5.3 Solution metrics

The implemented web application has significantly better performance than competitors' websites based on popular, non-optimised CMSs. The solution provides fast functions and pages load speed, low latencies, various elements of the system are highly optimised, including image compression. The website demonstrates high scores in third-party website performance evaluation tools.



Requested URL redirected to: https://[redacted]/

https://[redacted]/

▲ 0–49 ■ 50–89 ● 90–100

Field Data — The Chrome User Experience Report does not have sufficient real-world speed data for this page.

Origin Summary — The Chrome User Experience Report does not have sufficient real-world speed data for this origin.

Lab Data

● First Contentful Paint	0.5 s	● Time to Interactive	0.6 s
● Speed Index	1.0 s	● Total Blocking Time	0 ms
● Largest Contentful Paint	1.1 s	● Cumulative Layout Shift	0.004

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

Figure 10. A redacted Google PageSpeed Insights tools will hidden website link; desktop performance. Mobile performance is at 85 points.

6 Further developments

SF stakeholders are looking to develop the private area of the web application for a range of private partners to allow them request fine detailed project calculations within the private customers system.

Also, the enterprise in interested in integrating web-application resources with the external ERM system.

Some other prospect ideas include Machine Learning based super-scaling image processor to allow company using lesser quality images as sources, and also apply intelligent de-noise reduction filters, etc.

Full decoupling of front-end and back-end within web-applications is currently in development. Possible re-implementation of key web-services in non-blocking languages is also planned.

Migration from MongoDB to PostgreSQL is considered, while not prioritised.

If system and users load will grow, container orchestration (e.g. *Kubernetes* and derivatives) and IoC (such as *Terraform* and *CloudFormation*) will be considered.

SF is intended to continue it's technological progresses in the way of servitization.

7 Summary

This thesis demonstrates, on the primer of a case study from a specific industrial factory, how the process of developing an architectural software solution takes place.

Current work provides the principles of primary data collection and analysis from stakeholders to determine the qualitative and functional requirements of the project, constructing models of typical users and their scenarios of system use, argues the process of choosing the technological stack of solutions for implementation, analysis of compliance of models and selected solutions according to the primary requirements of the project.

Based on the data collected and analysed, a decision was made as to which system components needed to be developed from scratch and which could possibly be integrated from third parties.

The implemented solution has significantly improved the company's business processes. The tool is used internally and as a communication tool with partners and customers. The solution is affordable to maintain, has high reliability and performance. Based on the experience gained, the factory enterprise intends to continue the digital transformation of its enterprise by creating new software services based on this architectural solution.

References

[1] Nambisan, S. Wright, M. Feldman, M “The digital transformation of innovation and entrepreneurship: Progress, challenges and key themes” (2019); Research Policy, Volume 48, Issue 8, 103773, ISSN 0048-7333. DOI https://doi.org/10.1016/j.respol.2019.03.018

[2] Sebastian, I. Ross, J. Beath, C. Mocker, M. Moloney, K. Fonstad, N. "How big old companies navigate digital transformation", MIS quarterly executive, volume 16, issue 3 (2017): p. 202; The Kelley School of Business, Indiana University. DOI https://nbn-resolving.org/urn:nbn:de:bsz:rt2-opus4-15016

[3] Schwertner, K. "Digital transformation of business", Trakia Journal of Sciences 15, no. 1 (2017): pp. 388–389. DOI 10.15547/tjs.2017.s.01.065

[4] Hirsch-Kreinsen, H. “Digitization of industrial work: development paths and prospects”; J Labour Market Res 49, pp. 3–11 (2016). DOI https://doi.org/10.1007/s12651-016-0200-6

[5] Clements, P., Kazman R. Klein M. "Evaluating software architectures: methods and case studies. 2002."

[6] Žužek, T. Gosar, Ž. Kušar, J. Berlec, T. "Adopting Agile Project Management Practices in Non-Software SMEs: A Case Study of a Slovenian Medium-Sized Manufacturing Company" (2020); Sustainability, 12(21): 9245. DOI https://doi.org/10.3390/su12219245

[7] Dasanayake, J. Markkula, S. Aaramaa, M. Oivo "Software Architecture Decision-Making Practices and Challenges: An Industrial Case Study" (2015); M-Group, University of Oulu, Oulu, Finland. DOI https://doi.org/10.1109/ASWEC.2015.20

[8] Peffers, K., Tuunanen, T., Rothenberger, M.A., & Chatterjee, S. “A design science research methodology for information systems research” (2007). Journal of Management Information Systems, 24(3): pp. 45–77. DOI: https://doi.org/10.2753/MIS0742-1222240302

[9] Nielsen L. “Personas in Use” (2019). In: “Personas - User Focused Design. Human–Computer Interaction Series”. Springer, London. DOI https://doi.org/10.1007/978-1-4471-7427-1_5

[10] Bilski, M. “Migration from blocking to non-blocking web frameworks” (2014), p. 43; Dissertation. DOI http://urn.kb.se/resolve?urn=urn:nbn:se:bth-5932

[11] Ghimire, D. "Comparative study on Python web frameworks: Flask and Django" (2020), pp. 31-32; Metropolia University of Applied Sciences (Finland). DOI http://urn.fi/URN:NBN:fi:amk-2020052513398

[12] “Modular Applications with Blueprints” [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/blueprints/ [accessed on 18.04.2021].

[13] “The uWSGI project” [Online]. Available: https://uwsgi-docs.readthedocs.io/en/latest/ [accessed on 18.04.2021].

[14] Győrödi, C. Győrödi, R. Pecherle G. Olah, A. "A comparative study: MongoDB vs. MySQL" (2015), 13th International Conference on Engineering of Modern Electric Systems (EMES), p. 2. DOI 10.1109/EMES.2015.7158433.

[15] “PostgreSQL. E.27. Release 9.4” [Online]. Available: https://www.postgresql.org/docs/9.4/release-9-4.html [accessed on 21.04.2021].

[16] Martinez, K. Cupitt, J. "VIPS — a highly tuned image processing software architecture" (2005); IEEE Xplore. DOI https://doi.org/10.1109/ICIP.2005.1530120

[17] “libvips: Speed and memory use” [Online]; Available: https://github.com/libvips/libvips/wiki/Speed-and-memory-use [accessed on 24.04.2021]

[18] “imgproxy Documentation” [Online]. Available https://docs.imgproxy.net/ [accessed on 24.04.2021]

[19] Felter, W. Ferreira, A. Rajamony, R. Rubio, J. "An updated performance comparison of virtual machines and Linux containers" (2015), p. 8; IBM Research, Austin, TX. DOI https://doi.org/10.1109/ISPASS.2015.7095802

[20] “Docker Compose” [Online]. Available: https://docs.docker.com/compose/ [accessed on 22.04.2021].

[21] “Docker: Swarm mode overview” [Online]. Available: https://docs.docker.com/engine/swarm/ [accessed on 22.04.2021].

[22] “The Twelve Factor App” [Online]. Available https://12factor.net/ [accessed on 22.04.2021].

[23] “NGINX” [Online]. Available: https://www.nginx.com [accessed on 23.04.2021]

[24] Omid H. Jader, Subhi R. M. Zeebaree, Rizgar R. Zebari "A State Of Art Survey For Web Server Performance Measurement And Load Balancing Mechanisms" (2019); International Journal of Scientific & Technology Research. 8: pp. 535-543.

[25] Magnusson, F. “Implementing a Backup-Scheme with the 3-2-1 Strategy : A Comparison of the Active Solution with a New Implemented 3-2-1 Backup-Scheme” (2018), pp. 16, 38; Mid Sweden University. DOI http://miun.diva-portal.org/smash/record.jsf?pid=diva2:1246434

[26] “The Golden 3-2-1 Backup Rule”, Acronis (2021), [Online]. Available: https://www.acronis.com/en-us/articles/backup-rule/ [accessed on 23.04.2021]

[27] “Signed URLs”, Google (2021), [Online]. Available: https://cloud.google.com/storage/docs/access-control/signed-urls [accessed on 23.04.2021]

[28] “Using Alpine can make Python Docker builds 50× slower” [Online]. Available: https://pythonspeed.com/articles/alpine-docker-python/ [accessed on 23.04.2021]

[29] “Removing the Linux /dev/random blocking pool” LWN (2020) [Online]. Available: https://lwn.net/Articles/808575/ [accessed on 24.04.2021]

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

20.04.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

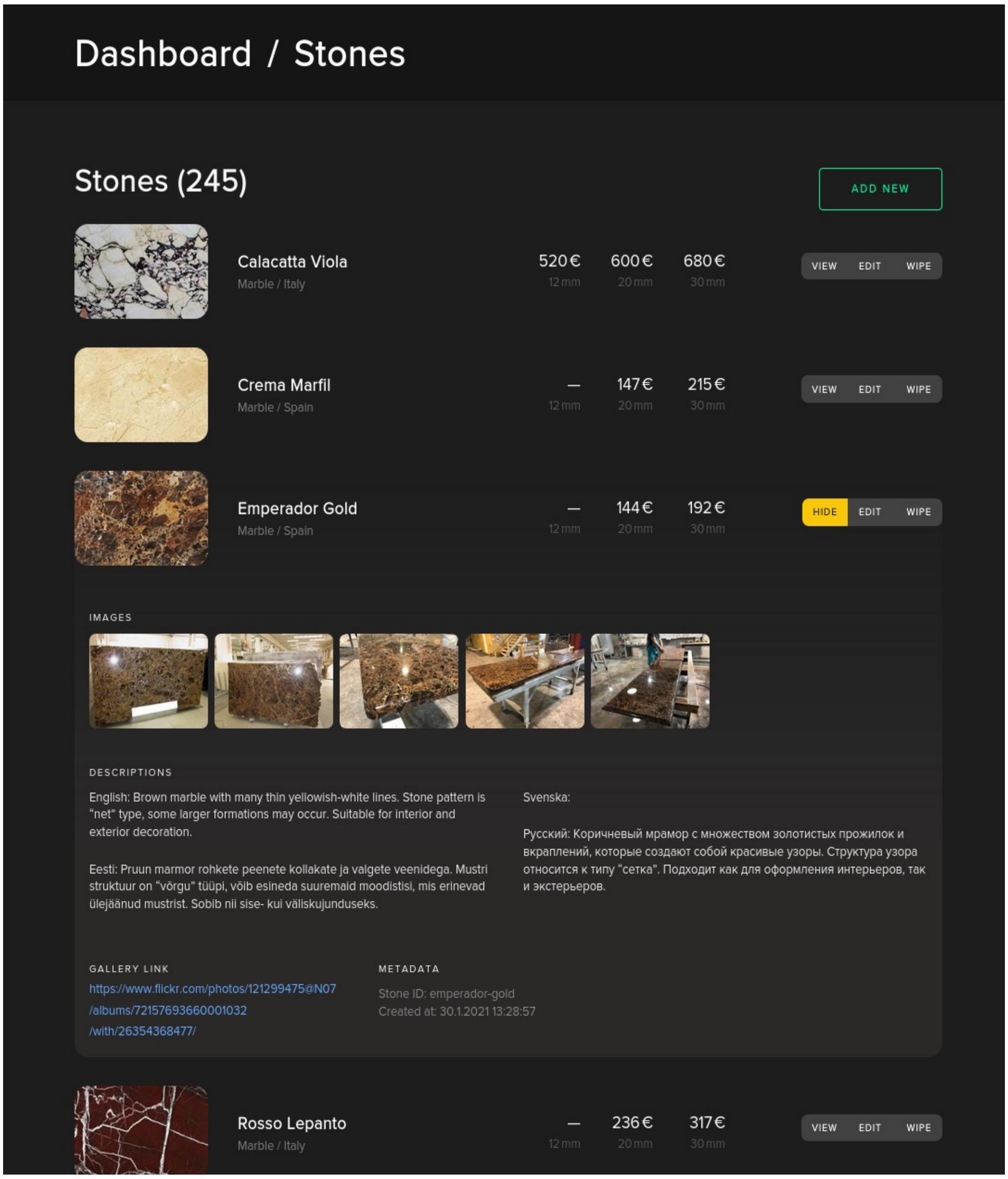
Appendix 2 – Build an HMAC signed URL to image server for accessing private images (called only by authorised users)

/app/imgserver-middleware/views.py:

```
def imgproxy_s3_signed_urls_handler(error, endpoint, values):
    """
    This handler triggers when `url_for` cannot build a URL. Checking, if the endpoint
    is pointed to "imgs_private" then invokes S3 URL builder
    """
    IMGPROXY_SRCSET = tuple(current_app.config["IMGPROXY_SRCSET"])
    IMGPROXY_WEB_HOST = current_app.config["IMGPROXY_WEB_HOST"]
    IMGPROXY_API_HOST = current_app.config["IMGPROXY_API_HOST"]
    # IMGPROXY_SRCSET = (180, 360, 540, 720, 900, 1080, 1296, 1512, 1728, 2048, 4472)

    if endpoint == "imgs_private":
        # Get IMGSRV_SRCSET; set 180px if empty; abort if invalid
        image_width = values.get("s", 180)
        path = values.get("path", None)
        if image_width not in IMGPROXY_SRCSET:
            return return_json_error(code=400, msg="Invalid image size specified")
        return IMGPROXY_WEB_HOST + get_signed_url_private_web_images(
            path, image_width, IMGPROXY_WEB_HOST
        )

    else:
        # External lookup did not have a URL.
        exception_type, exception_value, traceback = sys.exc_info()
        if exception_value is error:
            raise Exception(exception, exception_value, traceback)
        else:
            raise Exception(error)


def get_signed_url_private_web_images(path, image_width, host) -> str:
    signed_url = SignURL(
        current_app.config["IMGPROXY_KEY"], current_app.config["IMGPROXY_SALT"]
    ).generate(
        path=path, resize="fit", width=image_width, height=0, gravity="no",
        enlarge=0, extension="jpg",
    )
```

Appendix 3 – Private administration interface

Listing stone sample materials, collapsed and expanded view:



Dashboard / Stones

Stones (245)

Calacatta Viola Marble / Italy	520 € 12 mm	600 € 20 mm	680 € 30 mm
Crema Marfil Marble / Spain	— 12 mm	147 € 20 mm	215 € 30 mm
Emperador Gold Marble / Spain	— 12 mm	144 € 20 mm	192 € 30 mm

IMAGES

DESCRIPTIONS

English: Brown marble with many thin yellowish-white lines. Stone pattern is "net" type, some larger formations may occur. Suitable for interior and exterior decoration.

Eesti: Pruun marmor rohkete peenete kollakate ja valgete veenidega. Mustri struktuur on "võrgu" tüüpi, võib esineda suuremaid moodistisi, mis erinevad ülejäänud mustrist. Sobib nii sise- kui väliskujunduseks.

Svenska:

Русский: Коричневый мрамор с множеством золотистых прожилок и вкраплений, которые создают собой красивые узоры. Структура узора относится к типу "сетка". Подходит как для оформления интерьеров, так и экстерьеров.

GALLERY LINK

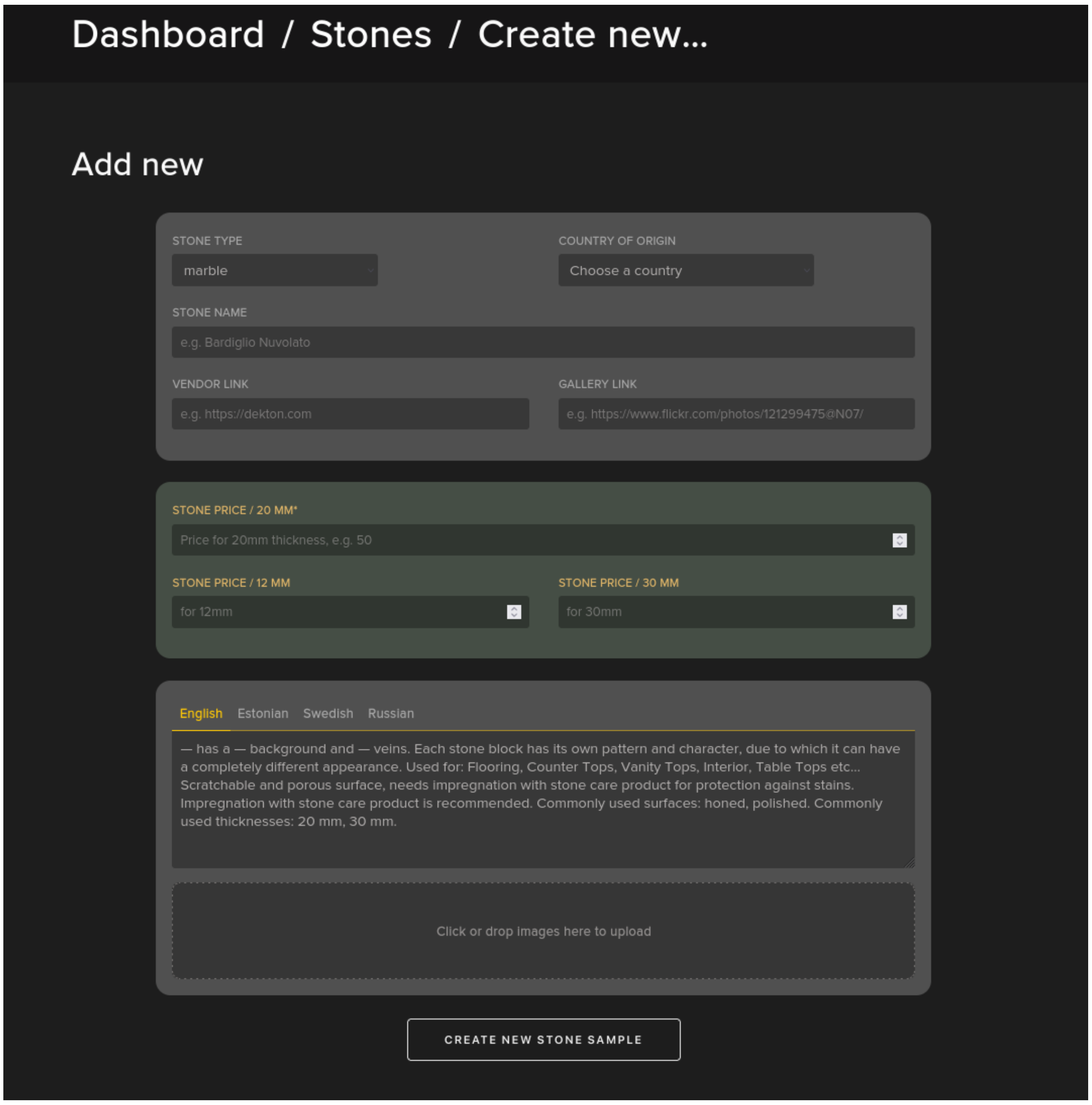
https://www.flickr.com/photos/121299475@N07/albums/72157693660001032/with/26354368477/

METADATA

Stone ID: emperador-gold
Created at: 30.1.2021 13:28:57

Rosso Lepanto Marble / Italy	— 12 mm	236 € 20 mm	317 € 30 mm

Creating new stone sample material:



Dashboard / Stones / Create new...

Add new

STONE TYPE	marble
COUNTRY OF ORIGIN	Choose a country
STONE NAME	e.g. Bardiglio Nuvolato
VENDOR LINK	e.g. https://dekton.com
GALLERY LINK	e.g. https://www.flickr.com/photos/121299475@N07/
STONE PRICE / 20 MM*	Price for 20mm thickness, e.g. 50
STONE PRICE / 12 MM	for 12mm
STONE PRICE / 30 MM	for 30mm

English Estonian Swedish Russian

— has a — background and — veins. Each stone block has its own pattern and character, due to which it can have a completely different appearance. Used for: Flooring, Counter Tops, Vanity Tops, Interior, Table Tops etc... Scratchable and porous surface, needs impregnation with stone care product for protection against stains. Impregnation with stone care product is recommended. Commonly used surfaces: honed, polished. Commonly used thicknesses: 20 mm, 30 mm.

Click or drop images here to upload

CREATE NEW STONE SAMPLE

Appendix 4 – Stone example marshalling/MongoDB schema

/app/api/products/schemes.py (simplified and redacted):

```python
class ProductSchema(Schema):
    """Schema of products.
    Defines the structure and validates product fields. Generates <product_id>
    (also used as URL slug) based on sample's title, checks if <product_id>
    is unique in MongoDB.

    Returning JSON formatted errors, if <product_id> already exists.
    """

    product_type = fields.Str(required=True)
    product_name = fields.Str(
        required=True,
        validate=[
            validate.Length(
                5,
                64,
                error="Product name has incorrect length (must be 5 to 64)"
            )
        ],
    )
    product_origin = fields.Str(required=False)
    product_price_20mm = fields.Decimal(
        required=True, allow_none=False, validate=validate_req
    )
    product_price_30mm = fields.Decimal(
        required=False, allow_none=True, validate=validate_opt
    )
    product_price_category = fields.Int(allow_none=True, required=False)
    product_gallery = fields.Url(
        relative=False,
        default=None,
        missing=None,
        schemes=set(["http", "https", "ipfs"]),
        validate=validate.Length(min=4),
    )
    description = fields.Dict()
    product_price_category = fields.Int(allow_none=True, required=False)
    product_gallery = fields.Url(
        relative=False,
        default=None,
        missing=None,
        schemes=set(["http", "https", "ipfs"]),
        validate=validate.Length(min=4),
    )
    description = fields.Dict()
    brief_en = fields.Str(
        allow_none=False, required=True, validate=validate.Length(min=5)
    )
    brief_ru = fields.Str(
        allow_none=True, missing=None, validate=validate.Length(min=5)
    )
    brief_et = fields.Str(
        allow_none=True, missing=None, validate=validate.Length(min=5)
    )
    brief_sv = fields.Str(
        allow_none=True, missing=None, validate=validate.Length(min=5)
    )
    photo_files = fields.Raw(many=True, read_only=True)
    product_images = fields.Raw(many=True, read_only=True)

    @pre_load
    def nullify_empty_fields(self, args, **kwargs):
        """Set `Null` to empty fields from HTTP FORM-object"""
        args["product_price_30mm"] = (
            Decimal(0) if not args["product_price_30mm"] else
args["product_price_30mm"]
        )
        args["product_vendor"] = (
            None if not args["product_vendor"] else args["product_vendor"]
        )
        args["product_gallery"] = (
            None if not args["product_gallery"] else args["product_gallery"]
        )
        args["descr_et"] = None if not args["descr_et"] else args["descr_et"]
        args["descr_sv"] = None if not args["descr_sv"] else args["descr_sv"]
        args["descr_ru"] = None if not args["descr_ru"] else args["descr_ru"]
```