Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

No-code platform for web pages hosting

Nikita Dvoriadkin

Supervisor: Ing. Karel Frajták Ph.D.
May 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Dvoriadkin**	Jméno: **Nikita**	Osobní číslo: **487601**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Softwarové inženýrství a technologie**		

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Architektura pro serverovou část no-code platformy pro vývoj a hosting webových aplikací

Název bakalářské práce anglicky:

No-code platform for web pages hosting (server side)

Pokyny pro vypracování:

Navrhněte vhodnou architekturu pro serverovou část no-code platformy pro vývoj a hosting webových aplikací. Uživatel systému bude moci vytvářet své stránky bez znalosti HTML, JS, apod. (frontend část ale není součást projektu). Systém by měl být navržen modulárně s maximální rozšiřitelností a dynamičností.
Systém by měl umožnit spravovat uživatelské účty včetně možnosti přihlašování se přes OAuth servery třetích stran (např. Google, Github, Facebook, apod.). Dále si vytvořit strukturu stránek a stránky upravovat (obsah, tagy, URL, apod.).
Systém dostatečně otestujte.
Součástí práce bude rešerše stávajích řešení.

Seznam doporučené literatury:

Moskal, Monika. "No-Code Application Development on the Example of Logotec App Studio Platform." Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 11.1 (2021): 54-57.
Schötteler, Sebastian, et al. "A No-Code Platform for Tie Prediction Analysis in Social Media Networks." International Conference on Wirtschaftsinformatik. Springer, Cham, 2021.
Ang, Raymund John. "Building Applications Using Low-Code and No-Code Platforms." Canadian Journal of Nursing Informatics 16.3/4 (2021).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Karel Frajták, Ph.D. laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Karel Frajták, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání Podpis studenta

Acknowledgements

I want to thank all the teachers who invested their time in me and shared precious knowledge with me during my bachelor's studies. Also, I want to thank my supervisor, Ing. Karel Frajták Ph.D., for helping and giving me helpful advice while working on this project. Finally, I want to thank my parents for constantly supporting me during my studies.

Declaration

Abstract

This thesis deals with design of an architecture and implementation of prototype of a server-side application for no-code web pages creation. The main feature of this application is a high modularity in terms of both data and display elements. First, the analysis of similar platforms is performed with an emphasis on their weak sides in order to improve them in my application. Then, the following chapters describe the concept of a solution and architecture design along with analysis of technologies which are used during the solution implementation. Next, the development process of a prototype is described. At the end, there is a reflection on the future of the product, things needed to improve and further product extension.

Keywords: No-Code, small business, web application, development, REST, Kotlin, Spring Framework

Supervisor: Ing. Karel Frajták Ph.D.
Department of Computer Science
FEE CTU in Prague,
Karlovo náměstí 13,
12135 Praha 2

Abstrakt

Tato práce se zabývá návrhem architektury a implementací prototypu serverové aplikace pro tvorbu nekódovaných webových stránek. Hlavním rysem této aplikace je vysoká modularita datových i zobrazovacích prvků. Nejprve je provedena analýza podobných platforem s důrazem na jejich slabé stránky za účelem jejich vylepšení v mé aplikaci. Následně je v následujících kapitolách popsán koncept řešení a návrh architektury spolu s analýzou technologií, které jsou při implementaci řešení použity. Dále je popsán proces vývoje prototypu. Na závěr je zamyšlení nad budoucností produktu, věcmi potřebnými ke zlepšení a dalšímu rozšiřování produktu.

Klíčová slova: No-Code, malý podnik, webová aplikace, vývoj, REST, Kotlin, Spring Framework

Překlad názvu: Architektura pro serverovou část no-code platformy pro vývoj a hosting webových aplikací

Contents

Figures

Tables

Chapter 1
Introduction

At the beginning of my last year of bachelor's studies I was obsessed by the idea of creating my own startup and releasing own software product. Although creating software projects from scratch could seem like a great thing, the main problem I encountered was lack of ideas.

Few days later, one of my friends called me with a problem. This person started a real-estate company and wanted to attract new clients to his business. As Bill Gates stated: "If your business is not on the Internet then your business will be out of business", my friend needed a website for his company. The main problem he faced was that he did not possess much money for outsourcing this task to any software company, nor he liked the solutions popular website constructors offered. His business required some specific data to be represented on a web page regarding to the real estate information.

Looking deeper, I found out that the vast majority of existing website constructors do not provide a user with flexibility of web pages creation. Thus, the same problem as my friend can have other small businesses, which can potentially lead to the loss of their clients and profit.

1.1 Motivation

Motivation for this bachelor's thesis was to design and implement a web application which would allow users to create their own web pages without having any programming skills. One of the main features of such a platform would be unlimited functionality in terms of web pages creation.

Currently there are lots of no-code or low-code existing solutions for website creation. The vast majority of them only allow creating web pages using already prepared blueprints or templates, which do not offer users flexibility in their website creation.

1.2 Goals

The main goal of this bachelor's thesis is to analyze and design the architecture of server-side no-code application for web page creation and hosting. This paper describes the process of creating a new product prototype, including analysis, design, and implementation of a new no-code platform prototype.

First, in Chapter 2, I will analyze existing solutions for no- or low-code website creation available on the Internet. I will describe their disadvantages and explore the points that can be improved. After analyzing the existing solutions, I will determine the functional and non-functional requirements for the system in Chapter 3. Next, in Chapter 4, I will describe the concept of my application. This section will give details about my idea of a solution with the help of a UML diagram and images. Further, in the 5th chapter, I will define the application architecture and describe it in detail.

After analyzing both business and technical, I will talk about the implementation itself. Chapter 6 will discuss the technologies I have used in my application. There will be some comparisons of them and my reflection on why I have chosen them. Next, in the 7th chapter, I will describe my implementation process of the application along with some notes on what has worked for me and what problems I have encountered while programming. After that, in the following 8th section, I will talk about the testing process of my application.

In Chapter 9, I will reflect on the future of my project. I will especially pay attention to technical improvements of my product and what should be done for scaling, and I will define the next steps for further extension of my application.

Finally, in the last chapter, I will summarize the result of my thesis. It will include my personal feelings about the project, whether it was successful or not, and also, I will describe what I have learned during my work on the thesis.

Chapter 2
Problem definition

The Internet has become an essential part of life for the majority of people in the world. By May 2022, there are more than 1.600.000.000 web pages on the Internet. Over 250.000 web pages are created everyday and 10.500 new websites are created per hour [1].

As the Internet grows, the number of active digital buyers is also increasing year by year. In year 2020 approximately two billion of people made online purchases and in the same year online sales surpassed 4.2 trillion U.S. dollars worldwide [2]. All these facts bring small business a huge opportunity to grow. Having a website not only helps to sell the products, but also it can attract new customers which leads to great potential to expand.

Year by year, there is a growing demand for website creation which comes from businesses. Usually, small companies do not possess the resources (including money or software engineers) to create websites for selling their products online. There are services that allow to create web pages relatively fast and cost-effectively without any software development skills.

2.1 No-code platforms

No-code platforms are the type of visual software development environments that allow developers and regular people to create web pages without writing any code quickly. Users build their products in a visual editor using customizable components. All the corresponding code for these components is created automatically. [3].

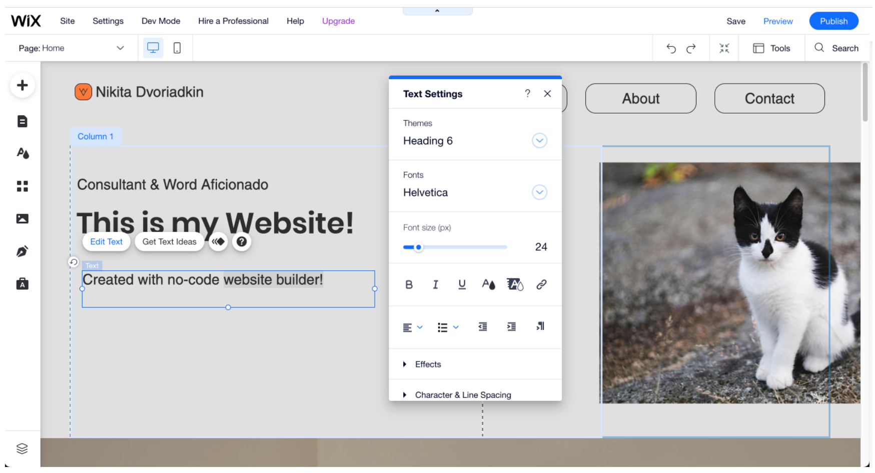


Figure 2.1: Example of creating a website on a no-code platform Wix.com

2.1.1 Benefits

- Since it does not require any programming experience, it makes no-code platforms incredibly easy to use.
- No-code platforms accelerate application development because users do not need to write code. It may lead to a decrease in the delivery time of a product to the market.
- No-code platforms are cost-effective. They mainly do not require any additional payments for any application changes. Also, the vast majority of all services work under the subscription model pricing.

2.2 Analysis of existing solutions

In this section, I analyzed existing no-code solutions. It describes popular services for website creation with their disadvantages, and further, there would be some reflection on what my product should have to be successful.

Solid Pixels

Solid Pixels[1]is a platform for different types of website creation without any code skills. (landing, multi-page, e-shop). Analyzing this product, I found out that it has lots of disadvantages. From the first opening, it was hard for me to navigate through their web page builder because the user interface they have is not much intuitive. Next, this platform offers minimal functionality without any additional payments. The other problem I faced was that I could not log in to my account after the first try. This system does not offer the functionality to have multiple projects for one user and does not support login via third-party services like Facebook, Google and others.

[1]https://www.solidpixels.com/en

Shoptet

Shoplet[2]is a platform for e-commerce system creation and hosting. Their primary goal is to let users create their online shops. This product seemed to me as a good tool for own e-commerce system creation, since it provides a great variety of plugins and other different tools. However, some of this functionality is available after additional payments. Some parts of websites cannot be customized without basic knowledge of front-end development. It also does not support login via other services.

WIX

Wix[3]is an online drag-and-drop platform allowing users to create different types of web pages. Although it offers a lot of advanced features, sites created using WIX still have to fall into predetermined categories. Another disadvantage is the hidden fees, the large learning curve.

WordPress

WordPress[4]is a popular open-source content management system that has been on the market for a long time. Although it is a proven technology for website creation, it still has disadvantages. Typically, users of this system need to purchase different templates or plugins to customize their web pages. Also, plugins and themes are frequently updated, so the user needs to keep them up-to-date constantly. Unfortunately, it is a highly complex tool for website creation with a significant learning curve. Also, to make minor customization, users typically need to insert HTML code.

2.2.1 What can be improved

After the research and competitor analysis I would like to highlight the points that will make out solution better and more effective over the competitors ones:

- Fully customizable website creation and editing. Competitors' solutions mostly allow for creating "cookie-cutter" websites. These are the websites that fit already prepared by platform templates. My solution needs to support the functionality that enables users to create and modify the structure of any interface components. It will bring unlimited modularity of data and display elements.

- Pricing without any additional charges. Most of the competitors' solutions charge for some additional features. For instance, a user needs to pay for plugins that extend some functionality of their website.

[2]`https://www.shoptet.cz/`
[3]`https://www.wix.com/`
[4]`https://wordpress.com/`

- The simplicity of use and small learning curve. A client needs to work with an intuitive and easy system to use. Unfortunately, some of the solutions mentioned above have great functionality, but, on the other hand, it takes a long time to learn how to use them.
- Multiple projects for a user. A user will be able to create multiple websites under his account in my product.

2.3 Potential customers

I introduced the idea of a no-code platform to some of my friends who have their businesses and, at the same time, need a website to sell and promote their services. In the following section, I am going to introduce the potential customers of the platform. They are divided into three primary groups. For each group, there is a real example of a user and his problem, which would be solved thanks to this platform.

2.3.1 Individual solutions

Target group: Individual entrepreneurs and small businesses
The users of this group would mostly like to have fully customizable and individual solutions to the specificity of their business and desires.

Real Example: A person wants to create a catalog website for his real estate business. The problem is that website creation for such a business as real estate is specific because of the data, so there is no such platform like WIX or Squarespace which would allow one to build a website using an existing blueprint/template.

2.3.2 Box solutions

Target group: Individual entrepreneurs and small businesses
The users of this group do not possess any big resources to purchase complete solutions developed by outsourced IT companies. At the same time they realize that some sort of 'box solution' would be enough for them.

Real Example: A person makes minimalistic furniture and sells it via Instagram account. In order to have a comfortable e-commerce platform for selling his goods and to reach more customers, he would like to have his own e-shop website.

2.3.3 Corporate clientele

Target group: Medium and larger businesses
The users of this group do potentially possess resources to purchase a complete solution but at the same time they realize that outsourcing the creation of

their websites would be more financially beneficial. Users of this group want a turnkey solution.

Real Example: A person organizes IT-related summits and wants to focus 100% of his attention on his organization. He would therefore like to completely outsource web issues.

2.4 Summary

Currently, there is a growing popularity of no-code services because of the high demand for website creation. Comparing the existing solutions and improving their shortcomings within this project, I believe that my product will benefit potential clients from businesses.

Chapter 3
Analysis of requirements

The output of this bachelor's thesis is the implemented base of a future no-code platform for web page creation and hosting. This base, also called a prototype, is a minimal product with a limited amount of functionality used to demonstrate the key features.

I defined functional and non-functional requirements which should be present in my application prototype.

3.1 Functional requirements

Functional requirements define the functionality that the resulting application must provide. The functional requirements are usually in form: "A system must do <requirement>." For my application I identified the following functional requirements:

1. User login: A system must provide a login functionality.
2. Website creation: A user can create one or more websites.
3. Website deletion: A user can delete their websites.
4. Website name setting: A user can set or edit their website name.
5. Page creation: A user can create a web page and assign it to his website.
6. Page no-code editing: A user can edit his web page without any code writing.
7. Page deletion: A user can delete his web page.
8. Page URL setting: A user can set URL of his web page.
9. Page name setting: A user can set or edit their web page name.

3.2 Non-functional requirements

Non-functional requirements define the properties of the system. The non-functional requirements are usually in form: "A system must be <requirement>". I identified the following non-functional requirements:

1. Integrability: Because the output of my bachelors work is a server-side application, it is important to ensure that the server can communicate with the client-side application, send data and receive data.
2. Security: The application should provide limited access to some functionality depending on the user currently logged in.
3. Extensibility: Because the goal of this thesis is to implement a prototype of server-side application, the application itself must be extensible, that is, it must provide the ability to easily add new features or extend current ones.

Chapter 4

Application concept

4.1 Inspiration

The goal of the resulting application is to give users functionality for creating their websites without any restrictions. To achieve this goal, it was necessary to look at the common web pages more closely and do research on them.

The idea of the solution lies in an observation of popular web pages. As an example, I looked through the design of popular websites like YouTube, Google, Instagram, and others. I noticed that each web page's interface is built from smaller components, which can be combined to make up a bigger UI pattern. Further, I came across an article where the author reflects on a similar problem of creating web pages. In it, he draws a parallel with chemistry, saying that every substance is essentially composed of smaller components: molecules and atoms. It looks like an application interface can be built from small components (atoms) that eventually assemble into large parts of the interface and finally form a web page [4].

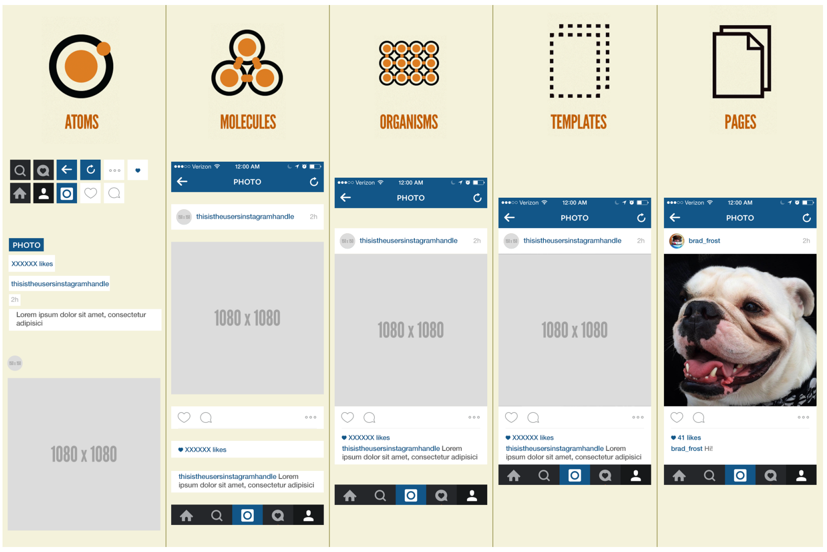


Figure 4.1: Instagram interface built from components [5]

Thinking this way, I noticed that the front-end development library for JavaScript React, with which I had some work experience before, exposes the same philosophy. A developer first creates some UI components with their implemented logic, combined to form a web page. This methodology is called Component Driven Development [6]. The benefit of such an approach is that it allows building web pages fast and with better reusability. Also, this concept is easy to understand as it reminds of building something from Lego.

Unfortunately, building web applications within this methodology requires some programming skills, which can be problematic for people coming from other than software development industries. For that reason, I decided to build an application that would allow the creation of web pages without any programming experience.

4.2 Solution

Analyzing the functional and non-functional requirements and also adopting some concepts from the previous section 4.1, I created a class diagram for the back-end application prototype, and identified necessary entities for the implementation.

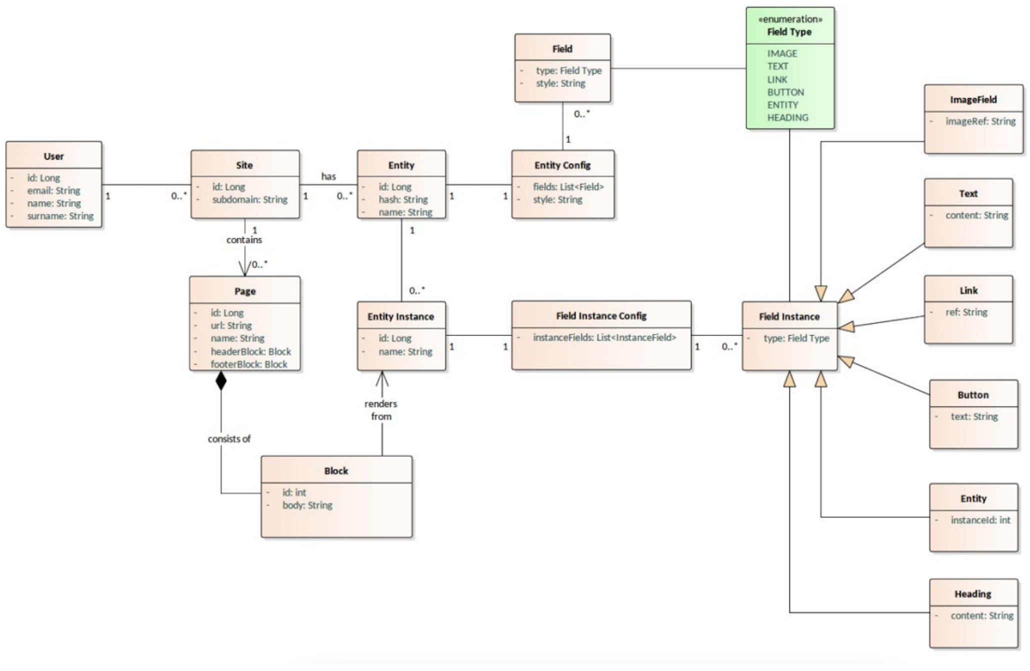


Figure 4.2: Class diagram of the application

4.3 User

A user of this system is a person who can manage their websites. The user is authorized via OAuth2 protocol through their existing Google account, thus it is not required to store user's credentials but just an *email*, *name* and *surname*. As soon as the user was authorized, the application will check if their profile already exists in the database. If not, it will create one.

4.4 Site

A site is a user's website in this system. It is possible for a user to create multiple sites. Each website must have unique *subdomain*. Later, in a complete application, a user will be able to access the websites with a url in the format:

https://[subdomain].droprr.com

4.4.1 Entity

Entity is a core of the whole system. This unit can be understood as a data type for a component on a web page. Entity defines what type of user's data should be presented. In it's configuration a user declares *fields*, their constraints and how should they be organized among with their styling.

4.4.2 Entity Instance

While Entity answers the question "What data structure should be displayed", the Entity Instance answers the question "What data should be displayed".

An Entity Instance is a specific instance of an Entity with specific user data that needs to be rendered. For simplicity, we can draw a parallel with object-oriented programming languages. In this case, the Entity can be considered a class, and the Entity Instance, in turn, is an instance of this class. Entity can have multiple Entity Instances.

4.4.3 Block

Further, a Block can be considered as a concrete representation of an Entity Instance on a web page. Since the Entity and Entity Instance describe the data that should be displayed and how it should be displayed, the Block stores the specific HTML code generated from the Instance.

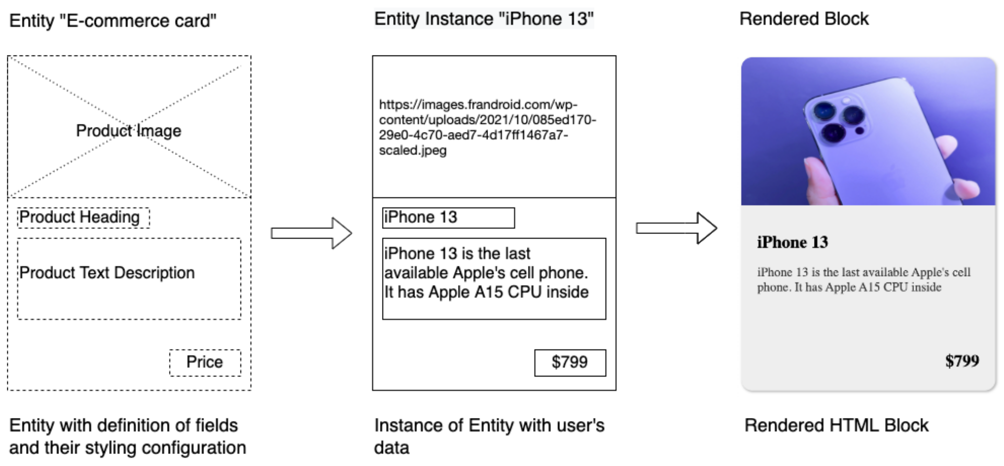

Entity "E-commerce card"
Product Image
Product Heading
Product Text Description
Price
Entity with definition of fields and their styling configuration
Entity Instance "iPhone 13"
https://images.frandroid.com/wp-content/uploads/2021/10/085ed170-29e0-4c70-aed7-4d17ff1467a7-scaled.jpeg
iPhone 13
iPhone 13 is the last available Apple's cell phone. It has Apple A15 CPU inside
$799
Instance of Entity with user's data
Rendered Block
iPhone 13
iPhone 13 is the last available Apple's cell phone. It has Apple A15 CPU inside
$799
Rendered HTML Block

Figure 4.3: Representation of creating HTML element through entities

4.4.4 Page

Each user's website consists of web pages. Each page stores Blocks, from which it will be rendered into a regular HTML page later. Also, to make it possible to get to this page, this entity has a unique *url* attribute within one site.

Chapter 5
Application design

5.1 Application Architecture

Software architecture is a set of rules and constraints that allow building a software system for a specific task. Software architecture plays an essential role in application performance, sustainability, and ability to scale. Choosing the right software architecture pattern is a significant parameter in software systems development [7].

Currently there are lots of existing architecture patterns which have their concrete use cases along with their advantages and disadvantages.

5.2 Layered architecture

The application within this project is built using Layered Architecture. The idea behind the Layered Architecture is to separate the application into logical parts called 'Layers', where each of these layers is responsible for one specific task. Every layer here is isolated from another, which means that editing the components of one layer will not affect components of another layer [8].

Layered architecture is the best choice for this project, because

- the application prototype built within this project is relatively small and development under this approach is fast thanks to the simplicity of the architecture
- each layer is responsible for only one exact task
- each layer is isolated which prevents from mutability of one part of application caused by changes in another one.
- testing is easier because of the separation of application components. Each component can be tested separately

There will be three layers: Database Layer, Persistence Layer and Business Layer.

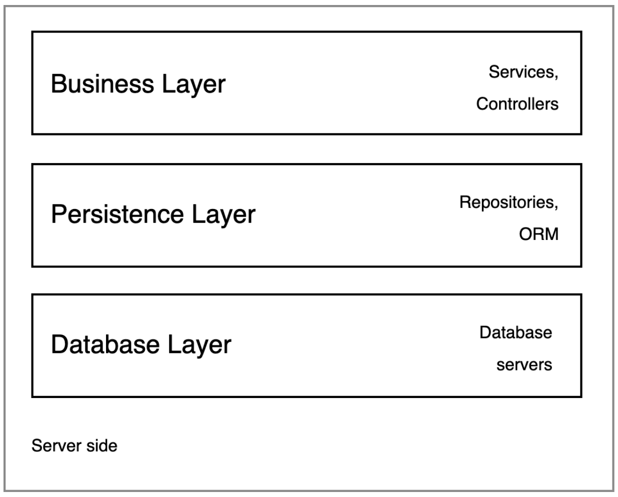


Figure 5.1: Layered application architecture diagram

- Database Layer

  This is the layer of my application where all needed data is stored.

- Persistence Layer

  This is the intermediate layer between the database and business logic. It isolates the application logic from the specific database engine. Persistence Layer uses the technique called Object Relational Mapping for mapping the raw database data to understandable for programming language objects.

- Business Layer

  This is the level of my application where all business logic is concentrated. Business Layer contains a set of rules and logic which are used to implement functional requirements. It also includes API for front-end communication.

5.3 Security

In order to secure users' data it is necessary to provide an application with some mechanism which will prevent from unauthorized access to them. Currently, there are lots of authentication and authorization protocols which can be used for securing data. The application within this thesis is going to have the OAuth2 authorization protocol.

The OAuth2 is currently the industry-standard authorization protocol [9], which was designed to allow an application to access user's data resources hosted on the other application. It grants access to and restricts some of the actions that a client application can take with resources on behalf of a user without sharing user credentials.

This protocol works with Access Token, a piece of data (typically a char sequence) used for accessing API. Users receive the access token after they successfully authorize, and then this token is sent with each API request as a credential. [10].

OAuth2 defines 4 roles [11]:

- Resource Owner - it is a user who has the resources that can be granted to from them.
- Client - an application which requires access to the resources.
- Resource Server - a system which receives and validates Access Tokens from a client and then sends the resources a user can access to.
- Authorization Server - a server which verifies the identity of a user and then issues the Access Tokens.

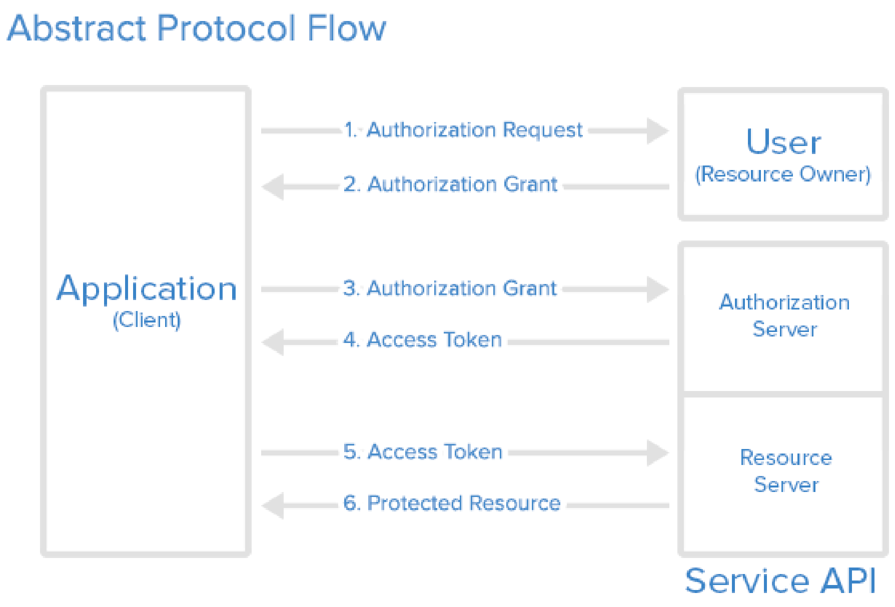


Figure 5.2: OAuth2 workflow [12]

Chapter 6

Technologies

6.1 Language and Framework

There are lots of existing programming languages available for the server-side application development.

During my bachelor's studies I learned a few of them: C, C++, Python, JavaScript and Java. All these programming languages can be used to develop server-side applications.

I have most experience in Java and I find it as a good programming language for server-side application development. However, it has such disadvantages as

- verbose code
- no null safety supported by language

which sometimes negatively affects my application development process.

After doing some research, I found the Kotlin Programming language, which seemed to me truly interesting, because one of its developers stated it was designed to be 'A better Java' [13]. For my bachelor's project I chose it, since it has small learning curve and it solves some Java problems.

6.1.1 Kotlin

Kotlin is a statically-type object oriented programming language running on Java Virtual Machine. It was designed and developed to be a "better language" than Java, but it is also fully compatible with Java code. It has few advantages over Java, which has also influenced my choice in favor of Kotlin.

Language null-safety

Unlike Java, Kotlin has embedded null-safety. It makes a distinction between nullable and non-nullable data types. All nullable variables must be declared with "?" postfix after the type name.
The following code snippet demonstrates an assignment of null value to a nullable variable and to non-nullable one.

```
val nullable: String? = null // ok
val nonNullable: String = null // compilation error
```

Extension functions

Kotlin allows developers to create extensions of functions without a need of class inheritance. In order to extend functions in Java, it is needed to create a new class and implement new functionality.
The following code snippet demonstrates the creation of an extension function to String class. The function adds " :)" to a string.

```
fun String.addSmile(): String {
    return "$this :)"
}

val stringWithSmile = "something".addSmile() // "something :)"
```

Functional Programming

Kotlin supports both Object Oriented and Functional Programming paradigms whereas Java is just Object Oriented Programming language. It brings a developer convenience to use features from Functional Programming when it comes to operation and mapping of huge data coming from different sources.

6.1.2 Spring Framework

Building an application without any tooling is a truly hard task. A developer should care about low-level functionality as handling user's requests or even application boot process. Framework lets a developer focus on the application logic implementation while building the web is framework's task.

There are several Java/Kotlin frameworks available to build web applications such as Spring Framework, Micronaut, Struts and others. During my studies I had a chance to only work with Spring Framework, so it was a the only candidate for my project.

Spring Framework is the most popular and open source framework for Java applications development. It gives Java developers more design freedom. In addition, it provides well-documented and easy-to-use tools for solving problems that arise when creating enterprise applications. Although Spring does not provide any specific programming model, it has spread in the Java community primarily as an alternative and replacement to the Enterprise JavaBeans model.

The key feature of Spring Framework is that most of the objects of an application are not created by a developer but by Spring itself. A developer configures the classes using Java Annotations or XML configurations to let Spring know which objects should be created for a developer. The feature responsible for this task is called the Inversion of Control Container. All the objects which were created and managed by it are called beans [14].

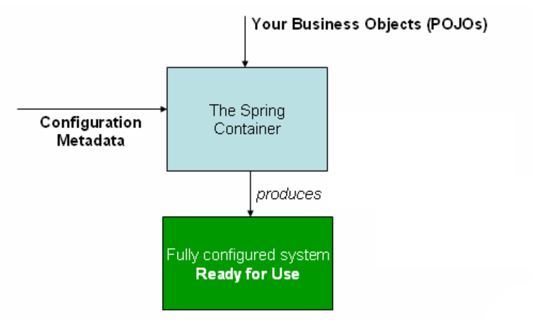
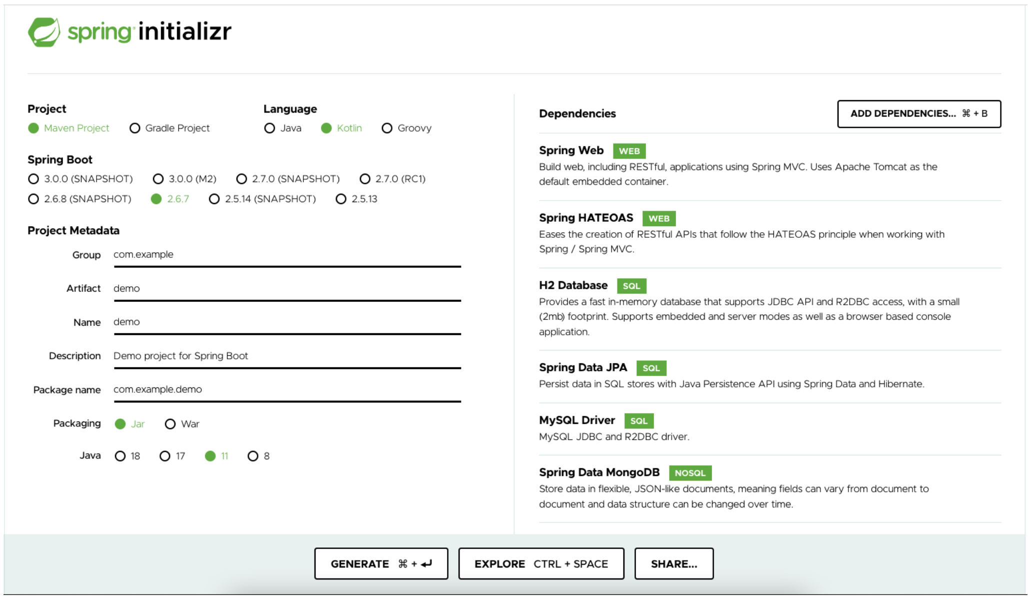


Figure 6.1: Spring IoC Container

Spring Boot

Spring Boot is an extension of Spring Framework. It allows the developers to create standalone applications with minimal configuration. The main advantage of Spring Boot is the automatic configuration based on the requirements. Spring Boot reduces the amount of code needed to perform simple tasks or configuration. It ships with Jetty or Apache Tomcat web servers without the need for explicit server settings [15].

Spring Boot project with all needed dependencies can be easily generated on `https://start.spring.io/`



Project
[✓ Maven Project] Gradle Project

Language
Java [✓ Kotlin] Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M2) 2.7.0 (SNAPSHOT) 2.7.0 (RC1)
2.6.8 (SNAPSHOT) [✓ 2.6.7] 2.5.14 (SNAPSHOT) 2.5.13

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	[✓ Jar] War
Java	18 17 [✓ 11] 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring HATEOAS WEB
Eases the creation of RESTful APIs that follow the HATEOAS principle when working with Spring / Spring MVC.

H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

Spring Data MongoDB NOSQL
Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Figure 6.2: Spring Boot project generation

6.2 Interaction with the application

In order to interact with server-side application a developer should implement Application Programming Interface (API) for it.

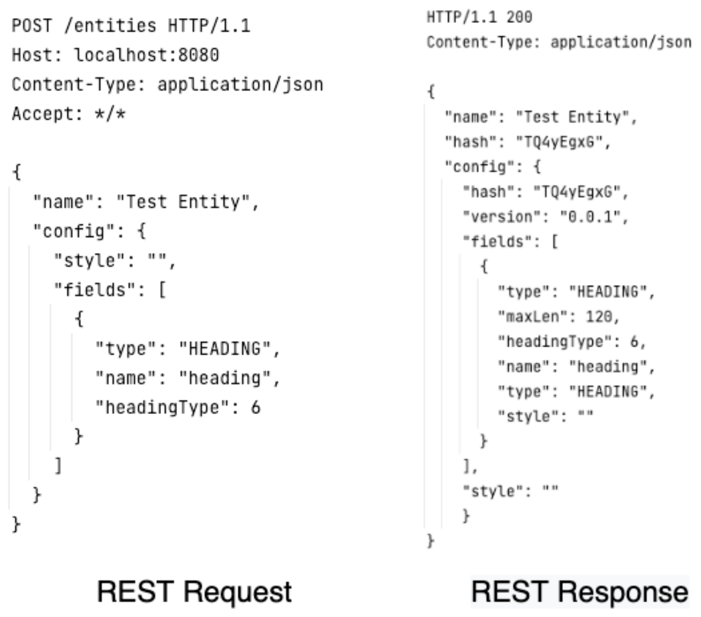
Every web application requires their API to be stable and reliable. For these reasons a developer has to define the architecture of it and their endpoints. There are different approaches for the implementation:

- REST - Representational State Transfer
- SOAP - Simple Object Access Protocol
- GraphQL

6.2.1 REST

REST or Representational State Transfer stands for a set of architectural principles and approaches on how to build a web-application API. Just because it is a set of guidelines, developers are flexible in their implementation.

REST works over HTTP protocol, which means that the communication server-client is done by sending hypertext messages. A response from the server is received in various formats: HTML, XML, JSON, or plain text. Mainly messages are transmitted to the server in XML or JSON format. [16]



```
POST /entities HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Accept: */*

{
  "name": "Test Entity",
  "config": {
    "style": "",
    "fields": [
      {
        "type": "HEADING",
        "name": "heading",
        "headingType": 6
      }
    ]
  }
}
```

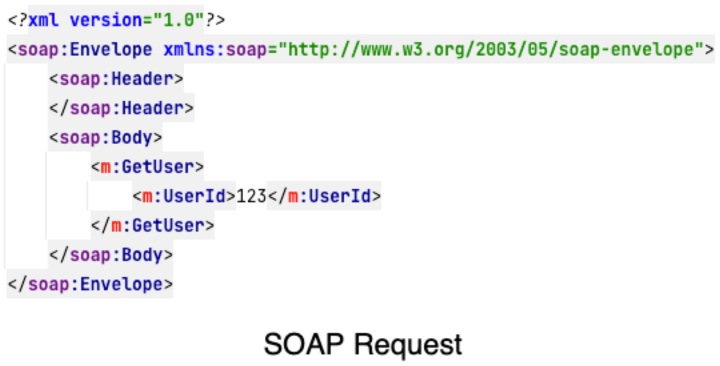
```
HTTP/1.1 200
Content-Type: application/json

{
  "name": "Test Entity",
  "hash": "TQ4yEgxG",
  "config": {
    "hash": "TQ4yEgxG",
    "version": "0.0.1",
    "fields": [
      {
        "type": "HEADING",
        "maxLen": 120,
        "headingType": 6,
        "name": "heading",
        "type": "HEADING",
        "style": ""
      }
    ],
    "style": ""
  }
}
```

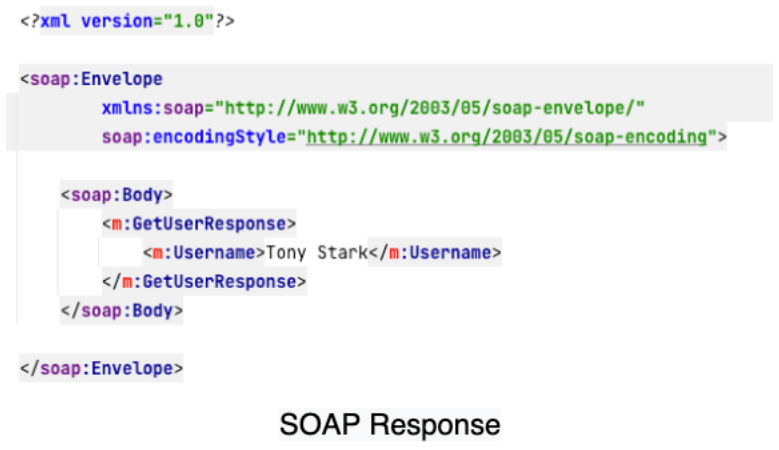
Figure 6.3: Example of POST REST request and response for storing Entity data

6.2.2 SOAP

SOAP or Simple Object Access Protocol is a messaging protocol which was designed so that applications written in different programming languages could communicate in decentralized and distributed environment using XML [17]. Requests to the SOAP API can be handled through different protocols: HTTP, SMTP, FTP and others. However, the response to this request must be returned as XML documents.



```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
    </soap:Header>
    <soap:Body>
        <m:GetUser>
            <m:UserId>123</m:UserId>
        </m:GetUser>
    </soap:Body>
</soap:Envelope>
```



```
<?xml version="1.0"?>

<soap:Envelope
        xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
        soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

    <soap:Body>
        <m:GetUserResponse>
            <m:Username>Tony Stark</m:Username>
        </m:GetUserResponse>
    </soap:Body>

</soap:Envelope>
```

Figure 6.4: Example of SOAP request and response for the user data retrieval

6.2.3 GraphQL

GraphQL is an open-source data query and manipulation language developed by Facebook[18]. HTTP is the most common choice for client-server protocol when using GraphQL. It allows clients to define data required to receive from the server and thus eliminating scenarios of data over-fetching[1]and under-fetching[2]. The main disadvantage of GraphQL is that it is not cacheable[19] and does not provide error reporting (always returns HTTP Status 200).

[1]A scenario, when there is too much unused data received from the server

[2]A scenario, when there is not enough data received from the server and thus forcing a user to make another endpoint call

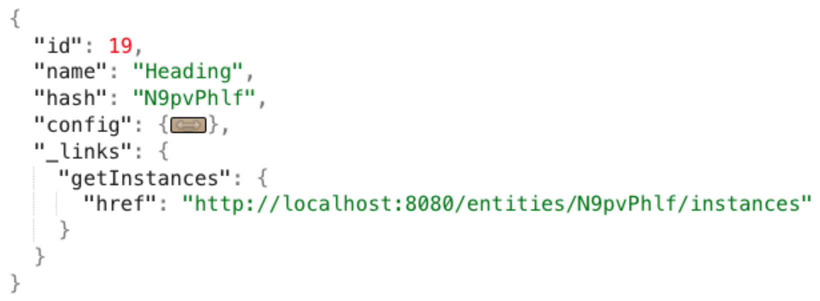
6.2.4 Chosen approach

The application within this project will expose REST API. It is a proven technology that has been on the market for a long time. Also REST is easy to learn and implement.

6.2.5 HATEOAS

HATEOAS stands for Hypertext As The Engine Of Application State. It is a constraint used for building RESTful API [20] in order to provide some notion of actions that can be performed on the data received from REST endpoints. Usually a developer who wants to implement some logic using someone's API has no clue on what action can be performed on received data from one endpoint. HATEOAS is a great tool for solving this exact problem. It allows not only to send data to a requester but also enables to specify the actions that can be performed on this data.

For example, response to HTTP GET request to */entities/N9pvPhlf* instead of just sending rough data will also send *_links* object with hypermedia links that a user of this API can traverse.



```
{
  "id": 19,
  "name": "Heading",
  "hash": "N9pvPhlf",
  "config": {↔},
  "_links": {
    "getInstances": {
      "href": "http://localhost:8080/entities/N9pvPhlf/instances"
    }
  }
}
```

Figure 6.5: Example of HATEOAS in the application

6.3 Database

The application built in this project is highly dependent on user's data and thus it needs to store it in a database. Currently, there are different types of databases available on the market including relational and NoSQL ones. Relational databases have strict data structure, every piece of data must be stored in tables having predifined constraints. NoSQL, on the other hand, does not have such strict data organization, which makes their usage very flexible, since the data stored there can be constantly changed.

The output application requires storing data in tables to have relations between them. For instance, it would be required to store tables for *Entities*, *Entity Instances*, *Sites*, *Pages*, and *Blocks*. This data structure is already predefined, and it would be easier to work with relations between these tables. On the other hand, the application would need to store Entity and Entity Instance configuration information. This data structure is considered to be highly changeable, and the table constraints of relational databases would decrease productivity. For example, a Form Field of Entity may consist of multiple inputs. Storing each input as a row in a strict table may become a problem because each "feature" of input will need its column, which means many columns, which can become messy. In this case, storing Forms and their Inputs is much easier in JSON-like objects, which NoSQL databases like MongoDB can provide.

I have chosen MySQL and MongoDB databases for this project.

6.4 Caching

With each API request, excluding endpoints for Site creation, users need to include *X-Droprr-Host* header with their website domain to make changes within this website. Each time server receives this header, it looks into a database in order to check whether this domain is present among the user's websites. This operation can be costly because of frequent database requests and their validations. For this reason, it would make sense to cache the website data to ease the load off the database.

In order to achieve caching, I have chosen Redis. It is an open-source key-value data store [21]. Redis stores the data in memory, which means it does not require a trip do disc, thus reducing latency to microseconds [22].

Chapter 7
Implementation

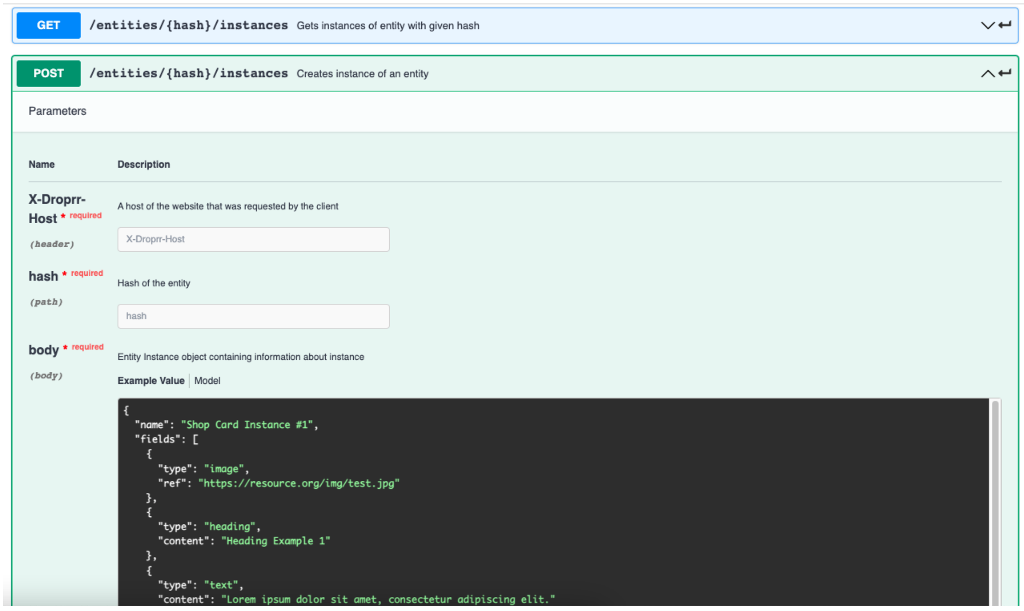
This chapter aims to describe the thesis application development process. All the skills acquired during the bachelor's studies concerning software development, including design, implementation, and testing, were applied during the implementation.

7.1 REST endpoints definition

After the requirements and system analysis, it was needed to design and document REST API endpoints.

First, it was necessary to identify the resources on which an API consumer would perform the actions. In this application, the resources are Sites, Pages, Entities, Entity Instances, and Blocks. Next, it was necessary to define the URLs for the resources and HTTP methods that can be applied to them. After all these steps, things like resource representations, required HTTP headers, error responses, and HTTP response stats codes were needed to resolve.

To specify the REST API endpoints, a tool Swagger was used. It is an open-source tool which allows developers to describe the structure of API. It uses JSON or YAML strurture to write an API definition, which can be further rendered to interactive API documentation [23].



GET `/entities/{hash}/instances` Gets instances of entity with given hash

POST `/entities/{hash}/instances` Creates instance of an entity

Parameters

Name	Description
X-Droprr-Host * required (header)	A host of the website that was requested by the client X-Droprr-Host
hash * required (path)	Hash of the entity hash
body * required (body)	Entity Instance object containing information about instance Example Value | Model

```
{
  "name": "Shop Card Instance #1",
  "fields": [
    {
      "type": "image",
      "ref": "https://resource.org/img/test.jpg"
    },
    {
      "type": "heading",
      "content": "Heading Example 1"
    },
    {
      "type": "text",
      "content": "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
```

Figure 7.1: Example of rendered API documentation in Swagger UI

7.2 Application Initialization

After the endpoints definition, the application base was created with the help of Spring Initializr. This tool generates Spring Boot projects with chosen by a developer dependencies. The pre-configured project is saved on a local machine afterwards.

The application was created as a Gradle project with Kotlin as a main language. The application also includes the following dependencies:

- Spring Web - includes configurations for building RESTful API.
- Spring HATEOAS - eases the RESTful API creation that follow HATEOAS principle.
- Spring Data JPA - enables data persistence in SQL stores.
- H2 Database - Provides in-memory database, which would be used for testing.
- MySQL Driver - provides JDBC driver for MySQL database to the application.
- Spring Data MongoDB - provides MongoDB database connection.
- OAuth2 Client - adds OAuth2 Client features

7.3 Application Development

7.3.1 Environment configuration

The first step after the project initialization was to configure the local development environment. First, it was necessary to set up a version control system in order to track the code changes. Such systems have become as one of the most popular tool in today's development [24], since they make the process of software development easier. Although such systems are commonly used while developing software in teams, they can also be beneficial in development alone. Sometimes comes the situation when a developer wants to abort their code changes because it is causing some software bugs. In this case, reverting the last change (or 'commit') can solve this problem. The most popular version control system is Git, which was used in this project.

Next, the databases MySQL and MongoDB used in this project were set up and run inside Docker containers. Docker is a platform for software development that provides the ability to package and run an application in an isolated environment called a 'container' [25]. It is a very powerful tool, especially when it comes to running different services on one machine. It was convenient to configure and run containers with desired databases and their configurations instead of installing these programs and their environments on a machine. Further, Docker makes the deployment process easier since it is only required to run a single container with an application instead of setting up the whole environment.

7.4 Business Logic implementation

The business logic implementation has started from the implementation of REST controller and Service class for Entity and Entity Instance.

First, in order to operate with these structures and their data, it was necessary to implement the logic for CRUD operations and save them to the database. Also, it was required to implement a service class with the logic of the Entity and Entity Instance Configurations. When creating an Entity, it is required to send a *config* object inside the request body with all necessary Entity Fields declaration, their constraints (for example, a restriction in amount of characters for the heading text) and styling. Because the application needs to operate with invalid inputs, it was necessary to implement the logic for the request object validation and the user's error handling.

Next, for Entity instantiating, it was essential to verify if the user's configuration object with specific data has the same field types and if the data in these fields comply with the constraints declared during entity creation.

A Block service had to be implemented for a user's data display. The logic of Blocks is straightforward: It takes the user's data from Entity Instance, the styling from Entities, and renders it all as an HTML block. Also, it is important to mention that since each Entity can contain other Entities, and thus Entity Instances contain other instances, it is also necessary to render them. In this fashion, Blocks rendering can be considered as a tree traversal.

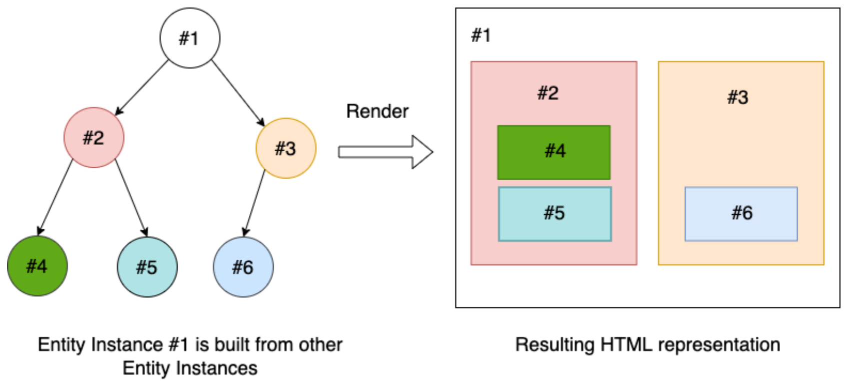

#1
#2
#3
#4
#5
#6
Render
Entity Instance #1 is built from other Entity Instances
Resulting HTML representation

Figure 7.2: Example of Block rendering from a complex Entity Instance

After implementing the Entities, their Instances, and Blocks logic, comes the implementation of Page and Site services. Fortunately, the development of these components was easier and faster. A web page must be renderable, and to achieve this, a service combines all its blocks and inserts the result inside *body* block of an HTML. A page must have a unique *url* attribute within one Site.

In order for users to edit their websites, a *X-Droprr-Host* header must be present. It contains the current site domain for which a user wants to make changes.

7.5 Authorization

As soon as the business logic of the application was implemented, there was left a final step towards the complete prototype - authorization. As it was mentioned in Section 5, the OAuth2 authorization protocol was used in this application. Since the whole protocol implementation from scratch would take a big amount of time, dependency OAuth2-Client for Spring Boot was there. The application was developed as a Client in the OAuth2 workflow, which was accessing users' profile data from their Google accounts. Google Services, on the other hand, was granted access to these resources and issued Access Tokens. This extension OAuth2-Client made it possible to implement all this functionality by just implementing a few Kotlin classes.

Chapter 8
Testing

Program testing is an essential part of any software project. It is a process of executing a program with different inputs in order to find errors of the system.

There are four main stages of software testing[26]:

- Unit testing - testing of small components of a program.
- Integration testing - testing of units, components and modules as a combined entity.
- System testing - testing of a whole application if it meets Quality Standards
- Acceptance testing - testing if an application is ready to release.

8.1 Unit Testing

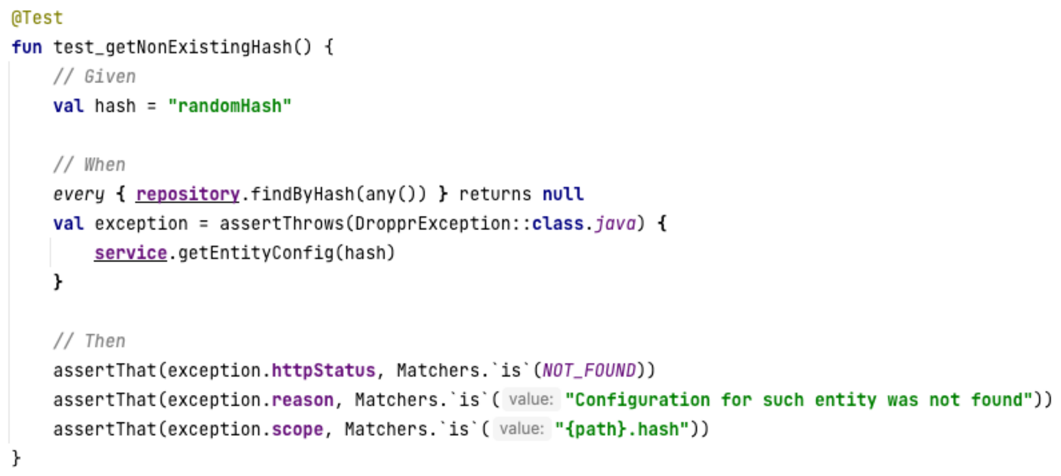
Unit testing is one of the software testing method in which small parts of application (called 'units') are being tested. The goal of it is to test isolated parts of a program in order to verify that these individual parts work correctly and as expected. Also the biggest advantage of unit testing is that it allows to find potential problems earlier in software development life-cycle.

In my application I wrote several test cases which check whether the required features are present in code and the individual parts of program work correctly. Within this part I tested CRUD operations with repositories and some business behaviour of my application with use of mocks[1] and stub methods[2]. Unit testing helped me to find errors and potential vulnerabilities in my code.

In my application I used JUnit 5 as a Unit Testing framework for Java and Kotlin programming languages. For object mocking I used the MockK library, which was developed mainly for Kotlin Programming language. Since each class is final by default in Kotlin, it can cause some problems in testing, especially mocking of objects. The MockK library solves this problem by creating proxies for mocking classes.

[1]Mock object is an object which simulates the behaviour of real object

[2]Method stub is a piece of code simulating the behaviour of existing code.



```kotlin
@Test
fun test_getNonExistingHash() {
    // Given
    val hash = "randomHash"

    // When
    every { repository.findByHash(any()) } returns null
    val exception = assertThrows(DropprException::class.java) {
        service.getEntityConfig(hash)
    }

    // Then
    assertThat(exception.httpStatus, Matchers.`is`(NOT_FOUND))
    assertThat(exception.reason, Matchers.`is`( value: "Configuration for such entity was not found"))
    assertThat(exception.scope, Matchers.`is`( value: "{path}.hash"))
}
```

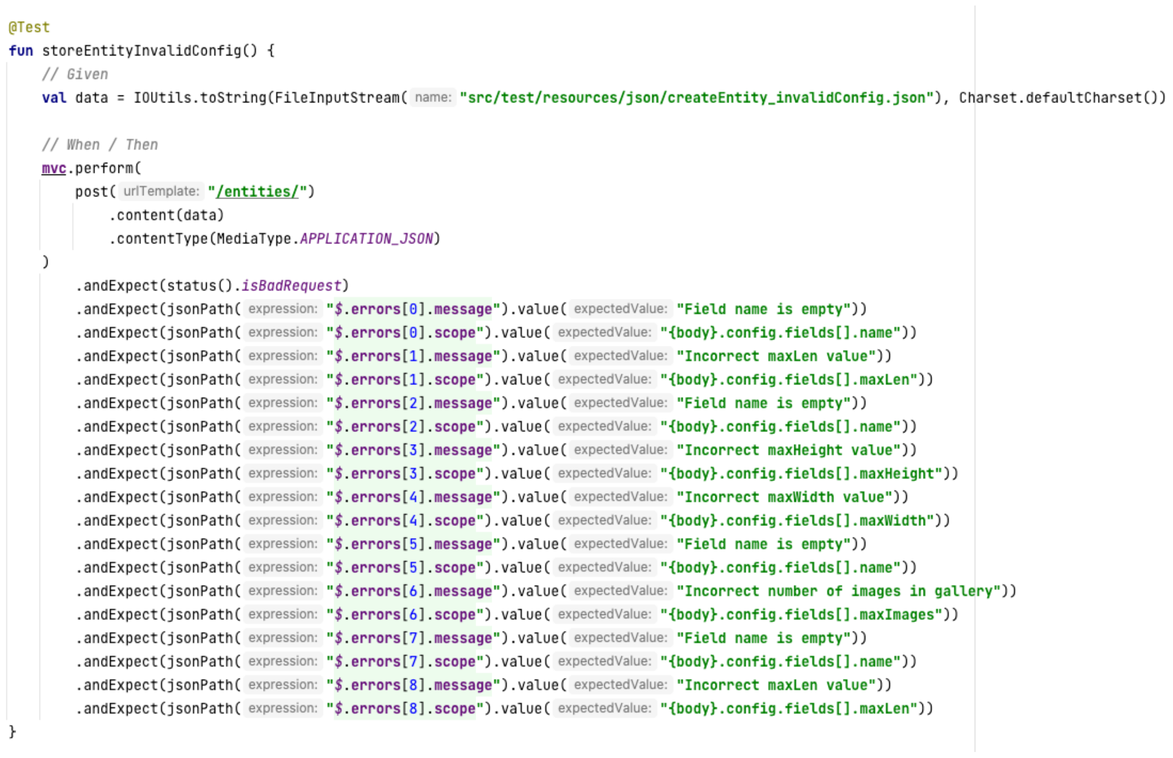
Figure 8.1: Example of test for testing the return of non-existing entity

8.2 Integration testing

Integration testing is a phase of software testing, where all the units are tested as a group. The reason for the integration testing is to find defects and problems which can arise between modules or functions. This testing was mainly focused on performing HTTP request and thus testing different parts of the application: Controllers, Services, Error Handlers and Repositories.

8.3 Detected problems

During testing, I found problems regarding the invalidity of user input, and because of this, I created separate objects that validate user data. Also, by writing tests, I found defects in my program regarding the interaction of several services, which I also successfully corrected.



```
@Test
fun storeEntityInvalidConfig() {
    // Given
    val data = IOUtils.toString(FileInputStream( name: "src/test/resources/json/createEntity_invalidConfig.json"), Charset.defaultCharset())

    // When / Then
    mvc.perform(
        post( urlTemplate: "/entities/")
            .content(data)
            .contentType(MediaType.APPLICATION_JSON)
    )
        .andExpect(status().isBadRequest)
        .andExpect(jsonPath( expression: "$.errors[0].message").value( expectedValue: "Field name is empty"))
        .andExpect(jsonPath( expression: "$.errors[0].scope").value( expectedValue: "{body}.config.fields[].name"))
        .andExpect(jsonPath( expression: "$.errors[1].message").value( expectedValue: "Incorrect maxLen value"))
        .andExpect(jsonPath( expression: "$.errors[1].scope").value( expectedValue: "{body}.config.fields[].maxLen"))
        .andExpect(jsonPath( expression: "$.errors[2].message").value( expectedValue: "Field name is empty"))
        .andExpect(jsonPath( expression: "$.errors[2].scope").value( expectedValue: "{body}.config.fields[].name"))
        .andExpect(jsonPath( expression: "$.errors[3].message").value( expectedValue: "Incorrect maxHeight value"))
        .andExpect(jsonPath( expression: "$.errors[3].scope").value( expectedValue: "{body}.config.fields[].maxHeight"))
        .andExpect(jsonPath( expression: "$.errors[4].message").value( expectedValue: "Incorrect maxWidth value"))
        .andExpect(jsonPath( expression: "$.errors[4].scope").value( expectedValue: "{body}.config.fields[].maxWidth"))
        .andExpect(jsonPath( expression: "$.errors[5].message").value( expectedValue: "Field name is empty"))
        .andExpect(jsonPath( expression: "$.errors[5].scope").value( expectedValue: "{body}.config.fields[].name"))
        .andExpect(jsonPath( expression: "$.errors[6].message").value( expectedValue: "Incorrect number of images in gallery"))
        .andExpect(jsonPath( expression: "$.errors[6].scope").value( expectedValue: "{body}.config.fields[].maxImages"))
        .andExpect(jsonPath( expression: "$.errors[7].message").value( expectedValue: "Field name is empty"))
        .andExpect(jsonPath( expression: "$.errors[7].scope").value( expectedValue: "{body}.config.fields[].name"))
        .andExpect(jsonPath( expression: "$.errors[8].message").value( expectedValue: "Incorrect maxLen value"))
        .andExpect(jsonPath( expression: "$.errors[8].scope").value( expectedValue: "{body}.config.fields[].maxLen"))
}
```

Figure 8.2: Example of test for response for invalid request of entity creation

Chapter 9

Further steps

The application prototype provides some basic functionality in order to represent the key-features of the system. This chapter describes the next steps for the development of this product along with their future features.

9.1 Linking Entities together

Currently, each Entity can be instantiated and rendered as an HTML block to form a web page. Unfortunately, due to lack of time, I did not manage to add a mechanism for linking multiple entities together so that actions performed on the first Entity can affect the second one. In order to achieve that, it is necessary to extend the Entity's Config and add new fields which will implement the logic of association between the Entities.

9.2 Front-end integration

In this project, I created a server-side application of a no-code platform for web-pages creation and hosting. Although it is possible to communicate with this application via REST endpoints, it is still unusable for end users. The obvious next step would be to develop a client-side application that would allow users to interact with the application and create their own websites.

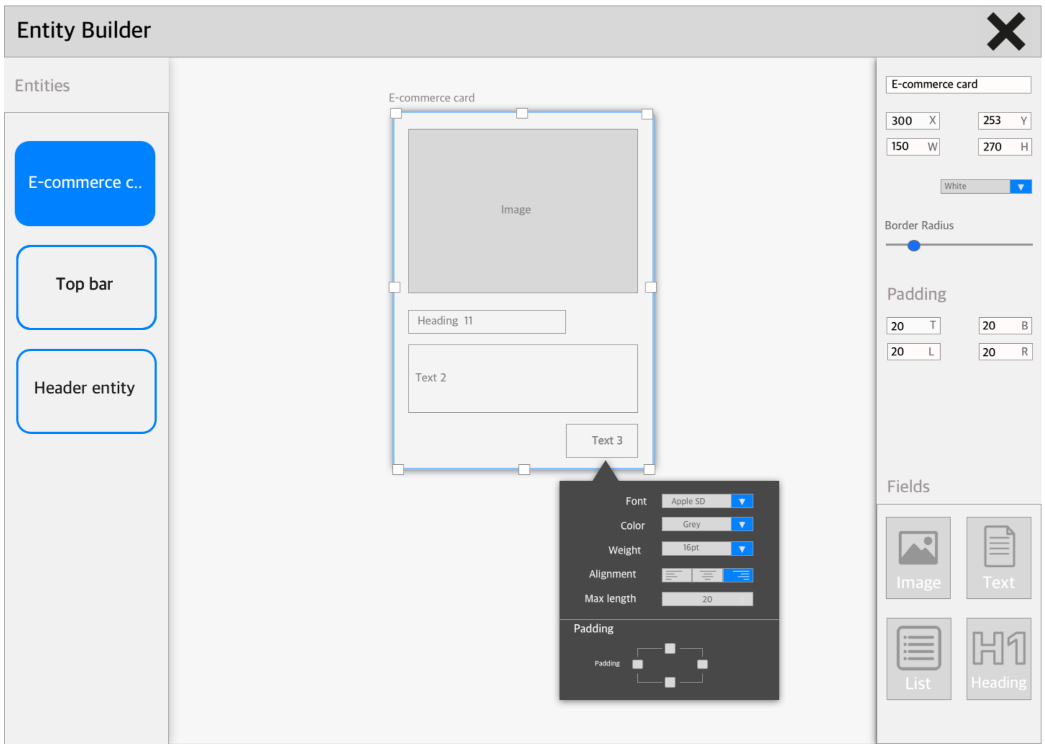


Figure 9.1: Wireframe of Entity Builder on front-end

9.3 Predefined Entities and templates

In order not to build a site from scratch, a user could choose a ready-made template with prepared Entities and configure them as needed. It would accelerate users' website creation as they only will need to configure fewer things.

9.4 Websites statistics

As a next step, the system would provide a user with their website statistics.

9.5 E-commerce support

As some of the users would want to make e-commerce systems for their business, the application will provide a functionality of cards and checkout process.

9.6 Localization

Application will allow users to have their websites in different languages.

9.7 Team collaboration

At the moment, only one user can make changes to his website. The application will make it possible to collaborate on the same project in the future.

Chapter 10
Conclusion

This bachelor's thesis aimed to design an architecture of the server-side of a no-code application for web page creation and hosting. First, I stated the motivation for this project, devoted to the issue definition and the analysis of existing solutions on the market, and the definition of prototype requirements.

Next, I described the process of the stated problem solution. In this part, I designed the concept and the architecture of a prototype. I also described my reasons to choose the particular technologies for the application development and further described the software implementation process. During my work on the thesis, I applied the knowledge about software development and the tools gained during my bachelor's studies. I also learned new technologies during the process of implementation. These were, for example, Docker, Redis, and MongoDB, with which I had no work experience before.

My work results are a working prototype, which will be possible to expand with new features. I believe that this solution will be commercially successful in the future, and, more importantly, it will make the process of website creation more straightforward and faster.

Bibliography

[1] N. Huss *How Many Websites Are There in the World?*, [online]. [Cit. 7.04.2022] URL: https://siteefy.com/how-many-websites-are-there

[2] Daniela Coppola *E-commerce worldwide - statistics facts*, [online]. [Cit 23.02.2022] URL: https://www.statista.com/topics/871/online-shopping/

[3] Avishay Cohen *No-code vs low-code vs developers*, [online]. [Cit. 17.01.2022] URL: https://www.animaapp.com/blog/industry/will-no-code-replace-developers/#what-is-low-code

[4] Brad Frost *Atomic Design*, [online]. [Cit. 10.06.2013] URL: https://bradfrost.com/blog/post/atomic-web-design/

[5] Brad Frost *Atomic Design Methodology. Chapter 2*, [online]. URL: https://atomicdesign.bradfrost.com/chapter-2/

[6] Jack Pritchard *Component Driven Development (CDD) In React*, [online]. [Cit. 14.11.2019]

[7] Babu, D.K., Rajulu, G. P., Reddy R. A., Kumari A. A. N. *Selection of Architecture Styles using Analytic Network Process for the Optimization of Software Architecture* [International Journal of Computer Science and Information Security, Vol. 8, No. 1]. [Cit. 04.2010]

[8] Chavan, Pramod and Mahalingam, Murugan and Chavan, Prajakta. *A Review on Software Architecture Styles with Layered Robotic Software Architecture* [DOI: 10.1109/ICCUBEA.2015.165] [Cit. 07.2015]

[9] *OAuth 2.0* [online]. URL: https://oauth.net/2/

[10] *Access Tokens* [online]. URL: https://auth0.com/docs/secure/tokens/access-tokens

[11] *What is OAuth 2.0?* [online]. URL: https://auth0.com/intro-to-iam/what-is-oauth-2/

[12] *An Introduction to OAuth 2* [online]. URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

[13] *JVM Languages Report: Extended Interview With Kotlin Creator Andrey Breslav*, [online]. URL: https://www.jrebel.com/blog/interview-with-kotlin-creator-andrey-breslav

[14] *Spring Framework Documentation 5.3.20* [online]. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core

[15] *Spring Boot* [online]. URL: https://spring.io/projects/spring-boot

[16] Red Hat *REST vs. SOAP* [online]. URL: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest

[17] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer *Simple Object Access Protocol (SOAP) 1.1* [online]. [Cit. 08.05.2000] URL: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383487

[18] Red Hat *What is GraphQL?* [online]. URL: https://www.redhat.com/en/topics/api/what-is-graphql

[19] *GraphQL Documentation: Caching* [online]. URL: https://graphql.org/learn/caching/

[20] *HATEOAS Driven REST APIs* [online]. URL: https://restfulapi.net/hateoas/

[21] *Introduction to Redis* [online]. URL: https://redis.io/docs/about/

[22] *Redis* [online]. URL: https://aws.amazon.com/redis/

[23] *Swagger Documentation - What Is OpenAPI?* [online]. URL: https://swagger.io/docs/specification/about/

[24] *Programming/development tools used by software developers worldwide from 2018 to 2021* [online]. URL: https://www.statista.com/statistics/869106/worldwide-software-developer-survey-tools-in-use/

[25] *Docker docs* [online]. URL: https://docs.docker.com/get-started/overview/

[26] LaTonya Pearson *The Four Levels of Software Testing*, [online]. [Cit. 11.09.2015] URL: https://www.seguetech.com/the-four-levels-of-software-testing/