



Assignment of bachelor's thesis

Title:	On-line Exploration of Fractals Powered by Julia
Student:	Martin Ondejka
Supervisor:	Ing. Tomáš Kalvoda, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Web Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

1. Get familiar with a few well-known fractals (e.g., Julia set, Mandelbrot set, Newton fractal, Twindragon, Lindenmayer system, Koch snowflake, Sierpinski triangle). Review their basic properties and construction.
2. Select at least four fractals and design their visualization in the Julia programming environment. Focus on the performance of your solution and select a suitable fast graphics library. Implement and test your solution.
3. Review possible ways how to interactively present results of your fractal computations on the web. Design a solution that will cooperate with your code as seamlessly as possible. Consider also purely Julian tools (e.g., Genie framework or Interact.jl).
4. Implement, deploy and test your solution.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

On-line Exploration of Fractals Powered by Julia

Martin Ondejka

Department of Software Engineering
Supervisor: Ing. Tomáš Kalvoda, Ph.D.

May 10, 2022

Acknowledgements

I would like to thank my supervisor, Ing. Tomáš Kalvoda, Ph.D., for all the time and dedication he provided during writing of this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No.121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 10, 2022

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2022 Martin Ondejka. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Ondejka, Martin. *On-line Exploration of Fractals Powered by Julia*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Hlavným cieľom tejto práce je implementácia webovej aplikácie na vizualizáciu fraktálov v programovacom jazyku Julia. Účel tejto aplikácie má slúžiť ako proof-of-concept pre budúce aplikácie s náročnými výpočtovými požiadavkami. Pre dosiahnutie tohto cieľu, sme navrhli architektúru riešenia s vlastnosťami prispôsobenými požiadavkám a následne sme ju implementovali s detailným vysvetlením.

Kľúčová slova fraktál, Mandelbrotova množina, Juliova množina, Kochova vložka, Twindragon krivka, webová aplikácia, Julia, Genie, React, PostgreSQL, Docker

Abstract

The main goal of this thesis is to implement fractals visualization web application in programming language Julia. The purpose of this application should serve as a proof-of-concept for future Julia web applications with demanding computation requirements. To achieve this, we proposed the architecture with the properties according to requirements and then implemented it with detailed explanation.

Keywords fractal, Mendelbrot set, Julia set, Koch snowflake, Twindragon curve, web application, Julia, Genie, React, PostgreSQL, Docker

Contents

Introduction	1
Goal of thesis	3
1 State of the art	5
1.1 Fractal	5
1.1.1 Mandelbrot set	6
1.1.2 Julia set	6
1.1.3 Twindragon	6
1.1.4 Koch snowflake	7
1.2 Existing solutions	8
2 Design, architecture and technologies	9
2.1 Requirements	9
2.1.1 Functional requirements	9
2.1.2 Non-functional requirements	9
2.2 Architecture	10
2.2.1 Julia package	10
2.2.2 Client-server architecture	10
2.2.2.1 Two tier architecture	10
2.2.2.2 Three tier architecture	11
2.2.3 Front-end	11
2.2.3.1 Document Object Model	11
2.2.3.2 Single Page Application	12
2.2.4 Back-end	12
2.2.4.1 API	12
2.2.4.2 REST API	12
2.2.4.3 JSON-RPC API	13
2.2.4.4 Active Object	13
2.2.4.5 Relational database	14

2.3	Technologies	14
2.3.1	Git	15
2.3.2	Julia	15
2.3.3	Julia Pkg	15
2.3.4	Luxor	15
2.3.5	React	15
2.3.6	Genie	16
2.3.7	SearchLight ORM	16
3	Implementation	19
3.1	Fractals Julia package	19
3.1.1	Environment	19
3.1.2	Julia Set	20
3.1.3	Mandelbrot Set	21
3.1.4	Koch Snowflake	23
3.1.5	Twindragon curve	24
3.2	Front-end	27
3.2.1	Environment	27
3.2.2	Gallery	27
3.2.3	Fractal form	28
3.2.4	Fractal detail	30
3.3	Back-end	33
3.3.1	Environment	33
3.3.1.1	Installation	34
3.3.1.2	Configuration	34
3.3.2	Database	35
3.3.2.1	Repository	35
3.3.2.2	Activation Queue	36
3.3.2.3	Migrations	36
3.3.2.4	Configuration	37
3.3.3	Web API	37
3.3.3.1	JSON-RPC	38
3.3.3.2	REST API	38
3.3.4	Dispatcher	38
3.3.4.1	Active object roles	38
3.3.4.2	Single dispatch method	39
3.3.4.3	Distributed computing	40
3.4	Deployment	41
3.4.1	Dockerfile	41
	Conclusion	43
	Viable future improvements	43
	Bibliography	45

A	Acronyms	49
B	Contents of enclosed media	51

List of Figures

2.2	Three tier architecture.	11
2.3	Active object diagram.	14
2.1	UML component diagram of Fractals portal.	17
3.1	Fractals directory tree.	19
3.2	Julia set fractal with $c = -0.8 + 0.156i$ for subset $K = \{z \in \mathbb{C} z = a + bi, a \in [-0.5, 0.5], b \in [-0.5, 0.5]\}$	21
3.3	Mandelbrot set fractal for subset $K = \{z \in \mathbb{C} z = a + bi, a \in [-2, 1], b \in [-1.5, 1.5]\}$	22
3.4	Koch snowflake for $n = 10$	24
3.5	Twindragon curve for $n = 9$	25
3.6	client directory tree.	27
3.7	Gallery of the Fractals portal.	28
3.8	Form of the Koch Snowflake fractal.	29
3.9	Image tab of fractal detail page.	31
3.10	Input data tab of fractal detail page.	32
3.11	server directory tree.	33
3.12	config directory tree	35

Introduction

Programming language Julia is high-level, high-performance and dynamic language. It's usage is very broad, but through it's advantages and special syntax it can be mostly found in computational science and numerical analysis. [1]

The aim of this thesis is to create interactive web demonstration of few selected fractals using Julia. Since fractal rendering is one of the mathematical problems, in which programming language Julia excels, it's suitable for this demonstration.

Main acquisition of the thesis is proof of concept of binding mathematical programming language Julia with development of modern web applications without using the overhead of other back-end frameworks.

First chapter aims to gain literature overview about the subject of fractals and to define basic mathematical properties of few chosen demonstration fractals.

Second chapter outlines the functional and non-functional requirements of the application, establishes the underlying architecture and determines the tools to achieve the goals.

Third chapter is dedicated to the implementation details of all of the parts of the application.

Goal of thesis

The main goal is to create an interactive web application to visualize a few well-known fractals.

The first chapter aims to gain a literature overview of a few well-known fractals (Julia set, Mandelbrot set, Twindragon, Lindenmayer system, Koch snowflake) and to review their basic properties, mathematical definition and construction.

The next chapter focuses on outlining the architecture of the final web application, which includes reviewing possible ways to visualize fractals on the web and deciding on used technologies.

The last objective is to implement designed solution. That includes using acquired math knowledge to implement package for visualization of mentioned fractals in Julia programming environment, and implement web application using chosen technologies.

State of the art

1.1 Fractal

Classical or Euclidean geometry studies geometric objects such as lines, squares, cubes — geometric objects with discrete dimensions. Fractals are also geometric objects, however they have specific properties that distinguish them and cannot be classified as objects of classical geometry. [2]

The term fractal, derived from the Latin word fractus (“fragmented”), was coined by the Polish-born mathematician Benoit B. Mandelbrot. [3]

According to K. Falconer, properties of fractals that distinguish them from classical objects are [4]:

- **Self-similarity:** Fractal objects have similar structure on different scales.
- **Irregularity:** Fractal’s image patterns cannot be described by traditional Euclidean geometry.
- **Fractional dimension:** In the world of fractals it is necessary to generalize the concept of dimension, introducing the possibility of fractional values. There are multiple definitions of fractal dimension, here is one from Michael F. Bamsley. For more detailed explanation see [5].

Definition 1.1.1 (Fractal dimension). Let (X, d) denote a complete metric space. Let $A \in \mathcal{H}(X)$ be a nonempty compact subset of X , where (X, d) is a metric space¹. Let $B(x, \epsilon)$ denote the closed ball of radius ϵ and center at a point $x \in X$. For each $\epsilon > 0$, let $\mathcal{N}(A, \epsilon)$ denote the smallest number of closed balls of radius $\epsilon > 0$ needed to cover A . If

$$D = \lim_{\epsilon \rightarrow 0} \frac{\ln(\mathcal{N}(A, \epsilon))}{\ln(1/\epsilon)}$$

exists, then D is called *the fractal dimension of A* . We will also use the notation $D = D(A)$ and will say “ A has fractal dimension D .”

¹https://en.wikipedia.org/wiki/Metric_space

1.1.1 Mandelbrot set

Mandelbrot set is fractal named after "father" of the fractal geometry Benoit B. Mandelbrot. Although, the set was first discovered by Robert W. Brooks and Peter Matelski [6], Mandelbrot later visualized it.

Mandelbrot set is set of complex numbers c for which the iterations of the function $P_c(z) = z^2 + c$ does not diverge to ∞ when iterated from $z = 0$. [7]

Definition 1.1.2 (Mandelbrot set). The *Mandelbrot set* is the set $M = \{c \in \mathbb{C} \mid \exists s \in \mathbb{R}, \forall n \in \mathbb{N}, |P_c^n(0)| \leq s\}$ where $P_c^n(z)$ is the n th iterate of $P_c : z \rightarrow z^2 + c$.

An illustration of the Mandelbrot set can be found in Figure 3.3.

1.1.2 Julia set

Julia set bears the name of French mathematician Gaston Julia of the 20th century. [8]

Let $c \in \mathbb{C}$ be a constant, Julia set is the set of complex numbers $z \in \mathbb{C}$ for which the iterations of the function $P_c(z) = z^2 + c$ does not diverge to ∞ . [9]

Definition 1.1.3 (Filled-in Julia set). The *Filled-in Julia set* is the set $K = \{z \in \mathbb{C} \mid \exists s \in \mathbb{R}, \forall n \in \mathbb{N}, |P_c^n(z)| \leq s\}$ where $P_c^n(z)$ is the n th iterate of $P_c : z \rightarrow z^2 + c$.

The Julia set is then the boundary between the complex numbers c that diverge and between those that converge.

An illustration of the Julia set can be found in Figure 3.2.

1.1.3 Twindragon

The Twindragon is fractal curve, that can be constructed recursively with method such as Lindenmayer system.

Lindenmayer (L-system) system is a type of formal grammar where generated string serves as a instruction list to generate geometric structure. [10]

L-systems are defined as a tuple

$$G = (V, \omega, P)$$

where

- V is the alphabet containing both variables and terminals
- ω is the initiator
- P is the set of production rules.

During every iteration of string structuring, as many rules as possible are applied, not only one rule per iteration, as it is in formal grammars.

The Twindragon curve can be constructed with the following context-free L-system. [11]

$$G_1 = (\{A, B, r, l\}, S, P)$$

Where P is

$$S \rightarrow ArB$$

$$A \rightarrow ArB$$

$$B \rightarrow AlB$$

A and B are instructions to go forward of predefined length, and r and l are instructions to turn right and left by 90° respectively.

So iteratively for:

- $n = 1 \rightarrow ArB$
- $n = 2 \rightarrow ArBrAlB$
- $n = 3 \rightarrow ArBrAlBrArBlAlB$

An illustration of the Twindragon curve can be found in Figure 3.5.

1.1.4 Koch snowflake

The Koch snowflake is fractal curve, one of the earliest described fractals by Helge von Koch in 1904. [12]

There are multiple ways to construct Koch snowflake. One of them is Lindenmayer system, similarly as Twindragon curve.

Let G be a context-free L-system

$$G_2 = (\{F, r, l\}, P, S)$$

Where P is

$$S \rightarrow FlFrrFlF$$

$$F \rightarrow FlFrrFlF$$

F is instruction to go forward of predefined length, and r and l are instructions to turn right and left by 60° respectively.

So for:

- $n = 1 \rightarrow FlFrrFlF$
- $n = 2 \rightarrow FlFrrFlFlFlFrrFlFrrFlFrrFlFlFlFrrFlF$

An illustration of the Koch snowflake can be found in Figure 3.4.

1.2 Existing solutions

There are already numerous web applications presenting fractal visualizations, for example usefuljs.net/fractals or jsdw.github.io/js-fractal-explorer. However they all have one thing in common, and that rendering in those applications is client-sided. This means one important implication, and that the computing power is only dependent on the user's device. We can interpret this consequence as advantage or disadvantage.

It can be advantage in case of many users, and not so computationally demanding tasks. Since server is only serving static files, application like that can be hosted on the CDN (Content delivery network) without dedicated server. This means cheaper hosting solution with higher availability.

However most user's devices don't have enough computation power to perform computationally demanding tasks. In this case the server has to perform the rendering of the fractal and return the final image. This can be extended not only for fractals, but any computationally demanding task, which has to be performed on the server. This thesis is aiming to create an application in Julia programming environment, that is able to do that.

Design, architecture and technologies

This chapter is dedicated to outlining the application requirements as well as the core application architecture. Later sections aim to determine the tool-set required to achieve the established requirements.

2.1 Requirements

2.1.1 Functional requirements

The Fractals web app must be able to offer multiple categories of fractals to render. It must offer multiple options to configure the desired fractal image. Back-end service must be able to validate user input. It must also forbid creating unachievable for malicious requests. Users must be able to interact with created fractals. These interactions include:

- listing created fractals,
- filtering own fractals,
- sorting fractals based on creation time,
- showing created fractal and its metadata,
- showing status of fractal request in queue,

2.1.2 Non-functional requirements

Fractals Julia package must be usable independently from the web app. It must be downloadable from the Internet.

The web app must be available on the Internet with little downtime. Loading times have to be low as possible, and waiting for fractal render has to be intuitive and apparent to the user.

The source code has to be scalable, to allow adding new fractals in future. Rendering fractals must use dedicated computing power as effectively as possible. That means being able to use single computer or cluster of computers, and distribute queued requests effectively between dedicated threads or cluster nodes.

2.2 Architecture

This section outlines architecture and used design patterns. UML component diagram of the outlined architecture can be seen in Figure 2.1.

2.2.1 Julia package

One of the requirements for `Fractals` source code is that it must be reusable in the future by anyone. Another substantial requirement is that it has to be versioned so that web app implementation will not break after updating `Fractals`' source code. Therefore it is needed to bundle `Fractals` source code in the form of a package. Julia environment offers a robust package manager called `Pkg`. A package manager deals with installed software packages, and allows you to easily install new software, upgrade software to newer versions, or remove software that you previously installed. Package management includes automated dependency resolution, conflict detection thus avoiding installation errors caused by incompatible dependencies, version tracking of installed packages and automatic download of new packages. [13]

2.2.2 Client-server architecture

There are two major kinds of modern architectures: two-tier client/server and three-tier—also called n-tier. It's important to note that each tier has many possible variations. At a high level, these architectures focus on the partitioning rules that can be applied based on business or technical requirements. [14]

Client/server is often a generic umbrella term for any application architecture that divides processing among two or more processes, often on two or more machines.

2.2.2.1 Two tier architecture

A two-tier architecture, also known as the client/server model, is a simple way to model the client/server relationship that exists between a user's application

and a server. The client is generally the presentation layer, and the server is the data storage layer. [14]

2.2.2.2 Three tier architecture

A three-tier architecture is an extension of the two-tier model, which adds a new layer that isolates data processing in a central location and maximizes object reuse. Figure 2.2 shows how this new third layer might fit into an application architecture. [14]

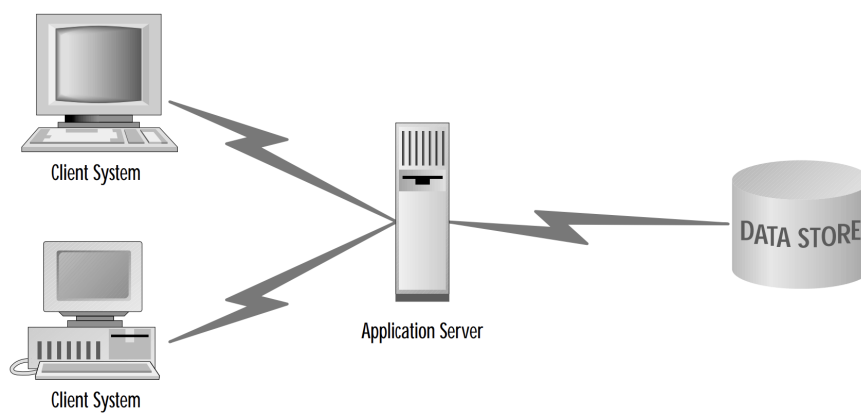


Figure 2.2: Three tier architecture.

This is a suitable solution to our use case. The client is front-end served in a browser for the end-user, the server handles client requests regardless of how a database is implemented, and the database stores client requests and rendered fractals with their respective metadata.

2.2.3 Front-end

Front-end in the Fractals web application will serve as a user interface for creating fractal requests and then interacting with already created fractal images.

Front-end web development is the development of the graphical user interface of a website.

2.2.3.1 Document Object Model

The Document Object Model (DOM) is a programming interface for web documents. It binds the script with HTML document structure so that programs

can change the document style and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page. [15]

2.2.3.2 Single Page Application

A Single-page application (SPA) is a web application that loads a single HTML file with injected Javascript code and then updates the body content via JavaScript APIs such as Fetch when different content is to be shown. [16]

This therefore allows users to use websites without loading whole new pages from the server, which can result in performance gains and a more dynamic experience, more effort is required to maintain state, implement navigation, and do meaningful performance monitoring. [16]

2.2.4 Back-end

Back-end in the Fractals web application has multiple essential roles. Firstly it must communicate with the front-end. Secondly, it must remember, process, and distribute user requests between dedicated working threads or cluster nodes. Lastly, it has to store the results of processed requests in the form of fractal images and be able to serve them to the front-end.

The following subsections dedicate to explaining strategies to satisfy mentioned requirements.

2.2.4.1 API

In order to establish communication between client and server, client needs access to the Application Programming Interface (API). An API is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software. [17]

2.2.4.2 REST API

A REST API (Representational State Transfer) is a software architecture for distributed hyper-medial systems. That means it conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. Entities in REST are treated as *resources*. Resource have type and set of methods which are following HTTP protocol conventions. GET method is used for retrieving data, POST method for creating data, PUT and PATCH for updating data and DELETE for deleting. Each resource is identified by URI, optionally with global ID. Moreover REST should follow HATEOAS (Hypermedia as the Engine of Application State) constraint. It means that hypertext should be used to find your way through the API. [18]

REST API will allow the client to communicate with the server to fetch required data for display.

2.2.4.3 JSON-RPC API

While REST API allows us to work with entities as resources, sometimes we want to run function on back-end without following REST constraints. Before REST became popular, most APIs were built on SOAP or XML-RPC. Web Services induce a massive overhead in terms of file size and parse effort mainly due to the use of XML as serialization format. JSON-RPC (JavaScript Object Notation Remote Procedure Call) comes as a compelling trade-off between REST API and XML based APIs for running remotely code on back-end. [19]

In the following code is the example HTTP request for creating fractal.

```
POST /createMandelbrotSet HTTP/1.1
Content-Type: application/json

{"iterations": 1000}
```

In Fractals web application JSON-RPC will be used complementary with REST for communication between client and server.

2.2.4.4 Active Object

The active object design pattern decouples method execution from method invocation for objects that each reside in their thread of control. The goal is to introduce concurrency, by using asynchronous method invocation, and a scheduler for handling requests. [20]

There are 6 components of an active object:

- A *proxy*, which provides an interface towards clients with publicly accessible methods.
- An interface which defines the *method* request on an active object.
- A list of pending requests from clients (*activation queue*).
- A *scheduler*, which decides which request to execute next.
- The implementation of the active object method, ie. business logic (or *servant*).
- A callback or variable for the client to receive the *future* result.

[20]

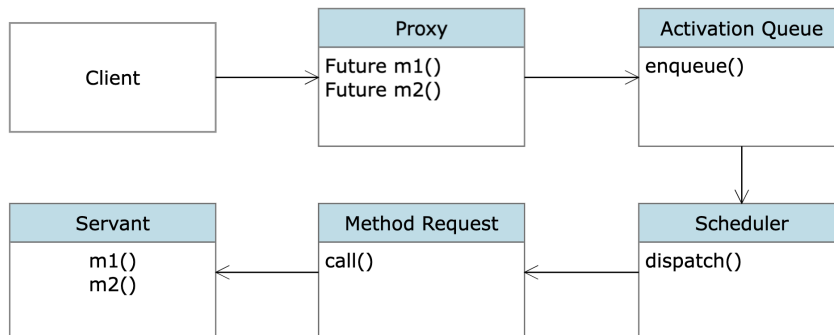


Figure 2.3: Active object diagram.

The Fractal web application will use the Active Object for scheduling and dispatching the queue of fractal requests from the client.

2.2.4.5 Relational database

Created fractals and their requests need to be stored somewhere and be accessible to the server processing the requests as well as the client showing results. Therefore, it must be saved in a database. Traditionally, relational (SQL) databases are used to handle storage and querying. Relational databases rely on a predefined schema, which must be updated to propagate a change in the domain model. Relational databases shine, especially when handling highly relational data, but come with some drawbacks like strict schema or lack of scalability. [21] Since Fractals web-app is not expected to have complex relational data to be stored, using NoSQL (Not Only SQL) database, like an in-memory key-value database, might be more beneficial. However, Julia, as of now, has very little support even for relational databases, I chose to stick with a more straightforward approach to avoid additional overhead and use relational databases. Using any other type of database would require implementing own driver for communication between Julia application and database.

2.3 Technologies

Building the whole application from scratch would not be attainable, therefore using third party libraries and services is essential for developers.

2.3.1 Git

“Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.” [22]

Fractals web app and Julia package use git to version all source code. The repository is stored remotely on the faculty GitLab server (<https://gitlab.fit.cvut.cz/ondejmar/bp-julia-fractals>) with public access to all faculty members. GitLab can also be used in the future for continuous integration and deployment.

2.3.2 Julia

Julia is a high-level, high-performance dynamic programming language for technical computing. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. [23]

Using Julia programming language is the main requirement of this thesis, therefore Julia is used in `Fractals` package and back-end of web application.

2.3.3 Julia Pkg

Pkg is Julia’s builtin package manager, and handles operations such as installing, updating and removing packages. [24]

Web app back-end uses `pkg` to manage dependencies, while Julia package also uses it to bundle it’s code for future use.

2.3.4 Luxor

Luxor is a Julia package for drawing simple static vector graphics. It provides basic drawing functions and utilities for working with shapes, polygons, clipping masks, PNG and SVG images, turtle graphics, and simple animations. Luxor uses `Cairo` for rendering. [25]

Julia `Fractals` package uses it to render vector fractals like Koch snowflake or Twindragon.

2.3.5 React

React is an open-source front-end JavaScript library developed by Meta for building UIs. React advocates a component-based approach, building encapsulated components that manage their state, then composing them to make complex UIs. Since component logic is written in JavaScript instead of templates, it is possible to pass rich data through the app and keep the state out of the DOM. [26] React can be used as a base in the development of SPA, mobile apps, or server-rendered applications using frameworks like Next.js.

For now, Fractals web app is SPA created using Create React App (CRA), a tool to bootstrap single-page React applications officially supported by the React team. By using a high-level front-end framework like React, the client of a web app is easily extensible in the future to a desktop application using Electron or a mobile application using React Native.

2.3.6 Genie

Genie provides a streamlined and efficient framework for developing modern web applications. It builds on Julia's strengths (high-level, high-performance, dynamic, JIT compiled), exposing a rich API and a powerful tool-set for productive web development. [27]

Genie covers most of the back-end infrastructure:

- **REST API:** Genie has a built-in HTTP server providing API to build REST endpoints.
- **Database Connection:** Genie provides tool-set called *SearchLight*, which includes Object-relational mapping (ORM) and connection drivers for few well-known relational databases like PostgreSQL, MySQL or SQLite.

2.3.7 SearchLight ORM

In programming usually data management tasks revolve around objects, as object can precisely represent entity with various properties. However relational databases don't store data as objects, so Object-relational mapping (ORM) is needed. ORM is technique for converting stored data from database into usable objects in programming languages. [28]

SearchLight serves as the ORM layer of Genie tool-set.

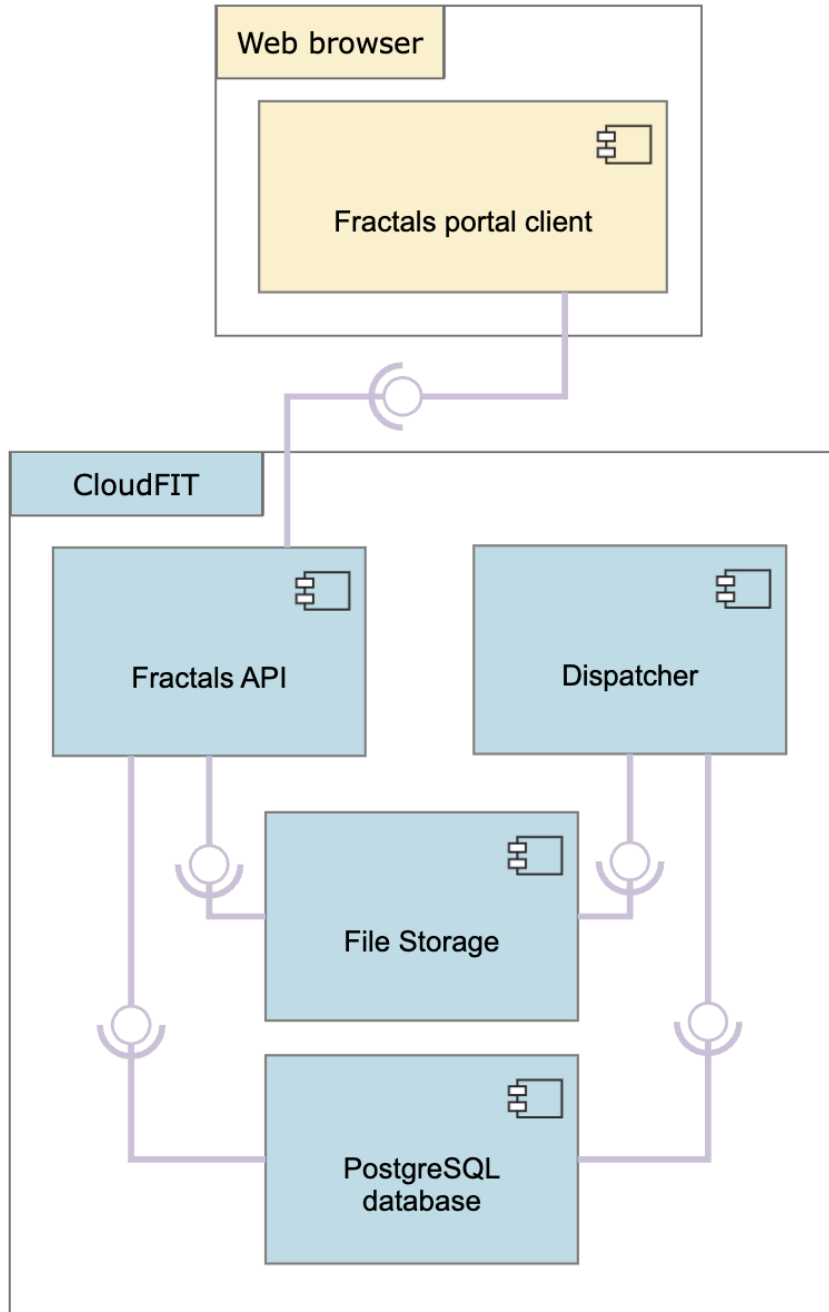


Figure 2.1: UML component diagram of Fractals portal.

Implementation

This chapter describes the implementation of all application parts into details.

3.1 Fractals Julia package

3.1.1 Environment

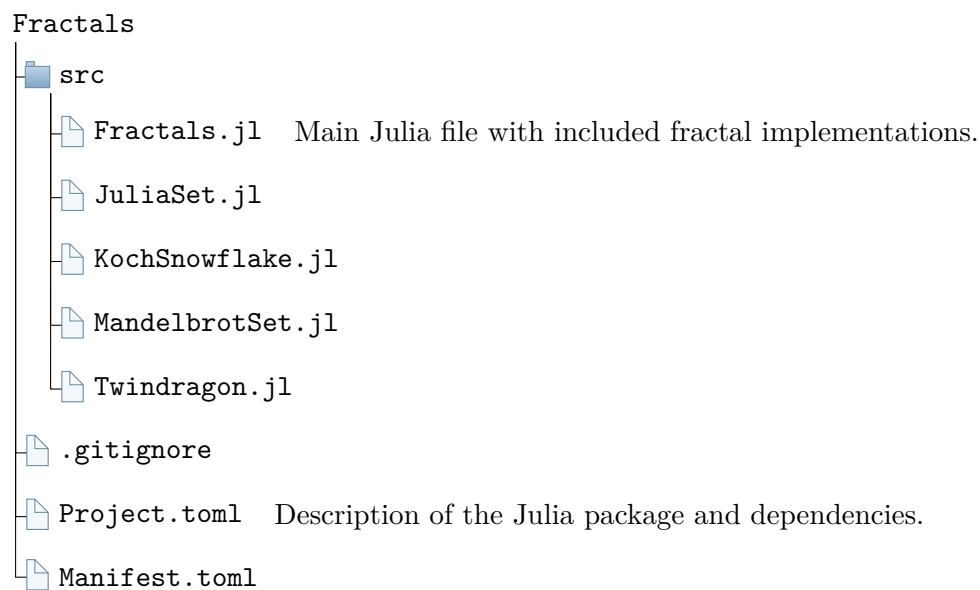


Figure 3.1: **Fractals** directory tree.

The **Fractals** package is the Julia package with fractal implementations, that can be install and reused. The package was generated using command

3. IMPLEMENTATION

generate `Fractals` which generates new folder `src` and files `Project.toml` and `Manifest.toml` (more info about those files in Section 3.3.1).

3.1.2 Julia Set

We can get the approximate result to what is described in Section 1.1.2 by setting some $R > 0$, number of iterations $M \in \mathbb{N}$ and for z from some subset of \mathbb{C} , we measure how many iterations of $f(z) = z^2 + c$ we have to do until we pass the R . This approach is described by Ing. Tomáš Kalvoda, Ph.D. in [9].

```
function julia_set(c, R, M, reals, imags)
    f(z) = z^2 + c

    for j = axes(reals, 1), k = axes(imags, 1)
        z = complex(reals[j], imags[k])

        for n = 1:M
            z = f(z)
            if abs(z) > R
                data[j, k] = n
                break
            end
        end
    end

    data = map(
        x -> 1.0 - min(x / M, 1.0) |> Gray,
        transpose(data)
    )
end
```

Listing 3.1: Source code of Julia set.

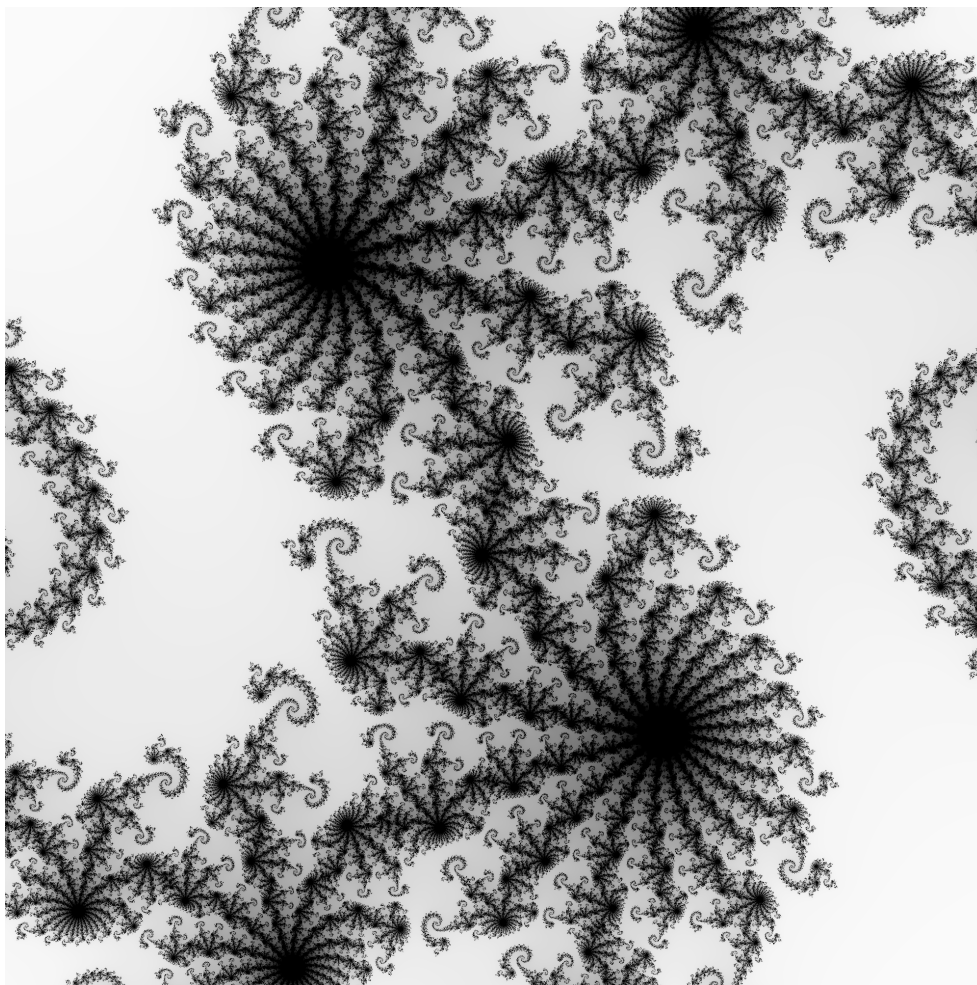


Figure 3.2: Julia set fractal with $c = -0.8 + 0.156i$ for subset $K = \{z \in \mathbb{C} | z = a + bi, a \in [-0.5, 0.5], b \in [-0.5, 0.5]\}$.

3.1.3 Mandelbrot Set

We can get an approximate result of the Mandelbrot set by following similar approach as in Julia set. We set some $R > 0$, number of iterations $M \in \mathbb{N}$ and for c from some subset of \mathbb{C} , we measure how many iterations of $f(z) = z^2 + c$, beginning with $z = 0$ we have to do until we pass the \mathbb{R} .

3. IMPLEMENTATION

```
function mandelbrot_set(R, M, reals, imags)
    f(z, c) = z^2 + c

    for j = axes(reals, 1), k = axes(imags, 1)
        c = complex(reals[j], imags[k])
        z = 0

        for n = 1:M
            z = f(z, c)
            if abs(z) > R
                data[j, k] = n
                break
            end
        end
    end

    data = map(
        x -> 1.0 - min(x / M, 1.0) |> Gray,
        transpose(data)
    )
end
```

Listing 3.2: Source code of Mandelbrot set.

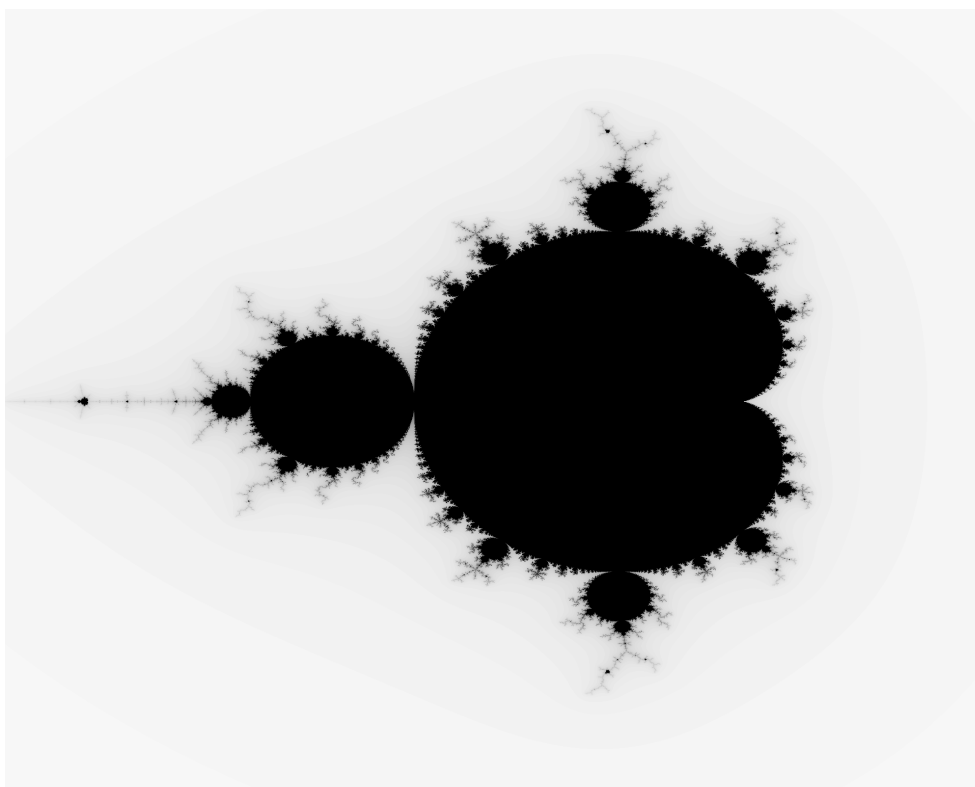


Figure 3.3: Mandelbrot set fractal for subset $K = \{z \in \mathbb{C} \mid z = a + bi, a \in [-2, 1], b \in [-1.5, 1.5]\}$.

3.1.4 Koch Snowflake

To draw curves, package Luxor provides turtle graphics module. Turtle graphics are vector graphics using turtle as a relative cursor.

Using the grammar defined in Section 1.1.4, we can use function Turn from Luxor to perform r and l instructions and create recursive function forward_recursive for instruction F .

```
function forward_recursive(t::Turtle, n::Integer, size::Number)
    if n <= 0
        Forward(t, size)
        return
    end

    size /= 3

    forward_recursive(t, n - 1, size)
    Turn(t, -60)
    forward_recursive(t, n - 1, size)
    Turn(t, 60)
    Turn(t, 60)
    forward_recursive(t, n - 1, size)
    Turn(t, -60)
    forward_recursive(t, n - 1, size)
end

function koch_curve(n::Integer, size::Integer, path::String)
    t = Turtle()

    @png begin
        Reposition(t, Point(-size / 2, size / 10))
        forward_recursive(t, n, size)
    end size size path
end
```

Listing 3.3: Koch snowflake source code

To draw snowflake instead of single Koch curve, we can repeat the process 3 times.

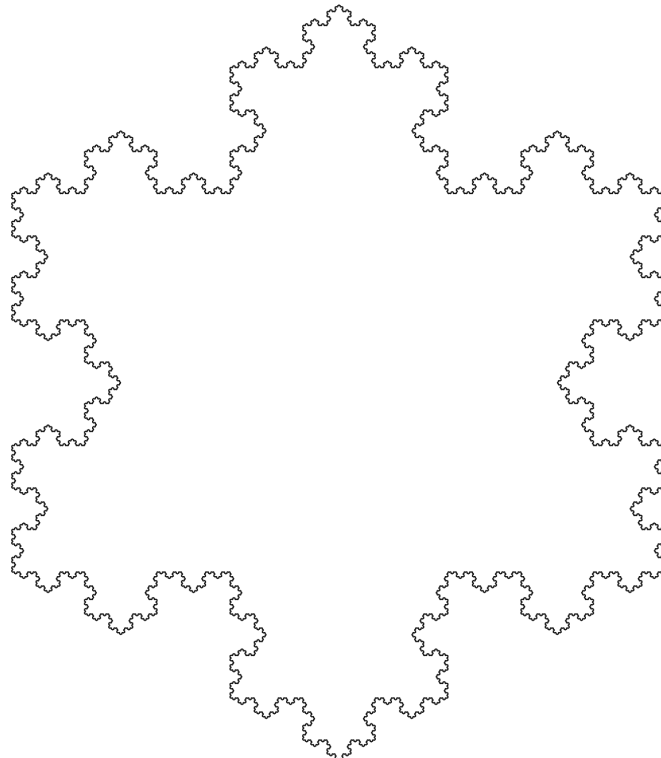


Figure 3.4: Koch snowflake for $n = 10$.

3.1.5 Twindragon curve

Grammar defined in Section 1.1.3 can be performed in similar manner as Koch snowflake.

To perform instructions r and l we can use function `Turn` and to perform instructions A and B we can define two recursive functions `forwardA` and `forwardB`.

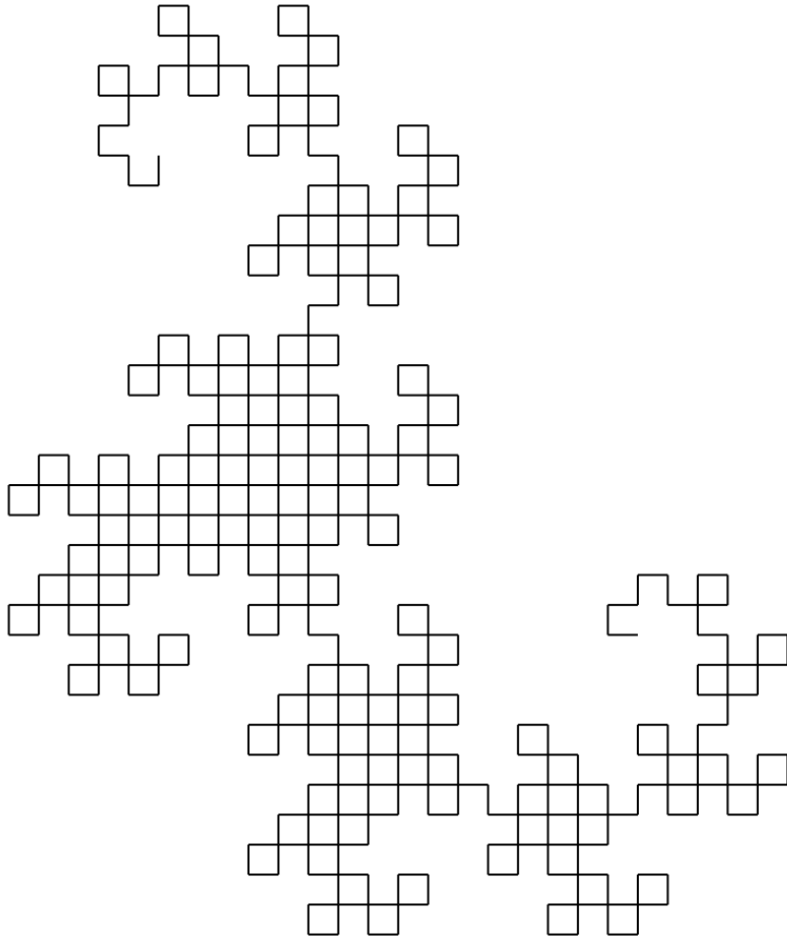


Figure 3.5: Twindragon curve for $n = 9$.

3. IMPLEMENTATION

```
function forwardA(t::Turtle, n::Integer, length::Number)
    if n <= 0
        Forward(t, length)
        return
    end

    forwardA(t, n - 1, length / sqrt(2))
    Turn(t, 90)
    forwardB(t, n - 1, length / sqrt(2))
end

function forwardB(t::Turtle, n::Integer, length::Number)
    if n <= 0
        Forward(t, length)
        return
    end

    forwardA(t, n - 1, length / sqrt(2))
    Turn(t, -90)
    forwardB(t, n - 1, length / sqrt(2))
end

function twindragon(n::Integer, size::Integer, path::String)
    t = Turtle()

    Reposition(t, Point(0, size / 6))
    Turn(t, -90 * (floor((n + 3) / 2)) % 4)

    @png begin
        forwardA(t, n, size / 3)
    end size size path
end
```

Listing 3.4: Source code of Twindragon curve.

3.2 Front-end

In this section I will describe the main front-end components of the Fractals portal client.

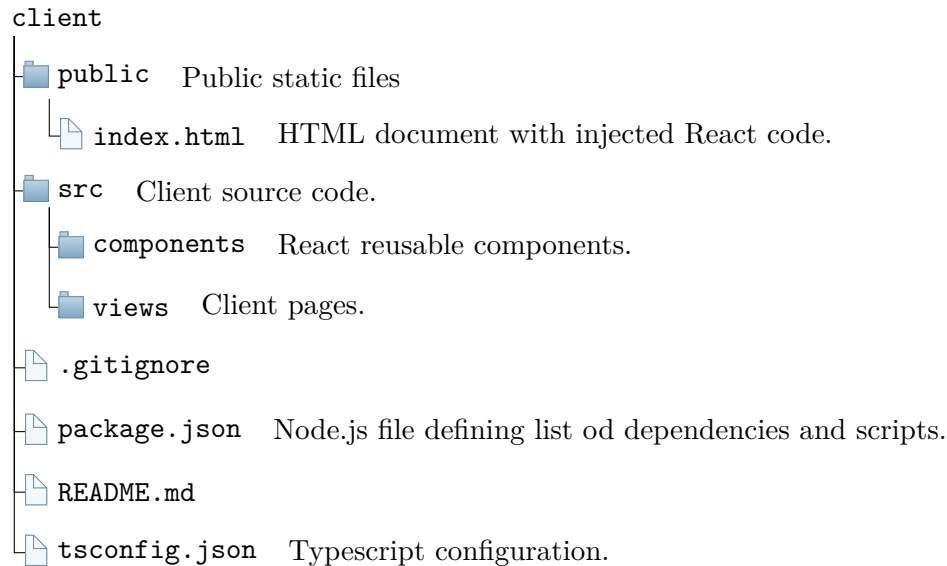


Figure 3.6: **client** directory tree.

3.2.1 Environment

A new React application can be instantiated in multiple ways. Nowadays React is pretty much only coding paradigm and meta-frameworks are used to handle compilation, bundling, bootstrapping etc. Fractals client is created by tool called `Create-React-App`², which creates new `Node.js` boilerplate environment.

Fractals client uses also a component library called `Mantine`³, which provides React components for broad variety of UI elements like buttons, input boxes, layouts, etc. and React hooks for common use-cases.

3.2.2 Gallery

The gallery is used as a landing page for Fractals portal. It lists all created fractals and fractal requests in the database.

On the left side is navigation with all the pages including Gallery and link to every possible Fractal request form.

²<https://create-react-app.dev/>

³<https://mantine.dev/>

3. IMPLEMENTATION

On the right side is the panel with filtering options. User is able to filter based on fractal type, request status or name.

Demonstration of the final gallery UI can be seen in the Figure 3.7.

It is also possible to filter only user's fractals. It doesn't work cross browsers, since in the background, the local storage is utilized for this feature. Local storage is storage object of the browser, which stores data across browser sessions. It is similar to session storage, except it has no expiration time. When Fractal client sends the request to create fractal, it saves the `uid` from the response to the local storage, and then filters only those that are present, when filter "Only my fractals" is toggled on.

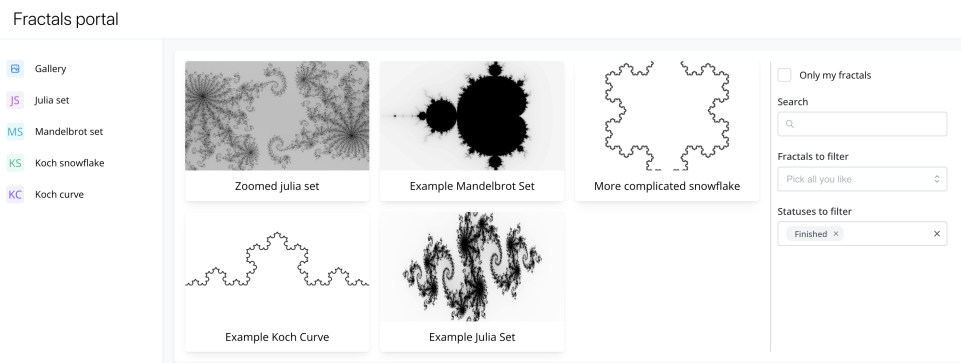


Figure 3.7: Gallery of the Fractals portal.

3.2.3 Fractal form

Every fractal type has its own page for creating request on the route `/create/:fractal_type`. The page consists of form, which on submit sends the request data to the respective API endpoint. Demonstration of the final form UI can be seen in the Figure 3.8. API responds with the `uid` of the created request, which is stored into the local storage for gallery filtering and redirects to the page showing this fractal.

```

const [
  myFractals,
  setMyFractals
] = useLocalStorage<string[]>({
  key: "my-fractals",
  defaultValue: [],
}); // React hook for using local storage

const send = async (values: JuliaSetInput) => {
  const result = await fetch("/julia-set", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(values),
  });

  const { uid } = await result.json();






  // Store uid to the local storage
  setMyFractals([uid, ...myFractals]);

  navigate(`/view/${uid}`); // Redirect to the detail page
};

```

Listing 3.5: Implementation of form handling.

Fractals portal

-  Gallery
-  Julia set
-  Mandelbrot set
-  Koch snowflake
-  Koch curve

Koch snowflake

Title

Iterations

Size

Figure 3.8: Form of the Koch Snowflake fractal.

3.2.4 Fractal detail

Every created fractal and fractal request has its own dedicated page on the route `/view/:uid`. There are 2 ways to get on this page, either from the gallery or by being redirected from the form.

The page layout is divided into 2 tabs:

- **Image:** This tab 3.9 shows the metadata info (title, type, status, date-time of creation) of the fractal request on the left side, and if request is finished, final image on the right side.
- **Input data:** This tab 3.10 shows the input data being sent to the server. Input data are essential to replicate or modify the fractal image.

Demonstration of the final detail UI can be seen in the Figure 3.9 or 3.10.

Fractal request has few possible states and fractal detail has to display every one of them accordingly. The states are:

1. **"WAITING"** and **IN_PROGRESS**: Client shows metadata, and instead of image, loading animation is being shown to indicate user that the request is being worked on. Additionally, client periodically tries to re-fetch status of the request, to automatically show a final image when it's ready.
2. **"FINISHED"**: Client shows metadata and final image like in Figure 3.9.
3. **"ERROR"**: Client shows metadata and instead of a fractal image, the error placeholder image is being shown.

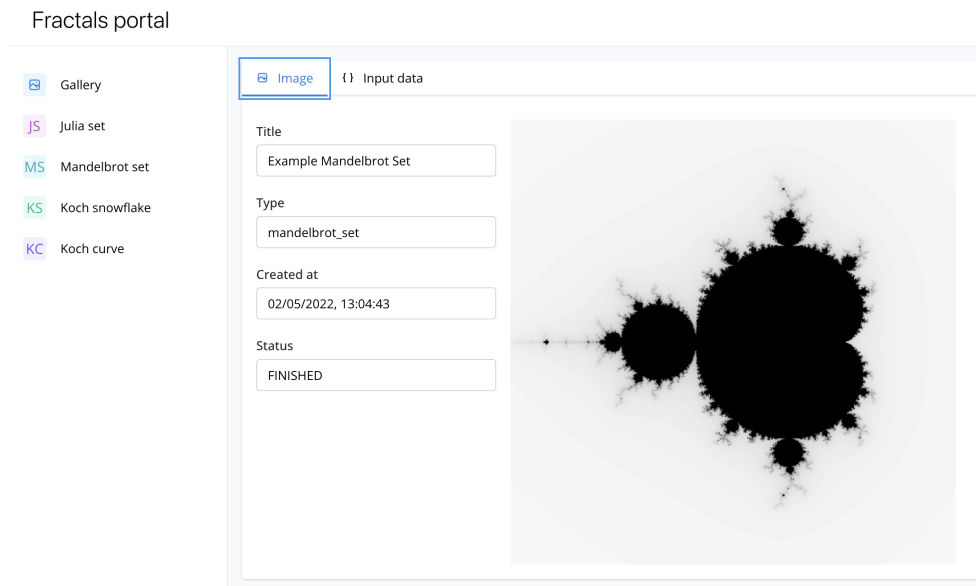





Figure 3.9: Image tab of fractal detail page.

Fractals portal

 Gallery

 Julia set

 Mandelbrot set

 Koch snowflake

 Koch curve

 Image [{ } Input data](#)

```
{
  "imax": 100,
  "max_x": 1,
  "type": "mandelbrot_set",
  "max_y": 1.5,
  "min_x": -2,
  "height": 500,
  "r": 10,
  "min_y": -1.5,
  "width": 500
}
```

Figure 3.10: Input data tab of fractal detail page.

3.3 Back-end

In this section I will further explain back-end infrastructure of Fractals web application. Folders structure follow Genie framework conventions.

All of the back-end code lives in the `server` subdirectory in the git repository. ⁴

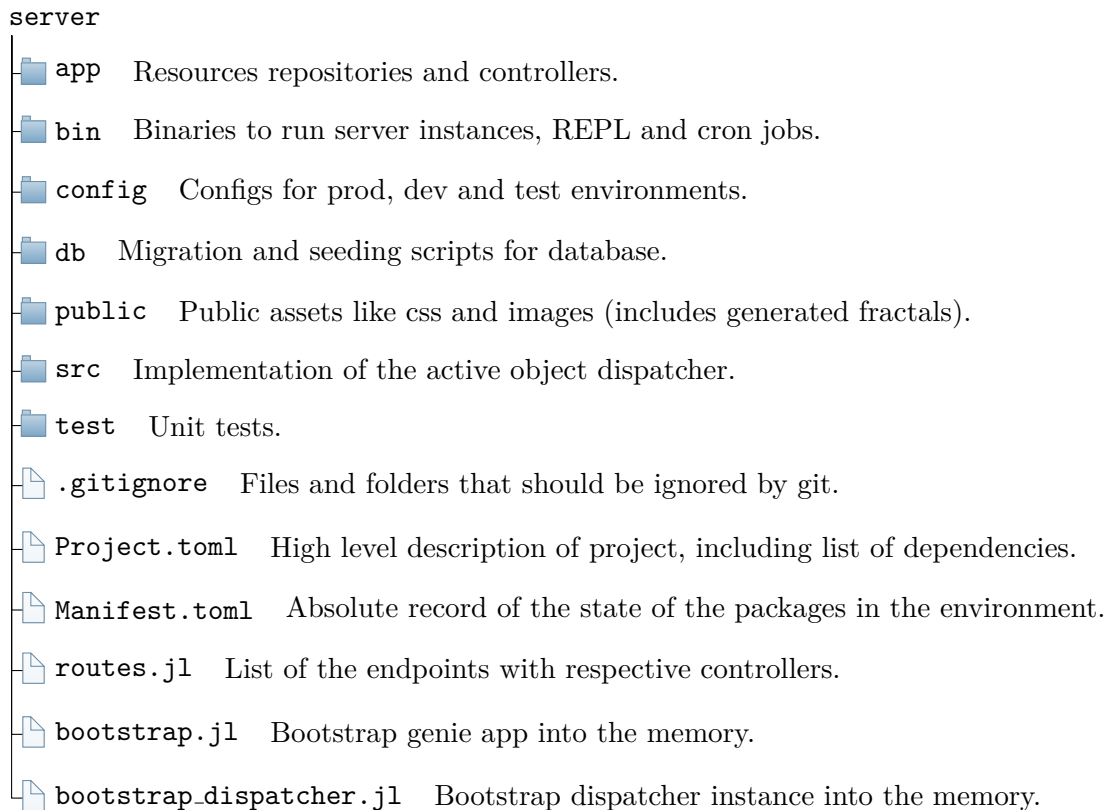


Figure 3.11: `server` directory tree.

3.3.1 Environment

Back-end has two server instances with different roles (public **REST API** and implementation of active object, **dispatcher**), which shares common environment. This environment is described in two files, `Project.toml` and `Manifest.toml`, which are then used by `Pkg` to instantiate the environment.

`Project.toml` describes the project on a high level, with information like project's name, author, version etc. Most importantly it includes the list of

⁴<https://gitlab.fit.cvut.cz/ondejmar/bp-julia-fractals/tree/master/server>

3. IMPLEMENTATION

library dependencies and compatibility constraints with libraries version and version of the Julia itself.

```
name = "Server"
uuid = "b278cb4a-0c3d-4ab5-88cd-cc79f0708b4f"
authors = ["Martin Ondejka <ondejmar@fit.cvut.cz>"]
version = "0.1.0"
```

```
[deps]
Genie = "c43c736e-a2d1-11e8-161f-af95117fbd1e"
Fractals = "c3bcf3c1-58f0-4432-ada4-aa8da7106ebb"
...
```

`Manifest.toml` is auto-generated file by `Pkg` on instantiating of the environment. It is an absolute record of the state of the packages in the environment, which includes exact information about dependencies in the project like version, uuid, dependencies upon which was this package instantiated etc. Usually this file is ignored by git, but in this particular case it is needed to be committed as it includes information about how our `Fractals` package was installed from relative path.

```
...
[[deps.Fractals]]
deps = ["Luxor"]
path = "../Fractals"
uuid = "c3bcf3c1-58f0-4432-ada4-aa8da7106ebb"
version = "0.1.0"
...
```

3.3.1.1 Installation

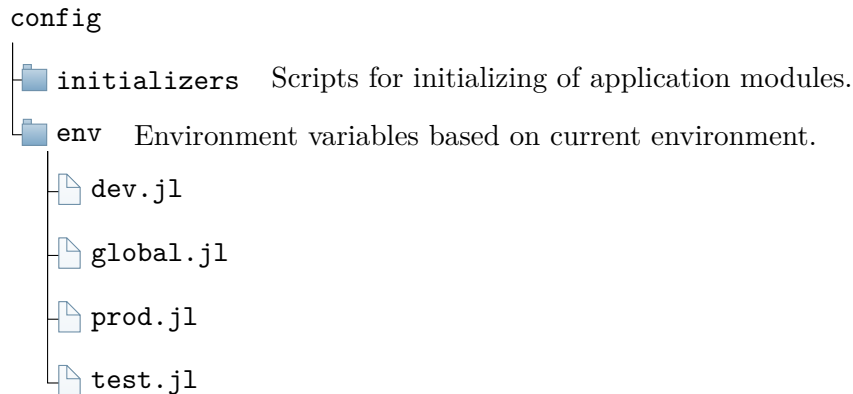
The back-end environment can be installed by running `make install` in the root directory of the project. The command runs simple Julia script which activates the local environment and instantiates packages.

```
using Pkg
Pkg.activate(".")
Pkg.instantiate()
```

3.3.1.2 Configuration

Folder `config` contains configuration files required to initialize environment for running the application. `config/env/global.jl` file contains all the common variables, while `dev.jl`, `prod.jl` and `test.jl` contain variables that are different in every environment.

Differentiating of the development environments is needed due to variables like `WORKER_NPROCS` which defines number of worker processes for dispatcher or `SERVER_URL`, which will be different in production than in local development environment.

Figure 3.12: **config** directory tree

3.3.2 Database

In Fractals back-end, PostgreSQL relational database is used. Genie ecosystem offers ORM SearchLight⁵, which will cover all of the SQL queries without having to write SQL.

In Genie entities are established as *resources*. To create new resource we can run in julia REPL command:

```
julia> Genie.newresource("Fractal")
```

This will create new `FractalRepository.jl` module, `FractalController.jl` module, and database migration to create `fractal` table.

3.3.2.1 Repository

In Genie Repository contains struct `Fractal`. ORM will then map the entity, and all of the database operations on this `Fractal` struct. This will ensure separation of database layer from business logic, as all of the database operations will be run from defined functions in repository. In Fractals web-app we have one repository for `Fractal` entity, which represents created fractal requests with the status `"WAITING"` and all of the created fractal images with the status `"FINISHED"`.

`Fractal` entity has 5 attributes. `uid` identifying fractal uniquely, `data` storing input data from the request, `created_at` to order fractals based on creation time, `location` defining location URI of the image and `status` describing the current status of processed request.

`Fractal` table supports following database operations:

- Creating new `Fractal` request.

⁵<https://genieframework.com/docs/searchlight/api/callbacks.html>

3. IMPLEMENTATION

```
@kundef mutable struct Fractal <: AbstractModel
  id::DbId = DbId()
  uid::String = ""
  data::String = ""
  created_at::DateTime = now()
  location::String = ""
  status::String = "WAITING"
end

function get_fractal(uid::String)
  find(Fractal, SQLWhere(:uid, uid)) |> onereducer
end

function waiting_fractals()
  find(Fractal, SQLWhere(:status, "WAITING"))
end

...
```

Listing 3.6: Source code of the Fractal repository.

- Fetching one Fractal request based on uid.
- Fetching all of the Fractal requests for client to list the gallery.
- Fetching all of the waiting Fractal requests for dispatcher.

3.3.2.2 Activation Queue

Database table `fractal_request` serves also the role of an Activation Queue in the Active Object architecture. By querying all of the Fractal requests with the status "WAITING", we get the queue ready to be processed.

3.3.2.3 Migrations

All of the database schema alterations in SearchLight are resolved through migration scripts. Relational databases require strict schema, which disallow changing requirements during development. Migrations solve this issue by having the queue of scripts with strict order to be applied on database once, which results in correct schema.

Migration script requires two functions:

- `up()`: to implement desired alteration.
- `down()`: to rollback implemented alteration from `up()`.

In SearchLight migrations can be created by creating new resource like in section 3.3.2. In Fractals web-app, migration is firstly used to create fractal table, and then during development to add required new fields.

3.3.2.4 Configuration

Connection configuration file is defined by ORM driver package for PostgreSQL, and it's located in `db/connection.yml`. File is divided into three configurations `dev`, `prod`, `test`.

Additionally for local development configuration of `docker-compose.yml` file is provided to easily install database instance on local machine without having to install PostgreSQL manually.

```
version: "3.9"

services:
  fractals-db:
    image: postgres
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=fractals
    volumes:
      - data:/var/lib/postgresql/data
    restart: always

volumes:
  data:
```

Listing 3.7: Docker compose configuration.

By running `make db-run` from root of the repository, docker will automatically download latest postgres docker image and create container with volume storage.

3.3.3 Web API

The Fractals web-app requirements of an API are:

1. Request generation of fractal image from the client.
2. Fetch data about in-progress and finished requests from the client.

Genie offers robust MVC framework for HTTP protocol communication. MVC stands for Model View Controller, which in our case we will use only Model (in the repository module) and Controller parts, as View is implemented separately in the client.

Respectively to the requirements an API is also divided into two components.

3.3.3.1 JSON-RPC

Remote Procedure Call API is used to enqueue fractal request into the active object scheduler called *Dispatcher*. Therefore this component of an API serves the role of *proxy* in the active object architecture 2.2.4.4.

Every fractal type has one endpoint procedure to enqueue new request with input data defined in the body of the POST payload.

- `/koch-snowflake`: payload of this fractal requires only n to define number of nested iterations, and $size$ to define size of final image in pixels.
- `/koch-curve`: koch-curve has exactly same input parameters as koch-snowflake.
- `/twindragon`: since Twindragon curve is similar fractal to Koch snowflake, the input is same.
- `/mandelbrot-set`: mandelbrot-set endpoint has all the input parameters required for computation of the image which include r, M , range of x axis, range of y axis and size.
- `/julia-set`: julia-set endpoint has the same input parameters as mandelbrot-set, except it adds the parameter c .

3.3.3.2 REST API

REST API is used to fetch resources to the client in form of created fractal requests or finished fractal images.

REST API contains two endpoints defined in `routes.jl` file:

- `/fractals`: fetch all created, waiting and in-progress fractals.
- `/fractal/:uid`: fetch single fractal resource based on unique ID.

3.3.4 Dispatcher

Dispatcher is the computing module of the Fractals web-app back-end. It shelters processing of Fractal requests efficiently based on the computing power provided, and stores the final image in the data storage.

3.3.4.1 Active object roles

Dispatcher is the main pillar of the Active Object architecture. It serves three most important roles:

- **Scheduler**: Separate process, which repeatedly checks the Activation Queue 3.3.2.2 for waiting requests.

- If there are any waiting requests, dispatcher delegates processing of the request to the other process.
- If there are no waiting requests, dispatcher sleeps for 2 seconds.

```
while true
  while count_waiting_fractals() === 0
    sleep(THREAD_SLEEP)
  end

  for request in waiting_fractals()
    @spawnat :any dispatch(request)
  end
end
```

Listing 3.8: Dispatcher scheduler implementation.

- **Servant:** Servants are all the processes delegated to the dispatcher. Setting environment variable `WORKER_NPROCS` to the number of processes, `Scheduler` process initializes them and uses for dispatching of requests.
- **Method:** Single dispatch method is used to have only one method for enqueueing all the requests, but ensure that every request will be processed correctly according to the request type.

3.3.4.2 Single dispatch method

`dispatch` function is defined in file `Dispatcher.jl`, which is then called by `scheduler`. Purpose of this function is to delegate Fractal request to the appropriate method from the `Fractals` package in this order.

1. Sets the status of the request to the `IN_PROGRESS`.
2. Parses the json request data.
3. Creates the `location` of the final image.
4. Checks the `"type"` field in request data and based on it parses rest of the request data and calls the method from `Fractals` package together with the location.
5. On success, saves the request with the status of `FINISHED`.
6. On error, logs the error and saves the request with the status of `ERROR`.

3. IMPLEMENTATION

```
function dispatch(request)
  uid = request.uid
  data = JSONParser.parse(request.data)

  try
    location = "img/fractals/${uid}.png"

    if data.type === "koch_curve"
      (; n, size) = data

      @info "Processing koch curve request $uid"
      koch_curve(n, size, "public/$location")
    end

    ...

    request.status = "FINISHED"
    request.location = location
    save(request)
  catch e
    request.status = "ERROR"
    save(request)
    @error "Unexpected error:" e
  end
end
```

Listing 3.9: Dispatcher servant implementation.

3.3.4.3 Distributed computing

Dispatcher allows for running the servants in separate processes or even on a remote machine, therefore scheduler serves as a proxy.

To run Dispatcher with multiple processes, it is needed to set `WORKER_NPROCS` environment variable to the number of processes.

To run Dispatcher servant remotely, it is needed to set two environment variables:

- `WORKER_SSH_TUNNEL=true`
- `WORKER_SSH_CONFIG` to the flags that will be passed to the `ssh` command. Example: `WORKER_SSH_CONFIG=user@host:port`

Dispatcher can be run by entering the command `make dispatcher-run`, this will run the script called `bootstrap_dispatcher.jl`.

Every spawned process has to be initialized and to load every library used to the memory same as in the parent process. Additionally database has to be used, so Genie initializes SearchLight database driver by loading all of the modules. Libraries can be loaded into the memory of spawned process by running `@everywhere` macro.


```

@info "Loading configuration"
using Genie
Genie.load_configurations(context=@__MODULE__)
@everywhere using Genie
@everywhere using SearchLight
@everywhere Genie.load(; context=@__MODULE__)

@everywhere using Fractals
@everywhere using FractalRepository

```

3.4 Deployment

In Fractals portal, there are 3 main components of the web application that need to be deployed in order to function properly:

1. Client
2. HTTP server with API
3. Dispatcher

Client can be built into the single Javascript bundle file and be served from the HTTP server. By running `yarn build` from the client directory, `client/build` folder will be created with full bundle of static files ready to be served. Client static files, together with fractal images can be served by any HTTP server like Nginx ⁶ or Apache ⁷. Currently Fractals portal is serving static files by built-in Genie HTTP server, which is not ideal from the performance perspective. Usually dedicated HTTP servers offer caching and other features to serve static files more efficiently.

Fractals API and Dispatcher are both Julia applications, so they can be deployed in the same way. In order to uniform Julia environments and provide easy way to deploy application on any unix machine, the Fractals portal can be containerized.

3.4.1 Dockerfile

`Dockerfile` is used to build docker image of Fractals portal. Docker images can be then run on server, which creates docker container — runtime of the Fractals portal.

```

FROM --platform=linux/amd64 julia:latest

# set up the app
RUN mkdir -p app/server
RUN mkdir -p app/Fractals

```

⁶<https://www.nginx.com/>

⁷<https://httpd.apache.org/>

3. IMPLEMENTATION

```
COPY ./server app/server
COPY ./Fractals app/Fractals

WORKDIR app/server

...

# instantiate Julia packages
RUN julia -e " \
using Pkg; \
Pkg.activate(\".\"); \
Pkg.instantiate(); \
Pkg.precompile(); "

EXPOSE 80

# set up app environment
ENV GENIE_ENV "prod"

CMD ["/run.sh"]
```

In order to build the image `docker build -t fractals-image .` can be run. Then to run the container:

```
docker run -e PORT=80 -e SERVER_URL=$domain_of_the_server \
-p 80:80 fractals-image
```

Conclusion

This thesis describes details on how the goals were achieved to implement web application to visualize fractals in programming language Julia.

The literature overview about the chosen set of fractals has been provided and well defined.

Requirements for an application were outlined the architecture and required tools have been chosen and introduced.

Implementation details of the Fractals portal have been detailed. The code was implemented and explained in the thesis.

The application has been deployed on CloudFIT environment and is running flawlessly.

To conclude this thesis, goals from the beginning, as well as the outlined requirements, were fulfilled.

Viable future improvements

Scope of this thesis didn't include some of the features that would be very valuable.

Currently there is no other option of removing the fractals, other than manually connecting to the database. Valuable addition to the application would be an admin interface behind the authentication layer.

To mitigate the impact of long lasting computations, method to segregate requests based on computation duration should be implemented. There should be always at least one dispatcher worker available for the short lasting tasks.

Bibliography

- [1] Bezanson, J.; Edelman, A.; et al. Julia: A fresh approach to numerical computing. *SIAM review*, volume 59, no. 1, 2017: pp. 65–98.
- [2] Theiler, J. Estimating fractal dimension. *JOSA A*, volume 7, no. 6, 1990: pp. 1055–1073.
- [3] Mandelbrot, B. B.; Evertsz, C. J.; et al. *Fractals and chaos: the Mandelbrot set and beyond*, volume 3. Springer, 2004.
- [4] Falconer, K. *Fractal geometry: mathematical foundations and applications*. John Wiley & Sons, 2004.
- [5] Barnsley, M. F. *Fractals everywhere*. Academic press, 2014.
- [6] Brooks, R.; Matelski, J. P. The dynamics of 2-generator subgroups of $PSL(2, C)$. In *Riemann surfaces and related topics: Proceedings of the 1978 Stony Brook Conference, Ann. of Math. Stud*, volume 97, 1980, pp. 65–71.
- [7] Avalos-Bock, S. Fractal Geometry: The Mandelbrot and Julia Sets. May 2022. Available from: <https://math.uchicago.edu/~may/VIGRE/VIGRE2009/REUPapers/Avalos-Bock.pdf>
- [8] Julia set. April 2022. Available from: <http://www.britannica.com/science/Julia-set>
- [9] Kalvoda, T. 07: Vizualizace a manipulace s grafikou. April 2022. Available from: <https://courses.fit.cvut.cz/BI-JUL/tutorials/bi-jul-07.html>
- [10] Prusinkiewicz, P.; Hanan, J. *Lindenmayer systems, fractals, and plants*, volume 79. Springer Science & Business Media, 2013.

BIBLIOGRAPHY

- [11] Großkopf, P. *Intersecting the twin dragon with rational lines*. Dissertation thesis, Wien, 2020.
- [12] Koch snowflake. April 2022. Available from: <https://mathworld.wolfram.com/KochSnowflake.html>
- [13] Aptitude user's manual: What is a package manager? March 2022. Available from: <https://www.debian.org/doc/manuals/aptitude/pr01s02.en.html>
- [14] Reese, G. *Database Programming with JDBC and JAVA*. "O'Reilly Media, Inc.", 2000.
- [15] Introduction to the DOM. March 2022. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
- [16] SPA (Single-page application). March 2022. Available from: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>
- [17] Reddy, M. *API Design for C++*. Elsevier, 2011.
- [18] What is a REST API? March 2022. Available from: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- [19] Samsel, C.; Gökay, S.; et al. Web Service to JSON-RPC Transformation. In *ICSOFT*, 2013, pp. 214–219.
- [20] Špaček, P. Lecture 7: Concurrency patterns. March 2022. Available from: <https://courses.fit.cvut.cz/NI-ADP/lectures/index.html>
- [21] Venkatraman, S.; Fahd, K.; et al. SQL versus NoSQL movement with big data analytics. *International Journal of Information Technology and Computer Science*, volume 8, no. 12, 2016: pp. 59–66.
- [22] Git. March 2022. Available from: <https://git-scm.com>
- [23] Wikipedia: Julia (programming language). March 2022. Available from: [https://en.wikipedia.org/wiki/Julia_\(programming_language\)](https://en.wikipedia.org/wiki/Julia_(programming_language))
- [24] The Julia Language: Pkg. March 2022. Available from: <https://docs.julialang.org/en/v1/stdlib/Pkg>
- [25] Introduction to Luxor. March 2022. Available from: <http://juliagraphics.github.io/Luxor.jl/stable>
- [26] React github. March 2022. Available from: <https://github.com/facebook/react>

- [27] Genie framework. March 2022. Available from: <https://github.com/GenieFramework/Genie.jl>
- [28] O’Neil, E. J. Object/relational mapping 2008: hibernate and the entity data model (edm). In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1351–1356.

Acronyms

API Application Programming Interface

CDN Content Delivery Network

CRA Create React App

DOM Document Object Model

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

JIT Just in Time

JSON Javascript Object Notation

MVC Model View Controller

NoSQL Not Only SQL

ORM Object-relational Mapping

PNG Portable Network Graphics

REST Representational State Transfer

RPC Remote Procedure Call

SOAP Simple Object Access Protocol

SPA Single Page Application

SVG Scalable Vector Graphics

SQL Structured Query Language

UI User Interface

A. ACRONYMS

UID Unique Identifier

URI Universal Resource Identifier

XML Extensible Markup Language

Contents of enclosed media

```
readme.txt ... the file with the instructions to run Fractals portal locally
src ..... the directory of source codes
├── server ..... the source code of the Fractals portal server
├── client ..... the source code of the Fractals portal react client
├── Fractals ..... the source code of the Fractals package
thesis .....
├── thesis.tex ..... the thesis source code
└── thesis.pdf ..... the thesis text in PDF format
```