MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Web-Based System for Developer Portfolio Management

Bachelor's Thesis

BARBORA PIATKOVÁ

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Web-Based System for Developer Portfolio Management

Bachelor's Thesis

BARBORA PIATKOVÁ

Advisor: RNDr. Martin Macák, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2025

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed in completed reference to the due source.

Throughout this thesis's preparation, I used various AI tools: Cursor to assist with coding, Grammarly for grammar checks, and ChatGPT to enhance my writing style. I confirm that these tools were used in accordance with the principles of academic integrity. I have thoroughly reviewed the content and take full responsibility for its accuracy and quality.

Barbora Piatková

Advisor: RNDr. Martin Macák, Ph.D.

Acknowledgements

I would like to express my gratitude to my advisor, RNDr. Martin Macák, for his guidance and support throughout the process of writing this thesis. I would also like to thank my consultant, Bc. Daniel Rozehnal, for his assistance and insightful feedback. Lastly, I am deeply grateful to my family for their unwavering support and encouragement during this journey.

Abstract

This bachelor thesis presents the design, implementation, and evaluation of a web-based system for creating developer portfolios that address key limitations of current tools, particularly in personalization and collaboration support. The thesis provides a theoretical context by reviewing relevant cloud platforms for software deployment and then defines the developed system's detailed requirements and data model. The implemented solution features functionalities such as an intuitive visual editor, a model for team collaboration through contributors, and direct integration with the Vercel platform to simplify project deployment. The thesis further presents the results of user testing of the implemented solution that confirmed the functionality, user satisfaction, and overall system effectiveness in portfolio creation.

Keywords

portfolio, developer, Vercel, NextJS, web application

Contents

Introduction **2**

1 Developer portfolios **3**
1.1 Existing solutions 3
1.1.1 Portfolio 3
1.1.2 Devswall 4
1.1.3 Simplefolio 4
1.1.4 vCard 5
1.2 Limitations of existing solutions 5

2 Cloud Deployment Models **7**
2.1 Platform as a Service - The Relevant Model 7
2.2 PaaS Platform Comparison 9
2.2.1 Github Pages 9
2.2.2 Netlify 10
2.2.3 Vercel 10
2.2.4 Heroku 11
2.2.5 Render 11
2.2.6 Railway 12
2.2.7 Fly.io 12
2.2.8 Summary of Platforms 13

3 Application analysis **15**
3.1 System requirements 15
3.1.1 Functional requirements 15
3.1.2 Non-Functional requirements 16
3.2 Data model 17

4 Technology Stack **19**
4.1 Next.js 19
4.2 TypeScript 19
4.3 Prisma 20
4.4 NextAuth.js 21
4.5 Supabase 21
4.6 Tailwind CSS 22
4.7 Shadcn 22

5 Portfolio System Features **24**
5.1 Content elements 24
5.1.1 Layout structures 24
5.1.2 Projects sections 25
5.1.3 Header 25
5.1.4 Timeline 25
5.1.5 Rich-text editor 25
5.1.6 Custom Code 26
5.1.7 Contact form 26
5.2 Portfolio contributors 27
5.3 Integration with Vercel 29
5.4 Portfolio Revisions 30

6 Deployment and System Evaluation **32**
6.1 Deployment 32
6.2 Testing 32
6.2.1 Testing Methodology and Participants 32
6.2.2 Feedback Collection 33
6.2.3 Results 33

7 Conclusion **35**

Bibliography **36**

A Attachments **40**

List of Tables

1.1 Comparison of Portfolio Platform Features 6
2.1 Analysis of Lowest-Tier Plans for Major Cloud Hosting Platforms 14

List of Figures

2.1 Cloud Computing Abstraction Layers [9] 8
3.1 Entity Relationship Diagram of Portfolio System 18
5.1 Portfolio Contribution Workflow 28
5.2 User interface for Portfolio Revisions 31

Introduction

The current job market in the technology sector is highly competitive, particularly for junior engineers seeking their initial roles [1]. Recent data have highlighted this difficulty. A CompTIA report revealed that only 21% of tech job postings in April 2025 were for candidates with 0-3 years of experience, a category that often still requires some prior experience, not exclusively targeting true entry-level positions [2]. Furthermore, the same report showed a rising technology unemployment rate, around 3.5%, underscoring the tight market [2]. Adding to these challenges is the issue of ghost jobs—postings for roles that companies may not be actively filling—with some analyses suggesting nearly a third of applications could be to such positions, significantly extending job search timelines [3]. These conditions underscore the demanding job market that junior developers face, where success hinges on their ability to stand out and demonstrate skills that go beyond the baseline requirements found in job offerings.

One effective strategy to meet this need for differentiation and to showcase practical abilities is to create a developer portfolio. While resumes and LinkedIn profiles are essential for outlining experience and passing initial screenings, a developer portfolio is crucial for standing out. Its main advantage lies in its capacity to showcase practical skills. Unlike traditional resumes, which are primarily designed to document work history, portfolios demonstrate practical skills through projects. This distinction is particularly important because a focus on documented work experience can disadvantage candidates without extensive job histories, such as recent graduates and self-taught developers [4].

Acknowledging the role of developer portfolios, this bachelor's thesis confronts the limitations of existing tools by designing and implementing a new web-based system for portfolio management. Current solutions do not provide adequate customization flexibility, support for collaborative work, and streamlined project deployment. Therefore, the primary goal of this work is to engineer a platform that specifically overcomes these deficiencies, offering users enhanced control over their portfolio's appearance, enabling seamless contribu-

tions from multiple collaborators, and simplifying the showcasing of technical projects via an integrated deployment feature.

The thesis documents this effort across several chapters. Chapter 1 delves deeper into developer portfolios, discussing common approaches to their creation and analyzing existing tools and platforms. Building on this foundation, Chapter 2 investigates suitable cloud deployment options, focusing on the platforms that can host developer projects. The core system design is then detailed in Chapter 3, which describes the system requirements and data model. Following the design phase, Chapter 4 discusses the technologies employed, and Chapter 5 focuses on key system features. Chapter 6 addresses the application deployment and reports user testing results. Finally, Chapter 7 concludes the thesis by summarizing the findings.

1 Developer portfolios

A developer portfolio plays a key role in demonstrating practical ability, especially in a competitive job market. Given this importance, two main approaches to creating one have sparked debate in developer communities. Advocates for building a portfolio from scratch highlight the benefits of learning opportunities, control over the final product, and the ability to showcase unique design and coding skills. However, their opponents stress the efficiency and convenience of dedicated portfolio builders.

They argue that this efficiency is achieved by allowing developers to focus on what truly matters: creating high-quality, meaningful projects to showcase in their portfolio rather than getting bogged down in website layout and styling. Ultimately, if the goal is to showcase abilities effectively, a tool that saves time on peripheral tasks can be helpful.

1.1 Existing solutions

Building on this discussion, various tools and platforms have been developed to help developers create portfolios. This section provides an overview of some of the most widely used developer portfolio solutions.

1.1.1 Portfolio

Portfolly[1] is a free SaaS[2] platform owned by Amigoscode, a UK-based company specializing in software development education. It simplifies developer portfolio creation by eliminating self-hosting complexities, allowing users to concentrate on presenting their work. A key aspect of Portfolly is its integration with GitHub. Users sign in via this service, and the platform can leverage this connection to convert a user's GitHub profile into a professional portfolio layout within minutes. The portfolio creation process involves inputting information into five predefined sections. As users build their portfolio, the interface provides

1. https://portfolly.io/
2. Software as a Service

a helpful side-by-side display with real-time, responsive previews. Regarding deployment, published portfolios are hosted on a predefined Portfolly domain and can be published with a single click. However, the platform's reliance on GitHub is exclusive. Currently, integration with other Git providers and custom domain use are not supported. In addition, Portfolly offers a single portfolio theme, which restricts visual customization.

1.1.2 Devswall

Devswall[3] is another no-code platform designed to rapidly create portfolios. A primary feature of Devswall is its strong focus on project presentation. The platform allows users to add projects that are then categorized and displayed according to their respective categories. Each project added to Devswall receives its own dedicated subpage, which allows users to provide detailed information about their work. Similar to Portfolly, Devswall simplifies data input by utilizing forms for users to provide the required information and also offers the convenience of signing up through GitHub. However, there are some limitations that should be considered. Unlike Portfolly, Devswall operates strictly on a subscription-based model. A key consideration for users is the lack of a permanent free option, as access requires a paid subscription after only a short three-day free trial. Furthermore, similarly to Portfolly, Devswall provides users with only a single portfolio template, meaning there is limited design flexibility for users who may want different layouts or styles.

1.1.3 Simplefolio

Simplefolio[4] is an open-source portfolio template built using HTML, CSS, and JavaScript and is licensed under the MIT license. This license allows users significant freedom to use, modify, and distribute the code, even for commercial purposes, provided that the original copyright and license notice are included. Unlike SaaS platforms, Simplefolio offers a hands-on approach that requires users to clone the

3. https://www.devswall.com/
4. https://github.com/cobidev/simplefolio

repository and edit the necessary files based on the provided instructions. The template features a clean and minimalist design, providing a responsive platform to showcase projects. Although it requires some web development knowledge for setup and deployment to a web server, Simplefolio offers a customizable and cost-effective solution as well as some freedom for developers.

1.1.4 vCard

Similarly to Simplefolio, vCard[5] offers an open-source template approach to create a developer portfolio. Developed by a Bangladeshi software developer and licensed under the MIT license, vCard stands out for its esthetic portfolio layout. The code is in a single HTML file, which further streamlines content modification. The portfolio-building process involves editing the HTML file directly, with optional CSS customization. Many users find the styling and layout of vCard to be fresh and visually pleasing, as evidenced by the numerous portfolios built using this template. As a static HTML template, much like Simplefolio, making vCard accessible requires deployment to a hosting platform, which will be discussed in the next chapter.

1.2 Limitations of existing solutions

Existing developer portfolio tools have several shortcomings. Devswall is not available for free. It operates on a subscription-based model, which can be a discouraging factor for developers when selecting a portfolio tool. In addition, customizability of all solutions is limited. While Portfolly and Devswall offer a single, non-customizable template, Simplefolio and vCard provide limited personalization options that restrict the ability to create a fully individualized developer portfolio. Moreover, unlike SaaS solutions, Simplefolio and vCard require users to set up and deploy the repository themselves. This adds unwanted complexity to the portfolio creation process. Finally, none of the reviewed platforms supports collaboration on portfolios.

5. https://github.com/codewithsadee/vcard-personal-portfolio

Table 1.1 presents a feature comparison of these tools, specifically evaluating them in terms of the aspects addressed in the preceding discussion, along with other key criteria.

Table 1.1: Comparison of Portfolio Platform Features

Feature	**Portfolly**	**Devswall**	**Simplefolio**	**vCard**
Free solution	✓	✗	✓	✓
Design adaptability	✗	✗	✓	✓
Managed hosting	✓	✓	✗	✗
Collaboration support	✗	✗	✗	✗
Personalized design support	✗	✗	✗	✗
Responsive previews	✓	✗	✗	✗

The preceding discussion and feature comparison reveal the shortcomings of existing portfolio solutions. To address these gaps, this thesis proposes the development of a new application. The core objective of this new system is to offer a better alternative by systematically incorporating all functionalities considered in the previous comparison. Therefore, this newly developed system will overcome the limitations of current tools and provide developers with a more robust, flexible, and holistically complete system to build portfolios.

2 Cloud Deployment Models

For a developer portfolio to serve its purpose, the showcased projects must be accessible to potential employers. This critical step is achieved through deployment. Deployment methodologies have evolved significantly, moving away from traditional server management toward more powerful and adaptable cloud-based solutions.

Historically, application deployment has been a cumbersome process involving manual configuration and ongoing maintenance of dedicated physical servers. This approach presents significant hurdles, especially when scaling resources up or down, and introduces complexities during software updates or installations [5].

Cloud computing has brought a new era to the deployment field, offering greater flexibility and efficiency. The proposed model allows for dynamic resource allocation and infrastructure customization depending on the requirements of the application. In addition, cloud-based systems lower operational costs and eliminate the management of physical infrastructure [6].

2.1 Platform as a Service - The Relevant Model

Given the shift toward cloud computing to release software, a closer look at specific service models is necessary. Cloud computing is generally categorized into service models based on their level of abstraction and the control provided. According to the National Institute of Standards and Technology, the three primary cloud service models are Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) [7]. These models form a layered hierarchy in which each ascending layer abstracts more of the underlying technical infrastructure from the user.

The highest level of abstraction is SaaS. This model delivers complete, ready-to-use software applications directly to end users over the internet. Common examples of this model include email clients and office suites, as well as the aforementioned portfolio builders. SaaS platforms provide significant convenience by managing virtually all infrastructure and application stack aspects. However, this high level

of management means that they are of limited relevance for developers seeking to deploy their own projects.

In contrast, IaaS operates at a lower abstraction level. Providers of this type of service, such as Amazon Web Services and Microsoft Azure, supply core computing resources, namely virtual servers, storage, and networking. The defining feature of this model is user control. Users manage the operating system, middleware, runtimes, development tools, application code, and data. This extensive control provides the flexibility required for complex enterprise applications with custom requirements. However, this also means that users are responsible for all configurations, security, patching, and maintenance of the operating system and virtual infrastructure [8]. Such management overhead is generally excessive and impractical for developer portfolio projects.

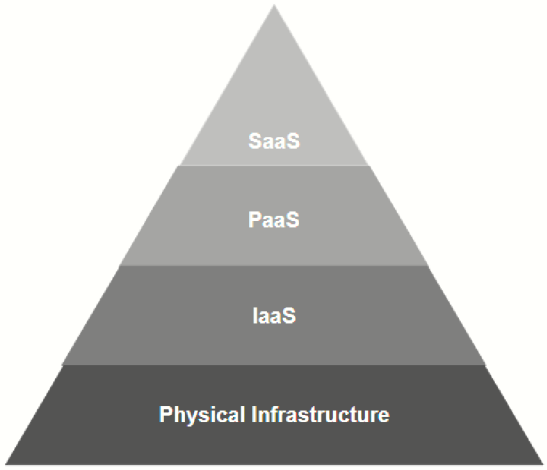

Figure 2.1: Cloud Computing Abstraction Layers [9]

This analysis directly reveals PaaS as the most relevant and practical model for hosting typical developer portfolio projects. It effectively balances the underlying infrastructure with the middleware, development tools, and deployment mechanisms required to run applications. This abstraction allows developers to focus on writing, testing,

and deploying their code without becoming mired in the complexities of infrastructure management. Platforms such as Netlify, Vercel, Heroku, and Render represent this model. They automate crucial aspects such as hosting configuration, scaling, and continuous deployment pipelines. As a result, they serve as ideal solutions for building, deploying, and showcasing web applications and static sites online.

2.2 PaaS Platform Comparison

After establishing PaaS as the most practical deployment model for developer portfolios, this section examines several popular platforms that offer this service type. A common benefit of these platforms is their ability to abstract infrastructure, which allows developers to focus on code and deployment. However, they vary significantly in terms of their features, complexity, cost structures, and suitability for different types of projects. This exploration aims to clarify these differences and guide the selection of the most suitable platform.

2.2.1 Github Pages

GitHub Pages is a simple and free solution to hosting static websites directly from a GitHub repository. It is deeply integrated into the GitHub ecosystem; thus, it is ideal for project documentation, personal sites, and simple projects [10]. Key features include free hosting for public repositories, deployment triggered by a git push, built-in support for the Jekyll static site generator, the ability to use custom domains, and automatic HTTPS provisioning [11, 12]. However, its primary limitation is its restriction to static content, which means that it cannot execute server-side code or connect to databases. While usage limits exist, such as a soft 1 GB site size and 100 GB monthly bandwidth limit, these are generally sufficient for typical portfolio projects [13]. Due to its direct integration and zero cost, GitHub Pages is suitable for developers familiar with GitHub and represents a good choice for purely static projects.

2.2.2 Netlify

Netlify is a platform optimized for building and deploying modern frontend projects, particularly those utilizing the Jamstack architecture. It provides a seamless developer experience focused on static sites and single-page applications, deploying them across a global edge network. Its core features include a robust Git-based CI/CD[1] workflow, deploy previews for reviewing changes, instant rollbacks, integrated serverless functions for adding backend logic, built-in form handling, custom domain support, and automatic HTTPS [14]. The free tier is generous, offering 100 GB of bandwidth and 300 build minutes monthly, but it comes with a critical limitation. Exceeding the free account usage limits on any site will result in the suspension of all sites. To reactivate them, one can either wait for your limits to reset next month or choose to upgrade to a paid plan, which offers higher limits and immediate reactivation [15]. Despite this shortcoming, Netlify's high ease of use, which requires minimal configuration, makes it an excellent choice for developers building static sites or Single-page applications with modern frameworks. This platform supports rapid iterations and preview capabilities, making it well suited for users with these specific needs.

2.2.3 Vercel

Vercel is a premier cloud platform for frontend developers. It excels at deploying high-performance websites and applications, particularly those built using its native Next.js framework. It emphasizes developer experience, focusing on speed and seamless integration with modern frontend frameworks. The core features of the proposed platform include a sophisticated Git-based CI/CD pipeline, automatic preview deployments for easy collaboration, and a global edge network for minimal latency. Beyond these fundamentals, Vercel integrates powerful serverless and edge functions, built-in analytics, automatic image optimization, and managed storage options, offering a range of tools beyond basic deployment. It also provides an AI SDK for AI-powered projects [16]. Although its generous free tier is appealing for personal use, it carries a strict non-commercial restriction, which

1. Continuous Integration and Continuous Deployment

must be considered before use [17]. Despite this limitation, Vercel's streamlined, zero-configuration approach makes it an excellent choice for performance-critical Next.js applications and modern frontend development.

2.2.4 Heroku

Heroku, a long-standing and influential PaaS, simplifies application deployment through its Git-based workflow and language-detecting buildpacks. It supports various programming languages, including Node.js, Ruby, Python, Java, and PHP. Key features include smart containers called Dynos, a comprehensive add-on marketplace for integrating databases and other services, managed databases such as Heroku Postgres and Heroku Data for Redis, easy-to-use scaling options, and CI/CD tools, including pipelines and review apps [18]. Heroku no longer offers an entirely free plan for web applications. The current entry-level plan puts a site to sleep after 30 minutes of inactivity, which introduces latency. The basic plan avoids this limitation and includes free SSL, making it more suitable for portfolio projects that require reliability. Database costs are managed through add-ons and typically require paid plans for practical use [19]. While its classic git push deployment remains relatively easy to use, the lack of a free tier and the sleeping nature of the cheapest option make it less attractive compared to the zero-cost hosting of some alternatives. Heroku is best suited for developers familiar with its ecosystem who want to host dynamic applications and are willing to pay a monthly fee.

2.2.5 Render

Render is a cloud platform that simplifies deployment and scaling via a zero DevOps approach. It is built to handle diverse applications, such as static sites, web services, background workers, and databases. It aims to be an easier-to-use alternative to large cloud providers and a modern successor to Heroku. Render supports multiple service types, deployment via Git with automatic updates, and native runtimes for languages, including Ruby, Node.js, and Python, alongside Docker support. Developers can deploy directly from a Docker registry or use a Dockerfile to define their app's environment. Managed PostgreSQL

and Redis instances, a global CDN[2] for static sites, and preview environments are also available [20]. Render offers free tiers for multiple service types. Static sites are free with 100 GB bandwidth and do not sleep, while free web services spin down after 15 minutes of inactivity, causing cold start delays. Free databases are also available, but they have limitations, such as expiration or data clearing. Free dynamic services share a pool of 750 instance hours monthly [21]. Render's user-friendly interface makes it ideal for static sites and simple full-stack applications. It is particularly suitable for developers seeking free dynamic hosting who can accept inactivity limitations or who are open to upgrading to affordable paid plans for consistent availability.

2.2.6 Railway

Railway is a modern cloud platform focused on an infrastructure-less experience that streamlines the deployment of full-stack applications, including backend services, databases, and frontend code. It abstracts away infrastructure complexity, allowing developers to deploy interconnected services from GitHub or Docker images easily. Standout features include its unified deployment model, automatic private networking between services within a project, integrated templates for databases, such as PostgreSQL and Redis, support for persistent volumes, and automatic language detection via Nixpacks [22]. Railway transitioned away from a persistent free tier. Users will use the paid plan after an initial trial credit, which requires a $5 monthly payment. This money covers the initial resource usage, with any consumption beyond that billed based on usage [23]. Railway particularly suits developers building full-stack projects that prioritize convenience. The high ease of use, especially when integrating databases with application code, justifies the small monthly cost.

2.2.7 Fly.io

Fly.io is a platform for deploying full-stack applications and databases globally, running applications as Docker containers in lightweight Firecracker virtual machines close to users [24]. Its focus is on low-latency global applications, edge computing, and providing developers with

2. Content delivery network

significant control over the deployment topology and resources [25]. Features include deployment to over 35 global regions, fast-booting Fly Machines, persistent storage volumes, managed PostgreSQL clusters, automatic private networking, a powerful command-line interface for management, secrets management, and custom domain support [25, 26]. Deployment primarily involves packaging applications into Docker containers and configuring them via a fly.toml file [24]. Fly.io operates on a pay-as-you-go pricing model based on resource consumption. Although Fly.io previously offered free allowances, these benefits remain available only to users who signed up before their cancelation in October 2024 [27]. The platform's CLI-centric workflow and Docker requirement result in a steeper learning curve than more abstracted PaaS options. Fly.io is best suited for developers comfortable with Docker and infrastructure concepts who require granular control over global deployment or are building latency-sensitive applications, understanding that costs are usage-based from the outset.

2.2.8 Summary of Platforms

Ultimately, the optimal platform for a portfolio project depends on the project's technical requirements, the developer's budget, and the balance between ease of use and control. For static sites, GitHub Pages, Netlify, Vercel, and Render provide solid free hosting solutions. Modern frontend applications benefit from the developer-centric features of Netlify and Vercel. When database integration and backend logic are required, Railway and Render offer integrated solutions with flexible free or low-cost plans, while Heroku provides reliable paid PaaS options. Fly.io stands out for providing unparalleled control for Dockerized, globally distributed applications on a pay-as-you-go basis.

Table 2.1 facilitates a direct comparison of the discussed deployment platforms. It summarizes the key specifications, limitations, and costs associated with their respective entry-level or free tiers. The focus is on quantifiable data, such as resource allocations and project limits.

Table 2.1: Analysis of Lowest-Tier Plans for Major Cloud Hosting Platforms

Platforms	Bandwidth	Build Execution	RAM	CPU	Disk	Price	Number of projects
Vercel	100 GB	100 h/mo	8 GB	2	23 GB	Free	200
Netlify	100 GB	300 min/mo	8 GB	3	N/A	Free	500
Railway	100 GB	N/A	8 GB	8 vCPU	5 GB	5 $/mo	N/A
GitHub Pages	100 GB	10 builds/h	N/A	N/A	1 GB	Free	N/A
Render	100 GB	500 min/mo	512 MB	2	8 GB	Free	1
Heroku	2 TB	75 builds/h	512 MB	N/A	N/A	5 $/mo	100
Fly.io	160 GB	300 min/h	N/A	N/A	N/A	Pay as you go	N/A

3 Application analysis

The previous chapter examined potential deployment platforms; however, the core motivation for the new system is a result of the limitations of current systems, as identified in Chapter 1. This chapter thus begins by outlining the system requirements and defining the necessary functionalities that the software must provide. Then, the data model is presented, illustrating the logical structure of the information managed by the application. These combined specifications serve as the blueprint for the implementation phase that follows.

3.1 System requirements

This section specifies the functional and non-functional requirements for a new software application developed as part of this thesis. This application is conceived as a SaaS web platform that allows users to create, manage, and present customized online portfolios. The development of this tool is driven by identified limitations in existing solutions, particularly in terms of collaborative workflows, customization depth, and integration of live project previews.

3.1.1 Functional requirements

The core functions that allow users to interact with and manage content on the platform are as follows:

- **User Identity and Access -** The system must distinguish between `User` and `Admin` roles. Users must be able to register and subsequently log in. A secure password reset mechanism must also be available.
- **Portfolio Content Management -** Portfolio owners should be able to create new portfolios, view existing ones, edit their content and styling through the integrated editor, control the portfolio's public visibility, and delete their portfolios.
- **Project Content Management -** Similarly, users must be able to create, view, edit, and delete project entries.

- **Collaboration Workflow -** To enable teamwork, portfolio owners must be able to invite other users. Collaborators should then be able to submit their own projects for inclusion in the shared portfolio. Such projects are reviewed by the portfolio owner, who then approves or rejects them. Rejected projects can be modified and resubmitted by the collaborator.

- **Portfolio Editor and Deployment Tools -** The system must provide an integrated portfolio editor for visual arrangement of content sections and style customization. Additionally, to support live project demonstrations, it must provide a project deployment option.

- **Public Access -** Anyone with the portfolio's unique URL[1] must be able to view a published portfolio, with no login required

- **Administration Functions -** Administrators must be able to view system-wide lists of users, portfolios, and projects. Furthermore, they must possess moderation powers, including the ability to delete any user account, portfolio, or project, and manage `Admin` role assignments for other users.

3.1.2 Non-Functional requirements

- **Responsiveness** - The published portfolios generated by the system must be responsive across various devices, including desktops, tablets, and smartphones.

- **Usability and Appeal** - The user interface should be modern, intuitive, and visually appealing.

- **Editor Environment** - The editor's core editing interface and its internal preview mechanism are optimized for desktop environments and are not required to be fully responsive or functional for editing tasks on mobile devices or small tablets.

- **Maintainability** - The system codebase should be designed following good practices to facilitate straightforward maintenance,

1. Uniform Resource Locator

updates, and the potential addition of new functionalities in the future.

3.2 Data model

The data model defines the logical structure of the information managed by the application. This involves identifying the key entities within the system's domain, their attributes, and the relationships between them, forming the basis for a relational database design. An entity relationship diagram is utilized here to visually represent this structure, as shown in Figure 3.1.

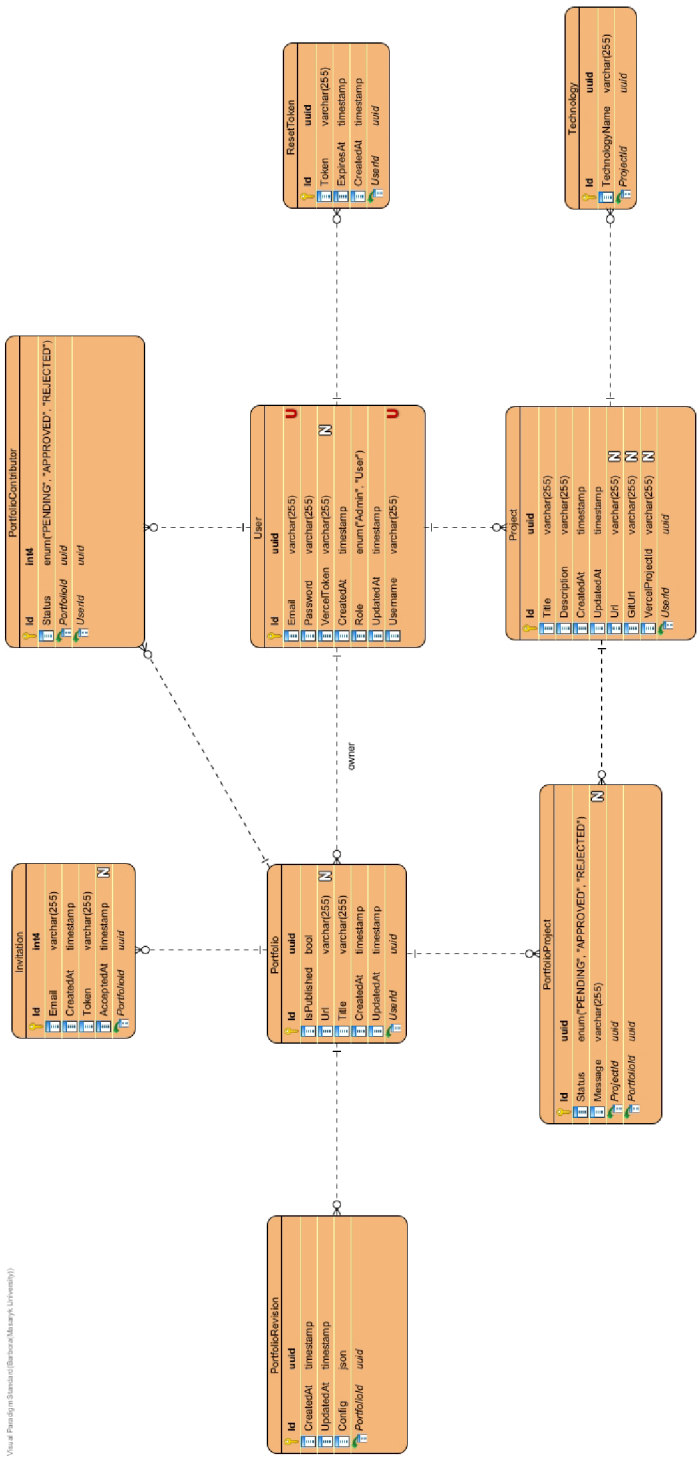

Figure 3.1: Entity Relationship Diagram of Portfolio System

4 Technology Stack

This chapter outlines the technologies used in the implemented system and explains the reasons for their selection.

4.1 Next.js

Next.js is a React framework developed by Vercel to build full-stack web applications. It enhances React by providing a structured environment with features like file-system routing, diverse rendering strategies such as server-side rendering and static site generation, automatic code splitting, client-side navigation with prefetching, and built-in optimizations for assets like images and fonts [28].

This framework was selected because it emphasizes performance, provides full-stack capabilities, and enhances developer experience. Its flexible rendering options allow per-page optimization for speed and SEO[1], while automatic optimizations enhance key metrics related to user experience, such as page load speed, interactivity, and visual stability. Features like API Routes and Server Actions enable building backend logic within the same framework, which eliminates the complexity associated with managing and deploying separate frontend and backend applications. Furthermore, its seamless integration with Vercel ensures an optimized deployment process. Overall, these characteristics make Next.js a powerful and suitable framework for this project's goals.

4.2 TypeScript

TypeScript is a strongly typed programming language that acts as a superset of JavaScript, introducing syntax for static types. Its main goal is to improve developer tooling and catch errors during development through static type checking. TypeScript code compiles to standard JavaScript, ensuring compatibility across all JavaScript environments [29].

1. Search Engine Optimization

TypeScript was selected over JavaScript for this project primarily because its static typing enhances code quality, maintainability, and understandability. This feature provides clarity and allows compilers to detect errors early, which results in more robust code. In addition to compiler capabilities, modern IDEs [2] such as Visual Studio Code and WebStorm, utilize static typing to offer features like auto-completion and immediate error feedback. These features increase productivity and shorten debugging time. TypeScript's structural tools, like interfaces and modules, provide advantages over plain JavaScript for long-term code maintainability, especially when the application scales. Next.js's native support for TypeScript ensures that all these benefits can be seamlessly integrated, making it the optimal choice for the project.

4.3 Prisma

Prisma is a next-generation, open-source ORM [3] designed for Node.js and TypeScript environments. It simplifies the interaction with relational databases such as PostgreSQL, MySQL, SQLite, and MongoDB by providing a type-safe database client, a declarative migration system, and a GUI [4] for data inspection. Prisma utilizes a declarative schema definition language as the single source of truth for database models, from which the type-safe client and database migrations can be generated [30].

Prisma ORM was incorporated into the stack primarily for its robust type-safety guarantees, its enhancement of developer productivity through an intuitive API and tooling, and its effective database migration management. Prisma Client, which is auto-generated from the schema, provides fully type-safe database queries, enabling compile-time error checking and superior editor auto-completion, which significantly reduces runtime errors and accelerates development compared to traditional ORMs or raw SQL. The declarative Prisma schema serves as a clear, centralized definition for database models, simplifying data modeling and improving team collaboration. Prisma Mi-

2. Integrated Development Environment
3. Object Relational Mapping
4. graphical user interface

grate automates the generation of SQL migration files from schema changes, streamlining the process of evolving the database structure in a controlled and predictable manner. This combination of type safety, developer experience, and migration tooling makes Prisma a compelling choice for efficiently and reliably managing database interactions within a TypeScript-centric stack.

4.4 NextAuth.js

NextAuth.js is an open-source authentication library designed explicitly for Next.js applications, and it aims to simplify the implementation of secure user authentication. It handles session management, sign-in and sign-out flows, and integration with various external authentication providers, supporting both server-side and client-side patterns within Next.js [31].

NextAuth.js was selected for its seamless integration with Next.js, extensive provider support, security focus, and flexible session management. Its tight coupling with Next.js offers an idiomatic and efficient authentication solution within that ecosystem, reducing development friction while relying on secure environment variable management, which is typically handled by Vercel.

4.5 Supabase

Supabase is an open-source BaaS [5] platform that provides developers with a suite of integrated tools built around a dedicated PostgreSQL database for each project. Its architecture includes key services such as automatically generated RESTful APIs directly from the database schema, real-time subscriptions for data changes, user authentication, object storage for handling files, and serverless edge functions. This integrated approach provides the core backend functionalities needed to build applications rapidly [32].

Supabase was chosen primarily for its ability to accelerate backend development through its BaaS model. Its foundation on PostgreSQL provides a robust, familiar relational database, which is enhanced by

5. Backend as a Service

features like integrated Row Level Security for fine-grained access control. A significant advantage is its seamless integration with Prisma via, its native Prisma provider adapter, which eliminates the need for the Supabase CLI and simplifies database interactions. This integration allows developers to manage and query the database efficiently while maintaining full control over operations. The developer experience is further enhanced by an intuitive dashboard, the auto-generated APIs that reduce boilerplate code, and comprehensive client libraries. Being open-source is crucial as it mitigates vendor lock-in and ensures data portability, thereby offering flexibility and potential cost-effectiveness. This structure also facilitates a hybrid architecture, allowing Supabase to handle standard BaaS tasks, whereas a framework like Next.js manages more complex backend logic.

4.6 Tailwind CSS

Tailwind CSS is a utility-first CSS framework for rapidly building custom user interfaces that comprises low-level utility classes directly in the HTML markup. It uses a build process to scan project files and generate an optimized CSS file containing only the styles actually used [33].

Tailwind CSS was selected to accelerate UI development, enforce design consistency through its configurable theme, offer deep customization, and ensure high performance via optimized builds. The utility-first approach accelerates implementation by reducing context switching between HTML and CSS. Its build process purges unused styles, resulting in very small CSS bundles for faster load times. Co-locating styles with markup can also improve maintainability. This choice reflects a preference for the utility-first methodology, prioritizing development speed and output optimization.

4.7 Shadcn

Shadcn is not a traditional component library; rather, it is a collection of reusable, accessible UI component primitives, often built using Radix UI and styled with Tailwind CSS. Its unique feature is that developers use a CLI tool to copy the source code of the desired components

directly into their project, effectively building their own library from these primitives [34].

Shadcn was selected for its exceptional control and customization by directly providing the component code. This open-code approach allows deep modifications to match project-specific needs without library constraints while still offering well-designed, accessible defaults. It integrates seamlessly with Tailwind CSS for styling and avoids the potential overhead of monolithic libraries. This choice prioritizes flexibility and ownership over the UI layer, accepting direct code management in exchange for ultimate customization freedom, and forming a cohesive UI strategy with Tailwind CSS.

5 Portfolio System Features

This chapter presents the key implementation features that shape the core functionality of the portfolio system. It details how users construct portfolios through configurable content elements. Additionally, the system supports advanced capabilities, including external deployment integration and collaborative editing through the contributor model. These features are designed to offer both technical depth and ease of use, enabling users to build professional, personalized portfolios with minimal effort while maintaining support for more complex workflows.

5.1 Content elements

To support personalized and structured content creation, the application offers a range of configurable sections that users can include in their portfolios. These include standard elements such as headings, text blocks, and social media links, each offering basic styling options to ensure visual coherence. In addition, specialized sections are tailored to more specific content types, which are discussed in the following sections. These features enable users to build meaningful and well-organized pages without requiring advanced design skills.

5.1.1 Layout structures

To support effective visual organization, the system provides a set of predefined layout options that accommodate typical presentation needs. Users can choose from four classic layouts: a three-column structure and three two-column variations, one with equal-width columns and two with asymmetric widths, offered in complementary left and right variants. These familiar layout patterns support common use cases, such as placing images beside text or separating different types of content within a section. By offering a limited set of well-designed options, the system balances layout flexibility with simplicity, enabling users to create visually coherent pages without being overwhelmed by complex configuration.

5.1.2 Projects sections

A central element of many portfolios is the presentation of selected projects. The application supports this through a dedicated project section, which allows users to showcase their work in a structured and visually appealing manner. Only projects explicitly linked to a given portfolio are displayed, ensuring relevance and curation. Any updates made to these projects are automatically reflected in the associated portfolio, maintaining consistency across the system.

5.1.3 Header

To enhance user experience and align with modern standards, the application incorporates headers. The header not only improves navigability by providing access to essential sections, such as navigation controls, but also contributes to the application's overall feeling and structure. In addition, the interface includes a theme toggler that allows users to seamlessly switch between light and dark modes. This feature acknowledges user preferences and environmental lighting conditions, thus supporting both accessibility and esthetic personalization. The dark mode is particularly beneficial for reducing eye strain in low-light environments, while the light mode maintains clarity and readability in well-lit settings. Together, the header and theme toggler play a crucial role in delivering a modern, user-friendly interface.

5.1.4 Timeline

To cater to common portfolio requirements, such as showcasing work experience, educational background, or project milestones, the system includes dedicated timeline sections. This provides a structured and visually appealing way to present chronological events, enhancing the narrative flow of the portfolio.

5.1.5 Rich-text editor

For text element manipulation, the system incorporates the Tiptap rich-text editor. This editor enables a range of formatting options such as highlighting, bold, italics, and changing text colors. A key factor

in its adoption was its seamless integration with the pre-existing UI components.

5.1.6 Custom Code

To provide users with maximum flexibility and personalization capabilities, the system incorporates a feature that allows the embedding of custom code snippets into portfolio pages. This allows users to create unique interactive elements or apply precise styling beyond standard content blocks.

However, enabling the execution of arbitrary user-provided code introduces significant security risks, primarily the potential for cross-site scripting (XSS) attacks. If not handled correctly, malicious code embedded by a user could be executed in the context of another visitor's browser, potentially leading to session hijacking, data theft, phishing, portfolio defacement, or malware distribution [35].

Executing user-provided code is hazardous and requires strict isolation. The standard, most effective, and selected method is to render the entire custom code block in an HTML frame element equipped with the sandbox attribute. The sandbox attribute instructs the browser to load the iframe's content with a severely restricted set of permissions.

In addition, another layer of defense was employed. All user-provided code is sanitized before it is stored or rendered. Sanitization involves parsing the input and removing potentially dangerous elements, attributes, or CSS properties based on a strict allowlist of known-safe tags and attributes. The DOMPurify library was specifically designed for this purpose in JavaScript environments. The sanitization library must be kept up-to-date because new browser features or attack vectors may require updates to the sanitization rules to prevent bypasses. This introduces an ongoing maintenance requirement for the application.

5.1.7 Contact form

A crucial feature of any portfolio website is the ability for visitors to contact the owner. The developed system implements dedicated contact form sections to serve as the primary communication channel. To ensure communication integrity and prevent spam, the proposed

form integrates robust anti-spam mechanisms. In particular, it uses Google's invisible reCAPTCHA v2. This version works discreetly in the background to distinguish between genuine human users and automated bots, often without requiring any direct interaction from the visitor unless suspicious activity is detected. This significantly reduces the influx of unsolicited messages while maintaining a seamless and trustworthy user experience for visitors trying to connect.

A key aspect of contact form implementation is its backend handling of message delivery. The form submission mechanism is active only for the live, published version of the portfolio. It is intentionally disabled in the editor and preview environments to prevent accidental submissions during the design phase. When a form is successfully submitted, the system automatically relays the message content to the portfolio owner's registered email address. This functionality is implemented using the Resend API[1]. The choice of Resend was motivated by its developer-friendly API and a suitable free tier providing a sufficient monthly email quota of 3000 emails/month for personal portfolio use cases. Crucially, this implementation means users do not need to configure any external email services or credentials, simplifying the setup and ensuring reliable message delivery is handled by the platform itself.

5.2 Portfolio contributors

The system includes a portfolio contributor feature that allows owners to invite individuals to submit projects. Contributors can propose projects but cannot alter existing portfolio content or structure. All proposals undergo owner review for quality. Owners can approve or reject submissions with feedback and suggestions, for contributors to refine and resubmit. Editing an approved project reverts it to pending state for re-review. This iterative cycle of submission, feedback, and refinement, illustrated in Figure 5.1, aligns contributions with portfolio standards.

1. https://resend.com/docs/send-with-nextjs

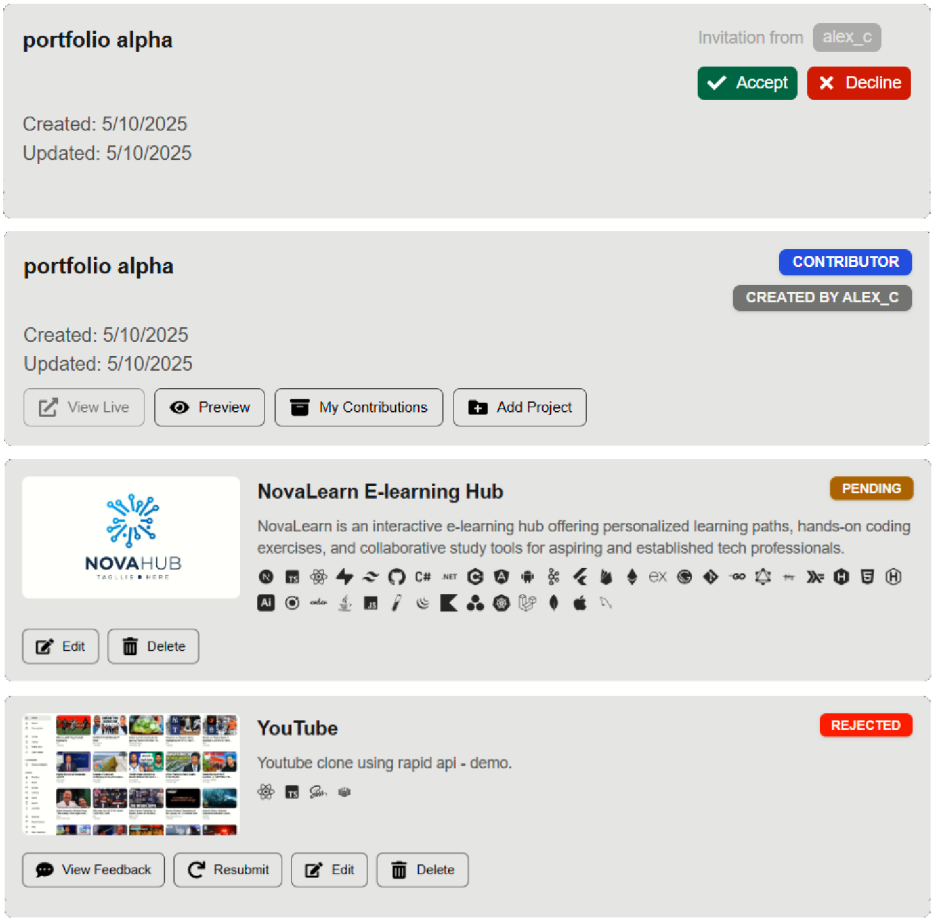

Figure 5.1: Portfolio Contribution Workflow

It depicts the workflow from an initial invitation, through which a user can accept or decline, to an approved contributor's view, which allows them to propose new projects. The figure further illustrates a submitted project with a pending status awaiting review and a rejected project where the contributor is clearly provided with options to view feedback and resubmit. These visual cues underscore the model's support for an iterative refinement process as previously described.

This contributor model, with its emphasis on guided revisions, enhances the overall process of collecting and refining project proposals. Owners can maintain high-quality portfolios by providing targeted

feedback, thus enabling contributors to improve their submissions to meet standards, rather than owners performing extensive edits themselves. This approach not only clarifies quality expectations for contributors, leading to more polished work and greater impact, but also reduces the administrative burden on owners through a transparent and accountable workflow. Ultimately, this design effectively balances broad participation with robust owner oversight, which is key to maintaining consistent quality, ensuring alignment with the portfolio's intended purpose, and protecting its overall integrity.

5.3 Integration with Vercel

The system offers integration with Vercel, allowing users to connect to their Vercel accounts and deploy selected projects directly through the application interface. This integration is facilitated via the Vercel API, streamlining the deployment process and eliminating the need for external deployment management. By embedding deployment capabilities within the portfolio system, users benefit from a unified workflow that supports both project presentation and deployment management. The ability to trigger builds and manage deployments from within the platform further strengthens the system's role as a centralized hub for showcasing and managing developer work.

The integration with Vercel was driven by both technical and strategic considerations. First and foremost, Vercel provides a generous free-tier hosting solution, making it an attractive option for individual developers and students seeking cost-effective deployment without sacrificing features. Another critical factor was the availability and quality of Vercel's API. The platform offers comprehensive and well-documented APIs that support project linking, build triggering, environment management, and deployment tracking. This level of programmatic control is essential for integrating deployment features directly into the application interface, thereby streamlining the user workflow and minimizing context switching.

In summary, Vercel was selected for its robust API, seamless Git integration, superior support for full-stack frameworks, and cost-effective hosting model. Although alternatives such as Netlify provide similar capabilities, Vercel's developer-centric design and flexibility

make it a more suitable choice for embedding deployment functionality within a portfolio system.

5.4 Portfolio Revisions

To enhance data integrity and user experience, the system incorporates a robust portfolio revision tracking feature. This functionality is crucial as it safeguards users against accidental loss or undesired modifications, allowing them to revert to previous states with ease. By maintaining a history of changes, similar to version control in common content builders and editors, users are empowered to experiment with their portfolio's structure and content, confident that they can undo changes or restore prior versions if required. This will ensure a more forgiving and productive environment for portfolio development.

The technical implementation of these revisions employs a dual strategy. Persistently saved portfolio versions are stored in the database as JSON objects. JSON is particularly well-suited for this purpose due to its flexibility in representing complex and evolving portfolio structures, and its widespread support across modern relational and NoSQL database systems facilitates efficient storage and retrieval. Complementing this, recent unsaved modifications are temporarily retained in the client's local browser memory. This local storage primarily serves to power an immediate undo/redo capability for changes made during an active editing session, providing a responsive experience without requiring constant database interaction for minor adjustments.

Users interact with the proposed system through an intuitive history interface, which allows them to view a list of previously saved portfolio versions, typically identified by timestamps. They can select any past version and, if desired, revert their current portfolio to that selected state. This, combined with the instant undo/redo for recent edits managed via local memory, provides a comprehensive safety net. The primary benefits of this approach include significantly improved user confidence, robust data-recovery options, and an efficient, responsive editing workflow that mirrors the expected functionality of modern content creation tools.

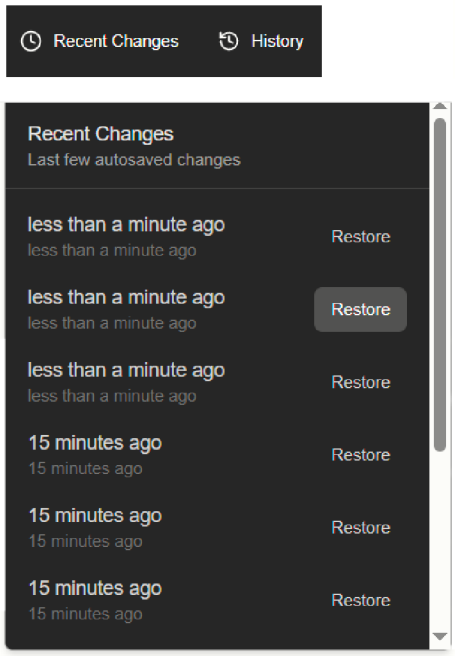

Figure 5.2: User interface for Portfolio Revisions

6 Deployment and System Evaluation

This chapter focuses on the final stages of the project lifecycle after the main development work. Making the portfolio system publicly available, verifying its functionality and user experience through testing, and analyzing the results of this evaluation.

6.1 Deployment

For the portfolio system, Vercel was selected as the deployment platform. The main reason for choosing Vercel is its seamless integration with Next.js and optimized performance for this specific framework with minimal setup time. As a result of this straightforward deployment process, the portfolio website was successfully deployed and is currently running live, accessible via a public URL[1].

6.2 Testing

To ensure that the portfolio website meets system requirements and provides a positive user experience, a dedicated user testing phase was conducted. This phase focused on qualitative user testing with a target group representative of potential technical users.

6.2.1 Testing Methodology and Participants

A group of 10 developers participated in the testing process. This group was specifically chosen due to their technical background, which allowed them to provide insightful feedback on critical aspects such as usability, design clarity, and potential technical issues. The decision to test primarily with programmers and other technical individuals aligns with the intended audience, who would most likely interact with the portfolio.

Each participant was tasked with navigating the deployed portfolio website, exploring its various sections, and interacting with key features. Their evaluation of the overall flow and presentation was a

1. https://portfolio-app-eight-mu.vercel.app/

key component of this phase. The testing was instrumental in identifying several bugs, which were subsequently fixed, leading to a more robust and reliable website.

6.2.2 Feedback Collection

Following the interaction with the site, participants completed a structured feedback form designed to gather detailed insights within an estimated 15 minutes as part of a broader 30-60 minute testing session. This form began by collecting basic demographic information and inquiring about prior experience with website builders. It then guided users through performing and rating the difficulty of specific tasks related to both individual use of the portfolio system and its collaborative features, including testing a collaborator role with provided credentials. The form also solicited open-ended feedback on any issues encountered, the user interface's clarity, overall experience, and suggestions for improvement. Finally, it included four questions adapted from the System Usability Scale (SUS) to quickly assess the perceived ease of use. The inclusion of these SUS questions offers a path to a quantitative measure of usability by potentially normalizing the results, complementing the qualitative feedback gathered.

6.2.3 Results

The feedback obtained from the participants yielded both quantitative and qualitative insights into the usability and overall user experience of the proposed portfolio system. The maximum achievable SUS across the four items was 16. Based on the responses from eight participants, the normalized aggregate score was calculated to be 90 on a 100-point scale, indicating a high level of perceived usability despite the abbreviated form of the SUS instrument.

Demographic data revealed that the majority of participants were aged between 18 and 24. Most respondents did not hold a university degree; however, three respondents held a bachelor's degree, and one held a master's degree. Half of the participants reported prior experience with website builders, providing a mixed-experience group for the evaluation.

Task-based evaluations indicated that most participants found the core functionalities of the system—including both individual and collaborative features—relatively easy to perform. When requested to use a collaborator role with provided credentials, participants generally succeeded without significant difficulty.

Qualitative feedback from the open-ended questions offered several valuable suggestions for improvement. Participants highlighted the need for clearer error messages and input validation, especially in tasks involving form submission. One commonly requested feature was the ability to change the account password. Additionally, suggestions were made to introduce new components and increase the system's modular flexibility. Despite these areas for improvement, the overall sentiment was positive, with participants describing the system as intuitive and expressing satisfaction with the interaction flow.

7 Conclusion

This bachelor thesis successfully addressed existing developer portfolio solutions limitations by designing and implementing a web platform offering enhanced customization, robust collaboration, and simplified project deployment. The primary goal was to equip developers, especially those early in their careers, with a more effective tool for showcasing their work.

The work began with an in-depth analysis of the developer portfolio landscape, which identified gaps in existing tools and offered a guide to suitable cloud deployment options for developers. Based on these insights, a system was designed by establishing its functional and non-functional requirements, followed by the definition of its core data model. The subsequent implementation delivered more than a template. It produced a dynamic web-based portfolio builder featuring a visual editor with diverse content elements, a distinctive contributor model for teamwork enabling collaborative portfolio development, seamless Vercel integration for direct project deployment, and a reliable portfolio revision system for tracking history and changes. This was all built upon a modern technology stack. User testing of the deployed system confirmed its high usability and positive user experience while providing valuable feedback for future iterations.

In essence, this thesis delivered and validated a web-based portfolio management system that directly counters deficiencies found in existing solutions. It provides developers with a valuable and flexible tool, offering customization, collaboration, and deployment capabilities to present their skills and projects effectively.

Bibliography

1. *Were We The Last Junior Software Developers?* [online]. Caroline Arnold, 2025 [visited on 2025-04-21]. Available from: https://ai.gopubby.com/were-we-the-last-junior-software-developers-fb4295294520.
2. *Tech Jobs Report* [online]. CompTIA, 2025 [visited on 2025-05-11]. Available from: https://www.comptia.org/content/tech-jobs-report.
3. *Tech Job Market 2024: Why You Can't Find a Job Right Now* [online]. Patrick Bohan, 2024 [visited on 2025-04-12]. Available from: https://www.pathrise.com/guides/tech-job-market-2024-why-you-cant-find-a-job-right-now/.
4. COMEAU, Joshua. *Building an Effective Dev Portfolio* [online]. Self-published, 2020 [visited on 2025-04-21]. Available from: https://www.joshwcomeau.com/effective-portfolio/.
5. *The Evolution of App Deployment and Packaging: 1990 to Now* [online]. Iron.io, 2020 [visited on 2025-04-22]. Available from: https://blog.iron.io/the-evolution-of-app-deployment-and-packaging-1990-to-now/.
6. GRINSTEIN, Doron. *What is Cloud Deployment? | Complete Guide* [online]. Doron Grinstein, 2022 [visited on 2025-04-22]. Available from: https://controlplane.com/blog/post/what-is-cloud-deployment-complete-guide.
7. MELL, Peter; GRANCE, Timothy. *The NIST Definition of Cloud Computing*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2011. Available from DOI: https://doi.org/10.6028/NIST.SP.800-145.
8. *What is IaaS?* [online]. Google Cloud [visited on 2025-04-27]. Available from: https://cloud.google.com/learn/what-is-iaas?hl=en.
9. B., Patel. Cloud Computing Deployment Models: A Comparative Study. *International Journal of Innovative Research in Computer Science Technology*. 2021, vol. 9. Available from DOI: 10.21276/ijircst.2021.9.2.8.

10. *What is GitHub Pages?* [online]. GitHub, Inc. [visited on 2025-04-25]. Available from: https://docs.github.com/en/pages/getting-started-with-github-pages/what-is-github-pages.

11. *Securing your GitHub Pages site with HTTPS* [online]. GitHub, Inc. [visited on 2025-04-25]. Available from: https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site.

12. *Creating a GitHub Pages site* [online]. GitHub, Inc. [visited on 2025-04-25]. Available from: https://docs.github.com/en/pages/getting-started-with-github-pages/securing-your-github-pages-site-with-https.

13. *GitHub Pages limits* [online]. GitHub, Inc. [visited on 2025-04-25]. Available from: https://docs.github.com/en/pages/getting-started-with-github-pages/github-pages-limits.

14. *What is Netlify?* [online]. Netlify [visited on 2025-04-25]. Available from: https://docs.netlify.com/platform/what-is-netlify/.

15. *Pricing* [online]. Netlify [visited on 2025-04-25]. Available from: https://www.netlify.com/pricing/.

16. *Vercel Documentation* [online]. Vercel Inc. [visited on 2025-04-25]. Available from: https://vercel.com/docs.

17. *Find a plan to power your projects.* [online]. Vercel Inc. [visited on 2025-04-25]. Available from: https://vercel.com/pricing.

18. *The Cloud Application Platform For Deploying, Managing, and Scaling Apps* [online]. Heroku [visited on 2025-04-25]. Available from: https://www.heroku.com/.

19. *Heroku Pricing* [online]. Heroku [visited on 2025-04-25]. Available from: https://www.heroku.com/pricing.

20. *For the next generation of web apps* [online]. Render [visited on 2025-04-25]. Available from: https://render.com/platform.

21. *Deploy for Free* [online]. Render [visited on 2025-04-25]. Available from: https://render.com/docs/free.

22. *Shipping great products is hard. Scaling infrastructure is easy.* [online]. Railway Corp. [visited on 2025-04-25]. Available from: https://railway.com/.

23. *Pay for usage, unlock the following* [online]. Railway Corp. [visited on 2025-04-25]. Available from: https://railway.com/pricing.

24. *Fly Apps overview* [online]. Fly.io [visited on 2025-04-24]. Available from: https://fly.io/docs/apps/overview/.

25. *Run User (or Robot) Code on Fly Machines* [online]. Fly.io [visited on 2025-04-24]. Available from: https://fly.io/.

26. *Networking* [online]. Fly.io [visited on 2025-04-24]. Available from: https://fly.io/docs/networking/.

27. *Fly.io Resource Pricing* [online]. Fly.io [visited on 2025-04-24]. Available from: https://fly.io/docs/about/pricing/.

28. *Next.js Documentation* [online]. Vercel, Inc., 2016/2025 [visited on 2025-03-18]. Available from: https://nextjs.org/docs.

29. *TypeScript for the New Programmer* [online]. Microsoft [visited on 2025-04-23]. Available from: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

30. *What is Prisma ORM?* [online]. Prisma Data, Inc. [visited on 2025-04-28]. Available from: https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma.

31. ORBÁN, Balázs. *Introduction* [online]. 2024-10-24. [visited on 2025-04-20]. Available from: https://next-auth.js.org/getting-started/introduction.

32. *Features* [online]. Supabase Inc [visited on 2025-04-20]. Available from: https://supabase.com/docs/guides/getting-started/features.

33. *Rapidly build modern websites without ever leaving your HTML.* [online]. Tailwind Labs Inc. [visited on 2025-04-25]. Available from: https://tailwindcss.com/.

34. *Introduction* [online]. shadcn [visited on 2025-04-19]. Available from: https://ui.shadcn.com/docs.

35. *Cross Site Scripting (XSS)* [online]. OWASP Foundation [visited on 2025-05-15]. Available from: https://owasp.org/www-community/attacks/xss/.

A Attachments

Attachments to this thesis are stored in the information system of Masaryk University and include the following:

1. **The text of this thesis in PDF format.**

2. **Source code archive (`implementation.zip`):**

   The system's implementation leverages app routing to structure its front-end pages, which are all contained within the `app` directory. The back-end logic, including API endpoints, is organized within an `app/api` folder. For those wishing to run the project locally, a `README.md` file is included with the setup instructions. Although local development is supported, it is geared toward a more straightforward setup and may not include all advanced functionalities in the deployed version. The fully operational application, complete with all features including a pre-configured admin role, is accessible at the following URL: `https://portfolio-app-eight-mu.vercel.app/`

3. **User study form (`form.pdf`):**

   This PDF document contains the structured questionnaire. It was utilized to gather feedback from developers during the user testing phase of the portfolio website.