UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Mykyta Voievudskyi

ProConnectX:
A Web Application for Freelance Marketplace

Bachelor's Thesis (9 ECTS)

Supervisor: Lidia Feklistova, PhD

Tartu 2025

ProConnectX: A Web Application for Freelance Marketplace

Abstract:

This thesis presents the design and development of *ProConnectX*, a secure Minimum Viable Product freelance marketplace aimed at empowering freelancers and small businesses in Estonia. The platform addresses three core issues identified in the local market: fragmented service offerings, lack of trust between clients and freelancers, and limited scalability for self-employed professionals and small businesses. The main goal of the thesis was to create a functional prototype that enables secure, transparent, and scalable interactions between users in a multi-role environment. The system was implemented using a modern full-stack technology stack, including Spring Boot for the backend, React with Redux Toolkit for the frontend, and PostgreSQL as the database. Role-based access control and REST APIs were used to support dynamic user workflows, including escrow-secured order management, dispute handling, and in-session role switching. The backend was enhanced with Aspect-Oriented Programming, Criteria API for complex queries, and secure architectural patterns to ensure data integrity and modular design.

Keywords: Web Application, MVP, Spring Boot, React, Redux, full-stack development, PostgreSQL, freelance marketplace.

CERCS: P170 Computer science, P175 Informatics

ProConnectX: Vabakutseliste turu veebirakendus

Lühikokkuvõte:

Käesolev bakalaureusetöö keskendub turvalise vabakutseliste turundusplatvormi ProConnectX loomisele. Tegemist on minimaalsetöötava tootega, mis on mõeldud toetada Eesti vabakutselisi ja väikeettevõtteid klientide hankimisel ning suurendada klientide usaldust läbi tagatiste ja arvustussüsteemi. Töö keskendub kolmele peamise probleemi lahendamisele: teenuste killustatus erinevates keskkondades, usalduse puudus tööandjate ja vabakutseliste vahel ning piiratud võimalused väiksemate teenusepakkujate kasvuks. Platvormi loodi kasutades kaasaegset täislahendustehnoloogiat: Spring Boot tagasüsteemina, React koos Redux Toolkitiga kasutajaliideses ning PostgreSQL andmebaasina. Rakenduses on toetatud rollipõhine ligipääs ja REST API teenused. Rakenduses on realiseeritud kasutajate vahelised töövood, nagu tellimuste haldamine tagatisega, vaidluste lahendamine ning rollide vahetamine

ühe seansi jooksul. Arhitektuuris kasutati ka aspektprogrammeerimist, Criteria API-põhiseid päringud ning turvalised mustrid andmete tervikluse tagamiseks.

Võtmesõnad: Veebirakendus, MVP, Spring Boot, React, Redux, täislahendustehnoloogia, PostgreSQL, vabakutseliste platvorm.

CERCS: P170 Arvutiteadus, P175 Informaatika

Table of Contents

Introduction

While small and medium-sized enterprises (SMEs) are often celebrated as engines of innovation and economic resilience, their susceptibility to rapid decline has reached a critical point in Estonia, demanding urgent attention. In the non-financial business economy, SMEs represent virtually all private sector firms in Estonia - 99.9% of registered companies - and generate 81.7% of total value added and 82.7% of employment in 202, underscoring their vital role in economic growth [1]. Globally, SMEs make up approximately 90% of all businesses and contribute over 50% of employment, highlighting their importance in both emerging and developed economies [2].

However, recent Eurostat business demography data has shown that Estonia is one of the few EU countries where the enterprise death rate 25.1% in 2022 exceeds the birth rate 16.3%, indicating a net reduction in active firms [3]. This decline poses significant risks to job creation and regional development, particularly outside major urban centers. SMEs are primary contributors to local employment, often providing jobs in smaller towns and rural areas where large corporations have a limited presence. When SMEs close at a faster rate than new ones are established, it directly reduces the number of available jobs, leading to higher unemployment and weakening the economic vitality of these regions. Additionally, the loss of small businesses diminishes local service availability, discourages new investments, and accelerates urban migration as people seek opportunities in larger cities. Over time, this can result in economic stagnation, reduced tax revenues, and widening disparities between urban and rural communities, ultimately undermining balanced regional development [4]. Late payments remain a significant barrier for Estonian SMEs - 53,7% reported payment delays in 2023 [5]. Such delays disrupt cash flow management, which is critical for small businesses that often operate with limited financial reserves and tighter liquidity margins compared to large corporations. When payments from clients are postponed, SMEs struggle to meet their own financial obligations, such as paying suppliers, employees, and covering operational expenses. This financial strain can lead to reduced investment in business development, hinder the ability to take on new projects, and, in severe cases, force businesses to downscale or close. For sole proprietors and micro-enterprises, which constitute a large share of Estonia's SME sector, these challenges are even more acute. Moreover, persistent payment delays create an atmosphere of financial uncertainty, discouraging entrepreneurship and making it harder for SMEs to access credit or external funding, further limiting their growth potential [6].

The concept for the proposed freelance marketplace platform has emerged from the author's personal observations of Estonian SMEs' reliance on traditional outreach methods.

Firstly, many small businesses or sole traders continue to depend on physical word-of-mouth networks rather than readily available - and often low-cost - digital marketing tools. This hesitation to use modern digital platforms makes SMEs slow to adapt to new technologies, which limits their growth and reduces how visible they are to potential customers [7, 8].

Secondly, while global platforms such as Fiverr and Upwork offer broad reach, they frequently overlook local requirements - particularly the need for robust identity verification and escrowed payments - and can expose users to vulnerabilities like fake profiles, unmediated disputes, and payment fraud [9]. In contrast, Estonia's e-government and digital ID infrastructure could furnish far stronger authentication and trust guarantees. The absence of such safeguards on worldwide marketplaces undermines confidence and prevents many Estonian freelancers and clients from engaging fully.

Lastly, without standardised review systems and escrow-based transaction management, payment disputes and mistrust remain endemic on existing services. SMEs and sole proprietors, lacking their own scalable and secure channels, are left unable to move beyond their immediate personal networks. By combining strong, digitally-backed identity checks with an integrated escrow-and-review workflow, the proposed platform seeks to overcome these barriers - enhancing both trust and market access for local businesses [10].

This thesis aims to design and implement a Minimum Viable Product (MVP) of a secure freelance marketplace that empowers small businesses and freelancers in Estonia to grow their client base while offering clients confidence through escrowed payments and a review system. The objectives are:

1. architect a full-stack solution that supports role-based workflows and transactional integrity;
2. enable clients to browse predefined service offerings, place escrowed orders, and submit reviews;
3. provide freelance professionals with tools to manage service listings, accept or reject orders, and handle disputes;
4. evaluate platform performance and user satisfaction through testing and pilot deployment.

Throughout the thesis-writing process, the author utilised AI-assisted tools, such as ChatGPT[1], to enhance readability, structure, and linguistic clarity, focusing solely on non-content-related editing. Additionally, GitHub Copilot[2] was employed to accelerate the development of boilerplate code and repetitive implementation tasks, thereby improving overall productivity.

[1] ChatGPT - https://chatgpt.com/

[2] GitHub Copilot - https://github.com/features/copilot

1. Mõisted ja terminid

SME - A business category defined by the European Union based on employee headcount and financial thresholds. SMEs are classified as companies with fewer than 250 employees and an annual turnover not exceeding 50 million euros or an annual balance sheet total not exceeding 43 million euros.

MVP - A development strategy focused on creating the simplest version of a product that delivers core functionalities to early users with minimal effort and resources. The MVP approach helps validate business hypotheses, gather user feedback, and iteratively improve the product while minimising development costs.

REST API - Representational State Transfer Application Programming Interface, A web service architectural style that uses standard HTTP methods to enable communication between client and server. REST APIs provide stateless, scalable, and cacheable interactions by exposing resources through URLs and using standard verbs such as GET, POST, PUT, and DELETE to manipulate these resources.

JPA - Java Persistence API, a specification for managing relational data in Java applications, enabling object-relational mapping between Java classes and database tables.

Bean - An object managed by the Spring container, representing a component, service, or configuration, which can be injected into other parts of the application.

Enum - Enumeration, a special Java type used to define a fixed set of constant values under a named type, providing type safety and clarity in representing limited possible values.

DTO - Data Transfer Objec**t**, a design pattern used to encapsulate data and transfer it between layers without exposing internal domain models, often used to reduce coupling.

JWT - JSON Web Token, a compact and self-contained mechanism for securely transmitting information between parties as a JSON object, widely used for authentication and authorisation in modern web applications.

Swagger - an API documentation tool used for designing, building, and documenting RESTful web services, often integrated with Spring Boot to provide interactive API documentation and testing interfaces.

CORS - Cross-Origin Resource Sharing, a browser security feature that restricts web applications from requesting resources from a different origin, requiring specific server-side configurations to allow cross-origin requests.

SPA - Single Page Application, an architectural solution for web applications where the entire content is loaded once, and navigation between views occurs dynamically without reloading the entire webpage.

UX - User Experience: the overall experience and satisfaction a user gains when interacting with an application or system, encompassing usability, accessibility, and emotional response.

OAuth - Open Authorization, an open standard protocol that enables secure delegated access to user resources on external services without exposing the user's credentials. OAuth allows third-party applications to obtain limited access to HTTP services through access tokens, facilitating secure Single Sign-On (SSO) and integration with external identity providers.

RBAC – Role-Based Access Control, a security model that restricts system access based on the roles assigned to users within an organisation. RBAC defines permissions according to roles rather than individual users, simplifying the management of user privileges and ensuring that users can only access resources necessary for their role.

2. Competitor analysis in Estonia

This section undertakes a comparative analysis of competitor applications within the Estonian market. Selected applications have a similar purpose as the proposed platform - facilitating the connection between service providers and clients - making it possible to assess their features, user experience, technology, and overall effectiveness.

In performing this analysis, a meticulous review of competitor applications is conducted to ascertain their key functionalities, ease of use, user interfaces, and overall user experience. This exploration provides valuable insights into the strengths and weaknesses of each application, enabling a nuanced understanding of the competitive landscape and preemptive measures to address potential challenges at the outset of development.

The author chose to analyse GoWorkaBit and Vabakutselised applications, as they are currently among the most prominent platforms in the Estonian market for connecting freelancers with potential employers. Their selection was based on their high visibility in search engine results for relevant queries and their widespread recognition among the general public. These platforms provide essential insights into existing solutions for connecting freelancers with potential employers.

The web application GoWorkaBit[3] (GWB) is a platform for freelancers to find jobs and for employers to hire employees. The author finds that the app offers a favourable overall user experience, although there is room for improvement in its interface design.

The user interface of the GWB platform is generally clean and readable, with effective use of colour coding - green for actions and purple for information. This supports quick visual recognition. The user experience and user flow are intuitive, guiding users step by step from selecting a date to viewing job details and applying, which helps streamline the process. However, the interface has notable drawbacks. The visual design lacks a clear hierarchy, making it difficult to distinguish between primary and secondary elements. Key actions, such as applying for a job, are not visually emphasised enough. Overall, the design feels outdated and overly reliant on text, with poor scalability for users browsing multiple listings, limiting usability and engagement.

[3] GoWorkaBit - promo.goworkabit.com/home/

Notably, the inclusion of OAuth 2.0 authentication enhances user convenience by enabling registration or login through Google or Facebook accounts. The performance of the app is high.

In addition to its strengths, GoWorkaBit integrates with the Intercom chatbot, enabling users to receive real-time assistance. This feature enhances the overall user experience by providing instant support for enquiries related to job postings, application processes, or platform functionalities. Employers can easily create and post job listings, while freelancers can browse and apply for projects through a structured and user-friendly application process.

To maintain a secure and trustworthy environment, GoWorkaBit implements an additional authorisation process before users can apply for jobs or post listings. Freelancers are required to enter their identification code and provide contact details, ensuring that only verified individuals can engage with job opportunities. Furthermore, the application process includes several steps aimed at improving employer decision-making. Applicants must submit a brief motivation letter, select relevant skills from a predefined list provided by the employer, and answer three standardised questions that remain consistent across all job postings. These measures help employers gain a clearer understanding of each candidate's qualifications, reducing hiring risks and streamlining the selection process.

Employers must also undergo a verification process before they can post job offers. This ensures that the person or company offering a job is legally accountable, further increasing trust and transparency on the platform. Such verification steps protect freelancers from fraudulent or unreliable job postings, reinforcing GoWorkaBit as a secure and professional environment for job seekers and businesses alike.

However, despite these advantages, the platform lacks a built-in real-time communication feature between freelancers and employers. Currently, all messaging occurs through third-party channels, which can slow down the hiring process and prevent direct collaboration. Introducing a direct messaging feature within the platform could significantly enhance user experience by facilitating faster interactions, reducing misunderstandings, and ensuring a smoother recruitment process.

In terms of pricing, GoWorkaBit operates on a commission-based model. Employers are charged a percentage of the freelancer's earnings, and they must also pay a fee to post job offers.

The freelance marketplace Vabakutselised[4] allows freelancers to offer their services. However, compared to GWB, Vabakutselised faces challenges with outdated design and lacks transparency in its interface, hampering user-friendliness. Additionally, the platform's non-responsiveness and lack of optimisation for mobile users further detract the overall user experience.

Applying for jobs on Vabakutselised is relatively straightforward, as freelancers list their services, and employers must manually search for suitable candidates rather than post projects. However, the review system is limited, as freelancers only receive star ratings without detailed feedback.

The interface of Vabakutselised presents a colourful and playful design, which may appeal to a younger audience, but overall lacks professional polish and usability. While the main categories are easily accessible from the sidebar and the site offers visual previews of services, the interface is cluttered, with excessive visual elements and inconsistent font usage, which can overwhelm users. The use of background illustrations and saturated colours distracts from core functionalities, reducing visual clarity. Furthermore, the user flow is not intuitive; key actions such as browsing, filtering, and purchasing services are not clearly prioritised or guided. There is little visual hierarchy, and important buttons like “Join” or “Log in” are not emphasised, making it harder for new users to engage. Overall, the platform lacks modern user experience standards, and its design feels outdated, limiting user engagement and trust.

From a technical standpoint, Vabakutselised lacks several modern features that could enhance user experience and security. The platform does not support OAuth 2.0 authentication, meaning that users must manually create accounts instead of conveniently logging in with existing credentials from services like Google or Facebook. Additionally, Vabakutselised does not provide chatbot assistance, making customer support less accessible compared to GoWorkaBit, which integrates with Intercom for real-time assistance. The absence of real-time chat functionality further complicates communication between freelancers and employers, requiring users to rely on external messaging platforms.

In terms of the pricing model, Vabakutselised operates on a commission-based structure, charging 15% service fees on all transactions. This means that freelancers must pay a portion of their earnings to the platform, which can be a considerable expense, especially for high-

[4] Vabakutselised - https://www.vabakutselised.com/

value projects. Unlike GoWorkaBit, which also charges employers for posting job listings, Vabakutselised does not have an upfront cost for employers, making it more attractive for businesses looking to hire without additional fees.

To provide a clear differentiation between these platforms and their primary features, the author presents a concise comparison in Table 1, elucidating the distinguishing characteristics of the mentioned platforms.

Tabel 1. Features a comparison of competitive applications.

	GoWorkaBit	**Vabakutselised**
Post projects as employer	x	-
Apply for projects as freelancer	x	-
Offer services as freelancer	-	x
Buy services from freelancer as client	-	x
One account for multiple roles	-	-
Role switching within one session	-	-
User-friendly interface	x	-
OAuth 2.0 authentication	x	-
Responsive design	x	-
Number of categories	32	21
Reviews of employers	x	-
Reviews of freelancers	x	x (only stars)
Modern UI	-	-
High performance	x	x
Interactivity	-	-
Live chat or chatbot support	x	-
Real-time messaging	-	-
Multilingual support	x	-
Smart upsales	-	-
Price	Fixed fee for job postings and an hourly commission based on the freelancer's salary	15% service fees on all transactions

3. Application requirements

In any software development project, the foundation for success lies in a thorough understanding of the project's requirements. Requirements analysis involves identifying, documenting, and validating the needs and expectations of stakeholders to ensure that the resulting software meets their objectives effectively [11]. This section delves into the comprehensive analysis of the various requirements identified for the application before its development.

Implementing a role-based access control (RBAC) system is paramount in any application, especially in a dynamic environment like a freelance marketplace. RBAC ensures that users are granted access only to the resources and functionalities necessary for their specific roles and responsibilities within the platform. This principle of least privilege minimises the risk of unauthorised access to sensitive information or critical functionalities, enhancing overall security [12].

The platform caters to four primary user roles:

- **Client**: Clients are users who purchase services from freelancers. They can browse available offerings, submit service requests, communicate requirements, and see the progress of their orders. Clients are also responsible for managing payments and providing feedback after service completion.
- **Freelancer**: Freelancers are individuals who offer their skills, expertise, and services to fulfil client projects. Their role entails creating and managing profiles, offering services, collaborating with clients, delivering project milestones, and managing payments.
- **Admin**: Administrators serve as the platform's overseers, responsible for maintaining its integrity, managing user accounts, resolving disputes, and ensuring smooth platform operations. Their role encompasses user management, content moderation, platform configuration, and generating reports and analytics.
- **Unauthenticated**: This role designates individuals who have yet to log in or register an account. While in this role, users are afforded limited access, primarily restricted to browsing public content. Such users are unable to partake in interactive features such as service posting or ordering. However, this role serves as a pivotal entry point for prospective users, enabling them to acquaint themselves with the platform's offerings before committing to registration.

Therefore, since the application uses the RBAC approach, all functional requirements are divided into three categories. Each category is a role and has specific requirements aligned with its responsibilities.

3.1 Functional Requirements

Functional requirements (FR) describe the specific functionalities or features that the software must perform to satisfy the needs of its users. These requirements typically outline the interactions between the system and its users or other systems, specifying what actions the software should enable or what outputs it should produce in response to given inputs [13].

For the correct functioning of the platform, the following functional requirements were conducted:

3.1.1 System

FR1: The system shall provide access to general platform information without requiring authentication.

The platform shall display details about its purpose, features, and pricing to all users, ensuring transparency before registration.

FR2: The system shall provide the ability for all users to search and filter service listings.

Users should be able to apply filters based on criteria such as service title, price, rating, category, and location. This functionality helps users efficiently discover relevant services that match their needs and preferences.

FR3: The system shall allow authenticated users to view and update their personal contact information, including first name, last name, phone number, and address.

This ensures that users can keep their profile data accurate and up-to-date for effective communication and service delivery.

FR4: The system shall allow authenticated users to upload files related to orders, services, or profile information.

This functionality enables users to provide necessary documents, images, or other relevant files to support their service listings or order fulfilment, ensuring clear communication and transparency in transactions.

3.1.2 Unauthenticated users:

FR5: Unauthenticated users shall be able to view reviews and ratings.

Basic review summaries and average ratings should be visible.

FR6: Unauthenticated users shall not be able to leave reviews or ratings.

Only registered users should be allowed to submit feedback on services or clients.

FR7: Unauthenticated users shall have access to registration functionality.

Users should be able to register using an email and password.

FR8: Unauthenticated users shall have access to login functionality.

A login system should be available for returning users.

FR9: Unauthenticated users shall be able to register as freelancers.

Users should be able to create freelancer accounts to access features such as job applications and service postings.

FR10: Unauthenticated users shall be able to register as clients.

Users should have the option to register as clients to post job offers and hire freelancers.

3.1.3 Client

FR11: Clients should be able to place orders.

This enables clients to engage freelancers for specific tasks through a structured and secure process.

FR12: Clients should be able to initiate a dispute regarding an order if the delivered work does not meet the agreed requirements.

This provides a formal mechanism for conflict resolution, ensuring fairness and maintaining platform integrity.

FR13: Clients should be able to reject proposals submitted by freelancers.

This feature ensures clients can decline unsuitable proposals, maintaining control over the selection process and ensuring alignment with project needs.

FR14: Clients should be able to notify the platform administrator to request assistance in resolving a dispute.

This functionality ensures that clients can escalate unresolved issues for impartial review and mediation by platform administrators.

FR15: Clients should be able to leave feedback and rate freelancers upon completion of an order.

This ensures service quality, promotes accountability, and builds trust within the platform community.

3.1.4 Freelancer

FR16: Freelancers must be able to post the services they offer.

Allowing freelancers to showcase the services they offer helps them attract potential clients and market their skills effectively.

FR17: Freelancers should have access to feedback and rating systems for collecting client reviews.

Implementing feedback and rating systems allows freelancers to receive feedback from clients, build their reputation, and establish credibility within the freelance marketplace.

FR18: Freelancers should be able to submit resolution proposals when a dispute arises regarding an order.

This allows freelancers to offer solutions, explain their work, or propose revisions, promoting fair dispute resolution and fostering transparent communication.

FR19: Freelancers should be able to notify the platform administrator to request assistance in resolving a dispute.

This feature allows freelancers to seek an objective intervention from the platform when direct communication with the client does not lead to a resolution.

FR20: Freelancers should be able to write feedback for clients upon completion of an order.

Providing freelancers with the ability to write feedback for clients enables them to share their experiences and impressions of working with the client. This feature promotes transparency and accountability within the platform, allowing freelancers to rate clients based on factors such as communication, professionalism, and payment punctuality.

3.1.5 Administrator

FR21: Administrators should be able to review, approve, or reject new user registration requests.

This process ensures that only verified and legitimate users gain access to the platform, maintaining service quality and trust.

FR22: Administrators should be able to resolve disputes by either refunding the client's payment or releasing the payment to the freelancer.

This functionality provides a final decision-making authority to ensure fair outcomes in cases where clients and freelancers cannot reach an agreement independently.

3.2 Non-functional Requirements

Non-functional requirements (NFR) define the quality attributes or constraints that the software must adhere to in addition to its functional capabilities. These requirements encompass aspects such as performance, scalability, reliability, usability, security, and regulatory compliance, which are critical for the overall success and acceptance of the software [13].

3.2.1 Performance and usability

In the competitive landscape of web applications, performance plays a pivotal role in determining user satisfaction, engagement, and ultimately, the success of the platform. In the case of a freelance marketplace platform, where users expect seamless interactions and swift access to information, optimising performance is paramount.

A crucial aspect of performance assessment is Lighthouse, a tool developed by Google that evaluates web pages across various metrics, including performance, accessibility, SEO, and best practices. Lighthouse provides a comprehensive audit of web application performance, generating a report with actionable insights and recommendations for improvement. By leveraging Lighthouse, developers can identify performance bottlenecks, optimise critical rendering paths, and enhance overall user experience. The metrics assessed by Lighthouse, such as First Contentful Paint (FCP), Largest Contentful Paint (LCP), and Cumulative Layout Shift (CLS), set clear benchmarks for performance optimisation and help ensure that web applications meet the rigorous standards of speed, responsiveness, and accessibility expected by users [14].

To define realistic and data-driven performance goals for the MVP phase, the non-functional requirements have been adjusted to align with Lighthouse's 10 median performance thresholds. Specifically, the platform targets a Lighthouse performance score of 50 or higher. This ensures that the system achieves at least an average level of performance before pursuing further optimisations.

As a result, the following non-functional requirements have been conducted:

NFR1: The system should achieve an FCP score of no more than 1.6 seconds on desktop and 3.0 seconds on mobile.

FCP measures how quickly the first visible text or image is rendered on the screen. Faster FCP times contribute to a smoother initial loading experience, enhancing user satisfaction. The specified thresholds reflect median performance levels, ensuring that users on both desktop and mobile devices can access content promptly without noticeable delays [14].

NFR2: The system should maintain an SI score of 2.3 seconds or less on desktop and 5.8 seconds or less on mobile.

Speed Index evaluates how quickly visible parts of a page are displayed during loading. Meeting these thresholds ensures that users perceive the page as loading swiftly, which is critical for keeping engagement high and minimising bounce rates, particularly on resource-constrained mobile devices [14].

NFR3: The system should limit TBT to under 350 milliseconds on desktop and under 600 milliseconds on mobile.

TBT measures the total amount of time between FCP and Time to Interactive (TTI) during which the main thread is blocked and unable to respond to user input. A lower TBT score indicates smoother interaction and faster responsiveness of the application. By keeping TBT within these limits, the system ensures responsive user interactions and reduces frustration caused by delayed input handling, especially for mobile users facing less powerful devices [14].

NFR4: The system should achieve an LCP score of no more than 2.4 seconds on desktop and 4.0 seconds on mobile.

LCP measures the loading time of the largest content element visible within the viewport. A lower LCP score indicates faster loading times for critical content, which is essential for delivering a positive user experience. These thresholds target the average performance levels, ensuring that essential content appears promptly, which is critical for user retention and perceived application quality across different device types [14].

NFR5: The system should have a CLS score of less than 0.25.

CLS measures the visual stability of a web page by quantifying unexpected layout shifts during page load. A lower CLS score indicates a more stable and predictable user interface, preventing elements from unexpectedly shifting position and disrupting user interaction. By maintaining

a CLS below 0.25, the system achieves median-level visual stability, preventing most content shifts and thereby ensuring a predictable and comfortable viewing experience [14].

3.2.2 Security and scalability

NFR6: The system should implement the HTTPS protocol for secure communication.

HTTPS encrypts the data exchanged between the user's browser and the web server, ensuring confidentiality and integrity. By implementing HTTPS, the system protects sensitive information such as user credentials, payment details, and personal data from unauthorised access and interception, enhancing overall security and user trust.

NFR7: The system shall restrict access to uploaded sensitive files, ensuring they are only accessible by their respective owners or authorised users.

This prevents unauthorised access to private documents and maintains data privacy for users uploading sensitive materials.

NFR8: The system shall securely hash user passwords before storing them in the database.

This ensures that sensitive authentication data is not stored in plain text, providing protection against data breaches and unauthorised access.

NFR9: The system should be designed to scale horizontally to accommodate increased user traffic and workload demands.

Scalability is essential for ensuring that the system can handle growing user traffic and workload demands without experiencing performance degradation or downtime. Horizontal scalability, achieved through the addition of more resources such as servers or instances, allows the system to distribute incoming requests across multiple nodes, effectively increasing its capacity to handle concurrent users and processing tasks. By designing the system to scale horizontally, it can adapt to changing usage patterns and accommodate future growth, ensuring consistent performance and availability under varying load conditions. Scalability is crucial for meeting user expectations, maintaining service reliability, and supporting business growth over time.

NFR10: The system must implement secure session management to ensure the confidentiality and integrity of user sessions.

Secure session management is essential to protect user data and prevent unauthorised access to sensitive information. By implementing secure session management practices, such as using secure cookies, encrypting session data, and implementing session timeout mechanisms, the system can mitigate the risk of session hijacking and unauthorised access.

NFR11: The system must securely manage session tokens and prevent session fixation attacks.

Secure management of session tokens is crucial to prevent session fixation attacks, where an attacker hijacks a user's session by forcing them to use a known session identifier. By securely generating and managing session tokens, the system can mitigate the risk of session fixation attacks and ensure the integrity of user sessions.

NFR12: The system should log and monitor user activities for security auditing and incident response purposes.

Logging and monitoring user activities enable the system administrators to track and analyse user interactions, identify suspicious behaviour, and detect potential security incidents in real-time. By maintaining comprehensive audit logs and monitoring user activities, the system enhances visibility into security-related events, facilitates timely incident response, and supports forensic investigations in the event of security breaches or compliance audits.

4. Used technologies

ProConnectX represents a software solution tailored to facilitate the seamless and efficient engagement of freelance professionals across diverse industries. Its core objective lies in ensuring accessibility, a pivotal aspect of its architectural design. To achieve widespread availability, the application must be available on as many devices as possible and be cross-platform. Consequently, the adoption of web-based technology emerges as the most fitting strategy, promising not only broad accessibility but also streamlined avenues for future expansion and enhancement.

In the realm of web application development, there are three fundamental components: frontend, backend, and database. The frontend is dedicated to crafting a user interface, encompassing the design and development of web pages that users interact with. Leveraging technologies such as HTML, CSS, and JavaScript, the frontend serves as the gateway to intuitive user engagement [15]. The backend focuses on server-side operations, where intricate business logic processes user requests and delivers timely responses. Widely popular programming languages like Python, Java, Ruby, and JavaScript fuel the backend infrastructure, ensuring robust functionality [16]. The database, constituting the third pillar, serves as the repository for essential system data, facilitating efficient data organisation and retrieval. With a spectrum of database options ranging from flat files and relational and object-relational databases to cutting-edge NoSQL databases, web applications are empowered with good flexibility to cater to a wide array of data management requirements [17]. By carefully selecting and integrating appropriate technologies and methodologies within each component, developers can construct a cohesive and high-performing application that meets the demands of modern users.

4.1 React

The importance of selecting a programming language and framework for front-end development has become increasingly apparent as modern user interfaces demand higher levels of sophistication. In the early days of the web, simple static pages sufficed and were constructed using basic technologies like HTML, CSS, and HTTP requests. However, as society's need for automation and productivity enhancement grew, so did the requirements for web applications. Today's users expect dynamic, interactive interfaces that adapt seamlessly to their needs. From real-time updates to personalised experiences, the bar for functionality has been raised significantly. This shift has transformed front-end development from mere presentation to a

realm encompassing complex logic, state management, and integration with back-end systems [18].

Furthermore, the proliferation of devices and platforms adds complexity to front-end development. With users accessing applications across various devices, including smartphones, tablets, and desktops, developers must ensure optimal performance and usability across diverse screen sizes. In this context, the choice of programming language and framework plays a crucial role. Frameworks like React, Angular, and Vue.js have emerged as frontrunners [19].

According to their comparison [20], each has unique advantages and is renowned for its comprehensive feature set. However, a notable difference between Angular and Vue.js or React lies in how they handle the Document Object Model (DOM) and manage data. Angular utilises the real DOM, which involves overwriting the entire DOM each time an object is modified. In contrast, React and Vue.js employ Virtual DOM (VDOM) technology, a streamlined version of the real DOM. The main idea is to compare the stored previous version of VDOM with the updated version, and based on the differences, frameworks calculate the minimal set of changes needed to update the actual browser DOM and eventually efficiently apply these changes. This approach aims to minimise rendering time by updating only the changed components, thus enhancing performance and efficiency [21].

Vue.js distinguishes itself from React in several key aspects. While both frameworks utilise Virtual DOM for efficient rendering, Vue.js offers a more gradual learning curve and simpler syntax compared to React. Additionally, Vue.js offers both bidirectional and unidirectional binding options for data manipulation. Despite Vue.js' notable adaptability in front-end development, its technical assistance may need to be more reliable due to the limited scale of its development team. Moreover, its smaller size suggests fewer features [20].

Analysing the prevalence of web frameworks, as documented in the Stack Overflow survey of 2023 [19], React stands out as the foremost front-end web framework and outpaces the popularity of its competitors by more than two times. Its widespread adoption can be attributed to several factors. Firstly, its open-source nature enables free utilisation, customisation, and dissemination, bolstered by a sprawling network that offers a wealth of knowledge-sharing resources. These resources include tutorials, comprehensive documentation, and vibrant online

communities, fostering a collaborative environment conducive to learning and skill development.

Furthermore, React has support for TypeScript, a statically typed superset of JavaScript that brings several advantages to React development. TypeScript enhances the development experience by providing type-checking capabilities during development, which helps catch errors early in the development process. By enforcing type safety, TypeScript reduces the likelihood of runtime errors and enhances code quality [22]. With TypeScript, developers can define clear and explicit interfaces for components, props, and state [23]. Additionally, TypeScript's strong typing system allows developers to navigate codebases more efficiently and provides better code editor support, including auto-completion and real-time error feedback.

Despite Vue.js simplifying the development of small and medium-sized applications, the author maintains a preference for React over other frameworks. This choice is motivated by a desire for ongoing learning and the careful consideration of factors such as performance, scalability, and adaptability within the project. Additionally, the seamless integration of readily available third-party libraries further reinforces the author's preference for React.

4.2 Spring Boot

The easiest and most modern way is to write a back-end in the same language as a client part of an application. This increases code simplicity, simplifies maintenance, and promotes consistency across the application. However, when choosing a framework for building RESTful APIs, factors such as performance, scalability, ease of use, community support, and integration capabilities must be carefully considered.

According to the Stack Overflow Developer Survey of 2023 [5], the top 3 most prevalent back-end frameworks are Express, Flask, and Spring Boot [19].

The comparison between Express, Spring Boot, and Django was conducted to explore efficiency and reliability by measuring server response time and fault tolerance using the Apache JMeter tool [24]. By using identical applications, researchers made tests for various request types, including GET, PUT, POST, and DELETE. As a result, the Spring Boot framework demonstrated the ability to handle 8000 requests simultaneously with the highest processing speed, while Express exhibited significantly fewer failed requests even under heavier loads. Conversely, Django was found to be less reliable and efficient. Therefore,

considering that ProConnectX is not anticipated to experience such a high volume of requests concurrently in production in the near future, the Spring Boot framework emerges as the optimal choice for developing this thesis application [24].

Spring Boot is an open-source framework designed for developing robust, dependable, and scalable enterprise applications. It simplifies web application development compared to traditional Java frameworks, making the process more transparent. Additionally, Spring Boot builds on the foundations of the Spring framework, offering greater user-friendliness than Spring while retaining all of its features. Developed as a microservices framework, Spring Boot facilitates rapid application deployment to production, automatically handling configurations [25]. Furthermore, its extensive community backing improves the framework's reliability and ensures ongoing enhancements and support for future projects. Java's enduring status as one of the world's most in-demand programming languages adds popularity and efficiency to the Spring Boot framework. Thus, the preference for Spring Boot is rooted in its proven track record, collaborative community, performance, and favourable impact of Java usage.

4.3 PostgreSQL

The selection of an appropriate data management system holds paramount importance in the realm of web application development. Within this context, the Spring Boot framework offers an invaluable toolset known as Spring Data JPA, which improves the way developers interact with a data access layer, particularly relational databases [26]. It provides a higher level of abstraction when working with JPA. JPA, or Java Persistence API, is a Java specification for managing relational data in Java applications [27].

According to the Stack Overflow Developer Survey of 2023, PostgreSQL is the most popular relational database [19]. PostgreSQL's popularity is further underscored by its extensive feature set and performance capabilities. Unlike some proprietary databases, PostgreSQL's open-source nature allows for greater flexibility and transparency in its usage. Additionally, PostgreSQL boasts strong support for advanced SQL features, including complex queries, indexing, and transaction management, making it well-suited for demanding workloads and complex data models [28].

The selection of PostgreSQL is justified for web applications due to its reliability, extensive feature set, strong community support, and adaptability to various project requirements.

5. User Interface Design

The platform's user interface is built in accordance with Material Design principles, using Material UI (Material UI) version 2 to ensure consistency, accessibility, and responsiveness across desktop and mobile devices. Material Design provides a unified system of layouts, components, and motion that helps establish clarity and hierarchy in complex applications.

Material Design layouts adhere to an 8-dp baseline grid to create visual balance and predictability in element placement and spacing [29]. Typography follows a predefined scale with discrete font sizes, line heights, and weights to guide hierarchy. This scale is implemented via MUI's Typography component [30]. To ensure readability across devices, the platform employs MUI's *responsiveFontSizes()* helper, which dynamically adjusts typography scale based on viewport size [33].

Motion and feedback are integral. Components use ripple effects, elevation changes, and transitions to signal user interactions and state changes, reinforcing a sense of depth and continuity in the UI [31].

A custom theme defines the color palette - soft blues and whites to evoke professionalism and trust, aligning with freelance marketplace branding guidelines [32]. The theme is configured via MUI's theming API, specifying primary/secondary colours, typography variants, and component shape radiuses.

To improve perceived performance and guide user focus during asynchronous operations, a global backdrop is displayed when pages initially load or critical actions are in progress. This dimmed overlay directs attention to active processes and signals state changes within the application.

For non-critical process notifications such as successful form submissions or error messages Snackbars appear at the bottom of the screen, persist briefly, and then auto-dismiss within 6 seconds without interrupting the user flow.

Layouts leverage MUI's Grid system and responsive breakpoints, adapting from a multi-column desktop layout to a single-column mobile view for widths below 600dp.

Although the current MVP uses a single language, the UI is wired for i18n support. All strings are externalised for runtime locale switching. The theming system is structured to allow future additions such as dark/light mode toggles or client-specific branding variants without refactoring component code.

6. Architecture

The created application is available from https://proconnectxui-production.up.railway.app/, and the code is located at https://github.com/NickPoint/ProConnectX. The architecture of the freelance marketplace platform has been designed with scalability, modularity, and maintainability as core principles. The system follows a distributed, layered architecture leveraging modern full-stack technologies to facilitate the interaction between freelancers and clients, ensuring secure and efficient service exchange.

The application is developed using a three-tier architecture consisting of a React-Redux frontend, a Spring Boot backend, and a PostgreSQL relational database. Communication between the frontend and backend occurs through RESTful APIs. Apache Kafka is integrated into the system for processing notifications. The system is containerised using Docker and deployed on the Render platform, ensuring scalability and platform independence.

6.1 Frontend Architecture

The frontend is implemented using React in combination with Redux Toolkit (RTK) for state management. For efficient API interaction, RTK Query is utilised, which abstracts caching and server communication logic, thus enhancing developer productivity and application performance.

A responsive and consistent user interface is achieved through the Material UI (MUI) component library. For complex form management and validation, Formik and Yup libraries are employed, respectively, ensuring form data integrity and user-friendly error handling.

Rich-text content, such as freelancer bios or service descriptions, is managed using Tiptap, a modern WYSIWYG editor built on top of ProseMirror. Geolocation and mapping features, for location-based service offerings, are integrated via the Google Places API.

To minimise manual API integration and enforce type safety, the application uses the @rtk-query/codegen-openapi tool that auto-generates RTK Query API slices directly from the backend's OpenAPI specification. This approach ensures tight synchronisation with backend contracts and significantly reduces boilerplate in data-fetching logic.

6.2 Backend Architecture

The backend is built using the Spring Boot framework, structured primarily as a monolithic application while remaining modular in design to support future service decomposition if needed. The data persistence layer utilises the Spring Data JPA framework with Hibernate and the Criteria API for type-safe and dynamic querying. Liquibase, occasionally in conjunction with Groovy scripts, manages database migrations and schema evolution.

To improve separation of concerns and facilitate modular functionality, the backend makes extensive use of aspects via Spring Aspect-Oriented Programming (AOP) owner checking. Additionally, event listeners are employed for decoupled event-driven behaviours, such as sending notifications upon different users' actions. Additionally, an event-driven architecture underpins notification logic, leveraging Apache Kafka for asynchronous message transport and Thymeleaf to construct email templates.

To enforce a consistent and predictable lifecycle across domain entities such as Order, the system uses explicit transition rules encapsulated in domain-level enums. For instance, the OrderStatus enum defines permissible state transitions through a *canTransitionTo()* (see Figure 1) method, which prevents invalid or out-of-sequence changes.

```
public boolean canTransitionTo(OrderStatus target) {
    return switch (this) {
        case CREATED -> target == IN_PROGRESS || target == CANCELED;
        case IN_PROGRESS -> target == SUBMITTED_FOR_REVIEW;
        case SUBMITTED_FOR_REVIEW -> target == APPROVED || target ==DISPUTED;
        case DISPUTED -> target == APPROVED || target == CANCELED;
        case APPROVED -> target == COMPLETED;
        default -> false;
    };
}
```

Figure 1. Example of *canTransitionTo()* method inside OrderStatus enum.

This approach helps separate transition logic from controller or service layers, improves testability, and safeguards business rules at the domain level.

The overall design also employs strategy patterns, polymorphism, and well-defined interfaces to encapsulate common logic across different user roles and actions, reducing boilerplate and simplifying horizontal scaling and future developments.

Object mapping between DTOs (Data Transfer Objects) and domain models is handled using the MapStruct library, which ensures compile-time validation and performance benefits. Code boilerplate is minimised using the Lombok library, promoting clean and readable class definitions.

The backend exposes a fully documented RESTful API using OpenAPI (Swagger) specifications. This API schema serves not only for documentation purposes but also as the foundation for generating client-side code, ensuring strict contract-based communication between frontend and backend.

6.3 Data Model and Persistence Architecture

The application is underpinned by a well-structured relational data model designed to support its complex business logic, including user management, service ordering, transaction handling, role-based authorisation, and service categorisation. This model is implemented using PostgreSQL, and data access is facilitated through Spring Data JPA (Hibernate), combined with the Criteria API for dynamic querying and the MapStruct library for mapping entities to DTOs. An overview of the domain entity-relationship (ER) model is presented in Figure 2[5].

The platform's primary entity is Principal, which encapsulates a user's authentication credentials (email, password) and tracks their most recently active role. By decoupling identity from role-specific data, the model allows a single user to assume both Client and Freelancer personas within one account. The Role table defines each possible role (FREELANCER, CLIENT, ADMIN, UNVERIFIED), and the principal_role join table implements the many-to-many relationship between Principals and Roles. This separation both simplifies authentication logic and provides a foundation for future expansion to additional user types without schema changes.

[5] A larger version of the image can be found from: https://tartuulikool-my.sharepoint.com/:i:/g/personal/voievuds_ut_ee/EQtC77mdQoxEsZOC3EG6WMYBQS_leVP-bKaJ9hbOmWj8Ow?e=n3zvZi

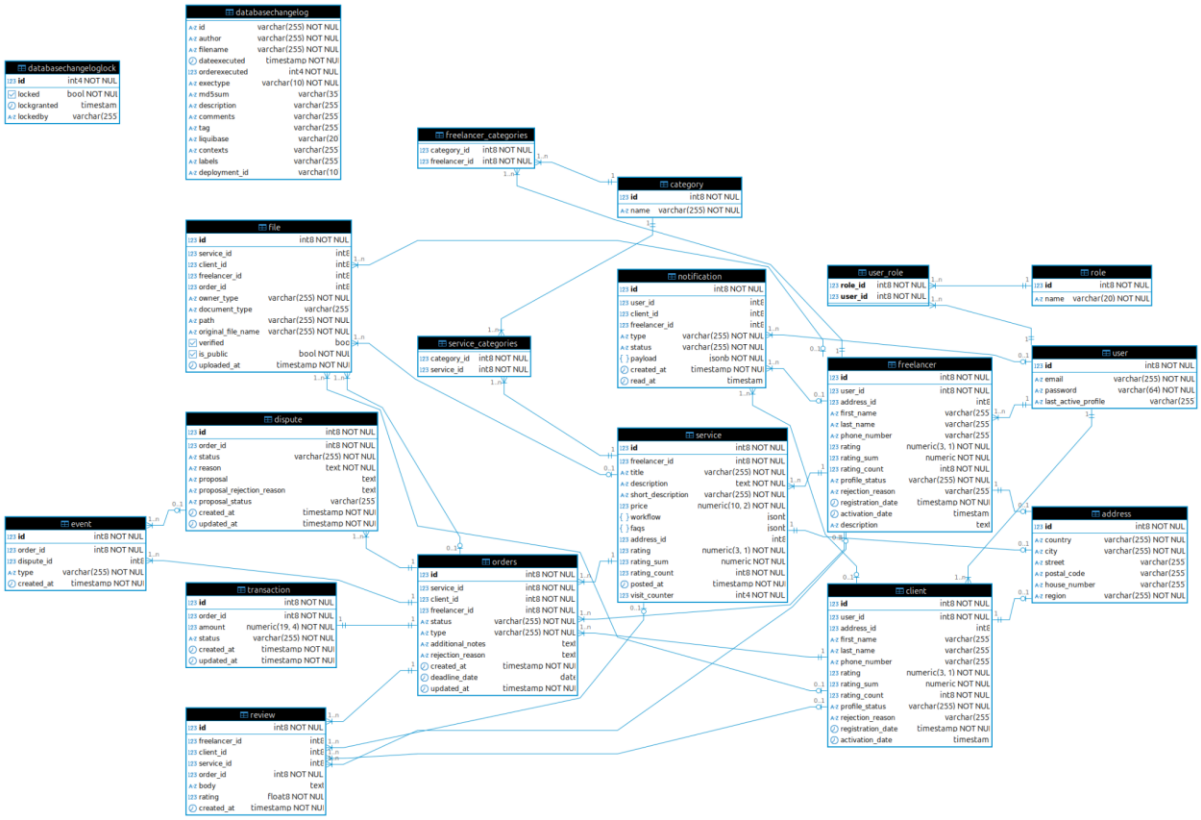

Figure 2. Domain entity-relationship of ProConnectX.

The Client and Freelancer entities extend the Principal construct to store role-specific details. Both share attributes for registration_date, activation_date, rating, and rating_count, along with an account_status field (UNVERIFIED, PENDING, ACTIVE, REJECTED) that governs access to platform features. Each also references an Address record for billing or service-delivery purposes and includes an optionall rejection_reason for manual account approvals. Freelancers carry an additional description and a link to multiple Category entries via the freelancer_categories join table, reflecting their areas of expertise.

The Service entity represents a freelancer's published offering: it includes a title, short and full descriptions (as large-object fields), a numeric price, and metadata such as posted_at, rating, and visit_counter. Services may belong to multiple skill or industry Categories through the service_categories join table, enabling faceted search and filtering. The faqs and workflow columns leverage PostgreSQL's JavaScript Object Notation Binary (JSONB) type to store variable promotional FAQs and process-definition steps, balancing schema flexibility with efficient querying. By normalising less-structured components into JSONB, the model accommodates evolving business logic without frequent migrations [34].

When a Client selects a Service, an Order record is created. Each order is linked to a client, a selected service, and a freelancer. The order progresses through a set of well-defined lifecycle stages via the status field, which captures states such as CREATED, IN_PROGRESS, SUBMITTED_FOR_REVIEW, APPROVED, DISPUTED, COMPLETED, or CANCELED.

In cases of dissatisfaction, clients may initiate a dispute tied to a specific order. This action creates a record in the dispute table, which includes references to the order in question, a selected reason, optional references to the resolution proposal and its rejection_reason, and a status field. The field possible values include: OPEN, IN_REVIEW, RESOLVED_REFUNDED, RESOLVED_FREELANCER_PAID, ADMIN_ACTION_REQUIRED, and REJECTED.

To facilitate accountability and platform transparency, the following supporting entities are employed:

- **Event Logging**: The event table captures transitions and system activities related to both orders and disputes. Each record includes a timestamp, associated entity IDs, and a type descriptor, enabling precise tracking of user actions and system responses.
- **File Management**: All user-generated documents, such as identification files, avatars, service images, order documents, and more, are stored in the file table. Each file is associated with relevant entities (client, freelancer, service, or order) and includes flags like verified and is_public to support controlled access and identity assurance. These safeguards help ensure that sensitive data, such as identification documents, remain protected from public access.
- **Notifications**: The notification entity handles asynchronous communication. Messages are generated for all roles, with delivery status tracked via a status field (NEW, UNREAD, READ, FAILED) and JSON-structured payload content to support rich interactions.
- **Review System**: Upon successful order completion, clients may leave structured feedback in the review table. Each review includes a numeric rating, an optional textual body, and references to the order, client, and service involved.

To support traceable workflows and stateful transitions across key entities, the system introduces status fields represented as enumerated types (enums). These enums encapsulate domain-specific lifecycle stages and are central to enforcing consistency in business processes.

Activation and verification lifecycle of platform participants' accounts defined by AccountsStatus:

- **UNVERIFIED**: The user has registered but hasn't made a choice what account type to choose.
- **PENDING**: The user has registered, made through corresponding to choosed role verification form and awaits administrative review.
- **APPROVED**: The account is verified and fully active.
- **REJECTED**: The registration request was denied.

OrderStatus represents the service order's progression, from initiation to closure:

- **CREATED**: The order has been placed and is awaiting freelancer response.
- **ACCEPTED**: The freelancer has accepted the order, and work has begun.
- **SUBMITTED_FOR_REVIEW**: The freelancer submitted work for client evaluation.
- **APPROVED**: The client approved the delivered work.
- **COMPLETED**: The order lifecycle is finalised.
- **DISPUTED**: The client was dissatisfied with the results of the job.
- **CANCELED**: The freelancer has declined the order.

TransactionStatus captures the state of the financial transaction linked to an order:

- **PENDING**: The transaction is initialised, but funds have not yet been secured.
- **ESCROWED**: Funds have been held in escrow pending service completion.
- **RELEASED**: Payment has been released to the freelancer after successful order completion.
- **REFUNDED**: Funds have been returned to the client, typically due to a dispute resolution.
- **CANCELED**: The transaction was voided, usually in tandem with a canceled order.

DisputeStatus defines the procedural states of a conflict registered between parties:

- **OPEN**: The dispute is newly created and pending resolution proposals.
- **IN_REVIEW**: The dispute has been escalated to administrative review.
- **RESOLVED_REFUNDED**: The client's claim was upheld and payment refunded.
- **RESOLVED_FREELANCER_PAID**: The freelancer's work was deemed satisfactory, and payment was released.

- **ADMIN_ACTION_REQUIRED:** The freelancer or client has requested administrator intervention.
- **REJECTED**: The dispute was dismissed without financial redress.

NotificationStatus tracks visibility and delivery of messages generated by the system or users:

- **NEW**: The message was persisted.
- **UNREAD**: The recipient has not yet viewed the message.
- **READ**: The message was opened by the recipient.
- **FAILED**: Message delivery was unsuccessful due to technical failure.

All database schema changes are managed using Liquibase, a version-controlled migration tool. Liquibase tracks applied migrations through the databasechangelog and databasechangeloglock tables. Migrations are primarily written in YAML, ensuring a structured approach to version control. However, for initialising essential platform data - such as roles, the admin user account, and categories - scripts are written using the GroovyDSL extension, making data initialisation more programmatic and dynamic.

The application adopts a normalised relational data model, utilising various types of associations to represent domain relationships. In cases of many-to-many relationships - such as those between freelancers and categories or services and categories - dedicated junction tables are employed to decompose the association into two one-to-many relationships. All entity relationships are explicitly defined using foreign key constraints, with appropriate cascading behaviour applied where data integrity must be enforced.

This structured approach ensures both referential integrity and long-term scalability. Furthermore, it supports seamless schema migrations across development, staging, and production environments, enhancing compatibility across application versions and deployment contexts.

To ensure robust data consistency, the system employs a layered validation strategy:

- **Database-level constraints**: including unique constraints (e.g., for email addresses), non-null columns, and referential integrity enforced by foreign keys.
- **Application-level validation**: implemented via Bean Validation (JSR 380) annotations such as @NotBlank, @NotEmpty, @NotNull, and @Email to enforce business rules before persistence operations are attempted.

6.4 Communication and Integration

The frontend and backend communicate securely over HTTPS using RESTful endpoints. Authenticated requests include JWT tokens stored in HttpOnly cookies with the Secure flag enabled to prevent access from client-side scripts and ensure transport-layer security. WebSocket sessions are likewise protected through session-based user verification and enforced Cross-Origin Resource Sharing (CORS) policies.

To support modularity and extensibility, the backend provides well-defined REST APIs, while the frontend organises its service logic using RTK Query slices. This architectural pattern promotes separation of concerns, enhances reusability, and facilitates streamlined unit testing. It also allows for a smooth future transition to alternative data-fetching paradigms, such as GraphQL, if the project requirements evolve.

6.5 Deployment and Containerization

The entire system is containerised using Docker, allowing consistent environments across development, testing, and production. Each component (frontend, backend, database) is defined in isolated containers with shared networking, orchestrated and deployed on Railway.

Environment variables and secrets, such as API keys and JWT secrets, are securely managed through Render's environment configuration, avoiding hard-coded values in the codebase.

6.6 Security Architecture

Security is a foundational aspect of the freelance marketplace application, given the presence of sensitive user data, financial transactions, and role-based workflows. The system adopts a layered security model that addresses authentication, authorisation, data access control, and auditing.

The application uses JWT (JSON Web Tokens) for stateless authentication, with tokens stored in HttpOnly cookies to protect against Cross-Site Scripting (XSS) attacks. This approach enables secure identity propagation without exposing sensitive tokens to the JavaScript runtime. Tokens are issued upon successful login and included automatically in subsequent requests.

While traditional Cross-Site Request Forgery (CSRF) protection mechanisms (such as synchroniser tokens or same-site headers) are not separately implemented, the usage of HttpOnly cookies mitigates most CSRF attack vectors by preventing direct JavaScript access to tokens.

The platform enforces fine-grained authorisation based on predefined user roles: UNVERIFIED, CLIENT, FREELANCER, and ADMIN. Access to controllers and service-layer methods is restricted via Spring Security annotations and role checks, ensuring that only authorised users can perform privileged actions.

In addition to static role checks, the backend includes ownership validation to ensure users can only interact with their own data. This is implemented via a custom @CheckOwner annotation and enforced using Aspect-Oriented Programming (AOP). Before executing sensitive operations (e.g., updating a service or completing an order), the system validates that the authenticated user is indeed the owner of the targeted resource.

To preserve platform integrity, a manual account verification process is enforced during user onboarding. After registration, users submit a verification form that is reviewed via an internal admin dashboard. Until approved, users retain the UNVERIFIED role and are restricted from engaging in client or freelancer activities. This process prevents anonymous actors from performing sensitive actions and allows platform administrators to vet users before granting access.

To protect the platform from cross-origin attacks, the CORS mechanism is explicitly configured. Only approved frontend origins are allowed to interact with the backend API. Preflight and actual request headers are validated to block unauthorised client applications.

The system maintains structured log files using the Logback framework, configured with daily rollover and a 30-day retention policy. Logs include sensitive security events such as authentication attempts, role changes, and order status transitions. These logs aid in post-incident analysis, administrative oversight, and regulatory compliance.

6.7 User Flows and Lifecycle Management

To ensure smooth interaction between various stakeholders on the platform, the application implements clearly defined user flows designed around two primary roles: Client and Freelancer. Each role experiences a distinct but interconnected lifecycle, governed by backend workflows and business logic. The platform also incorporates a robust registration and verification process, an order lifecycle with transaction management, and a review system for post-order feedback.

The platform enforces a multi-step user onboarding process to maintain quality and security:

1. **Initial Registration**: A user begins by submitting an email and password through a registration form and selects either the *Freelancer* or *Client* role,
2. **Role Selection**: Upon registration, the user retains the UNVERIFIED status and is restricted from any privileged actions.
3. **Verification Form Submission**: The system redirects the user to a role-specific verification form, which includes identity and profile data.
4. **Pending Review**: The submitted verification request receives a PENDING status and awaits manual approval by a platform administrator.
5. **Admin Review**:
    - If approved, the user gains full access to the platform as either a CLIENT or FREELANCER.
    - If rejected, the user gets notification about the declined registration request and can apply for registration.

This workflow ensures a controlled onboarding process, especially crucial in a trust-based freelancer marketplace (see Figure 3).

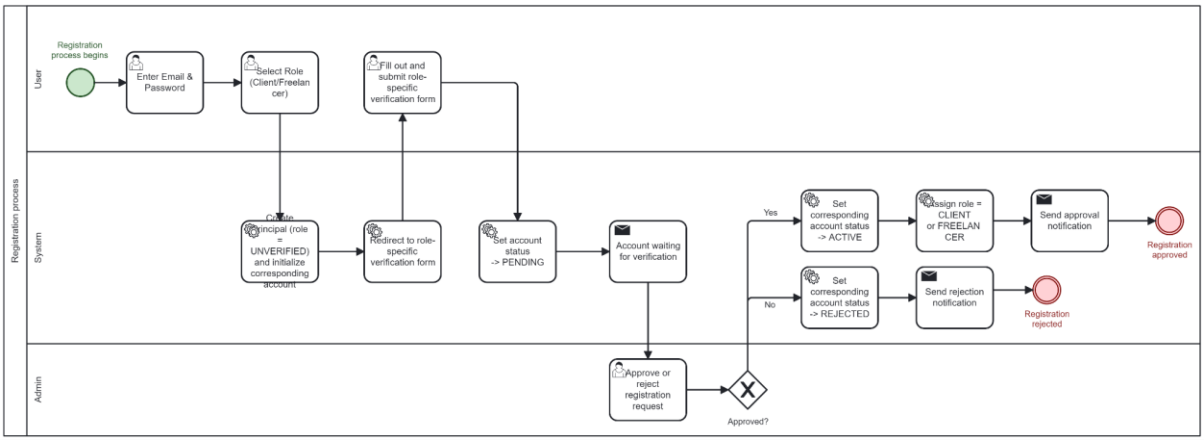

Figure 3. Process model showing a user registration process.

Once verified, clients can browse predefined services listed by freelancers and place orders as follows:

1. **Service Selection**: The client selects a predefined offering created by a freelancer.
2. **Order Placement**: The client initiates an order, optionally adding custom notes or instructions for the freelancer.
3. **Transaction Creation**: Upon order submission, a corresponding transaction entity is created with a PENDING status, storing the service price.
4. **Order Status – Pending**: The order enters a CREATED state, awaiting freelancer action.
5. **Freelancer Work**:
    a. Once the order is accepted, the freelancer performs the requested work, and the order status updates to ACCEPTED, and the transaction is moved into an escrowed state.
    b. If the freelancerncer decides to reject the order for some reason, the order status is set to REJECTED, and the transaction is discarded or voided.
6. **Completion**: The freelancer submits the completed work for review, and the order gets status SUBMITTED_FOR_REVEIW, signaling the client to review the results.
7. **Client Review**:
    a. If the client is satisfied with the results, the client marks the order as APPROVED.

    b. If dissatisfied, the client can mark the order as DISPUTED, prompting further discussion or administrative intervention. The detailed dispute user flow is described in the next section.
8. **Order Completed**: Upon approval, the system sets the order status as COMPLETED, and the corresponding transaction is released.
9. **Review Submission**: After approval, the client may leave a review.

This lifecycle ensures secure, milestone-driven collaboration and payment, with clear checkpoints for both parties.

If the client is dissatisfied with the submitted work and initiates a dispute, the platform transitions the order into a dedicated resolution flow. This process aims to ensure procedural fairness and maintain accountability for both parties. The flow follows a clear, status-driven progression:

10. **Dispute Opened:** The client marks the order as DISPUTED, prompting the system to create a Dispute entity with an initial status of OPEN. No changes occur to the escrowed transaction at this stage.
11. **Freelancer Proposal:** The freelancer may respond by submitting a resolution proposal. Dispute gets status IN_REVIEW.
12. **Client Evaluation:** Upon receiving the proposal, the client has two options:
    a. **Accept Proposal:** The dispute is resolved amicably. The system updates the dispute status to either RESOLVED_REFUNDED or RESOLVED_FREELANCER_PAID, depending on the terms agreed upon. The corresponding transaction is released accordingly - either refunded to the client or paid to the freelancer.
    b. **Reject Proposal:** The client declines the resolution offer. At this point, the dispute remains in the OPEN state. The transaction remains escrowed, and the freelancer is allowed to revise and submit a new proposal.
13. **Request Admin Resolution:** If no agreement is reached after proposals are exchanged, the client or freelancer may escalate the matter by selecting "Notify Admin", which transitions the dispute status to ADMIN_ACTION_REQUIRED. This step invokes administrative intervention.

14. **Administrator Review:** A designated platform administrator reviews all relevant materials, including submitted work, communication history, and any attached evidence. Based on the administrator's judgement, the dispute gets one of the following final statuses:
    a. **RESOLVED_REFUNDED:** If the client’s complaint is deemed valid, the transaction is refunded in full.
    b. **RESOLVED_FREELANCER_PAID:** If the administrator finds the freelancer's work satisfactory, the transaction is released to the freelancer.

This lifecycle ensures that freelancers retain control over the orders they accept while providing structured resolution pathways in cases of client dissatisfaction.

7. Testing

The evaluation of the developed system was conducted in two main areas: usability and performance. Usability was assessed using the System Usability Scale (SUS), a standardised questionnaire that measures users' perceived ease of use and overall satisfaction [35]. Performance was analysed with Google Lighthouse, an automated tool that evaluates web applications based on key user-centric metrics, including loading speed, responsiveness, and visual stability [14]. The following sections present the results of these evaluations and discuss their implications for the system.

7.1 Usability

To evaluate the usability and user experience of the developed freelance marketplace MVP, a SUS questionnaire was conducted with a group of nine participants. SUS is a widely adopted, standardised method for assessing perceived usability, offering a reliable benchmark for software products across various industries [35].

The choice of nine participants is justified by existing research indicating that a sample size of 8 to 12 respondents is generally sufficient to obtain reliable and representative SUS results. Studies have demonstrated that within this range, the usability scores tend to stabilise, offering an adequate reflection of overall user perceptions while balancing the practical constraints of time and resources [36].

Participants were asked to interact with the platform and complete the SUS questionnaire, which consists of ten statements rated on a five-point Likert scale ranging from "Strongly Disagree" to "Strongly Agree" [37]. The questions assessed aspects such as system complexity, ease of use, confidence in operation, and learnability. The SUS scoring method ensures a balanced measure of positive and negative sentiment towards the system.

The responses indicated a generally positive perception of the platform's usability. Most participants agreed that they would like to use the system frequently and found it easy to use. Statements such as "I thought the system was easy to use" and "I felt very confident using the system" received predominantly "Agree" and "Strongly Agree" responses. Additionally, the integration of various functions was viewed favourably, with multiple participants recognising the system as well-integrated and consistent.

Conversely, complexity-related concerns were minimal. Most respondents disagreed with statements like "I found the system unnecessarily complex" and "I found the system very

cumbersome to use," suggesting that the application was perceived as intuitive. The necessity for technical support was also rated low, indicating a high level of self-sufficiency among users when navigating the platform.

The calculated SUS score was 82.2 out of 100, which corresponds to an “Excellent” usability rating [35]. This result demonstrates that the MVP meets and exceeds typical usability expectations, validating its design decisions and user interaction flows.

In addition to quantitative data, participants provided qualitative feedback. One user highlighted an issue where certain service categories at the bottom of the main page were inactive, leading to dead links. Another participant suggested improving the accessibility of the “View Services” function. Currently, users are required to click “Exit Dashboard” to access service listings, which adds unnecessary complexity. The recommendation was to introduce a prominent “View Services” button directly within the dashboard interface, enhancing navigation efficiency and improving the user journey. These observations indicate minor, yet valuable, areas for improvement in future iterations.

While the conducted SUS evaluation confirms a strong usability baseline, further testing is planned for the platform’s public release, which is expected at the end of August. The primary objective of this phase will be to validate platform performance under real-world conditions, including user flows, order processing, transaction reliability, and dispute management. Additionally, user feedback will be gathered to refine the user interface and enhance the overall user experience.

The iterative feedback loop established through SUS and subsequent testing will be crucial for aligning the platform with end-user expectations, particularly among small business owners, tradespeople, and freelance professionals who represent the primary target audience.

7.2 Performance

In performance evaluations, it's essential to assess both desktop and mobile versions of a web application, as user experiences can vary significantly between these platforms. Mobile devices often have less processing power, smaller screens, and slower network connections compared to desktops, leading to different performance benchmarks. Therefore, conducting separate tests for each platform provides a more comprehensive understanding of the application's performance across various user environments.

The performance of the developed system was evaluated using Google Lighthouse 10, an industry-standard auditing tool embedded in Chromium-based browsers. To ensure consistent and unbiased results, tests were conducted in incognito mode, eliminating the influence of browser extensions, cache, and background activities. Both desktop and mobile versions of the application were assessed to capture the full spectrum of user experience across different platforms.

The results of the performance audit revealed notable discrepancies between the system's current performance and the predefined non-functional requirements, particularly in metrics associated with content loading and visual rendering speed. Specifically, the First Contentful Paint (FCP), Largest Contentful Paint (LCP), and Speed Index (SI) significantly exceeded acceptable thresholds on both desktop and mobile platforms. For desktop devices, FCP was recorded at 2.4 seconds, slightly surpassing the maximum acceptable value of 1.6 seconds, while on mobile, this metric reached an unsatisfactory 13.8 seconds, far beyond the recommended 3.0 seconds. Similarly, the LCP values were 9.7 seconds on desktop and 16.7 seconds on mobile, greatly exceeding their respective targets of 2.4 seconds and 4.0 seconds. The Speed Index followed a comparable trend, with scores of 3.5 seconds for desktop and 15.4 seconds for mobile, both falling short of their intended benchmarks.

The underlying reasons for this suboptimal performance can be attributed largely to the nature of React-based Single Page Applications (SPAs). React applications typically rely on extensive client-side rendering, where the browser must download, parse, and execute substantial amounts of JavaScript before any meaningful content becomes visible. This architectural choice inherently delays the First Contentful Paint and Largest Contentful Paint, as the Document Object Model (DOM) is constructed only after the initial bundle execution. Furthermore, the absence of Server-Side Rendering (SSR) or Static Site Generation (SSG) exacerbates this issue by requiring the client to handle the entire rendering workload, leading to prolonged loading times, especially on devices with limited computational resources, such as mobile phones [38].

Additional contributing factors include inefficient resource loading strategies, such as the lack of code splitting, resulting in the entire application bundle being delivered upfront, irrespective of its immediate necessity. Similarly, unoptimised static assets, including large images and fonts, can delay critical rendering paths, further impacting FCP, LCP, and Speed Index values. The observed performance bottlenecks are thus not platform-specific but rather stem from

shared architectural and optimisation challenges that affect both desktop and mobile environments [39].

Despite these shortcomings in rendering-related metrics, the system performed remarkably well in terms of interaction responsiveness and visual stability. The Total Blocking Time (TBT) was measured at 0 milliseconds on desktop and 100 milliseconds on mobile, both of which are well below the respective thresholds of 350 milliseconds and 600 milliseconds. These results indicate that, after the initial rendering phase, the application maintains a highly responsive main thread, allowing it to handle user interactions promptly. Similarly, the Cumulative Layout Shift (CLS) scored 0.051 on desktop and 0.0 on mobile, reflecting excellent visual stability with minimal unexpected layout shifts during loading. These positive outcomes suggest that, while initial rendering performance requires improvement, the application's runtime behaviour and user interface consistency are already aligned with industry best practices.

In order to address the identified performance deficits, several optimisation strategies should be considered. The implementation of Server-Side Rendering (SSR) or Static Site Generation (SSG) would significantly reduce time-to-first-paint metrics by offloading rendering tasks to the server and delivering pre-rendered HTML content to the client. Moreover, employing code splitting, lazy loading, and tree shaking techniques would reduce the initial JavaScript payload, expediting load times and improving user-perceived performance. Further enhancements could be achieved through the optimisation of static assets, including the adoption of next-generation image formats and the use of resource prioritisation techniques, such as inlining critical CSS and preloading essential resources. Finally, minimising render-blocking scripts and stylesheets would help streamline the critical rendering path, further enhancing FCP, LCP, and SI scores.

The complete Lighthouse audit reports, detailing both desktop and mobile performance evaluations, are available in the appendix of this thesis.

8. Further developments

While the MVP fulfils the core platform requirements, several advanced features and architectural improvements are planned to ensure the system remains scalable, secure, and user-friendly as it evolves. These planned enhancements can be grouped into three main categories: feature expansion, architectural evolution, and user experience refinement.

Firstly, the focus is on expanding platform functionality and improving user interaction. A third user role — Employer — alongside the existing Freelancer and Client roles. This change addresses a limitation of the MVP, where freelancers could only offer predefined services and passively wait for client orders. By introducing the Employer role, sole traders and SMEs will gain the ability to actively search for and apply to client-posted job offers, similar to traditional job marketplaces. This enhancement increases platform flexibility and inclusivity, catering to professionals who prefer bidding-based models over fixed service listings. Furthermore, it encourages higher platform engagement by enabling freelancers to proactively seek new opportunities rather than relying solely on client discovery.

To streamline onboarding and strengthen security, OAuth 2.0 integration is planned. This will allow users to authenticate via trusted providers such as Google or LinkedIn, simplifying account creation and offloading sensitive credential handling to specialised third-party identity services. In addition, the platform will evolve toward real-time interaction through the integration of WebSocket-based messaging, facilitating more immediate communication and collaboration between users. An AI-powered chatbot will also be introduced to handle frequently asked questions, guide new users through onboarding processes, and provide around-the-clock support, thereby improving platform accessibility and responsiveness. Moreover, smart upselling features will suggest context-relevant premium services or promotional boosts based on user behaviour and platform activity. These recommendations will be designed to appear at strategic moments — such as during checkout or after service completion — ensuring they add value without disrupting the user experience. Together with a shift toward a more event-driven design, these improvements will support the platform's evolution toward greater automation, responsiveness, and scalability.

Secondly, the architecture will evolve to support scalability and modularity. When it comes to managing the lifecycle of platform entities, it is important to emphasise that currently, state transitions are encoded as static logic within enums. However, the author intends to migrate to

a dedicated state-machine framework to realise fully event-driven workflows. At the same time, the architecture will evolve toward an asynchronous, microservices model, wherein responsibilities such as file uploads, data exports, and notification dispatch are decoupled from synchronous request handling through Spring's @Async mechanism and an Apache Kafka event bus. Kafka will not only continue to underpin existing email notifications but will serve as the primary conduit for all future messaging channels, including in-app alerts, mobile push notifications, and potential SMS or webhook integrations. By partitioning the monolith into independently deployable services that subscribe to Kafka topics, it is possible to achieve clear separation of concerns, horizontal scalability, and parallel team development. This event-driven, microservice-oriented strategy will streamline the introduction of new features, such as automated auditing, real-time UI updates, and robust retry or dead-letter handling, while preserving domain invariants and ensuring the platform can gracefully accommodate increasing load and complexity [40].

To support these changes and maintain system stability, the automated testing strategy will be significantly expanded. In addition to unit, integration, and end-to-end tests, contract testing will be introduced to ensure service interfaces remain consistent across deployments. Chaos and load testing will help verify the resilience and scalability of asynchronous and distributed components. These tests, integrated into a CI/CD pipeline, will ensure that performance benchmarks and security standards are consistently met with every release.

Finally, ongoing improvements to the user interface will aim to deliver a more intuitive, responsive experience. As the backend becomes increasingly event-driven and modular, the frontend will benefit from real-time updates, better feedback mechanisms, and mobile-optimised layouts. In the future, some components - such as the dashboard - may be split into standalone applications to allow independent deployment and specialised frontend technologies, further enhancing user experience and maintainability.

Summary

This thesis presents the design and implementation of an MVP for a secure and scalable freelance marketplace tailored to the Estonian market. The platform addresses key industry challenges, including market fragmentation, lack of trust mechanisms, and limited growth channels for small businesses and independent professionals. The proposed solution offers a unified environment where clients can discover services, engage freelancers through escrow-secured orders, and leave reviews, while service providers can manage offerings and resolve disputes.

ProConnectX is designed to overcome the limitations of existing platforms like GoWorkaBit and Vabakutselised by offering a modern, flexible approach aligned with the demands of today's freelance and gig economy. A key innovation is the implementation of two distinct user roles - Freelancer and Client - managed under a single account with seamless in-session switching. The project also incorporates modern UI principles and a responsive design. Combined with a free-to-use model, these features aim to position ProConnectX as a comprehensive, user-centred, and scalable alternative in the Estonian freelance marketplace.

From a technical standpoint, the application was developed using a modern full-stack architecture, combining Spring Boot on the backend and React with Redux Toolkit on the frontend. Core technologies include JPA with the Criteria API for persistence, PostgreSQL as the relational database, RESTful APIs for client-server communication, and Aspect-Oriented Programming (AOP) for managing cross-cutting concerns. For notifications, the system leverages Apache Kafka as its event bus to orchestrate email notifications via Thymeleaf templates. Particular emphasis was placed on a flexible data model that supports multi-role accounts with dynamic in-session switching between freelancer and client workflows.

During development, architectural decisions were carefully made to enforce secure state transitions and ensure data integrity. These were implemented using structured enums and centralised security policies. The MVP includes essential features such as order placement, dispute resolution, user reviews, and role-based interfaces.

Although the current version meets MVP requirements, this thesis outlines several directions for future development. These include the introduction of a dedicated Employer role, integration of OAuth 2.0 authentication, implementation of AI chatbot support and real-time

messaging, development of smart upselling strategies, and a shift toward microservices architecture. Further enhancements such as a more comprehensive automated testing framework and the adoption of state machines for workflow management are also proposed to increase system robustness, auditability, and long-term maintainability.

Overall, this work has delivered a functional MVP that validates the core concept of a localized, escrow-backed freelance marketplace for Estonia's SMEs and independent professionals. By implementing essential features - role-based workflows, escrowed orders, dispute resolution, and in-session role switching - on a modern Spring Boot/React architecture, the platform addresses key challenges of trust, fragmentation, and growth barriers within the local market. While this prototype intentionally limits its scope to establish feasibility and gather feedback, it also embeds an ongoing cycle of iterative testing and stakeholder input. This ensures that each release remains closely aligned with the real-world needs of Estonia's digital economy.

References

[1] “DocsRoom - European Commission.” Accessed: May 08, 2025. [Online]. Available: https://ec.europa.eu/docsroom/documents/60561

[2] “World Bank SME Finance,” World Bank. Accessed: May 08, 2025. [Online]. Available: https://www.worldbank.org/en/topic/smefinance

[3] “Business demography statistics.” Accessed: May 08, 2025. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Business_demography_statistics

[4] “(PDF) The Importance Of SMEs On World Economies,” in *ResearchGate*, doi: 10.36880/C11.02265.

[5] “Estonia Country Report 2024,” European Comission. Accessed: May 08, 2025. [Online]. Available: https://economy-finance.ec.europa.eu/document/download/34698aa7-9514-41bc-9c64-e97fb70a741e_en?filename=SWD_2024_606_1_EN_Estonia.pdf

[6] O. Kaya, “The impact of late payments on SMEs’ access to finance: Evidence from credit rationing and loan terms,” *Economic Modelling*, vol. 141, p. 106896, Dec. 2024, doi: 10.1016/j.econmod.2024.106896.

[7] D. Stokes, S. A. Syed, and W. Lomax, “Shaping up word of mouth marketing strategy: the case of an independent health club,” *Journal of Research in Marketing and Entrepreneurship*, vol. 4, no. 2, pp. 119–133, Jan. 2002, doi: 10.1108/14715200280001468.

[8] P. Centobelli, R. Cerchione, E. Esposito, and M. Raffa, “Digital Marketing in Small and Medium Enterprises: The Impact of Web-Based Technologies,” *Advanced Science Letters*, vol. 22, no. 5–6, pp. 1473–1476, May 2016, doi: 10.1166/asl.2016.6648.

[9] M. te Velde, “How is fair pay perceived? Qualitative research into gig workers working for online labor platforms.” Accessed: May 15, 2025. [Online]. Available: https://essay.utwente.nl/97800/

[10] “(PDF) Compendious research of Escrow Payment - Focusing on Future Considerations, Trends and Applications,” ResearchGate, doi: 10.24018/ejbmr.2020.5.4.347.

[11] *1. The essential software requirement*. Accessed: Feb. 05, 2025. [Online]. Available: https://learning.oreilly.com/library/view/software-requirements-3rd/9780735679658/ch01.html

[12] “What is role-based access control (RBAC)?” Accessed: Apr. 11, 2024. [Online]. Available: https://www.redhat.com/en/topics/security/what-is-role-based-access-control

[13] “Requirements: Functional vs. Non-functional | Baeldung on Computer Science.” Accessed: Apr. 11, 2024. [Online]. Available: https://www.baeldung.com/cs/requirements-functional-vs-non-functional

[14] “Lighthouse performance scoring | Chrome for Developers.” Accessed: Apr. 11, 2024. [Online]. Available: https://developer.chrome.com/docs/lighthouse/performance/performance-scoring

[15] G. Mohamad, “Web Applications – Basic concepts,” 2023. [Online]. Available: https://courses.cs.ut.ee/2023/WAD/fall/Main/Lectures?action=download&upname=Web%20Applications%20%E2%80%93%20Basic%20concepts.pdf

[16] H. M. Abdullah and A. M. Zeki, “Frontend and Backend Web Technologies in Social Networking Sites: Facebook as an Example,” in *2014 3rd International Conference on Advanced Computer Science Applications and Technologies*, Dec. 2014, pp. 85–89. doi: 10.1109/ACSAT.2014.22.

[17] “Databases in Web Application Development | Ramotion Branding Agency,” Web Design, UI/UX, Branding, and App Development Blog. Accessed: Dec. 04, 2023. [Online]. Available: https://www.ramotion.com/blog/database-in-web-app-development/

[18] K. Jacksi and S. Abass, “Development History Of The World Wide Web,” *International Journal of Scientific & Technology Research*, vol. 8, pp. 75–79, Sep. 2019.

[19] “Stack Overflow Developer Survey 2023,” Stack Overflow. Accessed: Dec. 04, 2023. [Online]. Available: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023

[20] J. Tong, R. R. Jikson, and A. A. S. Gunawan, “Comparative Performance Analysis of Javascript Frontend Web Frameworks,” in *2023 3rd International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS)*, Aug. 2023, pp. 81–86. doi: 10.1109/ICE3IS59323.2023.10335250.

[21] R. Vyas, “Comparative Analysis on Front-End Frameworks for Web Applications,” *IJRASET*, vol. 10, no. 7, pp. 298–307, Jul. 2022, doi: 10.22214/ijraset.2022.45260.

[22] “Documentation - TypeScript for Java/C# Programmers.” Accessed: Apr. 12, 2024. [Online]. Available: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-oop.html

[23] “Using TypeScript – React.” Accessed: Apr. 12, 2024. [Online]. Available: https://react.dev/learn/typescript

[24] D. Choma, K. Chwaleba, and M. Dzieńkowski, “THE EFFICIENCY AND RELIABILITY OF BACKEND TECHNOLOGIES: EXPRESS, DJANGO, AND

SPRING BOOT," *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, vol. 13, no. 4, Art. no. 4, Dec. 2023, doi: 10.35784/iapgos.4279.

[25] "Difference between Spring and Spring Boot," GeeksforGeeks. Accessed: Jan. 08, 2024. [Online]. Available: https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot/

[26] "Spring Data JPA," Spring Data JPA. Accessed: Apr. 10, 2024. [Online]. Available: https://spring.io/projects/spring-data-jpa

[27] "Jakarta Persistence Explained | The Eclipse Foundation," Jakarta® EE: The New Home of Cloud Native Java. Accessed: Apr. 10, 2024. [Online]. Available: https://jakarta.ee/learn/specification-guides/persistence-explained/

[28] "PostgreSQL: About." Accessed: Apr. 10, 2024. [Online]. Available: https://www.postgresql.org/about/

[29] "Material Design," Material Design. Accessed: May 08, 2025. [Online]. Available: https://m2.material.io/design/layout/spacing-methods.html#baseline-grid

[30] "React Typography component - Material UI." Accessed: May 08, 2025. [Online]. Available: https://mui.com/material-ui/react-typography/

[31] "Material Design," Material Design. Accessed: May 08, 2025. [Online]. Available: https://m2.material.io/design/interaction/gestures.html#types-of-gestures

[32] P.-J. Editor, P. Khandelwal, and N. Chaudhary, "The Psychology of Colors in UI/UX Design," *PRATIBODH*, no. NCDSNS, Art. no. NCDSNS, 2023, Accessed: May 08, 2025. [Online]. Available: https://pratibodh.org/index.php/pratibodh/article/view/154

[33] "Typography - Material UI." Accessed: May 08, 2025. [Online]. Available: https://mui.com/material-ui/customization/typography/#responsive-font-sizes

[34] "What is JSONB in PostgreSQL?," GeeksforGeeks. Accessed: May 08, 2025. [Online]. Available: https://www.geeksforgeeks.org/what-is-jsonb-in-postgresql/

[35] "JUS_Brooke_February_2013.pdf," *Usability Professionals' Association*, vol. 8, no. 2, pp. 29–40.

[36] T. S. Tullis and J. N. Stetson, "A Comparison of Questionnaires for Assessing Website Usability".

[37] I. E. Allen and C. A. Seaman, "Likert Scales and Data Analyses," pp. 64–65.

[38] S. Mondal, "Enhancing React Application Performance: Proven Strategies and Best Practices," vol. 11, no. 12, 2024.

[39] F. Pavić and L. Brkić, "Methods of Improving and Optimizing React Web-applications,"

in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, Sep. 2021, pp. 1753–1758. doi: 10.23919/MIPRO52101.2021.9596762.

[40] B. Bejeck, *Kafka Streams in Action, Second Edition: Event-driven Applications and Microservices*. Simon and Schuster, 2024.

Appendices

Lighthouse audit reports

Desktop: https://tartuulikool-my.sharepoint.com/:u:/g/personal/voievuds_ut_ee/EV-w7zO9G0tPguj-nYaMtrkBhx2MCbLxLecauOhX-SXCYA?e=UQtNlW\

Mobile: https://tartuulikool-my.sharepoint.com/:u:/g/personal/voievuds_ut_ee/EejF96e7HGpPu-CvHVW6HvEBATn9xBftKfTGSqxIhiyXNQ?e=zw2dej

Licence

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Mykyta Voievudskyi ,
(author´s name)

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

ProConnectX: A Web Application for Freelance Marketplace ,
(title of thesis)

supervised by Lidia Feklistova ;
(supervisor´s name)

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;
3. am aware of the fact that the author retains the rights specified in points 1 and 2;
4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mykyta Voievudskyi
15/05/2025