Polytechnic Institute of Leiria
School of Technology and Management
Computer Engineering Department
Master degree Computer Engineering – Mobile Computing

ROGERHR INTERNSHIP

FRANCISCO SIMPLICIO MELÍCIAS

Leiria, March 2025

Polytechnic Institute of Leiria
School of Technology and Management
Computer Engineering Department
Master degree Computer Engineering – Mobile Computing

ROGERHR INTERNSHIP

FRANCISCO SIMPLICIO MELÍCIAS
Number: 2222860

Internship Report under the supervision of Professor Doutor Leonel Santos (leonel.santos@ipleiria.pt).

Leiria, March 2025

ORIGINALITY AND COPYRIGHT

This internship report is original, made only for this purpose, and all authors whose studies and publications were used to complete it are duly acknowledged. Partial reproduction of this document is authorized, provided that the Author is explicitly mentioned, as well as the study cycle, Master degree in Computer Engineering – Mobile Computing, 2023/2024 academic year, of the School of Technology and Management of the Polytechnic Institute of Leiria, and the date of the public presentation of this work.

DEDICATION

This thesis is dedicated to all the professors who supported me throughout my master's journey, with a special thanks to Professor Leonel for his invaluable guidance and encouragement.

I also want to express my gratitude to my friends from Leiria, who have become like family, and to my friends around Portugal, whose unwavering support has been a constant source of strength. A heartfelt thank you goes to my girlfriend, who has been by my side during this major step in my life, especially as I made the move to Switzerland.

Finally, I extend my deepest thanks to Damien, Elias, and Romain for believing in me and giving me the opportunity to intern at RogerHR, a company filled with great people that brought me to Switzerland, a place I now call home.

ABSTRACT

This report presents the work conducted during an internship at RogerHR, a startup dedicated to revolutionizing HR workflows through innovative software solutions. Conducted as part of a Master's in Computer Engineering – Mobile Computing at the Polytechnic Institute of Leiria, the internship was focused on enhancing RogerHR's platform using Java Spring and React. The contributions were mainly focused on time management functionalities, timesheets, shift planning, and leave management, as well as optimizing code quality, refactoring core components, and introducing testing frameworks.

Transitioning from academic projects to a production-level codebase posed challenges, particularly in balancing rapid feature development with system stability in a fast-paced startup environment. Relocating to Switzerland added cultural and linguistic difficulties, which ultimately strengthened adaptability and collaboration skills. Over nine months, responsibilities expanded from backend development to full-stack responsibilities, contributing to frontend restructuring, CI/CD pipelines, and security best practices. Regular code reviews and Agile workflows refined the ability to write clean, scalable code.

Structured into five chapters, this thesis explores RogerHR's technical ecosystem, the contributions made to its HR platform, and the professional growth fostered by overcoming real-world challenges in a dynamic startup environment.

Keywords: Full-stack, Human resources, Internship, Startup, Web development

TABLE OF CONTENTS

LIST OF FIGURES

ACRONYMS LIST

API	Application programming interface.
BE	Backend.
CRM	Customer Relationship Management.
CRUD	Create, read, update, delete.
CTO	Chief technology officer.
DDD	Domain driven design.
DOM	Document Object Model.
FE	Frontend.
FS	Full stack.
HR	Human resource.
HTML	Hypertext markup language.
IDE	Integrated development environment.
ORM	Object-relational mapping.
PR	Pull request.
QA	Quality assurance.
SQL	Structured Query Language.

UI User interface.

UX User experience.

XP Extreme programming.

1

INTRODUCTION

RogerHR is committed to transforming human resources (HR) workflows through innovative software solutions, addressing inefficiencies in the HR tech market. This full-stack (FS) development internship centered on enhancing and expanding the RogerHR platform within a dynamic startup environment. The experience presented numerous technical challenges and growth opportunities, reinforced by a team deeply passionate about their product and its impact.

During this internship, the primary objectives are to develop both backend (BE) and frontend (FE) components, work on the public API, and contribute to creating new modules. A key focus is on integrating seamlessly with the team, adapting to established workflows, and collaborating effectively in a fast-paced development environment. These goals offer exposure to a wide range of technical and teamwork responsibilities, supporting personal growth and meaningful contributions to the platform.

My academic background in software development has prepared me for diverse projects, but integrating into an established team with a mature codebase presents new challenges. Although university taught problem-solving and rapid learning, working on a production-level application introduces complexities that require adaptability. There is some apprehension about keeping pace with a startup where features and improvements are continually released, and moving to Switzerland adds its own challenges, such as language barriers and cultural adaptation. Nonetheless, I anticipate growing in confidence as communication and teamwork skills develop alongside more efficient development practices.

During the internship, the focus is primarily on BE development using Java Spring and FE development using React, addressing tasks like new feature implementation, code optimization, and refactoring to improve maintainability and performance, especially within time management functionalities. Most of the work will be related to time management functionalities, including timesheets, shift planning, and leave management.

This thesis explores the key aspects of the internship, detailing the technologies, methodologies, and challenges encountered throughout the experience. It also high-

lights the impact of the contributions made to the system and how this experience has shaped growth as a developer.

The rest of this thesis is organized into four core chapters. Chapter 2 introduces RogerHR, its HR systems, and competitive positioning. Chapter 3 explores the methodologies, tools, and testing practices central to the technical work. Chapter 4 examines the project contributions through its modular architecture, detailing challenges addressed in each component and their collective impact on company workflows. Finally, Chapter 5 synthesizes the professional development outcomes, key lessons learned, and their alignment with career objectives in software engineering.

2

CHARACTERIZATION OF THE HOST INSTITUTION

RogerHR, a startup founded in September 2020 by Romain Beker and Elias Mirza in Vaud, Switzerland, identified a market gap for a Customer Relationship Management (CRM) to monitor a company's HR health. Their goal was to "offer better human relations" through innovative HR software, addressing the stagnation in the existing market where older companies struggled to evolve. As Elias and Romain explained in an interview (*RogerHR, la plateforme RH qui écoute les équipes* n.d.): "Existing tools are designed for the administrative side of HR. But we've reversed the paradigm and designed a platform for managers and their staff, which changes the whole logic. Not only is the tool more intuitive, ergonomic and accessible on smartphones, but above all it meets the strategic need to keep teams satisfied and involved. The real strategic value is to modernize the management approach. Today, even more so after the Covid, employees, especially Generation Z and millennials, expect a different way of working. Companies know there's a problem, but often do not know how to address it."

By December 2021, RogerHR had secured its first customers. However, during its initial development phase, the platform faced significant technical challenges that hampered its scalability and performance. Recognizing the need for a robust codebase, the team brought on Chief Technology Officer (CTO) Damien Bouclier in early 2022. Damien spearheaded a comprehensive refactor of the platform, resolving foundational issues and establishing a solid framework for future growth. This marked a turning point for the startup, enabling the launch of a more reliable and efficient product. By summer 2022, RogerHR began marketing its SaaS platform in Romandy (French-speaking Switzerland) and now serves over 140 customers across various sectors. As a Swiss company, RogerHR initially focuses on the Swiss market, aligning with the country's specific HR regulations and practices. The strategy is clear: gain market share in Switzerland.

Currently, the team (Figure 1) comprises of three salespeople and six developers, including the CTO. Jessy serves as the FE tech lead, with Pierre and Rabie as FE developers. Antoine is the BE developer, and Damien is the FS/devOps tech lead. The team is also focused on introducing innovative functionalities, simplifying

Figure 1: RogerHR team

HR processes, and enhancing the sales pipeline. With plans to integrate Artificial Intelligence and leverage cutting edge technology, RogerHR aims to stay ahead of the curve in the rapidly evolving HR landscape.

In this chapter, the key aspects of the HR system will be explored further, detailing the methodologies behind RogerHR's development, as well as an analysis of the competitors to better understand the market landscape and opportunities for product differentiation.

2.1 HUMAN RESOURCE SYSTEM

HR is at the heart of managing an organization's most valuable resource - its people. It will do so by overseeing employee data, supporting career growth, and ensuring compliance with labor laws and company policies. With a growing business, managing these aspects efficiently becomes increasingly complex, requiring robust

tools to streamline operations and maintain accuracy. An HR system is designed to simplify and automate people management by organizing employee data, enhancing communication between HR and staff, tracking work hours and absences, and delivering valuable insights through reports. The ultimate goal is to reduce the administrative burden on HR teams, enabling them to focus on strategies that foster a thriving workplace, rather than wasting time on compiling information.

At RogerHR, the mission is to deliver an HR software solution that is both easy to use and highly effective. The system caters primarily to medium-sized companies with 100 to 5,000 employees, although it scales seamlessly to suit organizations of any size. As mentioned earlier, the main focus remains the swiss market, tailoring the platform to meet the specific HR regulations and requirements unique to the region. Prioritizing HR functionalities ensures that RogerHR excels in managing people, schedules, and operational data. Unlike all-in-one platforms that attempt to handle everything from payroll to finance, RogerHR specializes only in HR. When external services are required, such as payroll management, integrations with other services are implemented, enabling the clients to leverage specialized tools without compromising efficiency.

RogerHR is built with a commitment to simplicity and speed. The interface is modern, intuitive, and continually refined to make even complex HR tasks straightforward. Clients appreciate the fresh, simple User Interface (UI) that eliminates the need for lengthy tutorials or steep learning curves. As a startup,the focus is on delivering new features and improvements quickly. The well-structured codebase allows faster building deployment of functionalities compared to many competitors. The platform is robustly tested, minimizing the risk of significant bugs. While minor edge cases or misunderstood feature requests can arise, the key advantage lies in the speed of the company's customer support, ensuring client satisfaction. Integration with external systems is also straightforward. If there's a high demand for connecting to a service, especially one that supports client needs or sales, the development of the necessary connections is prioritized.

Client feedback drives the development process, but how competitors handle similar use cases is also actively analyzed, by examining both industry leaders and newer players, it helps identify what works well and what falls short. This allows a refinement of RogerHR to provide an experience that is not only effective but also more intuitive and modern. While working to improve every aspect of the platform, the focus remains on the target audience, medium-sized companies, ensuring the system is not overcomplicated to cater to massive enterprise clients with vastly different needs.

The commitment is to simplify workflows, make processes clearer, and maintain faster development cycles while delivering a fresh, modern design. By blending insights from client feedback and competitor analysis, RogerHR is continuously shaped into a platform that is user-friendly, adaptable, and a step ahead of the market.

2.2 WORK METHODOLOGY

At Roger, the workflow balances the agility of a startup with structured processes to ensure quality and scalability. Task and ticket prioritization is led by Elias and Damien, based on client needs or commitments to deliver specific functionalities by a given date. This alignment between client expectations and internal goals enables the team to maintain focus and deliver results efficiently.

Each day begins with a stand-up meeting where the team discusses what was accomplished the previous day, shares any arising problems, and highlights recent changes in the code. This transparency ensures everyone is aligned and informed about ongoing tasks. Twice a month, a dedicated "refinement" session is held to break down and plan larger functionalities, allowing the entire team to provide input and collaborate on the best implementation strategies.

Typically, the day begins by reviewing assigned tickets, prioritizing high-urgency tasks such as bug fixes or production patches. Tasks are then addressed in order of importance. Each sprint typically includes one or two larger tickets per developer, focused on creating new functionalities or improving existing ones through refactoring. In addition to coding, time is allocated daily to review all opened Pull Requests (PR). This serves two purposes, staying up to date with code changes across the project and offering constructive feedback to colleagues when needed. This practice enhances understanding of the system and contributes to maintaining a consistent standard of quality across the team.

While all team members are encouraged to contribute across all modules, some naturally develop deeper expertise in specific modules. This specialization ensures a thorough understanding where it is most needed, while the team as a whole maintains flexibility and adaptability. To support this, consistent coding guidelines and practices are adhered to throughout the codebase. As a result, even if someone has not previously worked on a particular module, they can quickly familiarize themselves with it.

Rigorous testing is prioritized to maintain high code quality. Unit tests are written for new features and bug fixes, and on the FE, component tests are added to validate UI behavior. This approach minimizes bugs, fosters accountability, and ensures a stable, scalable foundation.

Elias and Luca, the Quality Assurance (QA) leads, handle most of the testing, but all team members thoroughly test their work. Staging and test environments are utilized, containing complex datasets that help identify potential real-world issues before deployment to production. If something is not executed properly, such as a feature malfunctioning or missing its intended purpose, the QA team will return the ticket, highlight the issues, and include steps to reproduce the problem. This allows the issue to be tracked down and resolved effectively.

As a startup, the foundational codebase is continuously refined to overcome early limitations and optimize performance. This proactive approach not only paves the way for new features but also ensures the system scales efficiently as the data grows. Handling vast volumes of HR data, such as shifts, timesheets, and leaves, demands robust performance, so the focus remains on raw performance optimization rather than relying on caching as a default.

To stay ahead, there is a bi-weekly "council" meeting where team members share new tools and ideas, sparking innovation and continuous learning. Additionally, Slack facilitates day-to-day communication, enabling quick sharing of tips, discoveries, and discussions about challenges.

The culture of collaboration, continuous learning, and commitment to high-quality code enables quick adaptation to evolving client needs. Every challenge becomes an opportunity to refine processes, ensuring the system remains agile, efficient, and prepared to meet future demands.

2.3 COMPETITORS

The HR software market is highly competitive due to the limited availability of effective solutions. In Figure 2, RogerHR is positioned at the top right, reflecting the strengths in modern technology and the delivery of a near-ideal solution. Some competitors may excel in specific areas, particularly when catering to enterprise-scale companies like Nestlé, which have over 50,000 employees. However, RogerHR has deliberately chosen a different path, focusing on small to medium-sized businesses, especially those with up to 5,000 employees. The SaaS model ensures simplicity

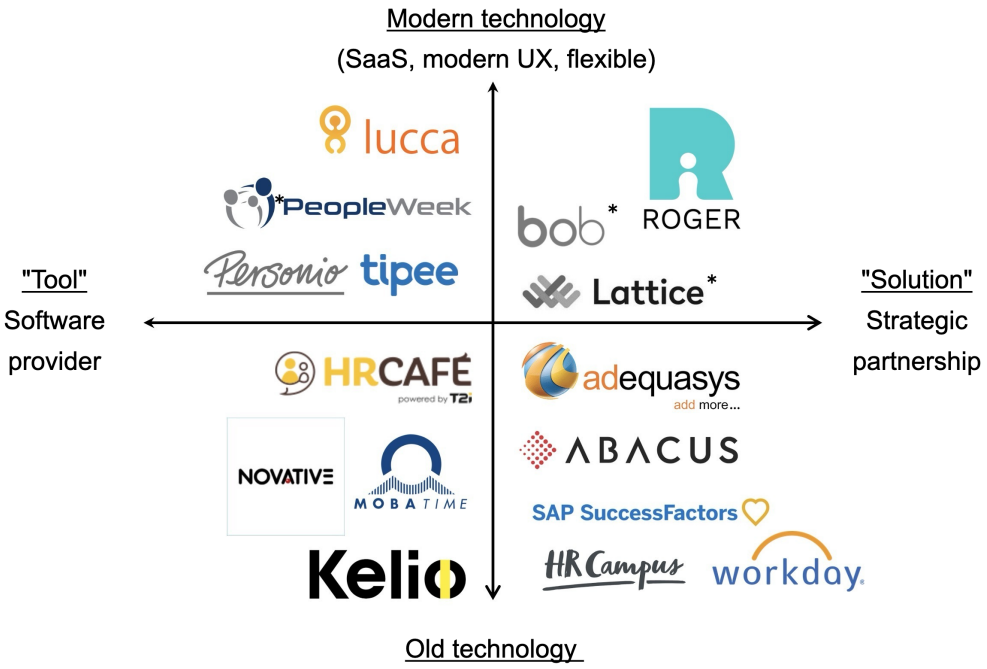

Figure 2: Competitors

and scalability, providing every client with the same product while maintaining the flexibility to customize and evolve rapidly.

This competitive analysis was conducted entirely in-house, with a detailed evaluation of market trends, product offerings, and technological approaches. At RogerHR, continuous competitor analysis is an integral part of the strategy, with regular assessments taking place every few weeks. This ongoing process ensures that the platform remains innovative and aligned with industry advancements while identifying opportunities to enhance its competitive positioning.

A major differentiator for RogerHR is its Swiss-centric approach. Unlike global competitors, the platform is currently designed exclusively for the Swiss market. This focus ensures adherence to Swiss labor regulations, particularly in areas like time management and compliance, keeping clients fully aligned with local requirements. Additionally, all data is stored securely within Switzerland, a critical factor for Swiss companies that prioritize data sovereignty and security. Many businesses in the region are hesitant to work with software providers hosting data abroad, giving RogerHR a significant advantage in this market.

While RogerHR does not claim to excel in every category, its strengths lie in modern design, a rapidly evolving platform, and exceptional user experience (UX). A comparison with key competitors highlights this positioning.

Lucca, for example, specializes in time management, online payslip distribution, and wage bill monitoring. While these tools are valuable, they lack the breadth of features RogerHR offers, making Lucca more of a niche solution.

Workday, a highly regarded US-based provider, delivers robust solutions for enterprise clients. However, its older technology and focus on large organizations often make it a poor fit for small to medium-sized businesses. RogerHR's lightweight, agile design is better suited for smaller companies looking for simplicity and speed without compromising functionality.

HRCafe, a Swiss-made and Swiss-hosted competitor, has strong local credentials but struggles with an outdated UX and feels more like a tool than a full solution. While they excel in time management, RogerHR offers a more balanced, user-friendly platform that combines compliance with modern design.

HiBob, one of the most formidable competitors in the market, benefits from over 10 years of experience, a large development team, and a strong international presence. Their platform includes features that surpass the current offerings in some areas. However, RogerHR's UX is more intuitive, and the modular architecture enables efficient development of required features. With additional time and resources, the gap can be closed, positioning RogerHR to surpass competitors in the future.

RogerHR's strength lies not only in its technology but also in its strong relationships with clients. As a startup, the company benefits from a unique edge, as clients work closely with its leadership, fostering a connection that is difficult for larger competitors to replicate. This proximity enables quick adaptation to client needs, delivering powerful solutions that often exceed expectations. While clients may initially have modest expectations due to the company's size, they are frequently impressed by the achievements within short timeframes. This combination of agility, responsiveness, and technical capability has resulted in high client satisfaction and loyalty. The startup culture fosters trust and provides personalized service that larger competitors often struggle to deliver.

3

METHODOLOGIES AND TOOLS

In this chapter, the primary methodologies and technologies used at RogerHR will be explored in detail. The discussion begins with an outline of the methodologies guiding the development process, followed by an overview of the key technologies employed. Next, the essential tools integral to the daily workflow will be examined, and finally, the comprehensive testing technologies will be detailed. Many of these tools and platforms were entirely new or significantly different from prior experience, playing a critical role in shaping the internship at RogerHR.

3.1 METHODOLOGIES

Methodologies form the core of the software development process at RogerHR. The approach draws inspiration from the Agile Manifesto, introduced in 2001 by 17 pioneering technologists (*Manifesto for Agile Software Development* n.d.), which emphasizes putting people and their interactions ahead of processes, delivering functional software over exhaustive documentation, collaborating closely with customers rather than rigid contract negotiations, and embracing flexible planning instead of adhering to strict schedules.

As a small, dynamic team, elements of Extreme Programming and Scrum are blended to suit specific needs. Extreme Programming encourages individual developer autonomy and a strong focus on code quality, while Scrum provides a structured framework for project management and fosters effective teamwork. Although neither methodology is followed to the letter, this hybrid approach ensures agility and responsiveness to changing requirements, enabling consistent delivery of innovative solutions at RogerHR.

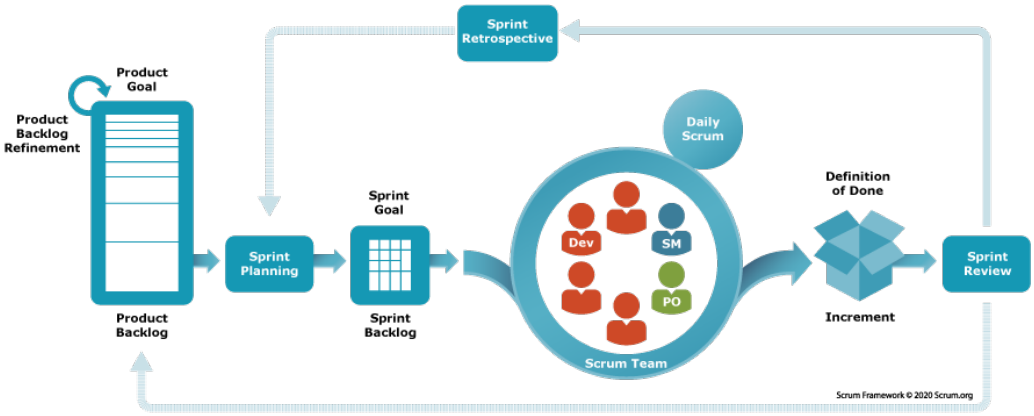

Figure 3: Scrum loop (*What is Scrum? | Scrum.org* n.d.)

3.1.1 *Scrum*

Scrum is a well-liked framework for complicated software and product development in agile project management. It offers an organized method for teams to work together productively and adjust to modifications in the project specifications.

Under Scrum (Schwaber and Sutherland, 2010), work is divided into sprints, which are fixed-length iterations which last two to four weeks on average. The team chooses a set of tasks from the product backlog, a prioritized list of features and requirements, at the start of each sprint during the planning meeting. The team strives to accomplish these tasks during the sprint, with the goal of delivering a possibly shippable product increment at the sprint's conclusion.

Scrum teams have all the abilities required to finish the task inside the sprint, since they are cross-functional and self-organizing. The group gets together for daily scrums, also known as stand-up meetings, to organize the day's tasks, discuss progress, and identify any roadblocks.

At the conclusion of each sprint, the team conducts a sprint review to present the finished product to stakeholders and gather valuable feedback, which is then used to update the product backlog and prioritize future development. Following the review, a sprint retrospective is held to evaluate the workflow and identify areas for improvement. For a visual representation of this complete Scrum cycle, please refer to Figure 3, which illustrates the entire loop from planning to review and retrospective.

With the help of Scrum's adaptable framework, which encourages iterative development, frequent inspections, and adaptation, teams are able to produce high-quality products quickly and effectively in response to shifting demands and feedback from clients.

3.1.2 *Extreme programming*

The agile software development framework known as Extreme Programming (XP) (Shrivastava et al., 2021) aims to generate high-quality software quickly. It is ideal for small teams, usually consisting of no more than 20 people, and prioritizes cooperative teamwork over individual accountability. In the effort to develop a lightweight process model, XP pushes the boundaries of conventional programming techniques.

XP is one of the most particular agile frameworks in software engineering because its main focus is on acceptable engineering practices. Teams can use it to develop software iteratively, producing regular releases that are customized to meet the needs of customers. XP contributes to the gradual enhancement of software quality by placing a strong emphasis on ongoing improvement and participation by users.

Some of the key principles and practices (Shrivastava et al., 2021) of XP are continuous feedback through pair programming and frequent releases, simplicity in code and process, incremental design that evolves with the project, pair programming for better code quality, test-driven development to meet requirements, continuous integration for early issue detection, and having an on-site customer for immediate feedback. These principles enhance collaboration, adaptability, and code quality in software development, which also goes along with RogerHR principles.

3.2 TECNOLOGIES

This section provides an overview of the core technologies that powered project development during the internship at RogerHR. A modern set of frameworks, libraries, and tools was combined to build a scalable, efficient, and user-friendly application. From the React ecosystem on the FE to robust BE solutions, each technology was carefully selected to address specific challenges and drive rapid innovation.

The integration and optimization of these technologies to support various functionalities will be explained, ensuring a seamless UX and high performance. Additionally, the strategic choices in development tools, testing frameworks, and continuous integration will be highlighted, demonstrating how they contributed to the overall success of the project, meeting both client needs and industry standards.

3.2.1 *PostgreSQL*

Originally created in the 1980s at the University of California, Berkeley (Stonebraker and Rowe, 1986), PostgreSQL has grown into a potent, Structured Query Language (SQL) based relational database management system available as open source software. With features usually only seen in commercial products, it is widely regarded as one of the most sophisticated open-source database systems in the world (Drake and Worsley, 2002). A global team of open-source developers actively maintains it. Also, accessible as a free, open-source version is PostgreSQL (*PostgreSQL: The world's most advanced open source database* n.d.).

Some of PostgreSQL the key features (Drake and Worsley, 2002) that made Damien select this tecnology are:

- Transaction Integrity: Explicit control options and reliable transaction support are provided via ACID (Atomicity, Consistency, Isolation, Durability) compliance.
- Advanced Querying: Powerful querying capabilities including recursive queries, common table expressions, user-defined functions, triggers and stored procedures in their own language, PL/pgSQL and/or others, such as, PERL and Python.
- Search and Indexing: Efficient full-text search and indexing for enhanced query performance.
- Replication and Availability: It offers robust replication methods that provide data redundancy, fault tolerance, and high availability in distributed contexts, while its effective full-text search and indexing significantly enhance query performance.
- Security Measures: Robust security features with encryption, authentication methods, and access control mechanisms.
- Data Integrity and Reliability: Preserves data integrity by allowing referential integrity through foreign regulations, enforcing constraints, and verifying data

types. It also provides write-ahead recording and crash recovery mechanisms to lessen the risk of data loss.

In terms of scalability, PostgreSQL offers features such as table partitioning and parallel query execution to handle large volumes of data and high concurrent workloads. It also provides various replication and clustering options for achieving high availability and fault tolerance, including synchronous replication and automatic failover.

PostgreSQL is a popular choice for developing applications that need strong data management capabilities because of its overall combination of performance, flexibility, and dependability (Smith, 2010). Whether it's handling transactional workloads, analytical queries, or real-time data processing, PostgreSQL provides a solid foundation for building scalable and resilient database solutions.

3.2.2 *Java spring*

Java is a widely adopted platform and programming language that underpins much of today's digital infrastructure. Its reliability, adaptability, and "write once, run anywhere" philosophy have made it indispensable for developing applications that run on any device with a Java Virtual Machine. With its object-oriented design, robust typing, and extensive standard library, Java supports a diverse range of applications, from desktop and mobile software to enterprise systems, ensuring its continued relevance in an ever-evolving technological landscape (*What is Java and why do I need it?* N.d.).

In 2003, Rod Johnson introduced Spring to simplify Java development by addressing common challenges such as configuration management, tight coupling, and excessive boilerplate code (*spring-reference.pdf* n.d.). Initially emphasizing dependency injection and aspect-oriented programming, Spring quickly evolved into a comprehensive ecosystem that provides solutions for web development, data access, security, and more, further enhancing Java's flexibility and efficiency in building modern applications.

The Spring Framework provides an extensive collection of tools and frameworks that enable developers to create reliable, scalable, and maintainable applications. Since its launch, Spring has developed into a well-developed ecosystem with a large selection of modules that address numerous aspects of programs development. Spring offers a comprehensive framework that makes enterprise application development

less complicated. It does this by offering specialized components for data access, web development, and security in addition to core elements like aspect-oriented programming and dependency injection (*Spring | Projects* n.d.).

A powerful and all-inclusive framework, Spring Security, was created to handle the complex issues of authorization, authentication, and general security in Java applications. It provides an extensive range of tools and capabilities to protect apps from different security risks and weaknesses. This contains support for a number of authentication techniques, including form-based authentication, HTTP basic authentication, and authentication using third-party systems like OAuth (Open Authorization) or LDAP (Lightweight Directory Access Protocol).

To sum everything up, Java Spring is a pillar of the Java development community, providing a stable and all-encompassing framework that streamlines the creation of applications while improving security, scalability, and maintainability. With its extensive feature set and array of tools, Spring gives developers the confidence to take on challenging tasks, facilitating the development of creative and robust applications. Java Spring continues to be at the forefront of modern software development, thanks to its constant innovation and persistent devotion to simplicity and flexibility. This allows developers to realize their vision in the always evolving digital landscape and propels technical developments.

3.2.2.1 *Hibernate and JOOQ*

Hibernate and JOOQ are both popular frameworks used in Java development for database access and manipulation. Although they both have the same goal of interacting with databases, they do so in distinct ways.

Hibernate is an object-relational mapping (ORM) framework that maps Java objects to database tables, allowing developers to work with databases using object-oriented paradigms. Developers can concentrate on working with Java objects instead of low level database operations, since it abstracts away the specifics of SQL queries and database interactions. Hibernate offers a handy method for carrying out Create, Read, Update, Delete (CRUD) actions on database entities by automatically generating SQL queries based on Java classes and annotations. To increase performance and maintainability, it also provides capabilities like transaction management, lazy loading, and caching. (*Your relational data. Objectively. - Hibernate ORM* n.d.)

However, jOOQ adopts a different strategy by offering a type-safe and intuitive Application Programming interface (API) for creating SQL queries straight within Java programs. jOOQ embraces SQL rather than abstracting it away, enabling programmers to write SQL queries using Java syntax while guaranteeing type safety and compile-time checks. Using fluent API methods, developers can create SQL queries that are transformed into native SQL queries at runtime. jOOQ creates Java classes that represent database tables and fields. With this method, developers have complete control over the SQL queries and can modify and optimize them in accordance with the needs of particular databases (*jOOQ: The easiest way to write SQL in Java* n.d.).

While Hibernate offers a more abstracted and convenient approach to performing CRUD operations, making it easier to save and update data, this convenience sometimes comes at the expense of performance, especially when handling large datasets or complex relationships. In contrast, jOOQ gives developers precise control over SQL queries, leading to significant performance gains when fetching data, though it does require a more complex setup and a deeper understanding of SQL. Ultimately, the choice between Hibernate and jOOQ depends on the specific needs of the application: Hibernate is ideal for simpler, more routine data manipulations, while jOOQ is better suited for scenarios that demand optimized data retrieval and the handling of intricate query logic.

3.2.3 *JavaScript and TypeScript*

JavaScript is a versatile, high-level programming language. It allows the developers to work with HyperText Markup Language (HTML) components, react to user input, and retrieve data from servers in an asynchronous way, websites can be made more functional and interactive. Because of its extensive compatibility with all major web browsers and seamless interaction with HTML and Cascading Style Sheets (CSS), it is an essential resource for developing websites and applications. Furthermore, JavaScript's flexibility goes beyond the browser thanks to server-side frameworks like Node.js, which let programmers create FS apps with the language (*JavaScript | MDN* n.d.).

Although JavaScript is incredibly powerful, it is missing multiple features, which is why TypeScript emerged. TypeScript is a superset of JavaScript that combines the syntax of JavaScript with extra type use guidelines. TypeScript maintains JavaScript's runtime behavior in spite of its strict error detection, making the

switch between the two languages easier. Notable are the features that JavaScript does not provide (such as interfaces, enums, generics, and declarations), which provide developers more control over the structure and readability of their code. Moreover, TypeScript compiles code with great care, removing type information to guarantee compatibility with pre-existing JavaScript environments without adding extra runtime libraries. Another great advantage of Typescript is the static type checking, which, strengthens code stability by detecting problems prior to execution based on the types of values involved (*TypeScript: The starting point for learning TypeScript* n.d.).

3.2.4 *React*

React (*React* n.d.) is an open-source JavaScript library developed by Facebook for building UI's for web applications. Its primary goal is to enable developers to create dynamic, interactive components that automatically update when data changes. By leveraging a virtual Document Object Model (DOM), React minimizes direct DOM manipulations, which enhances performance, while its declarative programming style simplifies the process of defining and managing complex UIs.

Moreover, React's component-based architecture promotes the development of reusable and modular UI elements, streamlining both construction and maintenance of scalable applications. It is frequently combined with libraries like Redux for state management and React Router for client-side routing, contributing to a robust and versatile development ecosystem. React's effectiveness, adaptability, and strong community support have made it one of the most popular choices in modern web development.

3.2.4.1 *FE libraries (MUI, AGGrid, DayPilot)*

At RogerHR, several key FE libraries are relied upon to build a modern, responsive interface that meets design and functional requirements. Among these, Material UI (MUI), AG-Grid, and DayPilot play essential roles.

MUI (*MUI: The React component library you always wanted* n.d.) is a React-based UI library built on Google's Material Design principles. It provides a comprehensive set of pre-built, customizable components that ensure a consistent look and feel across the application, while also accelerating development. By leveraging MUI, a polished and intuitive UX is delivered, aligning with contemporary design standards.

AG-Grid (*AG Grid: High-Performance React Grid, Angular Grid, JavaScript Grid* n.d.) serves as the go-to solution for handling large volumes of data. This powerful grid component supports advanced features such as sorting, filtering, and pagination, making it ideal for displaying complex datasets like employee timesheets, schedules, and reports. Its performance and flexibility enable users to navigate and interact with data smoothly, even under heavy loads.

DayPilot (*DayPilot - HTML5 Calendar, Scheduler and Gantt Chart Web Components* n.d.) is a scheduling and calendar library used to create interactive planning interfaces. It offers a robust set of tools for building dynamic calendars and scheduling views, enabling the design of responsive planning pages. With DayPilot, real-time updates can be implemented, and the scheduling interface can be tailored to the unique demands of HR planning, simplifying coordination of shifts and other time-related activities for managers and employees.

3.2.5 *Git and GitHub*

Git (**GitWhatis**) is a distributed version control system that enables developers to collaboratively manage and track code changes while preserving the complete project history. By allowing each team member to clone a repository, make local changes, and commit those changes with descriptive messages, Git fosters an environment of efficient, parallel development. Branching is one of its key strengths; developers can work on new features or bug fixes independently on separate branches and then merge those changes back into the main branch, ensuring smooth integration without disrupting ongoing work.

At RogerHR, Git is used in conjunction with GitHub (*GitHub · Build and ship software on a single, collaborative platform · GitHub* 2025) as the primary version control system. GitHub not only serves as a central repository but also integrates with Jira: when a ticket number is included in any merge request or commit, Jira automatically updates the corresponding ticket with details of the PR and the number of commits associated with it. This seamless integration enhances traceability and streamlines the project management process.

3.3 TOOLS

In this chapter, the key tools that played a crucial role in supporting development, design, documentation, and error identification processes during the internship will be discussed. Tools such as Jira, Figma, Docker, IntelliJ, BugSnag, and Swagger have proven invaluable in streamlining tasks, enhancing productivity, and improving overall project effectiveness. In the following subsections, each tool is presented in detail, highlighting its specific contribution to the workflow and the impact it had on the output.

3.3.1 *Jira*

Jira, developed by Atlassian, is a flexible project management tool that RogerHR relies on for planning, tracking, and managing tasks throughout the development process. Originally designed for Agile teams, Jira has evolved to support various methodologies, enabling efficient work organization.

RogerHR uses Jira to manage sprints. The workflow within a sprint follows five defined phases: "To Do" includes tasks that have been identified and prioritized but not yet started; "In Progress" captures tasks that team members are actively working on; "Ready for QA" marks tasks that have been completed by developers and are awaiting testing; "Accepted for Release" comprises tasks that have passed testing and are approved for deployment; and "Released" contains tasks that have been successfully deployed to users, typically at the end of a sprint or during a patch release. This structured approach ensures the team remains focused, accountable, and able to deliver value consistently.

3.3.2 *Figma*

Figma (*Figma: The Collaborative Interface Design Tool* n.d.) is a cloud-based design and prototyping tool widely used for creating interactive prototypes and UIs. Its intuitive interface and real-time collaboration features make it an essential tool for both designers and developers.

One of Figma's standout features is its support for vector editing, allowing designers to easily create and modify shapes, icons, and images. Additionally,

its component-based design system promotes the reuse of UI elements, ensuring consistency across the application.

At RogerHR, Figma plays a crucial role in the development process. The design team, led by Elias, prepares detailed designs for each task and attaches them directly to the corresponding Jira tickets. Developers use these designs to accurately replicate the UI in the application. This workflow minimizes ambiguity, ensures design consistency, and accelerates implementation while maintaining high-quality standards.

3.3.3 *Docker*

Another crucial tool employed during the internship was Docker (*Docker: Accelerated Container Application Development* n.d.), an open-source platform that automates the deployment of applications within software containers. Docker containers ensure that an application runs smoothly in a variety of settings, from development to production, by packaging up software and all of its dependencies into a single, standardized unit.

Docker encapsulates applications within lightweight containers, making the process of designing, deploying, and managing applications easier. To ensure consistency and get rid of environment-related problems, these containers contain all the necessary code, runtime, system tools, libraries, and settings for the software to function. Containerization, portability, efficiency, scalability and consistency between the production environment and all the developers local environments.

At RogerHR, Docker is used to mimic the production environment locally, specifically the localstack and PostgreSQL database. This ensures all developers operate in the same environment, improving testing and reducing dependency issues.

3.3.4 *IntellIJ*

IntelliJ IDEA (JetBrains, 2021) is a powerful integrated development environment (IDE) widely used for Java and other programming languages. Known for its advanced code intelligence, refactoring capabilities, and robust debugging tools, IntelliJ streamlines the entire development process and enhances productivity for professional developers.

At RogerHR, IntelliJ serves as the primary IDE. The configuration is standardized across the team to ensure consistent code formatting and quality. Essential plugins, such as Copilot, are used to boost productivity by automating repetitive tasks and suggesting code patterns or boilerplate, reducing manual effort in repetitive structures like mappers or utility functions. Additionally, the Save Actions plugin automatically formats code on save, and integration with SonarQube enforces strict coding rules, resulting in a clean and maintainable codebase.

3.3.5 *BugSnag*

BugSnag (*App Monitoring, Error Tracking & Real User Monitoring | Insight Hub* n.d.) is an error monitoring and reporting tool designed to capture and aggregate runtime errors in real time. It provides detailed error reports and stack traces, enabling developers to quickly diagnose and resolve issues as they occur. Its support for multiple environments makes BugSnag an invaluable tool for maintaining high application stability.

At RogerHR, BugSnag is configured separately for production and testing environments, ensuring that errors specific to each context are efficiently tracked. We also integrate BugSnag with Slack so that our team receives real-time notifications with complete error traces from both the FE and BE. Significant time was dedicated to analyzing these error reports, creating tickets, and resolving the root causes of issues. This proactive approach not only enables rapid problem resolution but also enhances understanding of recurring bugs, ultimately improving overall system reliability.

At RogerHR, BugSnag is configured separately for production and testing environments, ensuring errors specific to each context are efficiently tracked. BugSnag is also integrated with Slack, enabling the team to receive real-time notifications with complete error traces from both the FE and BE. Significant time is dedicated to analyzing these error reports, creating tickets, and often addressing the root causes of the issues. This proactive approach not only facilitates rapid problem resolution but also deepens the understanding of recurring bugs, ultimately contributing to the overall reliability of the system.

3.3.6 *Swagger*

Swagger *API Documentation & Design Tools for Teams | Swagger* n.d. is an open-source framework for designing, building, and documenting RESTful APIs. It allows developers to automatically generate interactive API documentation that simplifies understanding and testing of API endpoints by both internal teams and external partners.

At RogerHR, Swagger is used to document both our customer API and our public API. This automated documentation process ensures that every endpoint is clearly described and remains up-to-date. By leveraging Swagger, our external developers and internal teams can quickly grasp the functionality and structure of our APIs, which facilitates smoother integrations and more efficient collaboration.

3.4 TESTS

Testing represents a crucial component of the software development lifecycle at RogerHR, ensuring application quality, reliability, and performance. The comprehensive testing strategy enables early issue identification, improved code stability, and seamless user experiences. This chapter examines the key testing tools and frameworks employed at RogerHR, including JUnit, Cypress, and SonarQube.

3.4.1 *JUnit*

JUnit (*JUnit 5* n.d.) is an open-source framework for Java that allows developers to write and execute automated tests. Widely regarded as the standard for unit testing in the Java ecosystem, JUnit offers a simple syntax with annotations and a rich set of assertion methods. Its tight integration with build tools like Maven and Gradle further streamlines the testing process, enabling seamless incorporation into continuous integration pipelines.

At RogerHR, JUnit is a cornerstone of our testing strategy. Comprehensive test coverage through unit and integration testing verifies correct system behavior. By identifying bugs early in the development cycle and minimizing regressions, JUnit helps maintain high code quality standards. The tests are run automatically as part of our CI/CD workflow, providing rapid feedback and supporting our commitment to continuous improvement.

3.4.2 *Cypress*

Cypress (*Testing Frameworks for Javascript | Write, Run, Debug | Cypress* n.d.) is a modern, open-source end-to-end testing framework built specifically for web applications. It offers a robust environment that runs directly in the browser, allowing developers to write tests in JavaScript/TypreScript that simulate real user interactions. One of its advantages is its automatic waiting and real-time reloading capabilities, which significantly simplify the process of writing, debugging, and maintaining tests. This intuitive framework has quickly become a favorite for ensuring that web interfaces behave reliably under various conditions.

At RogerHR, Cypress now plays a crucial role in verifying the functionality and responsiveness of our FE components. Cypress is only used to create component tests that mimic user behavior. This proactive testing approach enables us to identify issues early and ensure a seamless UX across different browsers and devices. Integrating Cypress into our development process allowed us to catch regressions quickly, maintaining the overall stability and performance of our platform.

3.4.3 *Sonar*

SonarQube (*Code Quality, Security & Static Analysis Tool with SonarQube | Sonar* n.d.) is an open-source platform for continuous code quality inspection that helps developers detect bugs, security vulnerabilities, and code smells in real time. By providing detailed feedback on code health and enforcing coding standards, SonarQube plays a vital role in maintaining a clean, maintainable codebase. Its seamless integration with our CI/CD pipeline ensures that code quality issues are promptly addressed, contributing to the robustness and reliability of our applications.

RogerHR employs SonarQube for continuous code monitoring and pull request report generation, which systematically guides code review processes. The platform enforces strict quality standards, including a 70% minimum test coverage requirement for BE code prior to approval. FE standards maintain greater flexibility, reflecting the subsequent implementation of testing protocols during the internship period.SonarQube has been instrumental in highlighting issues and code smells, allowing our team to systematically refactor and improve our codebase whenever possible.

4

INTERNSHIP PROGRAMME

This chapter provides a thorough summary of the internship experience, covering everything from the first day to the end. It is broken into the following three major sections.

In the first section, Initial state, the onboarding experience, the application's initial state, and the early days of adjusting to the codebase and company culture are described. This includes the training sessions, the first tasks, and the challenges encountered.

The second section, Architecture, delves into the core aspects and strategic decisions behind the architecture of the BE and FE. This part covers the design principles, the technology stack, and the frameworks used, along with a discussion of how these decisions support the overall performance, scalability, and maintainability of the system.

The final section, Modules and Contributions, provides a detailed analysis of the project's different modules. This part focuses on the contributions made, outlining the features developed, the problems resolved, and the collaborative efforts with team members. Each module is discussed in terms of its functionality, significance to the project, and the involvement in improving and refining its components.

4.1 INITIAL STATE

The first day was dedicated to setting up the computer and installing the necessary software. This included configuring the IDE for code analysis and formatting. Afterward, Elias provided a user-focused tour of the application, offering an understanding of its features, UX, and how the work would fit into the bigger picture.

The next day, Damien explained the application from a developer's perspective. The BE architecture, public APIs, and external integrations were discussed, highlighting how the different components interact. Code conventions and the importance of writing clean and maintainable code were also covered. Damien recommended

Clean Code by Robert C. Martin (Martin and Coplien, 2009), which aligns with many of the practices followed.

The database architecture and the system for managing multiple environments across development, testing, and production were then reviewed. The rest of the week was spent exploring the app and identifying potential issues, providing a solid understanding of its functionality and areas for improvement.

The application, being an HR platform, has many interrelated parts and underlying logic. While certain elements may appear straightforward at first, their underlying complexity often makes them challenging to fully understand. As a result, the first week was particularly demanding as the function and goal of each element of the system were explored.

During this time, a few minor issues were addressed to become familiar with the existing codebase. It is important to note that, as a startup, the codebase was undergoing significant changes when the internship began. Specifically, large portions of the FE were being actively refactored to enhance the developer experience.

The original plan for the internship focused on BE development, however, initial tasks involved fixing minor FE issues. Over time, the scope expanded to include both FE and BE tasks, leading to more FS responsibilities.

4.2 ARCHITECTURE

In this subchapter the entire architecture of the application will be explained, covering both the FE and BE. It will in particular explain how each part is organized to provide a solid foundation that is easy to expand to meet new requirements.

The application employs a multi-tenant database architecture, meaning all clients share the same database while maintaining segregation through a tenantId column in every table. As only one database is needed for each environment (e.g., staging, test, and production), this strategy simplifies management. Consequently, our architecture is designed with this structure in mind to ensure scalability and maintainability.

Additionally, connectors were developed to enable integration with other systems. These interfaces allow leveraging pre-existing platforms for features that are not currently prioritized for in-house development or management. For instance, connections to TimeMoto, a clock-in/out device, and Opale, an administrative enterprise resource planning system specialized in the healthcare sector, were implemented.

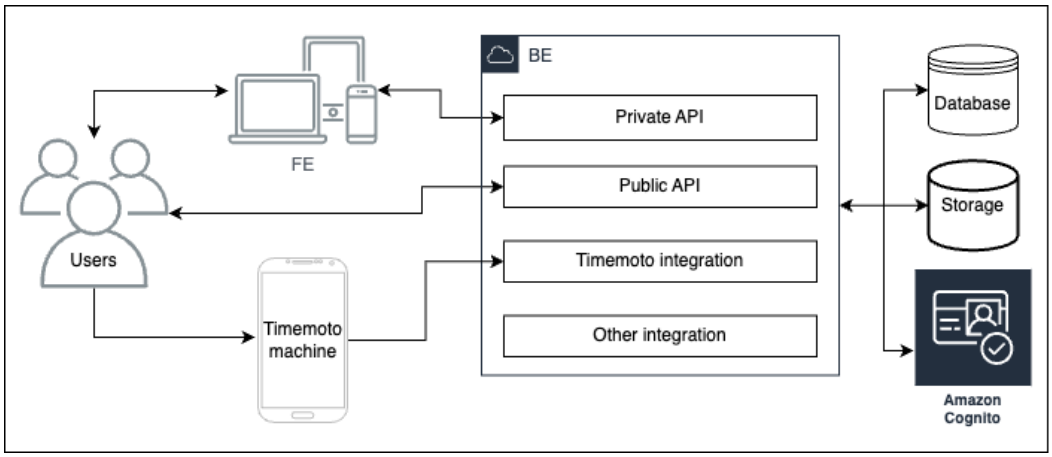

Figure 4: Architecture overview

These integrations extend the system's capabilities while maintaining a focus on core features.

As displayed in Figure 4, the system consists of the following main elements:

- **FE**: This is the UI that most users interact with. It communicates with the BE to retrieve and display the data.
- **BE**: The BE is divided into three main parts:
  - **Private API**: This is only used to communicates with the FE, handling internal data requests and business logic.
  - **Public API**: This exposes specific endpoints for clients who need access to certain functionalities or data. Not all clients/realms have access to this API.
  - **External Integration's**: This component manages connections with third-party services. An example of such integration is TimeMoto, which is used by a limited number of users.
- The BE connects to several critical components:
  - **Database**: Stores and manages application data.
  - **Storage Server**: Handles file storage and retrieval.
  - **Amazon Cognito**: Manages user authentication and authorization.

The application is structured to be robust, efficient, and scalable, ensuring seamless adaptation to new integrations and evolving requirements. This flexible design positions the system for continuous growth, making future expansions straightforward and efficient.

4.2.1 *BE architecture*

The BE has continually evolved and adapted to meet current requirements while anticipating future growth. A scalable architecture is employed to accommodate both present needs and anticipated expansion. As new functionalities are regularly added, maintaining a well-structured and extensible architecture ensures that new services can be developed and integrated efficiently.

To achieve this, a combination of three architectural patterns is utilized: Domain-Driven Design (DDD), Hexagonal Architecture, and Onion Architecture. Herberto Graça provides an excellent explanation of this approach on his website (Graça, n.d.).

These three architectures can be defined as follows:

- DDD: DDD introduces the concept of bounded contexts, which break down a large software system into smaller, more manageable contexts, each with its own models, rules, and ubiquitous language. By separating logic for each model, DDD makes it significantly easier to maintain and enhance both the code and business logic.
- Hexagon: Also known as Ports and Adapters, Hexagonal Architecture enables interaction between users or external systems and the application via clearly defined ports and adapters. This abstraction layer protects the core application logic from external tools and technologies, ensuring minimal disruption during changes. For instance, the database engine transition required minimal code modifications, enabling a straightforward and efficient migration.
- Onion: Built around a domain model, Onion Architecture connects layers through interfaces, keeping external dependencies on the outermost layers. The core is formed by domain entities and business rules. A practical example of its utility is when we migrated from Hibernate to JOOQ for data access; we only needed to adjust the domain layer without impacting other parts of the application.

The system's architecture presented by Herberto Graça is illustrated in Figure 5. This design separates the system into three core layers: the UI, the Business Logic (Application Core), and Infrastructure Code. The Application Core, central to the architecture, contains the essential business logic and interacts with the UI and infrastructure through clearly defined ports and adapters.

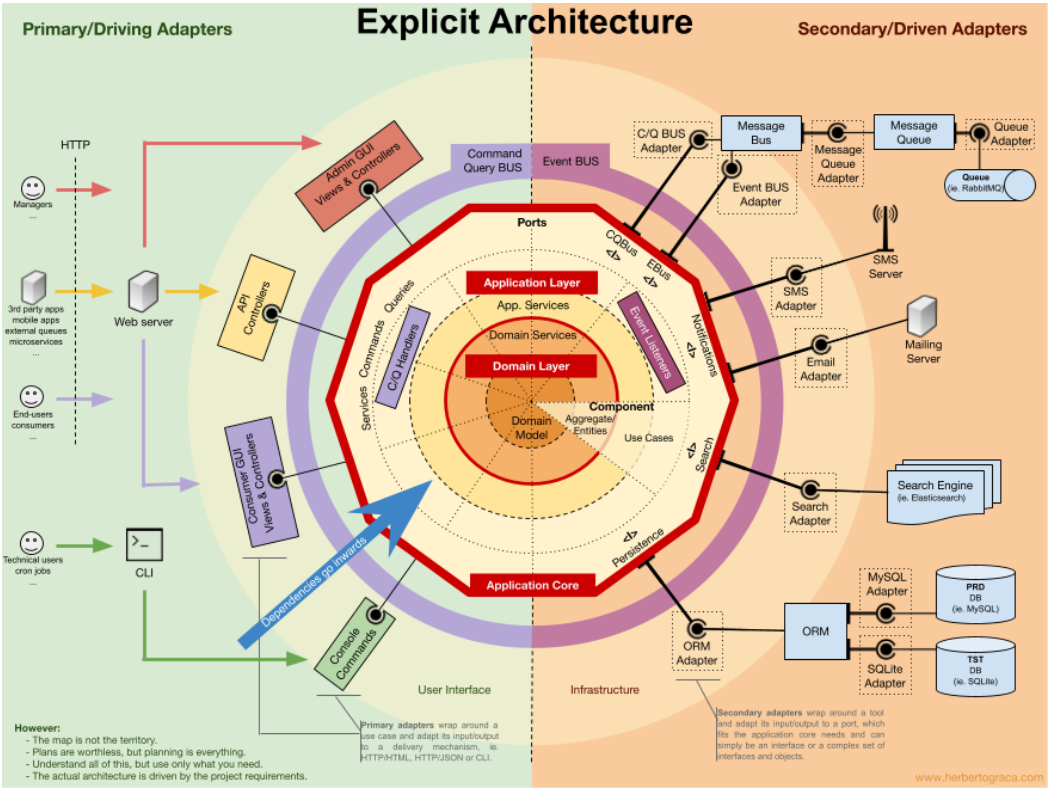

Figure 5: DDD, hexagonal, onion architecture (Graça, n.d.)

Each of the layers can be split even into more layers to make it even easier to change this up without breaking everything that is built. These interfaces ensure the core remains decoupled from external tools like databases and third-party APIs, enhancing maintainability and scalability.

Components are structured by features, such as Authentication and Billing, and are decoupled using principles like Dependency Injection and Dependency Inversion. Communication between components occurs via event dispatchers or HTTP calls facilitated by a discovery service, ensuring seamless interaction. The system uses a combination of primary (driving) adapters, which translate external inputs into actions for the core, and secondary (driven) adapters, which implement interfaces to communicate with external tools. This modular approach allows the architecture to adapt efficiently to changes, ensuring robust and scalable system growth.

At RogerHR, the BE architecture is structured into six primary folders: data, core, service, api, messaging, and public-api. Additionally, separate folders are maintained for each integration, such as timemoto and opale.

Figure 6 provides an overview of how the various BE components interact and work together. The architecture begins with the controller layer, which acts as the entry point for handling incoming requests. Its responsibilities include receiving and

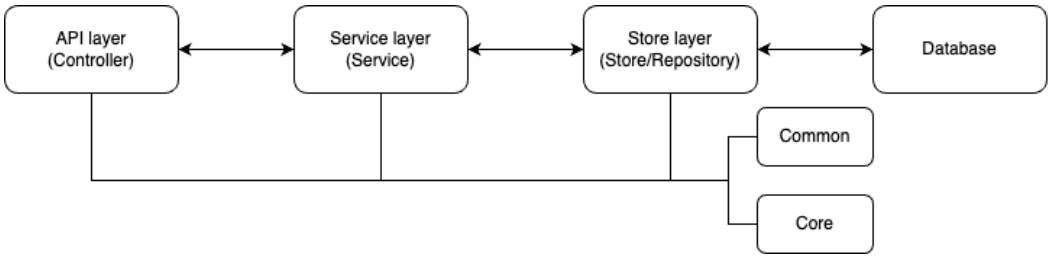

Figure 6: BE architecture

validating requests using annotations, mapping data between the domain model and data transfer objects, and forwarding the request to the appropriate service.

The service layer focuses on business logic, where all validation, data processing, and business logic are performed. Once the data is ready, the service layer delegates its persistence and retrieval to the store layer.

The store layer is responsible for interacting with repositories to handle data persistence and retrieval. It transforms entities from the database into domain objects before returning them to the service layer. Furthermore, it aggregates related data into complete domain objects. For example, when assembling complex objects that include related entities, the store layer ensures these are properly integrated, maintaining data integrity and coherence.

Shared resources, such as reusable classes, are located in the core folder, which is accessible across the API, service, and store layers. Additionally, utility files are housed in the common folder, providing functionality leveraged throughout the application. This structured approach promotes modularity, maintainability, and a clear separation of concerns.

4.2.2 *FE architecture*

At RogerHR, the FE architecture is built on a modern React framework and a suite of complementary tools. React Router is used to manage navigation throughout the application, and TanStack Query (formerly React Query) handles efficient data fetching and caching. For date manipulation, Date FNS is relied upon, while AG-Grid is employed to render large datasets such as employee timesheets. The UI is primarily built with MUI, which offers a rich set of pre-built components based on Material Design, and DayPilot is used for specialized scheduling components. Additionally, Yup is the choice for form validation, and Storybook helps document and showcase UI components. The build process is streamlined using Yarn and Vite.

The project is organized into several clearly defined folders that help maintain a modular and maintainable codebase. The 'api' folder houses all API calls along with the necessary data transfer objects, ensuring interactions with the BE are well-structured. The 'domain' folder contains reusable services and shared business logic, along with all custom types that tie the application together. Dedicated folders for custom hooks and utilities, such as those for date-time manipulation, math functions, and validators, promote code reuse and clarity. The 'pages' folder is organized by individual pages, with each page directory containing its main component (in TSX), associated Storybook files, and Cypress tests to ensure robust component functionality. This careful organization, which was significantly improved during the internship, has transformed the codebase into a much more comprehensible and consistent environment.

In terms of state management, Redux is deliberately avoided unless absolutely necessary. The preference is to leverage React's local state and Context API, which reduces complexity and minimizes the potential for over-engineering. However, Redux is used in cases where it is indispensable. This balanced approach helps keep the application lightweight while maintaining robustness to handle complex interactions.

The FE architecture, with its structured design, modular components, and robust testing strategy, enables the quick and efficient addition of new features. Comprehensive tests act as a safety net, immediately flagging any unintended issues, ensuring a stable, scalable, and maintainable application.

4.3 MODULES AND CONTRIBUTIONS

This chapter is organized by RogerHR modules, focusing on the areas where the most significant impact was made. While contributions were made to several modules, the discussion will delve into those where involvement was most substantial. Some modules will be mentioned briefly, as contributions were limited to smaller features or minor improvements.

The internship began with a focus on BE development, but curiosity and a willingness to learn led to taking on FE tasks as well. Initially, these were small changes, such as adding a button or updating text, but over time, responsibilities expanded. The transition to FS development started with simple modifications, like adding a new field in the database and making it available in the FE. This gradual shift introduced React, a framework previously unfamiliar, requiring quick

adaptation and learning. Despite a background more oriented toward BE development, proficiency in FE development was achieved, enabling independent handling of entire features without relying on others to bridge the gap.

The primary modules worked on extensively were leaves, shifts, timesheets, and timesheet settings, essentially covering the entire time management system within RogerHR. Contributions were also made to other areas, including realms, employees, employments, working patterns, and smaller supporting modules. Additionally, involvement included the Timemoto integration, a service designed to communicate with external clock-in/out devices, as well as the public API, which extended system functionality beyond the internal platform.

One major advantage of working across both FE and BE was the ability to implement changes seamlessly. Whenever a new feature required an additional field, an updated endpoint, or a change in how data was processed, alignment between BE and FE could be ensured. This eliminated dependencies between teams, reduced delays, and made feature development more efficient. As experience grew, contributions evolved from small improvements to full-scale refactoring of major components and the development of entirely new pages.

Another key aspect of the work was ensuring test coverage, which became a growing priority throughout the internship. Initially, there was no strict requirement for FE tests, but midway through, Cypress was introduced for component testing. In each module discussed, the integration of testing will be highlighted, ensuring stability and maintainability.

By working across the entire stack, a strong understanding of how the FE communicates with the BE was developed, as well as how BE logic should be structured to have the right impact on the UI. This full-picture perspective enabled faster and more effective feature development, ensuring smooth integrations and minimizing breaking changes.

Throughout the internship, a total of 468 tickets were worked on: 265 bug fixes, 177 stories related to new features and improvements, and 26 tasks.

This ticket distribution reflects the variety of tasks handled, ranging from fixing bugs and improving existing functionalities to adding new features and optimizing CI/CD processes. Many of the initial tasks involved bug fixing, which served as a natural introduction to the codebase and the different modules. The process of debugging was particularly engaging, as it required diving deep into the code, understanding the logic behind certain behaviors, and tracing the root cause of issues.

Beyond bug fixing, new functionalities were implemented, which occasionally surfaced unexpected edge cases later reported as bugs. Contributions were also made to performance improvements and code refactoring to enhance maintainability. Involvement in CI/CD tasks provided valuable insights into deployment pipelines and automation processes, broadening the understanding of the full development lifecycle.

4.3.1 *Realms*

Realms are central to the system's architecture, as every table in the database is associated with a realm. A realm represents a distinct client, and every table includes a reference to it to ensure that all rows belong to a specific realm. This design guarantees data isolation and prevents mixing data between clients. To achieve this, all database queries include the "tenant_id", which serves as the unique identifier for a realm.

Each realm also has its own login methods, accommodating clients with different authentication needs. For example, some clients may prefer the standard email/password login, while others might require integration with Microsoft or Google for authentication. The functionality enabling support for multiple login methods was implemented, with both the BE and FE components designed and coded.

In addition to login methods, realms define their functionality through "realm features," which determines the features available to a client. The platform supports features such as Advanced Employee Profile, Leaves, Documents, Timesheets, Planning, Surveys, Reviews, Payroll, Clock In/Out, Reports, Cost Centers, Third Parties, and Job Families.

When implementing the realm features, a key role was played in enabling clients to selectively activate or deactivate functionalities based on their specific business needs. This flexibility ensures clients interact only with the features they require while aligning with the platform's pricing model. Since each feature adds to the overall cost, clients are incentivized to choose only the functionalities that bring value to their operations, making the system both cost-effective and customizable.

The implementation of realm features focused more on the BE, enabling the selective activation or deactivation of functionalities based on client needs. This flexibility ensures clients interact only with the features they require while aligning

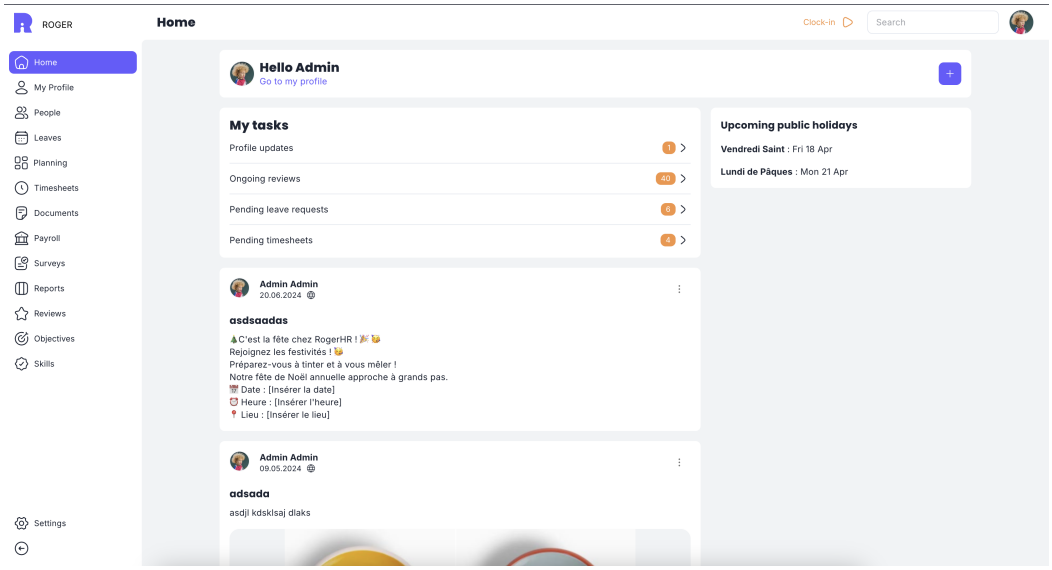

Figure 7: RogerHR landing page after login

with the platform's pricing model. Additionally, support was provided to the FE team by helping hide unnecessary pages. A hidden page was also created to simplify the process of toggling features on or off when needed.

As shown in Figure 7, most of the realm features are hole modules, accessible via the navigation bar on the left-hand side. However, not every realm feature represents a full-fledged module. Some are designed as smaller components tailored to meet the varying needs of different clients. For example, features like clock-in/out, cost centers, or job families can be disabled or hidden for clients who don't require them. This ensures that the interface remains clean and relevant, displaying only the tools and information that are essential for each client's needs. By minimizing unnecessary complexity, the system delivers a more focused and user-friendly experience, avoiding unused features that create clutter.

The creation of a new realm is a comprehensive process. Along with configuring realm features and login methods, several mandatory data and configurations are set up. These include creating a default admin employee and associated employment, job, department, location and permissions. Additional configurations include setting up the timesheet settings, weekly working time, and section definitions (such as employee basic info, address, etc.), as well as establishing leave types and planning settings. These elements are foundational to ensuring the realm is fully operational and tailored to the client's requirements.

Throughout the work, several aspects of the realm creation process were implemented and improved. The default realm notifications feature was developed,

allowing administrators to enable or disable notifications for all employees. This addition proved valuable in helping admins set up data without employees receiving unnecessary emails. Additionally, a critical role was played in refining the database structure by ensuring that every table included the 'tenant_id', a key element for maintaining data security and isolation between clients. Initially, this was challenging due to missing 'tenant_id' references caused by Hibernate's limitations. However, after transitioning to JOOQ, the database setup was successfully completed, incorporating the 'tenant_id' column across all tables and significantly enhancing security and consistency.

These improvements not only solidified the realm's foundational capabilities but also deepened expertise in its architecture. Together, the components of login methods and realm features form the backbone of the realm architecture, a structured design that ensures scalability, security, and high customizability for each client. The extensive work on realms, including significant enhancements and refinements, has built confidence in evolving this critical part of the platform to better meet client needs and ensure its robust functionality.

4.3.2 *Employees*

The second core component of the application is the employees, who serve as the backbone of the entire system. Nearly all functionalities and data structures are designed around employees, making them integral to the platform's operation. Some elements are essential for handling employees in all companies. For instance, each employee needs to have at least one employment and a work pattern. These fundamental HR data areas will be examined in detail throughout this chapter.

As shown in Figure 8, the employee profile page provides a central location for employees to navigate through various tabs, displaying only their own information. For companies using all functionalities, this page represents what a typical employee would encounter. The profile tab includes essential employee information, such as personal details, address, and any custom fields that the company may configure. The Job tab contains data related to the employee's contract, employment details, and work pattern. In the leaves tab, employees can view their leave balances and historical leave data. The planning tab offers a personal schedule overview, while timesheets display logged hours, highlighting whether an employee has worked under or over their expected hours. Additional tabs, such as reviews, skills, objectives, and documents, will be discussed in better detail in subsequent chapters.

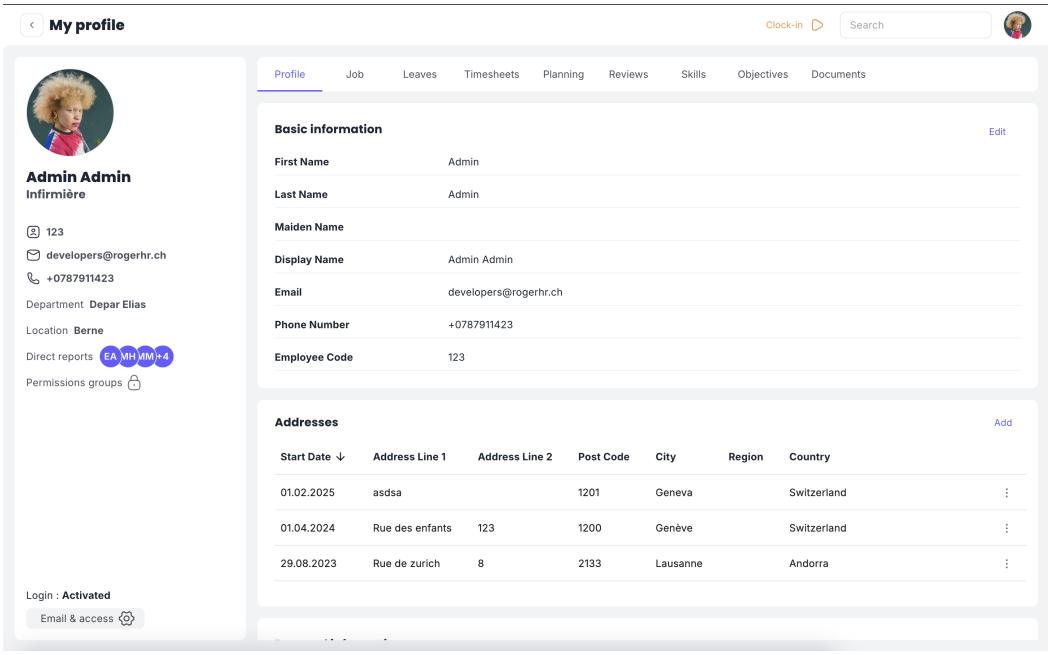

Figure 8: Employee profile page

The employee profile follows a modular architecture designed for extensibility and adaptability. While the initial modular structure was implemented previously, significant contributions were made to its evolution through the addition of new fields and validation logic. This modular approach streamlines the process of modifying profile fields, ensuring flexibility for future requirements. Key enhancements included implementing administrator hard-delete functionality for test account management, developing the Birthdays feature to display upcoming celebrations while maintaining data privacy, and optimizing employee information update processes.

Given the extensive amount of data associated with employees, challenges arose in optimizing how this information is loaded, particularly when fetching multiple employees. A functionality was developed to allow users to search for employees using keywords such as 'Francisco Leiria dev'. This feature involved filtering by various criteria, including location, department, manager, job, employee name, and display name.

The search logic was particularly complex due to the volume of data and the need to account for current employments and work patterns. The implementation began with a database-level search to filter initial results efficiently. Additional information, such as employments, work patterns, and personal data, was then loaded for each employee. Finally, further filtering was applied at the application level to ensure only relevant and current data was returned. This multistep approach balanced performance with accuracy, effectively meeting the platform's requirements.

To streamline onboarding, a dedicated page and endpoint were developed, enabling creation of all essential employee data through a single action. This onboarding workflow ensures that crucial information, such as personal details, employment, and work patterns, is created in one go. It also provides options to configure additional fields, including leave types (e.g., holidays, medical), planning settings (e.g., assigning a manager, defining positions, and granting team access), and scheduling performance or probationary reviews. Over time, this page/endpoint has evolved to include additional functionality, addressing the need for efficiency by minimizing repetitive tasks and consolidating multiple actions into a single endpoint. As part of this evolution, contributions were made to the onboarding process by adding several new fields as a FS task. This evolution has made the onboarding experience more intuitive and aligned with client needs.

Furthermore, a public API endpoint for onboarding was implemented, designed with a more minimalistic approach compared to the FE version. This API focuses on mandatory fields to ensure consistent data entry and reduce the risk of errors.

Contributions were also made to employee reports, including timesheet and leave balances, ensuring that critical metrics were calculated accurately. These reports provide valuable insights for both employees and administrators, though further details will be expanded later. Additionally, extensive tests were written to maintain the required code coverage of at least 75%, ensuring the stability and reliability of the developed features.

4.3.3 *Employments*

Employment data are central to the system's functionality, storing all information related to an employee's employment. Each record offers a thorough view of the employment, guaranteeing accurate historical tracking and operational data for analysis and decision-making. This includes important information like start and end dates, contract type, probation end date, location, department, job title, job family, managers, and cost center assignments.

To maintain data integrity, the system enforces strict rules for employments. The system follows stringent guidelines for employment data in order to preserve data integrity. To avoid conflicts between overlapping data, an employee can only have one active job at a time. Although employment records can be created at any point in time, the system makes sure they do not overlap in order to maintain a consistent employment history. A specific flag designates primary or secondary employment

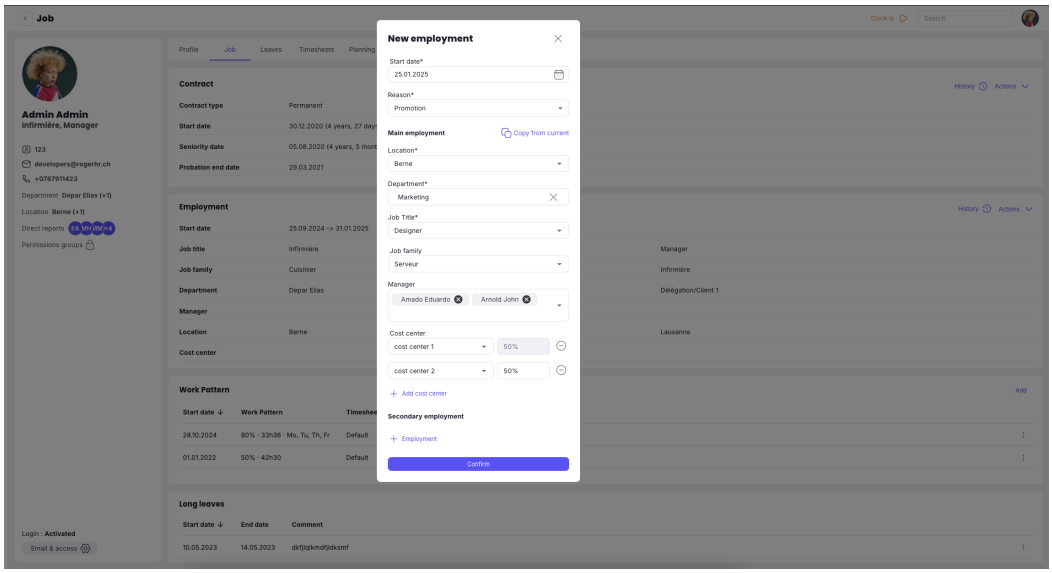

Figure 9: Employment dialog

data, which are then arranged chronologically according to their start dates. This implementation enables multiple employments with distinct data attributes while maintaining consistent start and end dates across all records. Figure 9 illustrates the employment creation dialog in the employee profile interface, presenting both required and optional data fields. In the case of clients that do not have the job family and cost center feature, these two fields are hidden so that the form is as simple as possible.

Significant contributions were made to enhancing the usability and functionality of employments within the system. One task involved improving the algorithm responsible for updating employment data. For instance, when the start date of an employment changes, the system automatically adjusts any linked employments to ensure consistency and maintain a proper chronological order. Additionally, new features were implemented, such as support for cost centers (only developed on the BE) and the introduction of the job family field (handled as a FS ticket).

Employments are a critical part of the application, as they provide essential information, such as where an employee is working and whether they are employed on a given date. Given that many features depend on this data, additional tests were added to ensure reliability and prevent future issues. By the end of the internship, the platform supported only a single employment per employee. However, the following month, a colleague implemented secondary employments, building on the groundwork that had been laid.

Given the complexity of employment management, significant emphasis was placed on optimizing the UX. Efforts to improve the UI have made it significantly easier to add and update employments, as well as to review both the current employment and historical data. These enhancements not only simplify employment management but also help users better navigate and understand the employee's employment timeline easily.

4.3.3.1 *Legal units, Locations and Areas*

Locations help organize employees and areas. Each location includes details like address, city, region, and country, along with a holiday calendar and operating hours for planning. Locations can also have multiple areas to represent smaller sections or functional spaces.

To streamline the management of multiple locations, it was introduced the concept of an "Entity" or legal unit. A legal unit is a simple grouping mechanism that consolidates several locations under one unit, identified by a name. This feature enables the generation of reports that span multiple locations, making it easier for organizations to analyze data across their operational locations.

For example, consider an organization like IPLeiria. IPLeiria serves as the legal unit, while its specific campuses, such as ESECS and ESTG, are represented as locations. Within ESTG, the B building could be defined as an area, highlighting the hierarchical relationship between legal entities, locations, and areas.

The integration of locations with employment data is critical for the planning. The system establishes where employees may go to work and makes it easier to create shifts, timesheets, and workforce plans by relating employees to specific locations. This helps to plan in advance because it is easier to replace someone as they both work in the same location, so, if an employee is sick or cannot come to work on a specific day, the manager can see if there is someone available at a specific location to replace him.

Significant contributions were made to the location feature, addressing a variety of tickets and resolving several bugs. These improvements included refining the UI to create a more user-friendly layout, enhancing usability, and adding new fields to better represent location data. One of the most impactful updates involved enabling the use of opening and closing hours, which had previously been stored in the system but were not utilized. Additionally, two new fields, region and country, were introduced based on a client request, implemented as a full-stack (FS) ticket. For

legal units, the BE implementation was handled, while a colleague managed the FE changes, as the BE update did not introduce any breaking changes.

4.3.3.2 *Departments*

Departments play a central role in organizing employees and managing permissions within the system. Each department is defined by its name, parent department, assigned managers, and associated cost centers. A department may have more than one cost center, just like the employments, with the overall percentage distribution among all cost centers being 100%.

To maintain a manageable hierarchy, departments can have parent-child relationships, but these are limited to six levels to avoid excessive recursion and complexity. When a department is no longer useful, it can also be archived to preserve its historical data and prevent it from interfering with current activities.

Departments are crucial for permissions management and are mainly used to group employees. Workers in a particular department frequently need permissions specific to their roles. Members of the HR department, for instance, usually manage private employee information. As a result, they need department-specific permissions that allow them access to employee data while maintaining system security and confidentiality. This way, when someone joins the HR department, they will be automatically granted the appropriate permissions.

Considerable work was done for the department feature, including implementing the ability to assign multiple managers and introducing cost centers. To ensure data integrity, extensive BE validations were added to verify the accuracy of data sent from the FE, addressing a gap that previously existed. Additionally, substantial enhancements were made to the endpoint responsible for retrieving all department nodes. This improvement enabled the construction of the FE department tree with correct child-parent relationships and the display of proper managers for each department.

4.3.3.3 *Job families and Jobs*

The Job Family feature provides a simple yet powerful way to group related jobs under a common category. Defined by a single attribute, its name, a job family helps streamline the organization and management of roles within companies. For example, a job family like "Doctor" could encompass jobs such as Cardiologist, Neurologist, and Dermatologist, making it especially useful for clients with large

groups of employees and intricate job hierarchies. While this feature is optional, it is typically enabled for larger companies that require a more structured classification of roles.

Job families also play an important role in managing skills, as they allow skills to be specific to a particular job family or general across all employees. For instance, a "Doctor" job family may need unique skills that are irrelevant to other job families. This capability improves skill organization and application, with detailed examination following in a subsequent chapter.

The job family feature was done as a FS ticket, integrating it into both the jobs and skills modules and linking it to the employment module. This implementation extended the system's functionality, enabling users to categorize roles more effectively and align them with organizational needs.

The Job module itself is designed to encapsulate essential information about specific roles within an organization. Each job entry now includes a title, gender-specific display names to ensure accurate representation based on the employee's gender, the associated job family, a brief description outlining its purpose, and a detailed account of its responsibilities.

Jobs are also essential in supporting hierarchical management structures and defining permissions. For example, a "CTO" job might be authorized to supervise employees assigned to the "Developer" job. This hierarchical organization of jobs is invaluable for managing permissions and responsibilities effectively, ensuring the system can accommodate the complex structures of modern organizations.

When the work on the job module began, it only included a name field. By introducing additional fields, the scope of the module was significantly expanded, laying the foundation for future enhancements. While the fields for purpose and responsibilities have been fully integrated, their broader applications are still evolving as the system continues to grow.

4.3.3.4 *Cost centers*

Cost centers, at present, consist of only a name field, serving as a foundational tool for tracking and organizing employee-related data. They give companies information about how employees are distributed among various cost centers, allowing them to examine workforce allocation by cost center. Clients can use this feature, for example, to determine how many employees are working at maximum capacity in each cost center.

Cost centers are an optional feature within the application and can be enabled or disabled through realm settings, catering to clients who require this level of detail. Depending on their particular requirements, some companies, particularly smaller ones, may find this data unnecessary, but for others, it may be essential for operational analysis.

The BE development of the cost center feature was focused on seamless integration with both the departments and employments modules. While some FE contributions were made for this feature, the more complex FE tasks for departments and employments were primarily handled by a colleague.

4.3.4 *Employee work patterns*

An employee's working pattern defines when and how many hours they are expected to work each week, as well as the specific days they are assigned to work. This information is crucial for managing schedules, calculating holidays, and ensuring compliance with employment agreements. When creating a working pattern, key fields such as the start date, related employee, calendar, and timesheet settings are always required. While an employee may have multiple working patterns, overlapping patterns at different start dates are not allowed. This ensures clarity in determining the total hours an employee is expected to work on a specific date.

Working patterns are deeply integrated into system modules like timesheets, shifts, and leaves. They serve as the foundation for determining daily work hours, which is essential for scheduling and accurate hour calculations. Timesheets use these patterns to identify discrepancies between scheduled and actual hours worked, while leave management relies on them to calculate entitlements and track usage. By centralizing this information, working patterns streamline operations and support efficient workforce management.

To cater to diverse company needs, the system supports four distinct types of employee working patterns:

- Fixed: Requires only the weekly working time, specifying the number of hours an employee is expected to work per week, making it ideal for employees with predictable schedules.
- Rate: In addition to the weekly working time, includes a percentage rate (0–100%) that defines the proportion of the base hours an employee will work.

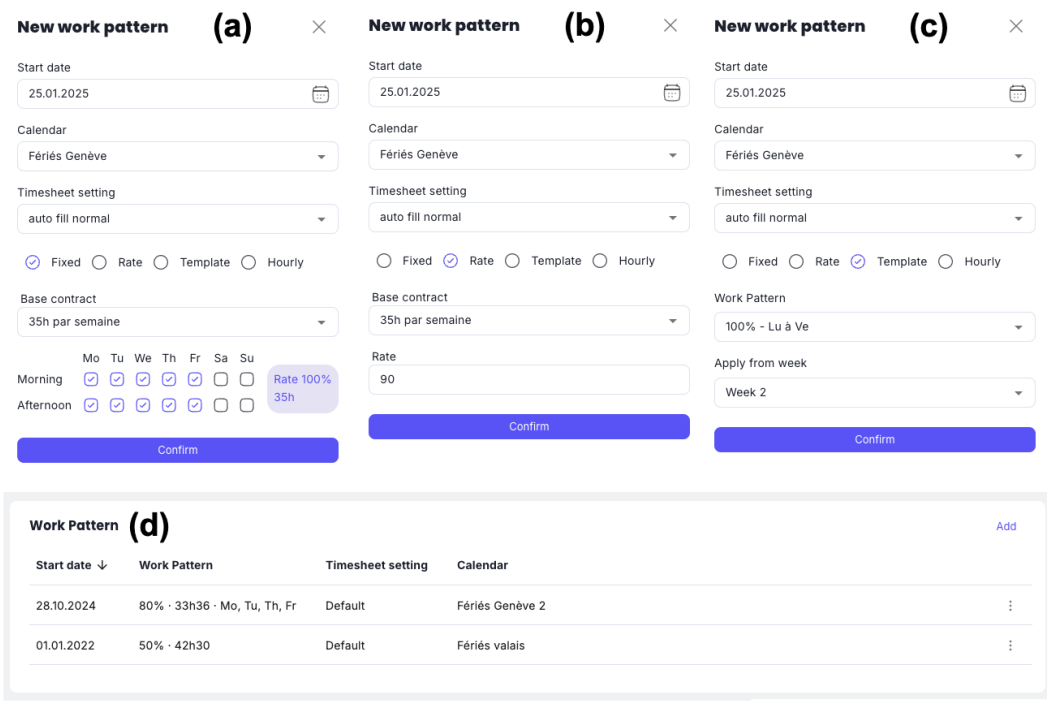

Figure 10: Employee Working pattern dialogs and history

This is particularly valuable in countries like Switzerland, where part-time work at rates such as 80% or even 50% is common.

- Hourly: Does not rely on predefined contract hours, instead dynamically determining hours based on actual work logged, offering maximum flexibility for roles with irregular schedules.
- Template: Enables clients to set up recurring work schedules spanning up to 10 weeks, with the ability to specify a starting week within the template. This advanced flexibility reduces the need for multiple unique work patterns, addressing specific client needs with efficiency and re-usability.

Figure 10 illustrates three examples of the dialog when using Fixed (a), Rate (b), and Template (c) work patterns. For the Hourly work pattern, no additional details are required, as it is inherently based on hourly rates. The lower section of Figure 10(d) displays historical data available in the employee profile module.

Developing and maintaining the working pattern functionality posed several challenges, particularly in calculating daily expectations for employees under different work patterns. This complexity required extensive testing, which was addressed by writing numerous test cases to validate the accuracy of calculations and workflows across all scenarios. One contribution was the addition of the rate type, initially designed for a specific client but later adopted by others. This addition has proven

especially useful for smaller clients with fixed schedules. Similarly, the starting week feature for templates was implemented, allowing clients to efficiently reuse templates without creating new patterns, further enhancing the system's flexibility. Contributions were also made to migrating the calendar from employments to working patterns and transitioning the timesheet settings from the employee table to the employee working pattern. This enhancement provided clients with the flexibility to adjust an employee's timesheet settings without impacting past hour calculations.

The working pattern module is already a robust tool, offering clients the ability to configure a variety of work arrangements with ease. This flexibility has been instrumental in client satisfaction, as organizations can adapt the system to their unique requirements without compromising simplicity. For example, the rate type has become invaluable for clients requiring part-time arrangements, while the template feature addresses more complex scheduling needs with minimal effort.

4.3.4.1 *Weekly working times*

The Weekly Working Time module is a simple yet essential part of the system, capturing critical details about an employee's weekly schedule. It includes fields such as the name, total minutes worked per week, daily start and end times, break start time, and break duration.

With this information, the system can accurately calculate daily working hours and generate missing timesheets, ensuring that both work periods and breaks are correctly accounted for. This capability is vital for creating reliable employee schedules and aligning data with company policies.

This module was enhanced by introducing the optional field for break start time, adding flexibility for companies that require it while remaining unobtrusive for those preferring a simpler approach.

4.3.4.2 *Advanced working patterns*

The Advanced Working Pattern module provides a highly flexible solution for managing employees with non-standard or alternating work schedules. It includes fields for the name of the pattern, the associated weekly working time to establish base weekly hours, and a detailed list of weeks that define the specific schedule. Each pattern allows for a minimum of one week and a maximum of ten weeks.

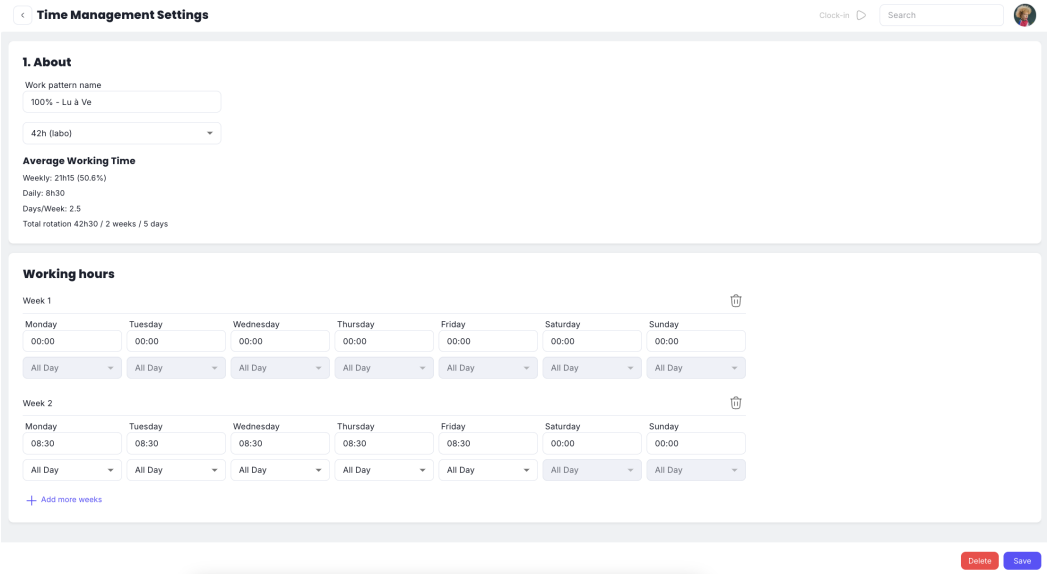

Figure 11: Advanced working pattern page

Within each week, all seven days (Monday to Sunday) are represented, with fields to specify the working hours or mark a day as non-working by assigning zero hours.

While the weeks in a pattern can technically be identical, the module is primarily designed to manage distinct weekly variations, making it an optimal solution for employees with irregular schedules. As these patterns repeat cyclically, there is no need to recreate identical configurations, ensuring a streamlined and efficient process for managing extended schedules.

Figure 11 showcases the Advanced Working Pattern page, featuring an intuitive interface for creating and managing schedules. This visual highlights the ease of configuring patterns to support diverse working arrangements.

This functionality is especially useful for rotating or alternating schedules, such as employees working every other Friday, resulting in a 90% work pattern. By defining alternating weeks, the module seamlessly handles such scenarios, making it ideal for industries like restaurants and hospitals, where flexible and non-standard schedules are essential.

This functionality was enhanced by refining the method used to calculate which week of the Advanced Working Pattern an employee is in, based on their start date. For example, if an employee began working on 01/01/2024 with a 10-week pattern defined, the applicable week on 20/02/2024 would be calculated as the 8th week of the cycle. Previously, issues with these calculations led to inaccuracies.

To address this, the method was fixed, and extensive testing was implemented to ensure its reliability. Accurate calculations are critical, as they directly impact the determination of working and non-working hours. Errors in this process could result in incorrect hours being recorded, leading to significant consequences, including incorrect payroll calculations. Therefore, thorough tests were essential to safeguarding data integrity and ensuring employees are paid accurately for their work.

4.3.5 *Leaves*

The Leave module is a crucial feature within the system, serving as the primary interface for employees to manage their holidays, medical leaves, and other time off requests. While some features of the system, such as HR-specific modules, may primarily be used by administrative staff, the leaves module sees significant engagement from employees across all levels, making it an essential component of their UX.

A leave request includes mandatory fields such as the leave type, start and end dates, and the time period (morning, afternoon, or all day). For hourly-based leaves, additional details such as start and end times in hours and minutes are required. Certain leave types, such as medical leaves, allow for end date omission to accommodate long-term absences. Furthermore, the system enables percentage based leaves for example, an 80% leave means the employee works 20% of their usual schedule while the leave is active.

Attachments can also be required for specific leave types, ussually for medical leaves where official documentation is often mandatory. This ensures that all necessary proof is provided to support the leave request. Whether mandatory or optional, the attachment requirement can be tailored per leave type, adding another layer of customization to the module.

The request workflow is straightforward yet robust: requests typically begin in a pending state, followed by approval or rejection, and can be deleted if needed. This process is designed to be intuitive for employees while providing the necessary oversight for HR or management.

Managing leave balances, hours used, and remaining time is where the complexity of the module shines. Calculating these numbers requires loading and analyzing a variety of data points, such as the employee's working pattern, leave type policies,

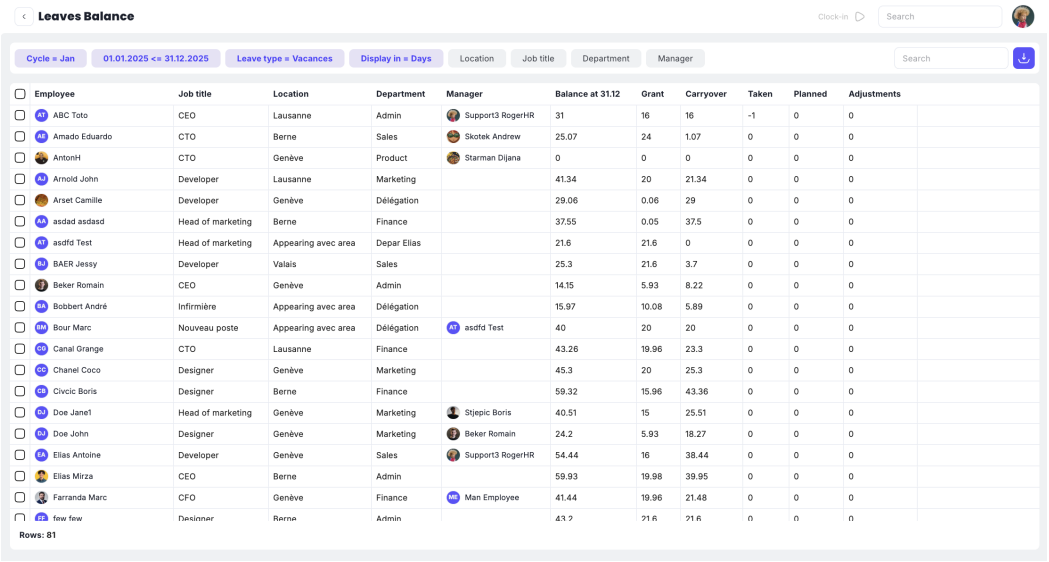

Figure 12: Leaves balance page

all the leave corrections, and all the leaves for this leave type. Additionally, at the end of each year, leave balances are reset, ensuring a clean slate for the next period.

The BE endpoints were developed, and the FE page was implemented to display all employees' leave balances in a single view, as shown in Figure 12. This page is designed for HR personnel or managers, allowing them to efficiently review employee balances. It includes mandatory filters, such as the cycle (e.g., from January to January or September to September), which is hidden if only one cycle exists, the start and end dates for the period being analyzed (with the start date typically defaulting to the beginning of the cycle), and the leave type for which the balance is being viewed.

Although the page may appear straightforward, its implementation was complex due to the numerous factors that needed to be considered. To ensure accuracy and maintain functionality, extensive tests were added for both the BE and FE, providing robust coverage to prevent regressions and ensure the page continues to operate as intended.

The module's integration with timesheets and employee working patterns is essential for accurate calculations. By understanding the employee's specific work schedule, the system determines how many days or hours a leave request will actually use. For example, it can distinguish between a full-time employee's week-long vacation and a part-time employee's equivalent leave, ensuring precision and fairness in tracking leave usage.

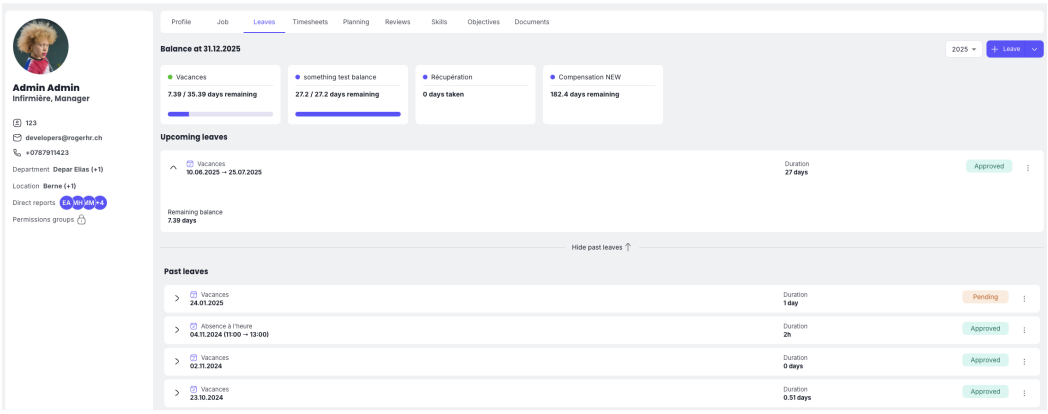

Figure 13: Employee leaves balance page

Figure 14: Leave pending my approval page

An employee can easily view their leave balances directly from their profile, as illustrated in Figure 13. This page allows employees to check their current balances for each leave type for a specific year, which can also be adjusted to view balances for other years. Additionally, it displays both upcoming and past leaves, providing a comprehensive overview of their leave history and future plans. Several enhancements were made to this page, including improving the display of the leave type balance cards. These now clearly show the remaining days or hours for each leave type, making it easier for employees to track their balances at a glance. The layout was also refined by creating a clearer distinction between upcoming and past leaves, ensuring the information is better organized and more user-friendly.

New endpoints were developed to enable mass approval of leaves and to retrieve all leaves requiring an employee's approval, typically used by HR personnel or managers to approve employee leave requests. These endpoints were integrated into the page created, as shown in Figure 14. This page displays all pending leave requests that an employee is authorized to approve. One of the most challenging aspects of building this functionality was handling cases where certain leaves could not yet be approved. Extensive logic was implemented to determine whether a leave request was approvable or not. For instance, a leave might be unapprovable if it

is missing an end date, if an attachment is required but not yet provided, or due to other specific conditions. Ensuring that this logic aligned seamlessly between the BE and FE was particularly challenging, especially for correctly enabling or disabling buttons based on the leave's status. However, these efforts resulted in a cohesive and user-friendly experience for managing leave approvals."

While the Leave module is already powerful and highly customizable, it does have room for improvement. The underlying code, particularly the logic for calculating balances and corrections, is complex and occasionally requires refactoring. However, the team has been addressing this incrementally, optimizing the codebase whenever updates or changes are needed. This ongoing refinement ensures that the module remains both reliable and scalable as new features are added or existing processes evolve. The development process includes continuous test expansion alongside code modifications to maintain comprehensive test coverage.

A significant role was played in enhancing the Leave module, contributing to both its FE and BE development. One major achievement was refactoring the entire leave request dialog, which will be discussed in more detail later, streamlining its functionality and improving the UX. On the BE, several new endpoints were created to provide more flexible and efficient ways to fetch leave data, catering to various use cases and improving overall system performance. New settings for leave types were also introduced, enabling greater customization. This required significant updates to the existing logic to ensure these settings were seamlessly integrated across the module.

Another feature implemented was the ability to create multiple leave requests in a single operation. For example, users can now create multiple leave requests for different days simultaneously, using the same input data. These contributions to the module have not only expanded its capabilities but also improved its usability, adaptability, and alignment with the needs of diverse clients.

4.3.5.1 *Leave types*

Leave types, as shown in Figure 15, are essentially a configurable table used to define specific settings and rules for different types of leaves. There are five main leave types: Not Working Paid, Not Working Unpaid, Medical, Working, and Compensation. Each type affects how worked time is calculated. For example, "Not Working Paid" implies the time will be paid, while "Not Working Unpaid" means the time will not count toward working hours. The "Working" leave type allows us to represent situations like home office work. Although not perfect, this approach has been

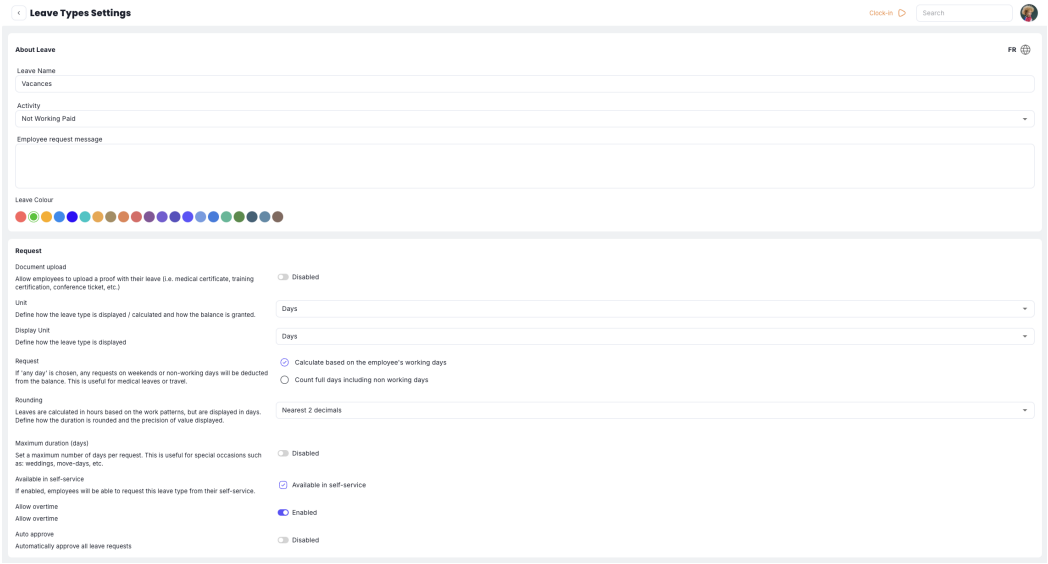

Figure 15: Leave setting page (upper half)

effective, and clients have expressed satisfaction with its functionality. The "Medical" leave type includes unique conditions, such as not requiring an end date, making it suitable for medical scenarios.

The leave module includes several settings, each with its own complexities. For example, the document upload setting determines whether attachments are allowed for a leave type. If enabled, additional rules can require a document after a certain number of days. Another critical setting addressed is leave duration calculation, where clients can choose between basing it on the employee's working days or counting all calendar days, including non-working days. This flexibility is especially valuable in Switzerland, where sick leave information sent to insurance must include all days, not just working days.

The leave module includes several additional settings designed to enhance its flexibility and functionality. For instance, the maximum leave duration defines the longest leave an employee can take, while the decimal precision setting allows for controlling the number of decimals displayed for leave balances, or removing decimals altogether for simplicity. Among these features, the implementation of overtime restrictions was handled, preventing employees from working overtime on days when certain leaves, such as sick leave at 50%, are applied. This feature had a significant impact on working hour calculations and addressed specific client requirements. Another setting, auto-approval, streamlines the process by automatically approving valid leave requests upon creation, even when the employee does not have approval permissions.

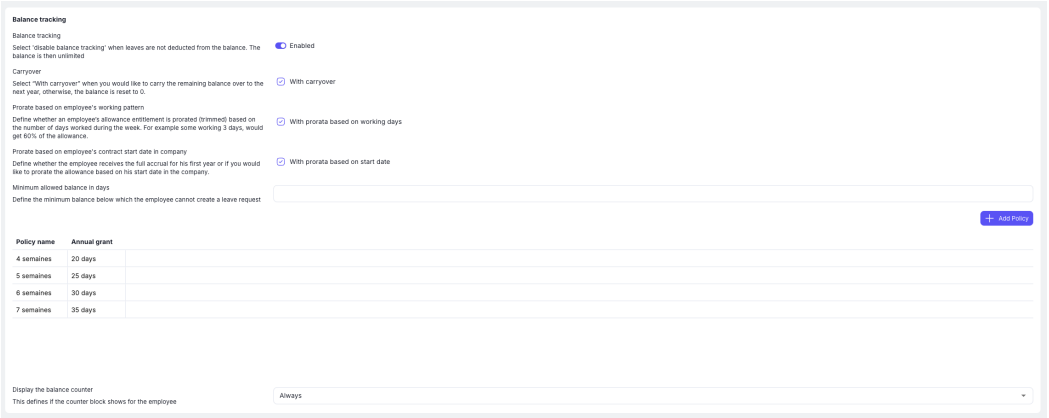

Figure 16: Leave setting page (lower half)

A crucial role was played in employee balance calculations, particularly for leave types. Several bugs and edge cases within the balance tracking features were addressed. This included working on settings (Figure 16), such as unlimited or limited balances, which determine whether a leave type, like medical leave, has no annual limit or if a specific cap applies, such as 22 vacation days per year. Contributions were also made to improving the carryover functionality, which determines whether unused leave balances roll over to the following year. Additionally, the ration settings were refined to ensure accurate balance adjustments based on an employee's working pattern or start date, enabling proportional allocation for employees who join mid-year. Another area addressed was the minimum balance configuration, which allows for a negative balance limit, such as up to -2 days, providing employees greater flexibility when requesting leaves.

Another feature of the leave module is the concept of policies. Multiple policies can be associated with the same leave type, each specifying a different allocation of days per year. For instance, an employee entitled to 20 days could be assigned one policy, while another entitled to 30 days could be assigned a different policy. This approach eliminates the need to duplicate leave types and significantly streamlines leave management, and this is how we know each employee got access to each leave type, is based on the policy he is a part of.

On the FE, the balance counter display was enhanced, enabling users to toggle its visibility. While working on the leave request card (Figure 13), significant portions of the related code were refactored, improving its structure, readability, and maintainability. Many of the settings implemented impacted both the FE and BE, requiring substantial additions to the codebase to enhance functionality.

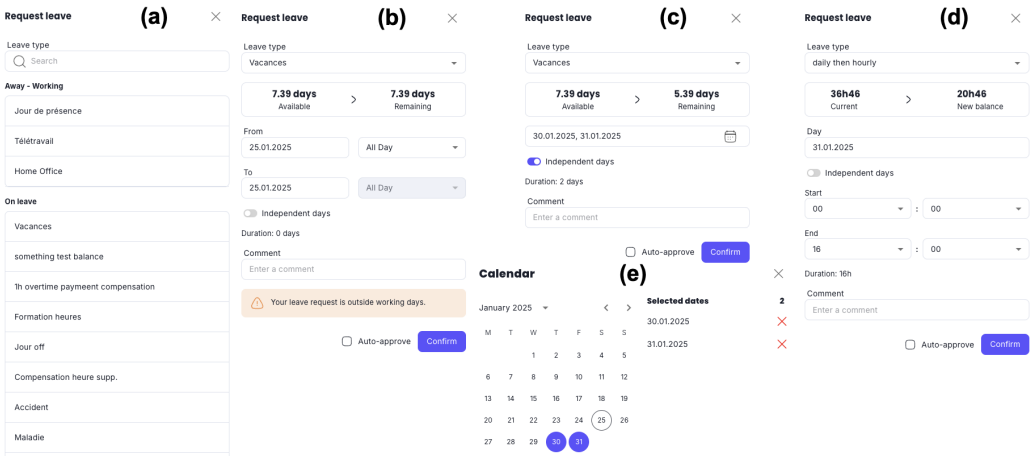

Figure 17: Multiple leave request dialogs

Since much of the existing leave module code had not yet been refactored, the opportunity was taken to improve its overall quality, ensuring it was more organized and easier to understand. Given the significant impact of these settings across various parts of the system, extensive test coverage was added for both the BE and FE to account for all edge cases. As these tasks often involved breaking changes or required seamless integration, they were consistently approached as FS tickets to maintain smooth workflows and minimize disruptions.

4.3.5.2 *Leave request dialog*

The leave request dialog (Figure 17(a/b)), the primary interface for creating and updating leave requests, was refactored and enhanced through several improvements. One of the most impactful enhancements was the development of a reusable multi-date selection component. This component (Figure 17(e)) allows users to select multiple dates directly from an interactive calendar: clicking on a date toggles its selection, and a dynamic list displays the currently selected dates. Users can remove a date by clicking on it in the list, ensuring the calendar and selection remain perfectly synchronized.

This streamlined approach simplifies the leave creation process by allowing users to submit a single request for multiple dates, thereby reducing the number of BE calls. Not only does this improve usability, but it also contributes to a more maintainable codebase by promoting reusability and consistency across the application. Overall, these enhancements significantly elevate the UX and make our leave management system more efficient and scalable.

Integrating this functionality required significant enhancements in both form validation and state management. By leveraging React Hook Form and Yup, redundant state handling was reduced, ensuring that every form update correctly triggered the necessary BE calls. Existing logic was refactored to support dynamic leave type changes during a request, allowing users to switch between hourly and daily leave formats seamlessly. This entailed splitting the code to handle three distinct scenarios—hourly leaves (Figure 17d), daily leaves (Figure 17b), and multiple leaves (Figure 17c)—while unifying shared logic into one comprehensive dialog.

Moreover, every change in the UI triggers multiple BE requests to fetch overlapping leaves, leave previews (with duration and remaining balance), and detect conflicts with other leave requests. This ensures users always see the most accurate and current data. In parallel, various parts of the leave approval workflow were refactored to consolidate complex logic, such as verifying overlapping shifts and validating leave requests, into a single, unified system, reducing redundancy and improving maintainability.

4.3.6 *Time management (timesheet/shifts/leaves)*

Throughout the internship, significant involvement in time management tasks was maintained, covering areas such as timesheets, leaves, shifts, and planning. As responsibilities grew, a deep familiarity with this part of the system was developed, resulting in a thorough understanding of the code and the interactions between its components. Initial tasks were smaller, helping to build a solid foundation. Over time, an interest in algorithms and code optimization became evident, leading to more complex assignments focused on enhancing time-related functionalities and improving performance.

It is worth noting that the work built upon a solid foundation of code, making the task more manageable. The focus was primarily on coding new features, optimizing existing logic, and refactoring older code that had started to slow down as new features were added.

Initially, the logic around timesheets was relatively simple. However, as more settings and features were introduced, the system grew significantly in complexity, increasing the challenge. Given the strong understanding of the logic and rules in this field, FS tickets were often taken on. Once a new endpoint was developed, integration with the FE became easier. As a result, several FE pages and dialogs

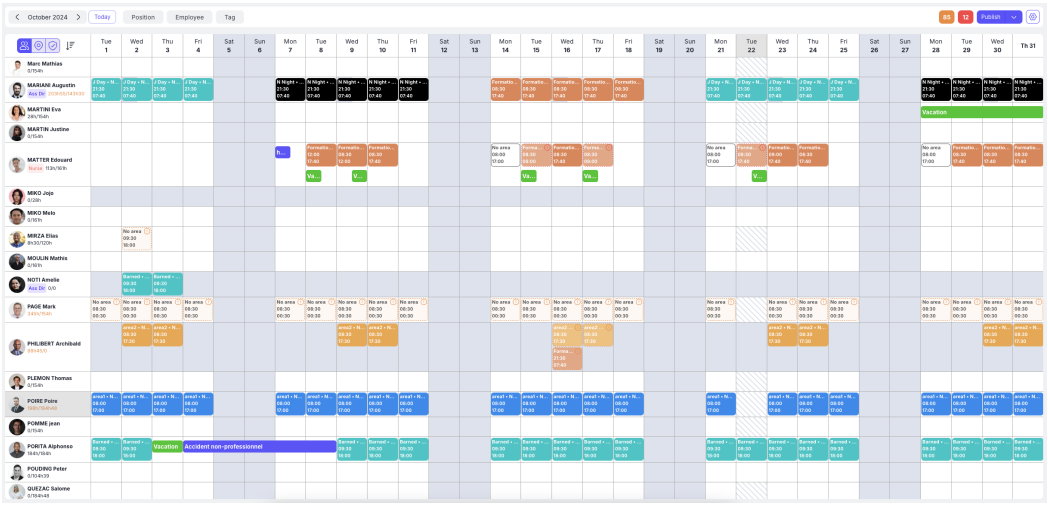

Figure 18: Planning page (employee view)

were created or refactored to better align with BE changes and improve overall system behavior and UX.

In the following chapter, these concepts will be explored further, with a detailed explanation of the contributions made to each area.

4.3.6.1 *Planning (shifts and recurring shifts)*

Shifts are one of the most important aspects of planning, as the planning module essentially revolves around shifts and leaves. The module provides three views: one for employees, one for areas, and one for coverage. The most significant improvements were implemented in the employee and area view components. For instance, in the employee view, functionality was added to allow users to create new leaves by simply right-clicking on any square in the schedule, making the process much faster and more intuitive.

The planning page (Figure 18) was mostly built using DayPilot, a library designed for scheduling interfaces, and contains a significant amount of FE logic to render these views properly. The majority of development efforts focused on BE implementation for shift management. Shifts can exist in two states: drafts or published. Drafts serve as temporary, editable versions that do not impact employees, while published shifts represent the finalized schedule that guides employees' work.

Shifts are divided into two main types: normal shifts and recurring shifts. A normal shift contains fields such as the start and end dates (typically no more than 12 hours apart), a break, an area, and an employee. Recurring shifts, on the other hand, include similar fields but are more complex due to the addition of recurring

rules. These rules allow shifts to repeat over a specific period, such as every X weeks, and include fields like excluded dates and the days of the week the shift applies to. Unlike normal shifts, recurring shifts can span several years, making them more challenging to manage.

One of the functionalities developed was an endpoint to release employees from their assigned shifts when they take a leave during a specific period. For example, if an employee requested a week-long leave, the system would remove or adjust their shifts accordingly. This functionality required handling multiple scenarios, such as splitting a shift when a leave partially overlapped with it or removing the shift entirely when it was fully within the leave period. For normal shifts, the logic was relatively straightforward. However, for recurring shifts, the process was more complex. Since recurring rules are not stored as individual shifts but as a set of repeating patterns, a mapping system was created to transform the recurring rules into individual shifts for processing. Once the necessary calculations were made, the recurring rule was updated by adding the affected days to its excluded dates table.

To illustrate how this works, imagine a scenario where an employee takes a leave during their lunch break. The system must split the shift into two: one for the morning and one for the afternoon. Additionally, the system removes the lunch break day from the recurring rule by adding it to the excluded dates table, or it might completely remove the shift in case it was just a simple shift.

This functionality turned out to be more challenging than initially anticipated due to the number of variables and rules that needed to be accounted for. It took nearly a week to fully develop and test the feature as a FS ticket. A test-driven approach was followed, with tests written even before the implementation was complete to ensure each function behaved as expected. This approach not only validated the code but also ensured that no regressions occurred later. In the year following its implementation, the functionality proved to be robust, with no bugs reported.

Toward the end of the internship, work was done to improve the performance of the system, particularly when converting recurring shifts into timesheets. Using New Relic, a performance bottleneck was identified in the process, where certain parts of the code were running with $O(n^2)$ complexity. By optimizing these calculations and reducing unnecessary loops, the performance was improved by 10 times, bringing the process closer to $O(n)$. Given how frequently this transformation is performed, this optimization significantly enhanced the UX and system responsiveness.

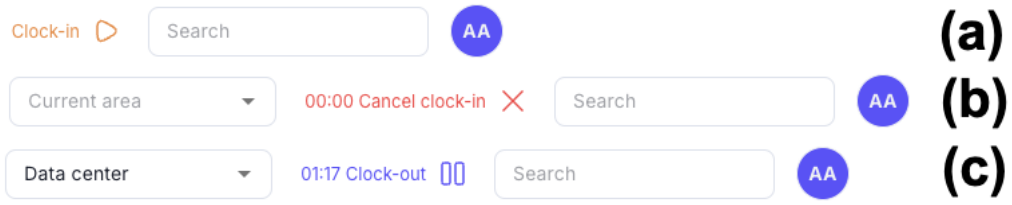

Figure 19: Clock in/out component

4.3.6.2 *Timesheets*

Timesheets are a fundamental part of tracking actual worked hours, distinguishing them from shifts, which represent planned working hours. A timesheet consists of a start date, an end date, a reference date (which assigns the timesheet to a specific day), the employee, the break, and a status that can be pending, approved, canceled, or declined.

Several new fields were introduced to the timesheets table over time to enhance functionality. These include the ‘original start date‘ and the ‘original end date‘, which were added due to the clock-in/clock-out feature. This feature evolved to support timesheet settings that allow modifications to the start and end times, making it essential to retain the original values for historical reasons. The ‘reference date‘ field was introduced to properly group timesheets per day, especially for clients with many night shifts, ensuring that data remained consistent when applying new configurations.

Further enhancements include tracking the clock-in and clock-out IPs, a feature requested by clients to ensure employees only log their time upon arriving at the office. The ‘clock in method‘ and ‘clock out method‘ field was introduced to store how the timesheet was created, whether through clock-in/out, FE input, Timemoto integration, Excel import, or by the public API. The ‘system correction‘ field allows automatic system adjustments based on specific timesheet settings, such as mandatory break, ensuring data accuracy while reflecting system-driven modifications.

A key achievement in this module was the implementation of clock-in and clock-out functionality as a FS feature. This involved integrating complex settings that govern time calculations, which will be elaborated on in the next section. The clock-in functionality is accessible through a small button in the FE, available only to users with the necessary permissions. Three distinct scenarios are handled: clocking in (Figure 19(a)), canceling a clock-in within one minute (Figure 19(b)), and clocking out (Figure 19(c)).

To enhance the clock-out process, a dedicated dialog was developed to manage breaks effectively. This dialog is triggered only when a mandatory break setting is enabled, ensuring that if a timesheet exceeds the predefined duration, users specify when the break occurred. The system then automatically splits the timesheet into two separate entries, accurately reflecting the break period.

Despite timesheets being standalone objects, they integrate into various categories, including shifts, payments, leaves, and adjustments. This consolidation allows the system to compute worked hours effectively without handling multiple independent objects. Adjustments modify an employee's balance by adding or removing minutes, while payments save compensated hours, deducting them from the accumulated balance. Timesheets represent worked hours, whereas leaves typically denote non-working periods. Other types include missing timesheets, which indicate a missing working time on specific days, and shift timesheets, which link directly to shifts. Depending on settings, shifts can also be considered worked time and not only planned.

Calculating an employee's worked time poses significant performance challenges. Since an employee's history spans multiple years, retrieving all related timesheets, shifts, recurring shifts, leaves, payments, and adjustments becomes computationally expensive, particularly in large companies. To optimize performance, unnecessary data loads were eliminated, filters were refined to fetch only essential data, and core logic was streamlined. Additional endpoints were developed to establish a shared logic layer that all endpoints leverage to manage extensive data efficiently. Additionally, the BE was modularized by splitting the existing 'TimesheetService' into three services, 'TimesheetSearchService', 'TimesheetService', and 'TimesheetClockInOutService', ensuring that each service handles only relevant logic while maintaining reusability where necessary.

Significant FE development included comprehensive refactoring of the timesheet dialog (Figure 20), a legacy component in RogerHR since its creation. The existing implementation was prone to errors and difficult to maintain, prompting a complete rewrite. The updated version re introduced the two modes: normal and simplified. In normal mode, users specify exact working hours, while in simplified mode, they enter the total worked duration without precise time ranges. Managing these modes while integrating numerous timesheet settings made this task particularly complex. Furthermore, a planned improvement involves implementing a preview endpoint to detect errors before submission, enhancing UX. The refactor also served as an opportunity to deepen the understanding of React, hooks, 'useEffect', and Yup

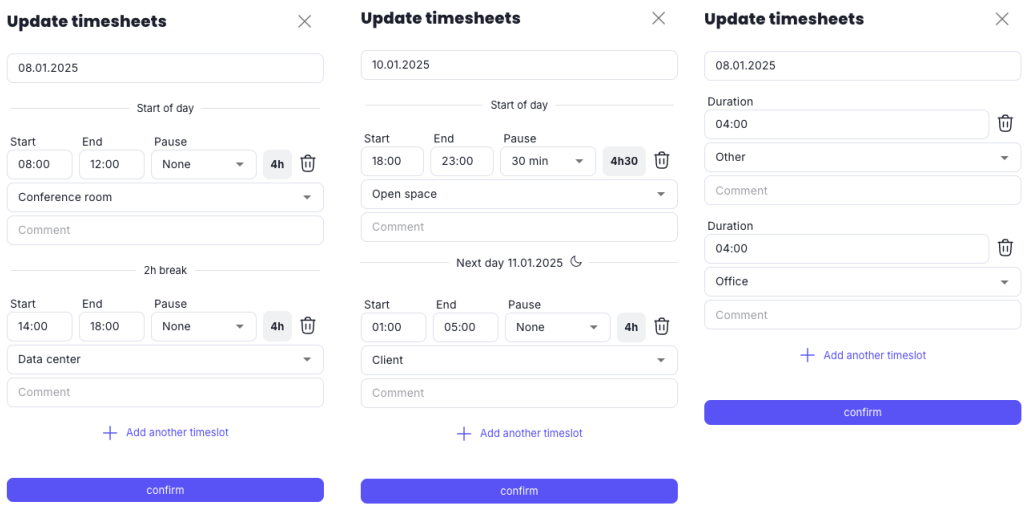

Figure 20: Timesheet dialog

validation. Additionally, it was one of the first areas where Cypress tests were introduced, which posed a unique challenge.

Another crucial page in the timesheets module is the employee timesheet history page (Figure 21), which is one of the most frequently used pages for tracking timesheets in detail. Unlike summary pages, this page provides a comprehensive daily breakdown of an employee's timesheets for a specific month. It displays key information, including the date, timecard details, planned hours (if applicable), worked time, leave duration, contractual hours, daily differences, overtime, time balance, and the status of each timesheet. Additionally, most of this data can be edited directly within the page, providing flexibility for adjustments.

At the top of the page, users can also view aggregated information on payments and adjustments, offering a complete overview of an employee's working hours and modifications. Functioning as the core data layer for the timesheets module, this page underwent significant optimization of functionality, usability, and performance to support all dependent summary views.

Another feature developed was the overtime balance card, which displays the current and projected balance at the end of the year. It also provides insights into payments and compensations, offering employees and managers a clear overview of time balances. Throughout the work on timesheets, new fields were introduced, existing logic was refactored, exports were improved, filters were enhanced, and various components were refined. Initially, contributions focused on BE improvements, but as work continued on timesheets and their settings, the transition to FS development occurred.

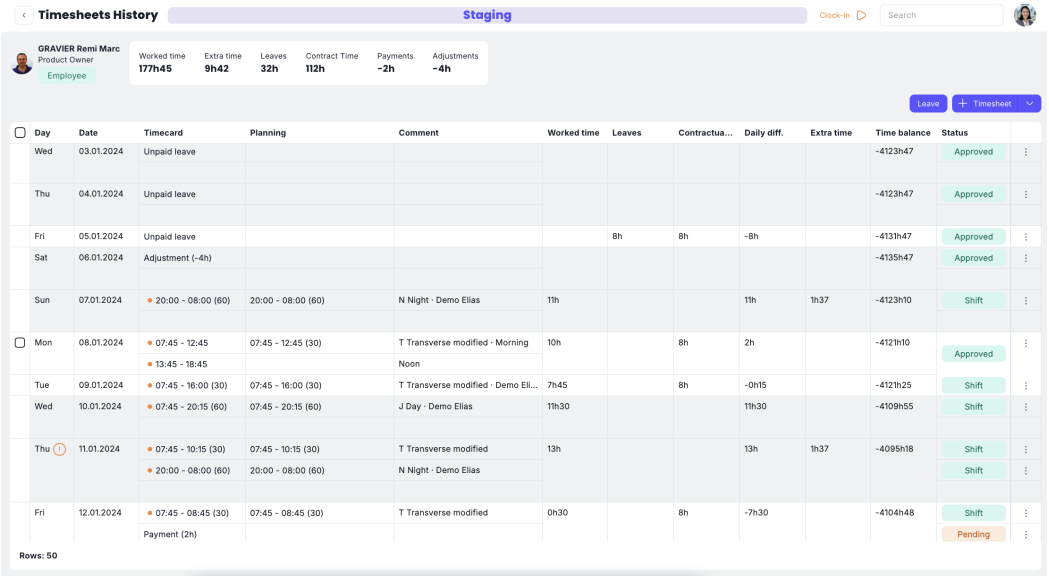

Figure 21: Timesheet history (January 2024)

The timesheets module relies on a wide range of endpoints to manage data efficiently, including those retrieving pending timesheets, annual summaries, and detailed monthly breakdowns. Complex business logic in these endpoints was handled, such as computing time balances, enforcing rules like mandatory breaks, and managing automatic corrections, making the system both robust and scalable.

4.3.7 *Timesheet settings*

Timesheet settings were one of the areas where the most time was spent, not just on the settings themselves but also on their broader impact across the system. Nearly every change in this module affected the calculation of worked time, making it one of the most critical and sensitive parts of the application.

When work began, timesheet settings were relatively simple, consisting of only six configurable options. Over time, this expanded to around 24 different settings, significantly increasing the flexibility available to users. With this growth, the FE was redesigned to ensure the UI remained intuitive and users could easily understand the available configurations. To maintain structure, the settings were categorized into five main groups: Basic, Restrictions, Bonus, Breaks, and Clock-In/Clock-Out. While some of these settings had a minor impact on the FE, the BE required extensive custom logic, making this the area where the most development time was spent.

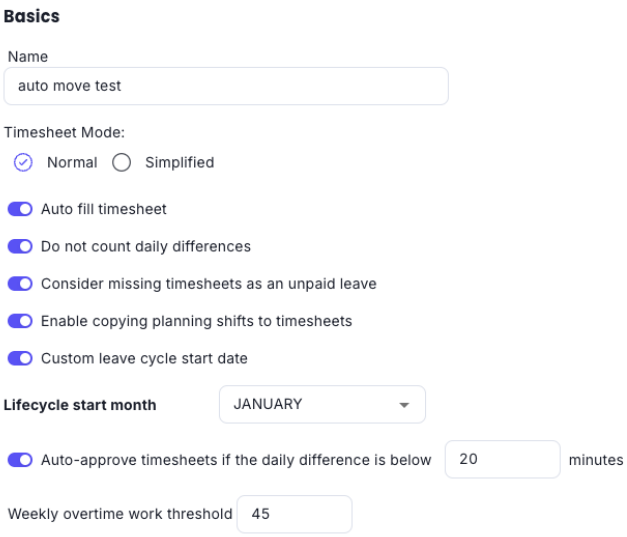

Figure 22: Timesheet setting basic menu

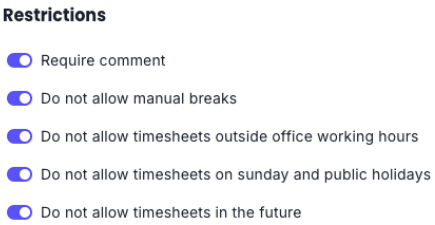

Figure 23: Timesheet setting restrictions menu

Basic settings (Figure 22) include fundamental configurations such as timesheet mode, which determines whether users enter detailed hours ("from" and "to") or just the total duration worked. Some settings, like auto-approving timesheets when the daily difference is below X minutes or treating missing timesheets as unpaid leave, automate common workflows. One of the settings worked on was copying planned shifts into timesheets, eliminating the need for employees to manually create timesheets when their schedules are predefined. Another feature implemented was the custom leave cycle start month, primarily used for leave management but later extended to also influence some timesheets pages.

Restrictions (Figure 23) determine how employees interact with timesheets. Clients requested several new options here, including requiring comments, not allowing manual breaks, and restricting timesheets to working hours, weekends, and public holidays. Unlike other settings, these are absolute rules, there are no overrides or exceptions, ensuring consistent enforcement across the platform.

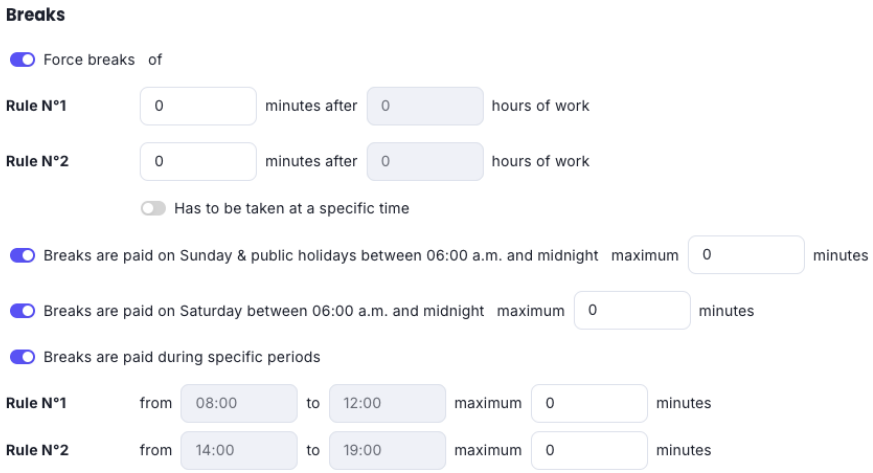

Figure 24: Timesheet setting breaks menu

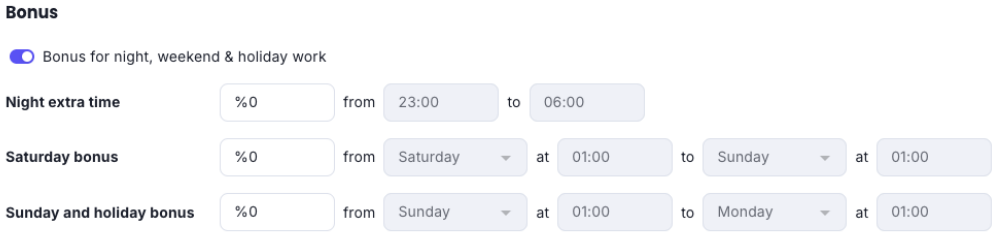

Figure 25: Timesheet setting bonus menu

The forced break (Figure 24) feature was particularly challenging. It enforces mandatory break times by automatically adjusting timesheets to include breaks after a certain number of worked hours. The logic ensures that a second break, if applicable, is always longer than the first, simplifying calculations. To distinguish system-enforced breaks from user-entered breaks, two new fields were implemented, ‘system Correction‘, which flags when the system has made an adjustment, and ‘break taken at‘, which specifies the exact time a break must occur, often used to enforce mandatory lunch breaks.

Another complex implementation was paid breaks. The system had to determine which periods qualify as paid breaks, ensure that breaks do not overlap with existing rules, and efficiently calculate paid versus unpaid break time. To achieve this, an algorithm was designed to generate a list of valid paid break periods, split timesheets into smaller segments that align with those periods, and associate each segment with its corresponding break rule. With this structure in place, the system can accurately distinguish between paid and unpaid time, ensuring reliable worked-time calculations.

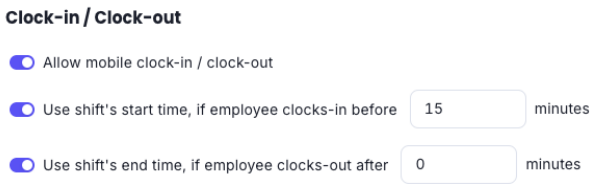

Figure 26: Timesheet setting clock in/out menu

Bonuses (Figure 25) were initially restricted to night hours, weekends, and public holidays. However, client feedback revealed the need for greater flexibility, prompting several enhancements. The system was extended to support customizable start and end times for weekend bonuses, allowing, for instance, a Saturday bonus to begin on Friday at 20:00 and continue through Sunday. To prevent calculation errors, logic was implemented to handle overlapping rules, ensuring only one bonus applies per period. When night and weekend bonuses intersected, the highest percentage was prioritized.

Introducing paid breaks added further complexity, as these breaks could also qualify for bonuses, depending on the configured rules. To accommodate this, much of the existing bonus calculation logic was refactored, ensuring seamless integration with the new paid break functionality.

The clock-in rounding rules (Figure 26) were introduced at the request of clients who wanted minor differences of one to two minutes to be rounded for cleaner data processing. To accommodate this, the system allows clock-in's that occur within X minutes of a shift's scheduled start time to be rounded to the exact shift start. Similarly, clock-out's within Y minutes of the scheduled end time are adjusted accordingly. Since these adjustments require a complete view of the timesheet, the rounding rules are applied only at the moment of clock-out, ensuring accuracy in the final recorded hours.

Working on timesheet settings required a FS approach. While most of the work focused on the BE, handling complex business logic, the FE page was also completely redesigned to accommodate the growing number of settings. The most impactful contributions included developing the paid breaks algorithm, one of the most intricate and precise calculations in the system, expanding the bonus system to provide greater flexibility for clients while maintaining system integrity, and implementing clock-in rules. Each of these features plays a crucial role in how companies manage and validate employee working hours. The work required not just coding but also a deep understanding of the system, optimization strategies, and careful planning for long-term scalability.

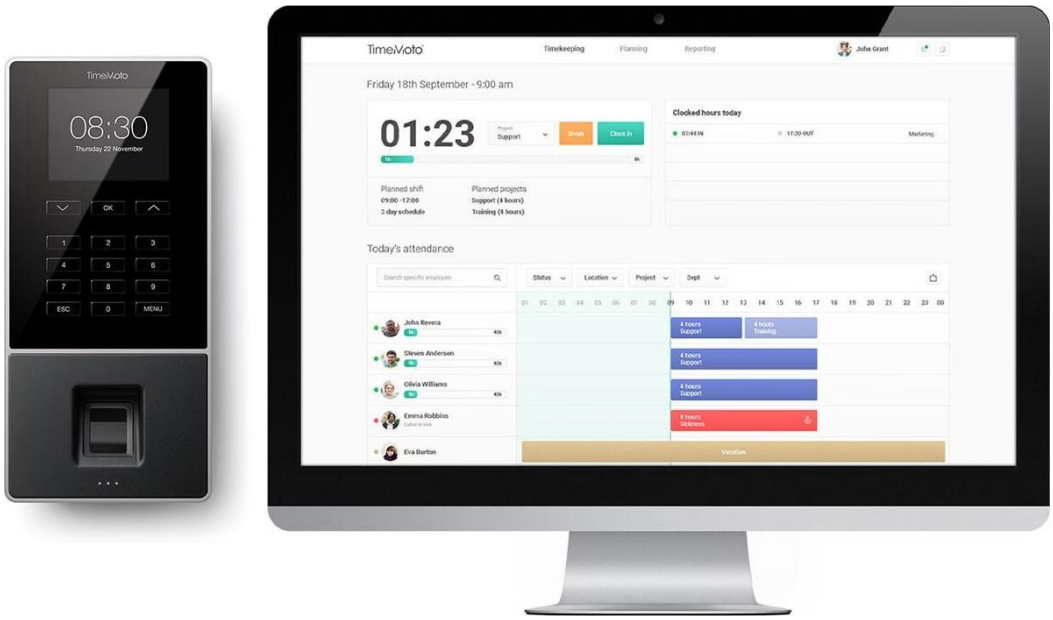

Figure 27: Timemoto machine and software

4.3.8 *Timemoto integration*

Some clients required a physical clock-in/out system so that employees could authenticate their presence at the office using a machine rather than logging into RogerHR. This functionality was achieved through integration with an external service using webhook technology for real-time clock-in/out data transmission.

Timemoto (Figure 27) was chosen due to its easy integration, reasonable price, and ability to meet the immediate needs of clients. While it may not be the best long-term solution, this implementation provided a foundation for future improvements. If demand for this feature grows, building a fully integrated solution has already been considered, but this would take time and is not currently a priority.

The most challenging part of the implementation was connecting the webhook to RogerHR and ensuring everything worked correctly. Timemoto's documentation was limited, requiring significant trial and error. Better documentation would have made the process smoother, but this challenge also helped improve debugging skills.

To ensure security, data from Timemoto is only accepted if it includes a valid token from their webhook, preventing unauthorized requests. However, one limitation of the integration is that Timemoto does not currently support automatic employee creation and updates. A workaround was implemented by manually setting up employee profiles and ensuring their email addresses in Timemoto matched those in RogerHR. If an employee's email changes in RogerHR, it must also be updated in Timemoto,

which is not ideal. If Timemoto ever supports automated user management, it will be integrated to prevent these issues.

To handle errors, a logging system was implemented to track any issues that arise, with the most common problem being unmatched email addresses. Other than that, the aim is to retain all received data, even if imperfect, rather than discarding it. This ensures that no clock-in/out records are lost due to minor inconsistencies.

Full end-to-end tests with Timemoto were not implemented, but tests covering most possible cases on the RogerHR side were written. This helped verify the integration's behavior under different scenarios.

The main benefit of this feature is that companies can now place Timemoto machines at different locations, allowing employees to clock in and out using a badge instead of logging into RogerHR. This simplifies the process and improves convenience for both employees and administrators.

This was a completely new type of task, offering valuable learning opportunities. Hands-on experience was gained with webhooks, a new service was built from scratch to handle an external integration, and problem-solving skills were developed by working with limited documentation. It was a great opportunity to test the ability to learn independently and grow as a developer.

While Damien, the CTO, initially explored the integration and provided some basic scaffolding, the details of the implementation were independently handled. Many aspects had to be figured out, making the task even more valuable as a learning experience.

4.3.9 *Public API*

The public API is designed primarily to provide CRUD functionality, ensuring clients can access and manage their data efficiently. Heavy computations are deliberately kept to a minimum, as most business logic is implemented on the client side. Almost every endpoint added is a direct request from clients, as features are not developed without a demonstrated need.

Significant contributions were made to expanding the public API, with approximately half of the current endpoints added. The work includes:

- Enhancing the leaves' endpoint;

- Creating CRUD operations for timesheet adjustments, including batch creation and updates;
- Developing the onboarding endpoint;
- Implementing CRUD for employments;
- Tracking from where the creation was made by adding metadata to indicate whether the data was created via the public API, our internal API or from timemoto.
- Building a flexible timesheets search endpoint that allows clients to filter specific data types, such as shifts and recurring shifts, while excluding adjustments, timesheets, or leaves. This required significant refactoring to ensure the API could handle these complex filtering requests.
- Adding an endpoint to fetch timesheet balances at a given date.
- Introducing a leave balance endpoint, which one client needed for a Power BI integration. Direct support was also provided to assist the client in troubleshooting connection issues.
- Adding endpoints to expose additional fields, such as location, areas, calendar, jobs, job families, cost centers, departments and timesheet settings.

One ongoing challenge is API change management. While versioning is supported and deprecation warnings are provided, no deprecated fields have been removed to avoid breaking client implementations. The aim is to improve this by implementing a notification system that alerts relevant users about upcoming changes before a major release, giving them time to adjust.

API documentation is another area for improvement. Currently, Swagger is relied upon, which works well since Java Spring automatically generates documentation based on annotations. However, because external users may not have in-depth knowledge of the system, descriptions need to be improved, and more context must be provided to make integration easier.

Although working on the public API was similar to other API development tasks, extra care was taken with documentation, usability, and backward compatibility. Ensuring external users could easily understand and integrate with the system required additional effort in writing clear comments and structuring endpoints in a way that made sense to people outside the organization.

4.3.10 *Permissions*

Permissions are a core aspect of RogerHR, ensuring that all endpoints remain secure and that employees only access data they are authorized to see. Given that this is an HR platform, permissions are critical, any misconfiguration could result in sensitive employee information being exposed. BConsequently, permission management receives rigorous attention in the system design, ensuring that every access control rule is correctly implemented and thoroughly tested.

Permissions are stored in a complex set of database tables, which track all access control rules. The system operates on two levels: company-based permissions, where an employee is granted general access to view or modify specific data across the entire company, and employee-based permissions, which determine whether an employee has access to another employee's data based on predefined rules.

The system supports two methods for defining access. The first is explicit employee-to-employee permissions, where specific employees are granted permissions over others. The second is condition-based permissions, where access is granted dynamically based on attributes such as department, location, or employment status. To enable flexibility, the system includes six different conditions for defining permissions: lifecycle status (Hired, Employed, On Long Leave, Terminated), location, department, job title, contract type (Permanent, Temporary, Internship, Contractor), and legal unit. Each condition supports "equals" and "not equals" rules, allowing granular control over which employees can access specific information.

Two of these conditions were implemented: lifecycle status and contract type. The lifecycle condition allows permissions to be assigned based on an employee's current employment status, helping distinguish active employees from those on long leave or terminated. The contract type condition enables permissions based on contract categories, ensuring that permissions are correctly assigned based on employment arrangements. The biggest challenge during implementation was learning the entire permission module from scratch. Since prior experience with the module was limited, understanding how permissions were structured and stored in the database required significant effort. Trial and error were heavily relied upon to understand how the system processed permissions. However, once the logic was grasped, the actual implementation became straightforward.

Permissions are stored as they are at a given moment, meaning they do not automatically update when an employee's details change. For example, if an employee is moved to a different department, they may lose their previous permissions and

gain new ones based on the new department's rules. This static approach ensures that access rights always reflect the employee's current status without requiring additional recalculations.

Since permissions control access to sensitive data, security was a top priority throughout the implementation. A mistake in access control could lead to unauthorized data exposure, so meticulous verification was required to ensure the permissions worked as expected. To ensure reliability, unit tests were written to validate the new permission rules, and extensive manual testing was conducted to confirm that permissions were applied correctly. One key advantage of the system is that there are no conflicting permissions. Access is always granted based on explicit rules—either a permission exists or it does not. If an employee has multiple permissions from different sources, the system simply checks whether any of them allow the requested action. Since there are no 'deny' rules in the model, conflicts do not arise.

4.3.11 *DB improvements and refactor from hibernate to JOOQ*

Since joining RogerHR, the team has been working on migrating from Hibernate to JOOQ to address performance issues. Hibernate's abstraction layer made data fetching slow, as there was little control over the underlying queries. In contrast, JOOQ allows for manual query building, resulting in significant performance gains. This migration process was complex and spanned about two months, during which both Hibernate and JOOQ were run concurrently. Many hours were spent debugging regressions that arose from the transition, especially in areas handling large amounts of data, such as timesheets, shifts, leaves, and reports, where even the smallest issues could lead to missing data that Hibernate's lazy loading had previously handled automatically.

One of the key benefits of moving to JOOQ was the ability to refactor many parts of the codebase to perform filtering at the database query level rather than in Java, greatly improving efficiency. A rule was adopted that any change on a table still managed by Hibernate should be migrated to JOOQ, as this was inevitable for long-term performance improvements. This approach allowed for optimized queries, resulting in more effective data retrieval and processing.

In addition to the migration, responsibility was taken for executing around 70 database migrations. These migrations involved adding new tables, reformatting data storage methods, introducing new indexes, and appending numerous fields to existing tables. Although these tasks might seem straightforward, many of the

migrations were particularly challenging because any disruption could impact clients. To mitigate risks, extensive testing was conducted in staging and test environments, and changes were validated in a pre-production setup with backups from production.

Overall, the transition to JOOQ and the accompanying database migrations have provided significantly better performance and greater control over data, ensuring the platform can scale effectively while maintaining the high standards clients expect.

4.3.12 *FE refactor*

During the time at RogerHR, a major FE refactor was undertaken to improve the developer experience and make the codebase easier to maintain. The previous structure was messy, making it difficult to locate files. With the new organization, everything became more structured, improving navigation, especially for new developers.

A key part of this refactor was introducing a dedicated API layer in the FE, centralizing all API calls. This allows the fetching library to be changed without modifying every component. Permission handling was also improved by introducing an endpoint that returns all employee permissions in a single request. This not only reduces unnecessary BE calls but also enables permissions to be checked before making a request, preventing unauthorized API calls.

Component structure was standardized, always using YUP for validation and following a clear order: loading state, YUP model, state variables, custom logic, and JSX. Larger components were split into smaller, reusable parts when possible.

This refactor took 1–2 months, with changes applied gradually. Active contributions were made throughout the process, migrating components and improving existing logic. By the end, a more maintainable and scalable FE architecture was achieved, enabling faster development and easier updates.

4.3.13 *Empty time without tickets*

At RogerHR, a simple rule is followed: if a task can be completed in under two hours, it is tackled immediately. Each team member is expected to estimate task durations, ensuring efficiency and responsibility with time. When there are no pressing tickets or collaborative discussions scheduled, the focus shifts to bug fixing and code improvements.

Regularly reviewing PR's is a key part of the daily routine. This practice not only maintains code quality but also broadens understanding of modules not extensively worked on, enabling better maintenance and contribution to the system as a whole. Bug fixing, driven largely by issues identified in Bugsnag, ranges from promptly resolving null pointer exceptions to addressing more critical bugs that prevent data from being saved or updated or that break pages entirely. These fixes are prioritized based on their impact on functionality.

Occasionally, New Relic is monitored to identify slow API calls. When performance issues are detected, trace logs are added to pinpoint bottlenecks and optimize queries, further enhancing system responsiveness and UX.

5

CONCLUSION

The RogerHR internship provided intensive FS development experience within a fast-paced startup environment, significantly enhancing critical HR systems including timesheets, shift planning, and leave management modules. Beginning with limited React knowledge, the experience involved rapid skill acquisition in both FE with React and BE with Java Spring technologies, ultimately producing clean, maintainable code designed for collaborative development. This accelerated learning environment not only fostered exceptional technical adaptability but also demonstrated the crucial balance between innovation and system stability. The hands-on experience with complete feature implementation - from database design to UI components - translated academic foundations into practical software engineering competencies while emphasizing the importance of robust, scalable solutions in production environments.

Transitioning from academic projects to a production-level codebase required rethinking problem-solving approaches. Under Damien's mentorship, the focus shifted toward performance optimization, PostgreSQL query efficiency, and architectural foresight—skills that significantly improved code quality. Relocating to Switzerland introduced cultural and linguistic complexity, further enhancing collaboration abilities. Through Agile workflows and rigorous code reviews, the importance of intentional design and readability in building scalable, user-centric software became evident.

Beyond technical growth, this internship reinforced the importance of user-centric design and collaboration in software development. It provided valuable insights into best practices, including code quality, security, and CI/CD pipelines, while regular team discussions and code reviews improved maintainability and scalability. Working in RogerHR's fast-paced environment also highlighted the balance between speed and stability—an essential skill in real-world software development.

More than just a technical experience, this internship solidified a passion for solving real-world problems through empathy-driven design. Software became more than just code—it served as a bridge between functionality and human needs, where intuitive interfaces and robust logic coexist. The challenges faced fostered

adaptability, problem-solving skills, and the confidence to navigate complex projects. Ultimately, all proposed objectives were successfully achieved, but the learning extended far beyond them. This experience was an opportunity to contribute to a meaningful product, develop professionally, and gain the resilience to thrive in evolving environments, where collaboration fuels innovation.

For future work, there are three key areas that need to be prioritized: technical optimization, interface refinement, and feature expansion. System reliability will improve through expanded test coverage and performance enhancements, including query optimization and Redis caching for faster API responses. FE updates will address UX issues while adopting a modular component architecture with Storybook and Cypress for consistency and testing. Planned features include recurring leave management, multi-currency expense tracking, real-time HR analytics, and improvements in the reviews.

RogerHR served as both a professional environment and a career accelerator, establishing foundational development principles that extend beyond technical implementation. The experience instilled critical practices, including code structuring for maintainability and database optimization for production-scale demands—principles that now form core engineering values. Most significantly, this opportunity reinforced the importance of building tools that meaningfully empower users, demonstrating how technical excellence and human-centered design can synergize. These insights will continue to inform approaches to software engineering challenges throughout future professional endeavors.

BIBLIOGRAPHY

AG Grid: High-Performance React Grid, Angular Grid, JavaScript Grid (n.d.). [Online; accessed 2025-01-19]. URL: https://www.ag-grid.com/.

API Documentation & Design Tools for Teams | Swagger (n.d.). [Online; accessed 2025-02-11]. URL: https://swagger.io/.

App Monitoring, Error Tracking & Real User Monitoring | Insight Hub (n.d.). [Online; accessed 2025-02-10]. URL: https://www.bugsnag.com/.

Code Quality, Security & Static Analysis Tool with SonarQube | Sonar (n.d.). [Online; accessed 2025-02-11]. URL: https://www.sonarsource.com/products/sonarqube/.

DayPilot - HTML5 Calendar, Scheduler and Gantt Chart Web Components (n.d.). [Online; accessed 2025-02-10]. URL: https://www.daypilot.org/.

Docker: Accelerated Container Application Development (n.d.). https://www.docker.com/. (Accessed on 05/17/2024).

Drake, Joshua D and John C Worsley (2002). *Practical PostgreSQL*. " O'Reilly Media, Inc."

Figma: The Collaborative Interface Design Tool (n.d.). [Online; accessed 2025-02-15]. URL: https://www.figma.com/.

GitHub · Build and ship software on a single, collaborative platform · GitHub (Jan. 2025). [Online; accessed 2025-02-15]. URL: https://github.com/.

Graça, Herberto (n.d.). *DDD, Hexagonal, Onion, Clean, CQRS, ... How I put it all together – @hgraca*. https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/. (Accessed on 07/19/2024).

JavaScript | MDN (n.d.). https://developer.mozilla.org/en-US/docs/Web/JavaScript. (Accessed on 04/30/2024).

JetBrains (June 2021). *IntelliJ IDEA – The IDE for Professional Development in Java and Kotlin*. [Online; accessed 2025-02-15]. URL: https://www.jetbrains.com/idea/.

jOOQ: The easiest way to write SQL in Java (n.d.). https://www.jooq.org/. (Accessed on 04/28/2024).

JUnit 5 (n.d.). [Online; accessed 2025-02-11]. URL: https://junit.org/junit5/.

Manifesto for Agile Software Development (n.d.). https://agilemanifesto.org/iso/en/manifesto.html. (Accessed on 05/10/2024).

Martin, Robert C. and James O. Coplien (2009). *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ [etc.]: Prentice Hall. ISBN: 9780132350884 0132350882. URL: https://www.amazon.de/gp/product/0132350882/ref=oh_details_o00_s00_i00.

MUI: The React component library you always wanted (n.d.). [Online; accessed 2025-01-19]. URL: https://mui.com/.

PostgreSQL: The world's most advanced open source database (n.d.). https://www.postgresql.org/. (Accessed on 04/27/2024).

React (n.d.). https://react.dev/. (Accessed on 04/30/2024).

RogerHR, la plateforme RH qui écoute les équipes (n.d.). https://blog.genilem.ch/rogerhr-la-plateforme-rh-qui-ecoute-les-equipes/. (Accessed on 06/07/2024).

Schwaber, Ken and Jeff Sutherland (2010). "What is scrum". In: *URL: http://www.scrumalliance. org/system/resource/file/275/whatIsScrum. pdf,[Sta nd: 03.03. 2008]*.

Shrivastava, Anchit et al. (2021). "A systematic review on extreme programming". In: *Journal of Physics: Conference Series*. Vol. 1969. 1. IOP Publishing, p. 012046.

Smith, Gregory (2010). *PostgreSQL 9.0: High Performance*. Packt Publishing Ltd.

Spring | Projects (n.d.). https://spring.io/projects. (Accessed on 04/27/2024).

spring-reference.pdf (n.d.). https://docs.spring.io/spring-framework/docs/2.0.x/spring-reference.pdf. (Accessed on 04/27/2024).

Stonebraker, Michael and Lawrence A. Rowe (June 1986). "The design of POSTGRES". In: *SIGMOD Rec.* 15.2, pp. 340–355. ISSN: 0163-5808. DOI: 10.1145/16856.16888. URL: https://doi.org/10.1145/16856.16888.

Testing Frameworks for Javascript | Write, Run, Debug | Cypress (n.d.). [Online; accessed 2025-02-11]. URL: https://www.cypress.io/.

TypeScript: The starting point for learning TypeScript (n.d.). https://www.typescriptlang.org/docs/. (Accessed on 04/30/2024).

What is Java and why do I need it? (N.d.). https://www.java.com/en/download/help/whatis_java.html. (Accessed on 04/27/2024).

What is Scrum? | Scrum.org (n.d.). https://www.scrum.org/resources/what-scrum-module. (Accessed on 05/10/2024).

Your relational data. Objectively. - Hibernate ORM (n.d.). https://hibernate.org/orm/. (Accessed on 04/28/2024).

DECLARATION

I solemnly declare, on my honor, that the work presented in this dissertation, titled *"RogerHR Internship"*, is original and was carried out by Francisco Simplicio Melícias (2222860) under the supervision of Professor Doutor Leonel Santos (leonel.santos@ipleiria.pt).

Leiria, March 2025

Assinado por: **FRANCISCO SIMPLICIO MELICIAS**
Num. de Identificação: 15962947
Data: 2025.03.29 15:33:14 +0100

Francisco Simplicio Melícias