Professional Growth of a User Interface Developer

Linnea Möller

Haaga-Helia University of Applied Sciences
Bachelor of Business Information Technology
Bachelor's Thesis
2025

Abstract

Author
Linnea Möller

Degree
Bachelor of Business Information Technology

Thesis Title
Professional Growth of a User Interface Developer

Number of pages and appendix pages
72 + 0

This thesis explored professional growth of a software developer through diary entries. The work entailed software development and -testing in a user interface project called Nebula. Ten consecutive weeks were covered in July-Oct 2024 where the authors goals and software development tasks were monitored. Furthermore, the thesis covered a description of the initial situation, ending in a discussion regarding the observation weeks findings.

The thesis purpose was to develop and analyze the author's professional competence as well as describe related theoretical framework. Topics covered during observation weeks are software development, software testing, user experience, REST APIs and ICT project management.

The thesis began with an introduction and initial assessment. The introduction presented the thesis and outlined main sources and concepts of the study. The initial assessment analyzed the initial situation and workplace. The thesis did not include the company or partner names due to confidentiality obligations. Furthermore, project stakeholders were introduced, examining the authors interactions with them.

Each observation week began with weekly goals, following with daily diary entries. The diary entries included project related work tasks, excluding all unrelated work. Every observation week was concluded with a weekly analysis, where the author analyzed weekly observations and findings, reflected against professional sources and concepts.

In the final part of the thesis, the author reflected on her professional growth during the observation period with an emphasis on original goals. The author was able to achieve her goals in TypeScript, Robot Framework, debugging and project management skill improvement. However, the author did not achieve her goal regarding BPMN development nor did she have time to focus on writing clean code. Furthermore, the author reflected over how she will utilize her improved debugging skills and focus on refining her TypeScript skills. Lastly, the author reflected on her future development goals and described how she intends to continue building her skills. The author concludes that growth comes not only from doing, but from taking the time to reflect, learn, and intentionally evolve with each experience.

Keywords
Software development, user interface, user experience, software testing, debugging, Robot Framework, React, JavaScript, TypeScript, REST API, ICT project, project communication

Contents

Abbreviations and definitions

Abbreviation	Term	Definition
API	Application Programming Interface	Created by developers to allow other applications communicate with the application programmatically. It is the rules that must be followed to communicate with the application in question. (IBM n.d.)
AWS	Amazon Web Services	A cloud computing platform provider.
BPMN	Business Process Model and Notation	A visual modeling language that defines business process workflows (Visual Paradigm n.d.)
CSS	Cascading Style Sheets	A stylesheet language that defines how HTML and XML elements are displayed across various media (MDN n.d.b).
Git	Global information tracker	Git is a version control system that tracks changes to file(s), allowing you to recall specific versions (Chacon & Straub 2024, Chapter 1.1)
ISAE	International Standard for Assurance Engagements	Provides a framework for assurance engagements, consistency, quality, and reliability (Martech Zone n.d.)
JSON	JavaScript Object Notation	A human readable lightweight data-interchange format which machines parse and generate (JSON n.d.)
MVP	Minimum Viable Product	The minimal version of a product built for first market sale (Nidhi n.d.)
REST	Representational Rest Transfer	Software architecture that enforces the conditions on *how* an API should work (IBM n.d.)
REST API	Representational Rest Transfer Application Programming Interface	A flexible and lightweight interface that integrates applications and connects components (IBM n.d.
UI	User Interface	Point where humans interact with computers using visual and audio elements possibly through keyboard, display, mouse and desktop appearance. (Hashemi-Pour & ChurchVille 2024)

UX	User Experience	The ISO 9241-210 (2010) standard defines UX as a “person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service”.
VPN	Virtual Private Network	Microsoft Azure (n.d.) defines a VPN as “a digital connection between your computer and a remote server owned by a VPN provider, creating a point-to-point tunnel that encrypts your personal data, masks your IP address, and lets you sidestep website blocks and firewalls on the internet”.

1 Introduction

In this diary type thesis, we will follow my professional growth as a software developer. Currently, my main work assignments consist of frontend development in an internal user interface (UI) project called Nebula. I develop new features, debug and test the UI as well as create REST API connections to external providers. Furthermore, I create automated tests using Robot Framework and documentation in Atlassian Confluence. The role requires skills in React JavaScript and TypeScript, Cascading Style Sheets (CSS), Robot Framework, Git, Jira, Microsoft Office and Amazon Web Services (AWS).

The observations will be done during ten consequent weeks in July-Oct 2024, and each week is completed with a weekly analysis. At the start of the observation weeks the project is at its final stage before launch, the approval testing phase. The weeks will also cover the week of the launch of the MVP version 1.9.2024, as well as the launch of the MVP Plus version 1.10.2024.

My objective during these 10 weeks is to refine my skills in TypeScript further by defining missing types and writing clean code according to high quality coding standards. I plan to write tests with Robot Framework; to utilize the skills I learned during a testing course last spring. Furthermore, I will most likely develop my debugging skills, as malicious testing usually discovers hidden bugs. I also hope that I will develop my skills in drawing new processes with the BPMN tool, as we desperately need more team members with skills in that area. Lastly, I will focus on improving my project management skills, as my work duties strongly revolve around the project.

The company itself offers outsourcing services to companies, such as HR, payroll, accounting, advisory, and financial management, technology and software services, with a strong presence in Northern Europe. The Nebula MVP project is focusing on developing the payroll process, and new projects will be established for each other business operation as well. Flexible workhours and working location are a part of the company policy. The automation team is based in Helsinki but has team members in other Finnish cities as well as Tallinn, Estonia. Therefore, we stay connected to each other remotely, using Microsoft tools. The team has a voluntary office day on Wednesday’s, in which I like to participate for team relationship building purposes.

The knowledgebase required for my work and explored in my thesis encompasses software development, software testing, user experience, REST APIs and IT project management. The theoretical framework for my analysis consists of books, articles, blogposts and websites. This diverse combination of sources provides a solid foundation for evaluating both my professional development and applied software solutions.

Themes covered in software development include problem-solving, clean code, refactoring, debugging, error handling and Git. The discussion on improving problem solving and debugging skills is based on Humble's (2024) method presented in his book *Improve Your Problem-Solving Skills.* Bug classification follows the official ISTQB Glossary. Refactoring and clean code are explored through Fowler's. and Beck's (1999) book *Refactoring: improving the design of existing code* and Contieri's (2023) book *Clean Code Cookbook.* Error handling is examined using articles and online resources*.* The use of Git commands in software development is based on technological websites.

Themes covered in software testing include various test types, testing techniques, test planning and test case design. The discussion is based on Sommerville's (2016) book *Software engineering,* Homes' (2012) book *Fundamentals of Software Testing,* along with insights from different technological websites. The benefits and limitations of automated testing, as well as the impact of defective applications, are analyzed using Dustin's, Rashka's, and Paul's (1999) book *Automated Software Testing: Introduction, Management, and Performance*. A closer examination of the Robot Framework is based on its official documentation. The successful implementation of automated tests follows on Hegde (2019) blogpost *How to Develop a Test Automation Strategy*. Other types of tests, as well as test planning and test case design, are explores using insights from technological websites and Bisht's (2013) book *Robot framework test automation*.

User experience is analyzed based on de Voil's (2020) book *User Experience Foundations* as well as Nielsen's research from 1993 on website response-time limits and later article in 2010.

REST API coverage is based on different technological websites such as AWS, IBM and GeeksforGeeks. REST API optimization is further examined based on Pragada's (2024) article *API Optimization for React Apps: Minimizing Data Fetching Overhead.*

IT project management is discussed at a general level, with a deeper focus on project closing, the importance of communication, and project scope. The definition of a project is based on Brewer's and Dittman's (2023) book *Methods of IT Project Management.* The importance of project communication draws from Minois' (2023) blogpost *The Importance of Communication in Project Management* and Schwalbe's (2019) book *Information technology project management.* Project scope is examined using Landau's (2023) blogpost and additional online resources. The project closing phase is based on the *PMBOK Guide* (Project Management Institute, 2017).

This thesis utilized ChatGPT to structure some sentences into coherent text and to identify correct English terms by explaining the meaning to the artificial intelligence (AI). Additionally, references were collected and generated with Mendeley.

2 Description of the initial situation

Before diving into the observation weeks, it is beneficial to provide some background of me and my employer. I have worked at the company since October 2019 as a Payroll Analyst. I started a slow transition to my current main role as a Junior Software Developer in September 2023. Approximately 80% of my working hours are spent in Nebula project frontend development, and the rest of the time in payroll or meeting participation. Therefore, I have chosen to set this thesis scope to cover only the Nebula software development. Nebula is an internal user interface project that combines the usage of integrations, robotic processes and a Business Process Model and Notation (BPMN) tool. This chapter provides an analysis of my current responsibilities, skills and contributions within the Nebula project. It introduces the key project stakeholders and highlights the different interactions scenarios, emphasizing the critical role of effective communication in fostering collaboration, ensuring progress, and achieving project success.

2.1 Analysis of your current work

Personal development

I would rate myself to be a skilled performer as I am able to perform my job independently and rarely need to ask for help. In addition, I can see what needs to be done in a prioritized manner and take initiative. However, this only covers my frontend developer role in the Nebula project. The next step in my career is to work on my Python skills and take an active role in Python integration development. I will receive my own small internal integration project in January 2025 after which the difficulty level of the projects is increased bit by bit. In our team the Python integration developers also take turns in managing Freshdesk tickets on a weekly basis. These cover tickets that are sent by people in our organization as well as integration error notifications. I will be starting to shadow these at the end of the year. I am a novice when it comes to the Python integration development role. As I haven't yet coded any integration with Python, nor have I had to manage any tickets other than those assigned to me, I need a colleague's instructions and support to complete these tasks. The language itself is thankfully familiar, as I have completed Python courses at Haaga-Helia.

Nebula project role

My role in the project is frontend development. I am programming the frontend with React TypeScript and automated tests with Robot Framework. In addition to these the role requires skills in Cascading Style Sheets (CSS), Git, Jira, and some cloud computing in AWS. I have learned basic programming skills as well as both frontend with React and Robot Framework at Haaga-Helia courses. I was

chosen to the Nebula project team based on my React skills, as it was a new programming language for the automation team.

Skill analysis

As a Junior Software Developer in the Nebula project, my role primarily involves frontend software development. At a broad level, this includes general development practices such as creating and maintaining user interfaces, whilst ensuring seamless functionality, and contributing to the overall quality of the software through testing and collaboration. My work has centered around frontend development using React and TypeScript, but I plan to explore Robot Framework automated software testing and AWS Cloud computing during the observation weeks. By practicing these skills among others, I aim to strengthen my professional competence while addressing key areas for success within the project.

Before the project I didn't have much experience of TypeScript but have learned a lot with the support of my senior colleague. TypeScript (n.d.) documentation defines it as "a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale". I still need to practice TypeScript by creating interfaces and types, which I hope to do during the thesis observation period. Otherwise, I do feel confident in my React skills as I have proved to myself continuously that I can code with React independently. React is a web and native user interface library with which it is possible to build user interfaces with building blocks called components (React n.d.).

My Robot Framework skills are at a total beginner's level, as my only experience is from a Haaga-Helia Software Testing course. Robot Framework is an open-sourced, Python-based, keyword-driven test automation framework that is ideal for acceptance testing (Robot Framework n.d.). I aim to create a testing plan and the tests themselves during the thesis observation period. Hopefully, with a bit of training, I will reach an intermediate level. Until this day we have done all testing manually. There are a lot of test cases for which it would be excessive to have automated test created. However, there are some parts of the software that have broken several times without the developer noticing, and for those the automated test will prevent the occurrence of returning bugs and thereby increase the quality of the product and customer trust.

During this Nebula project I have become quite the detective when it comes to styling with Cascading Style Sheets (CSS). CSS is a stylesheet language that defines how HTML and XML elements are displayed across various media (MDN n.d.b). As we are using many ready React components such as MUI and display forms created in the BPMN tool, the overriding of existing styles has been quite demanding and time consuming. I have become faster in resolving the styling issues though and

have succeeded in creating a consequent design in accordance with the customer defined requirements.

My Git skills have developed a lot during my working journey as a software developer as well. Git is a version control system that tracks changes to file(s), allowing you to recall specific versions (Chacon & Straub 2024, Chapter 1.1). I started out afraid of doing any merges with other branches, as I have bad experiences from merge conflicts. Now I can resolve merge conflicts with confidence and have learned about other version control commands such as stashing and the usage of pull requests. Nevertheless, I do feel like I have a lot to learn in this area still. The best way to learn is to make mistakes and learn from them. Google is a great help, and I have found that artificial intelligence is a great support when you have messed up in git and need to undo your mistake.

We use Atlassian Jira to organize and plan the different development topics, as well as assigning them to different developers. Atlassian Jira is an agile project management tool with which it is possible to plan, track, release and support software projects (Atlassian n.d.a). Every branch always starts with the Jira tickets code, which also enables the automated status update when a branch is merged into the main branch. I have only used Jira in my role as a software developer, and I must admit that I have yet only scratched the surface. I will take a course in the project management tool once I find the time for it. But for the time being I will continue developing my Jira skills by simply using it.

It is not mandatory for me to have skills in Amazon Web Service (AWS) Cloud computing in my junior role. I only need to know how to approve a new release to production at the AWS Code Pipeline. Cloud computing delivers IT resources over the Internet with pay-as-you-go pricing, allowing access to services such as storage, computing power and databases without owning or maintaining physical infrastructure (AWS n.d.a). As cloud computing is scarce within my team, I find it beneficial to add it to my professional skill palette. I do not yet have much knowledge in this area but will take the AWS Cloud Architecture course at Haaga-Helia this autumn.

2.2 Stakeholders

For a project to succeed, it is important to identify stakeholders and consider their motivations and interests. Stakeholders are the persons who can affect or are affected by the project. (Eskerod & Lund Jepsen 2016, chapter 1) Each stakeholder has their unique expectations and priorities that need to be understood and managed to ensure the project’s progress and success. I interact with a variety of stakeholders, originating from our company, customers and different service providers. As my job is related heavily to the Nebula project, I will present the project stakeholders from my point of view.

My project role involves many different stakeholders, which are visualized in Figure 1 below. First, we have my team members (Implementation team), which consists of our Project Manager / Team leader, Technical Lead and different developers. These are naturally the stakeholders that I am most in touch with, as we connect daily in morning meetings. In addition, we communicate via Teams chat and extempore Teams calls.

One could argue that the company itself is the most important stakeholder, as they are the Nebula project investors. Ensuring that they are content and kept up to speed is crucial for the project to succeed. They have the power to “pull the plug” and we need to ensure that their investment becomes profitable over time. The business stakeholder group consists of the management level as well as the appointed contact person.

The end users are another important stakeholder group. A successful project requires that the end-product is used. If the end-product is difficult to use or the end users do not see the added value, they won’t use it. This would result in losing the invested time and money, i.e. a failed project. For the current BPMN process it is the payroll department employees that are the end users. We will create other BPMN processes in the future, which will expand the end user group to consist of other departments as well.

Another stakeholder is the BPMN tool provider. We develop the BPMN process by buying the product from the provider and developing the processes ourselves. The provider is responsible for everything that has to do with the tool itself, such as creating production, testing and development environments.

The user interface is built on top of a Cloud Solution, which is provided by Amazon Web Services (AWS). We have a Software as a Service (SaaS) cloud computing model, which means that AWS provides the application software, and we manage affairs such as customer access, account management and resource provisioning (AWS Marketplace n.d.). We have one contact person at AWS whom we can contact whenever we need help with any of the setups for the Nebula website.

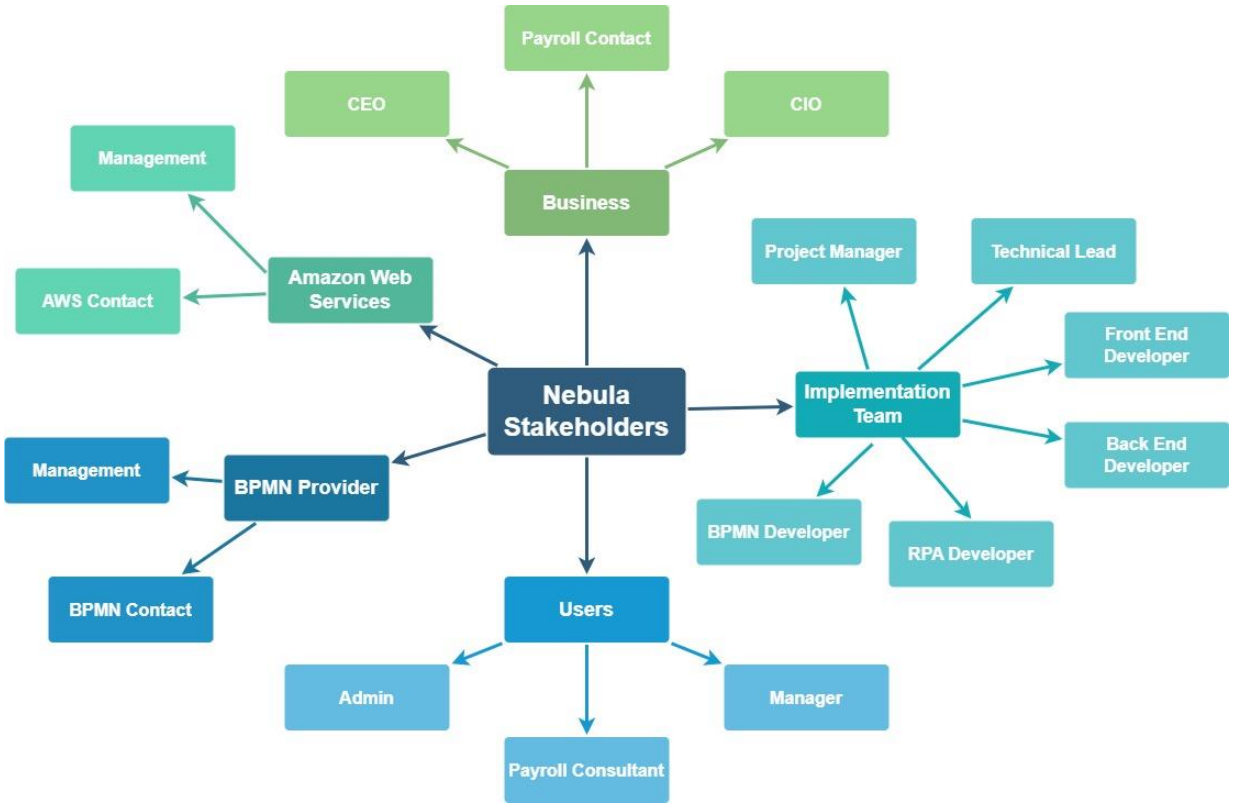

Figure 1 Stakeholders

2.3 Interaction situations

Interactions consist of communication situations and is affected by the communication climate. A healthy and open communication climate fosters collaboration and learning in a workplace. The organizational communication climate affects the threshold to ask colleagues for help, sharing knowledge, voicing improvement suggestions and expressing disagreement. (Kvalnes 2023, 5-7) Every interaction, no matter if it is in written or spoken form, is important. Clear and frequent communication reduce the risk for misunderstandings. I find it beneficial to think of every interaction as a customer service interaction. As a business, we offer services that are dependent on each other, and one customer's question might require the work effort of multiple departments. Therefore, we need to make ourselves easily approachable, acknowledge a colleague's message in a friendly manner and give time estimations when needed. Even though a colleague isn't a customer, we need to ensure that we communicate in a positive manner, avoiding spreading negativity. Destructive communication can in the worst-case scenario affect the other party so negatively that they aren't fit to continue their workday.

In our company the importance of customer experience is emphasized in the form of customer service trainings and workshops. In my opinion, we have a healthy and open communication climate that fosters collaboration and learning in the workplace. We collaborate over department lines and

ensure together that we exceed the customers' expectations. We also share knowledge with each other, creating a shared knowledge base. Never have I felt that someone would withhold information only to gain a personal benefit. I would estimate that the threshold to ask for help is also low. At least within my team, because I can both ask for help when I need it and get help requests from my colleagues frequently. By fostering an open and healthy communication climate, combined with a shared mindset of creating the customer experience together, we lay the foundation for success.

Communication within the team is essential for fostering innovation, sharing knowledge, and building trust among team members. Clear and consistent communication ensures that tasks are well-coordinated, helping us avoid duplication of efforts or the risk of tasks being overlooked altogether. It is also important to distribute everyone's workload evenly and ensure that no one is overwhelmed by the workload while others remain underutilized. Furthermore, regularly communicating task progression and possible challenges allows the team to identify possible bottlenecks early on and adjust plans accordingly. Effective communication not only improves collaboration but also strengthens accountability, as everyone understands their roles and how their work contributes to the team's shared goals.

In the Nebula project we interact with the customer on weekly project meetings as well as in a designated project group chat. It is important to communicate with the customer in a clear and timely manner. One must remember that they do not have a technical background, and therefore the language needs to be stripped of too technical terms. It is also better to communicate excessively than to keep the customer in the dark. They need to know whether the project is proceeding according to the schedule and whether any adjustments need to be made. Any time the customer has a question we must react to it as soon as possible, only to communicate that it has been noticed. Whenever we cannot answer the question right away, we need to communicate that we will investigate it and give a time and date when we will give the next update. In my experience the customer experience is increased if you give a new update before the given deadline, as you are then exceeding the customers' expectations.

We do not communicate directly with the end users, as everything goes through the customer contact person. We have a user test group that gives feedback to the customer contact person, and they forward it to us. It is important to take their feedback into account, whilst keeping in mind that we do not have the resources to accomplish every improvement at once.

We are the customers in the relationship with the BPMN tool provider and AWS. As the BPMN tool is a central part of our whole Nebula payroll process, it is crucial for us to get answers and change request completed in an effective manner. Whenever the provider doesn't deliver on time our schedule suffers, and we need to explain to our customer why the project is delayed.

3 Diary entries

The diary observations span 10 weeks, beginning in late summer July 29, 2024, and concluding in early autumn on October 4, 2024. During these ten weeks we will follow my journey as a software developer and witness both the highs and lows along the way. During this time. the Nebula project will transition from the final testing stage to its official release.

Nebula has two types of spaces: DataSpace and WorkSpace. DataSpace consists of tables generated with React DataSheetGrid that display data used by robotics, integrations and the BPMN tool. WorkSpace is as the name suggests the users working space. Here the BPMN process is projected in the UI in the form of tables created with React MUI, forms and buttons. DataSpace and WorkSpace will be mentioned in the diary entries to differentiate work tasks.

3.1 Observation week 1

The main goal for this week is to work on the Nebula project's Minimum Viable Product (MVP) version in preparation for the test users' launch in September. This will involve resolving identified bugs and performing thorough testing on the current version. Simultaneously, we will need to maintain clear communication with the stakeholders.

Monday

Returning from vacation, I caught up on emails and Teams conversations and attended two key Nebula project meetings. The first, with the project manager, customer representative and the development team, covered recent progress and upcoming milestones, emphasizing a controlled release process to maintain testing integrity. The second, a technical meeting, focused on implementation details, including reminding the customer that user-created filter views were stored in the cookies rather than the database.

I had a few issues on my worklist, but decided to work on the filters cache bug, as it had been marked as an MVP issue. The issue was that an edited, unsaved filter disappeared upon page refresh. It took a while for me to figure out how to solve the filter cache bug but decided in the end to create a new cookie in which the user edited filter was stored. Then I refactored the code so, that in case the user edited filter cookie is empty, the active view in the pre-created default filter views is shown. As soon as the user starts to edit the filter, is it added to the edited filter view cookie. Now when the user refreshes the page the past saved filters are kept intact, and it is possible to save, save as and undo the making of a new filter view.

Tuesday

I started my day with testing the cache I'd built for the filter the previous day by using *Glass Box* testing. I added logging to the code and examined what happened in the browser cookies as I simultaneously inspected the code. This led to discovering two bugs: the 'Approve' button failed to clear the edited views filter cache and the date filter crashed the website when removing one of the selected dates (start- or end date). After fixing both, I ran a linter check and submitted a pull request.

I then merged an earlier pull request, which updated navigation logic to track active states based on the URLs rather than user clicks. Resolving merge conflicts took some time but was manageable. Additionally, I tackled a Jira issue where it was possible for the user to save a new filter without naming it. The solution to the problem was simple, I only added an error message that would pop up in case the user would try to save the filter without naming it. I verified the change locally with *Black Box* testing, started the linter and submitted a pull request.

Later, I investigated with my colleague a Jira issue where popups caused unexpected page shifts. Unable to replicate the defect, we concluded it had already been resolved. We also discussed a bug where the business status is cleared when you edit the filter and navigate from the dashboard to the task page. This appeared because the business status was filtered out when displaying the task page, as there is no business status on the task page. Hence, we deleted the filtering out of the business status. Now the business status is not filtered out and remains intact in the filter when returning to the dashboard from the task page.

Then I took upon me a Jira ticket regarding cache memory of the total amount of rows shown in the tables. The customer had found an issue where they chose e.g. 30 rows, but it was reverted to the default 15 rows whenever they refreshed or visited another page. I solved the issue by creating yet another cache memory in the cookies, in which the number of rows shown last is saved. Now the page takes the information from the cookies, instead of always having the default 15 rows on reload.

The day ended with testing in the test environment, revealing two new bugs. One caused crashes due to missing error handling in the filter cache. I had forgotten to add a null check to the error handling of the new edited filter view cookie cache, which resulted in the site crashing the first time the page is opened, and no cookies exist yet. This worked smoothly after the error management logic had been added. The other bug removed task rows from the task page. It turned out that the code that filtered out the business status that I and my colleague removed from the task component wasn't uncalled-for. When the removed code was reinstated, the task rows were visible again.

Wednesday

Morning testing with the *Black Box* method confirmed that the previous fixes were stable. I then improved the filter editing logic by refining how filters were stored and updated. Before the edited view cookie existed, the users unsaved edited filter view was saved in default views and was managed there. Now this is unnecessary, as the users own filter views are saved in edited views. Therefore, I refactored a function where the active filter is being determined and made a new function that manages the update of the filter. This simplified other functions as well, e.g. cancelling changes in the filter view, as it was possible to determine the filter with the same name in default views as active and clear the edited views cookie. Saving the filter also became easier, as you could simply call the function that updated the saved filter. As the change was quite large and needed changes in multiple files, working on this took most of my workday.

Later in the day a bug in the task's popups had been found, where the user filled in data wasn't transferred to the BPMN tool. Debugging required collaboration with a colleague, as I lacked admin permissions to the BPMN tool. Through network analysis and code inspection, we discovered that the wrong payload had been added to the post method. Instead of posting the popups payload it was posting the original tasks payload, resulting in the data filled in by the user in the pop up being lost.

Thursday

Today we published the next version and started testing it. I found one bug in the user experience of emptying the calendar filter. It wasn't possible to empty it, and when it was emptied, the filter wasn't working anymore. This resulted in displaying the wrong rows on the dashboard and the task page.

In the payrolls internal automation project meeting we agreed upon the Nebula projects prioritizations. I demonstrated the issues I had found in the calendar filter and my proposed solution to it. I also presented the bug fix to the saved filters cancellation logic. We agreed these fixes can be tested immediately and deployed the next day.

After I finished testing, I worked on the calendar's usability. We are using Reacts DatePicker component, and improving the usability turned out a lot harder than I had originally thought. I got it to work by adding a button outside of the DatePicker, but I was unable to get the button inside the DatePicker. Then I found an update to the React component where this had been used:

```
slotProps={{
    field: {clearable: true},
}},
```

This made the icon (x) visible, but still didn't work correctly. The issue was that when you removed the filter, it remained as 'xx.dd.yyyy' and no rows were shown. Finally, I found out what caused the problem. The value that was entered into the filter was in fact 'Invalid Date', resulting in the filter not working as that isn't a comparable date value. I solved this by changing the value to null instead of the mentioned string. The solution code is shown in Figure 2 below.

```
const handleDateChange = useCallback( callback: (newValue, type) : void  => {
    setEditedFilter(true); // approve, edit & cancel buttons
    setIsCustomView(true);
    const formattedDate = dayjs(newValue).format( template: 'YYYY-MM-DD');
    const isValidDate : boolean  = formattedDate !== 'Invalid Date';
    const updatedValue = isValidDate ? newValue : null;
    const isStartType : boolean  = type === 'start';

    // Validate date range
    if ((startDate && formattedDate != 'Invalid Date') && (endDate && formattedDate != 'Invalid Date')) {
        if (type === 'start' && endDate && newValue.isAfter(endDate, 'day')) {
            // Don't update startDate if it's after endDate
            setErrorMessage( value: "Alkupäivä ei voi olla suurempi kuin loppupäivä");
        } else if (type === 'end' && startDate && newValue.isBefore(startDate, 'day')) {
            // Don't update endDate if it's before startDate
            setErrorMessage( value: "Loppupäivä ei voi olla pienempi kuin alkupäivä");
        } else {
            setErrorMessage( value: '');
        }
    }
    isStartType ? setStartDate(updatedValue) : setEndDate(updatedValue);
    setUpdate(false);
    isValidDate ? (
        setSelectedFilters(prevFilters => {
            return updateFilter(prevFilters,  name: type === 'start' ?
                'dPalkkapaivaStart' :
                'dPalkkapaivaEnd', formattedDate);
        })) : (
        setSelectedFilters(prevFilters => {
            return updateFilter(prevFilters,  name: type === 'start' ?
                'dPalkkapaivaStart' :
                'dPalkkapaivaEnd',  value: null);
        }));


}, deps: [setSelectedFilters, startDate, endDate, setIsCustomView, setEditedFilter, setUpdate]);
```

Figure 2 handleDateChange solution 1

Friday

Today I continued working on the filters calendar selector by refactoring it. Whilst refactoring I found a bug where the error message wasn't working correctly when either start- or end date always was null the first time the value was inserted, as it gets its value later in the function 'updateFilters()'. The new solution to the handleDateChange is shown in Figure 3 below.

```
const handleDateChange = useCallback( callback: (newValue, type) : void  => {
    setEditedFilter(true); // approve, edit & cancel buttons
    setIsCustomView(true);
    const formattedDate = dayjs(newValue).format( template: 'YYYY-MM-DD');
    const isValidDate : boolean  = formattedDate !== 'Invalid Date';
    const isStartType : boolean  = type === 'start' ;
    const oppositeDate : Dayjs  = isStartType ? endDate : startDate;
    const filterKey : "dPalkkapaivaStart" | "dPalkka...  = isStartType ? 'dPalkkapaivaStart' : 'dPalkkapaivaEnd';
    // Validate date range
    if (isValidDate && oppositeDate) {
        if((isStartType && newValue.isAfter(endDate, 'day')) ||
        (!isStartType &&  newValue.isBefore(startDate, 'day'))){
            const errorMessage : "Alkupäivä ei voi olla suuremp...  = isStartType
            ? "Alkupäivä ei voi olla suurempi kuin loppupäivä"
                : "Loppupäivä ei voi olla pienempi kuin alkupäivä";
            setErrorMessage(errorMessage)
        } else {
            setErrorMessage( value: '');
        }
    }

    setUpdate(false);
        setSelectedFilters(prevFilters =>
            updateFilter(prevFilters, filterKey,  value: isValidDate ? formattedDate : null)
        );

}, deps: [setSelectedFilters, startDate, endDate, setIsCustomView, setEditedFilter, setUpdate]);
```

Figure 3 handleDateChange solution 2

The rest of the day I was occupied with other work tasks and investigated how I could fix the issue with the disappearing business status from the filter when in editing mode but found no solution yet.

Weekly analysis 1

This week I have worked on bugs that have already been detected and experienced difficulties in testing my code comprehensively before pushing it from my local branch to development and testing. When I edit code that has anything to do with the browser cookies, I must remember to delete the cookies and refresh the browser. If I had done that when I was testing my changes to the filter cache bug, I would have noticed the missing null check that now crashed the whole site. Preferably an automated test should be created, where all cookies are cleared and the application run.

Automated tests are according to Dustin, Rashka and Paul “The management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated test tool.”. Automated tests provide a control mechanism that ensures the correctness and steadiness of the software in each build. It is also convenient in regression testing, in which it is verified that modifications in the software haven’t caused any new defects in the existing functionalities. To get the most advantage out of automated tests they should be introduced as early as possible in the project. Preventing errors early on decreases the cost (amount of time and resources required) to fix them. (Dustin, Rashka & Paul 1999, chapter 1)

We haven’t created any automated tests yet even though we have returning bugs that could have been avoided. The problem is that automated test requires a large time effort in the beginning, which is then paid back later. We have worked under great time pressure to get the job done and have therefore had to cut on making automated tests and do all tests manually instead. It isn’t possible to build automated tests for everything, and developers should focus on creating automated tests for functionalities that break easily in further development (Dustin, Rashka & Paul 1999, chapter 1).

The impact of a defective application also underlines how important it is for us to find as many defects as possible in the software before the launch. Any defects or errors in the operational phase can result in more comprehensive retesting, involve several organizations input and cause system downtime (Dustin, Rashka & Paul 1999, chapter 1). I have also experienced that a user doesn’t trust that the application will work as it should if it has too many defects. This again results in lesser usage of the software and returning to old working patterns. This scenario would be unbearable, as the business wouldn’t get to cash in on the benefits that the new software offers.

Elfriede Dustin has introduced a structured methodology for ensuring successful implementation of automated tests called “Automated Test Lifecycle Methodology (ATLM)”. *First* a decision of creating automated tests is made. During the decision phase it is important to manage expectations and convey the potential acquired benefits to the business management. In the *second* phase an automated test tool is chosen. There is no “one fits all” solution, and the choice depends greatly on

how the software has been built. In the *third* phase test goals and strategies are outlined and test processes recorded. In the *fourth* phase roles and responsibilities are defined as well as schedule, test plan, design activities etc. In the *fifth* phase the test units are executed. In the *sixth* phase lessons learned and other metrics are collected to improve the process. The ATLM process is visualized in the Figure 4 below. (Hedge 1 October 2019)

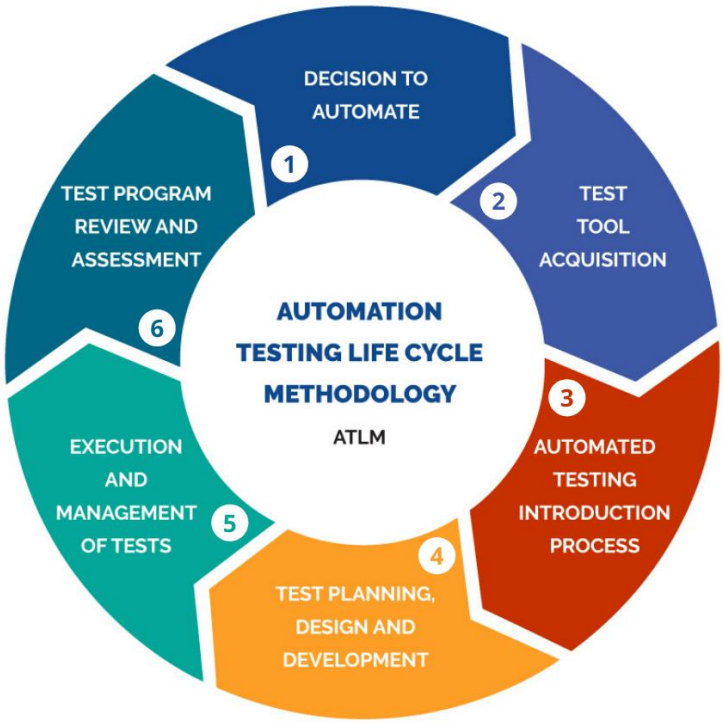

Figure 4 Automation Testing Life Cycle Methodology (ATLM) (Hedge 1 October 2019)

In addition to the null check, I should have also noticed that removing the code on the task page where the business status was filtered out of the filter resulted in no visible rows. That was the whole reason behind why I had added it in the first place but had forgotten. Adding more comments and information in the code would have also improved readability and decreased the chance to “break the code”. Furthermore, I should take the time to think each process through recognizing all the different possible user actions. When I created the feature that allowed users to save their own filter views, it didn’t cross my mind that a user could save a filter without naming it. Adding this kind of error handling is important to minimize unwanted application behavior. Therefore, I need to take the time to think through all scenarios in the future.

A project consists of three main characteristics, it is temporary, unique and progressively elaborate. It is a *temporary* endeavor that ends when objectives have been achieved, it's decided that the objectives won't be achieved, or it is terminated due to being redundant. *Unique* means that the project produces a unique deliverable that hasn't been done before and has a well-defined purpose. *Progressively elaborate* means that the project is developed in steps and increments of adding features and definition. In IT projects we often build something that we have never built before. It can therefore be helpful to follow a standard project management process to acquire repetitive success. Project management consists of the processes "initiation, planning, executing, monitoring, and controlling, and closing" to achieve the project objectives. (Brewer & Dittman 2023, 13-14)

The Nebula project is currently in execution and monitoring phase of the project life cycle. We have worked in small iterations one feature at a time and are now in a place where we are polishing the details and fixing faults in the software. Each iteration has a backlog plan, and is designed, developed and tested before it is presented to the customer for approval testing. We are now concentrating on testing the software to ensure that the developed software meets its requirements and that most defects are detected. It is done by running the program using mock data and then reviewing the results. The goal of testing is to find errors, abnormalities and information about non-functional attributes (Sommerville 2016, 227).

3.2 Observation week 2

This week's primary focus will be on approval testing, which requires stability in the BPMN tool and prevents development of the payroll automation process. As a result, the payroll automation process developers will focus on completing the training process. Stress testing is another critical priority, alongside ensuring that the MVP version is completed and thoroughly tested ahead of schedule. My personal goal for the week is to solve some of the discovered bugs assigned to me. One of the major ones is the business status filter bug, where the filter is sometimes invalidly cleared.

Monday

During project meetings we discussed the start of approval testing and its impact on development. Unlike the UI, where work can be done in branches, the BPMN tool doesn't allow parallel development, limiting progress on the payroll automation process. Therefore, the BPMN process developers will focus on finalizing training process and contact the BPMN tool provider to determine whether hotfixes could be applied. Stress testing was flagged as an urgent task, as it needs to be completed well in advance of the MVP version launch. Finally, it was agreed that everyone creates a workload estimate for their MVP Plus version tasks. We also discovered the need to refine backlog grooming for the MVP Plus cards as well as a prioritization of the version 1.0 cards. The customer also needs to clarify version 1.0 requirements, as the current scope remained ambiguous.

Workwise I worked on a solution for the bug where the business status filter is cleared when the user visits the task page in editing mode, in addition to other non-development work. Despite efforts, I couldn't figure out the solution to the bug yet.

Tuesday

Project meetings covered approval testing progress, which was scheduled to continue for two more weeks, as well as the stress tests that hadn't been started yet. As the stress tests are critical components of the MVP testing phase we need to get on top of them as soon as possible. Failing to do so could jeopardize the whole release as the customer would cancel the release. We also discussed the BPMN tools version handling but hadn't received any news from the BPMN tool provider yet. We can only hope for the best and that they get back to us soon.

I continued investigating the business status filter bug. I analyzed the interaction between different components and their state management, focusing on how filters were stored and retrieved. The root cause is that the business status is not defined in the task page, which is why it needs to be filtered out when loading the page. Otherwise, no task rows would be shown, as there is no row that has a business status. However, filtering out the business status means that it is removed from the

filter, resulting in the found defect. Furthermore, I handled non-development work, ensuring that my other work obligations and customers weren't neglected.

Wednesday

I finally resolved the business status filter bug. I solved the problem by creating a new state called *'businessView'* and added it to the App.js file as it is the lowest common level for the components that will be using it. When the user opens the task page, the system will check whether there is a business status present in the current edited filter. In case it is found, is it added to the *'businessView'* state using Reacts useState hook. When the user returns to the dashboard, the system checks whether there is a value in the *'businessView'* state. If it isn't null, it is added to the filter and the state *'businessView'* is set to null. I added the same check to the filter when it is saved when the user is still on the task page, as it would otherwise be lost when the filter is saved on the task page. I feel very proud of my solution and solving this complex issue felt particularly rewarding.

With that resolved, I turned my attention to another UI inconsistency. The customer had reported that the task forms didn't have enough "air" and lacked a horizontal line that was in the BPMN tools version. The task forms themselves are presented as they have been created in the BPMN tool, but we've had some issues with how it is displayed in the UI, and it has required quite a bit of manual CSS detective work to get the colors and such the same. With raising the issue that "air" and the horizontal line is missing, the customer means that there is more padding in the BPMN tool version than there is in the Nebula version. The difference originates from how the UI interprets rendered styles, which by default are the same as the React components default styling. Debugging required a lot of CSS adjustments to align padding, spacing, and visual elements. I didn't however have time to find a solution and will continue tomorrow.

Thursday

In addition to other duties, I conducted post-merge testing for the business status filter fix. Since my pull request had been approved, I needed to verify that the feature still worked after integration, as the main codebase had evolved since my original implementation. Testing confirmed that the fix remained functional, and I submitted a new pull request to merge it into the testing branch.

Returning to yesterday's UI task issue, I continued my detailed CSS analysis to pinpoint where the discrepancies originated and the best solution for it. Comparing the BPMN tool's version of the task to Nebula's, I could clearly see what the customer meant. I attempted to see whether I could somehow edit the BPMN tool's task UI styles, but the tools for that fell a bit short. I will have to solve it somehow in the UI's CSS stylesheet. Adding padding is quite easy but conjuring a line that doesn't

exist in the UI is another story. I experimented with various fixes, but didn’t have that much time on my hands to work on this as I have a lot of payroll customer work today.

Friday

No software development work today.

Weekly analysis 2

I find that I have been successful in refining my React and general coding skills this week. It was quite demanding to solve the error where the filters business status was cleared when visiting the task page in editing mode. Even though it was frustrating at times, it doesn't beat the feeling when you finally solve it. Perhaps that makes it ever so satisfying when you have had to really work for it. I enjoy solving difficult logical scenarios and I believe that solving problems are one of the best ways to develop yourself as a programmer. The more problem solving you do, the more your logical thinking improves.

Charles Humble has written an interesting text titled "Improve Your Problem-Solving Skills". Humble discusses the frequent mistakes in software problem solving as well as the steps to successfully debug the code. First mistake is to go straight into coding. This occasionally works but usually results in a waste of time. The second mistake is to go straight into a search engine, Stack Overflow or a large language model (LLM) and attempt to fix the problem by copy-pasting the result. Problem solving requires that one understands the problem, and therefore this approach doesn't solve a problem. And even if it does solve it, the programmer might not understand the code they have pasted. (Humble 2024)

Humble (2024) introduces a four (4) step process for problem solving:

- **Understand the problem.** When debugging code, ask questions like "what do you think the code *should* be doing?" and "what is it *actually* doing?"
- **Gather information**. Read through documentation (e.g. REST API), investigate the commit history of the codebase or talk with fellow developers who worked on the code in the past.
- **Produce potential solutions**. Take the time to come up with and iterate several possibilities instead of going with the first idea that comes to your mind. During this step it is also beneficial to **explain the problem to someone**. This is called confessional programming or rubberducking. as it involves explaining the code to a rubber duck or another person. Explaining the code to someone else out loud reveals whether the original problem has been understood and might uncover new insights.
- **Start coding the preferred solution**. During this step it is important not to get too hung up on one solution too early and be prepared to discover that the solution does not work. This step also involves testing, verifying and documenting the solution.

Humble also discusses the importance of doing nothing in problem solving. It is better to take a break and go for a walk or sleep on it when you are truly stuck with a problem. This is described by Holly Cummins as the default mode network which is "a pattern of brain activity which kicks into life when the rest of the brain goes into an idle state". (Humble 2024)

I have experienced this approach to work well during my career as a software developer, which is why I often decide to sleep on it and continue another day whenever I have struggled with a problem long enough (often multiple hours). I have somewhat followed the four (4) problem solving steps even though I haven't done so consciously and not always in that very order. I do not take the time to produce several solutions before I start coding a solution, but I do pivot to another solution if the first one doesn't work. I have also experienced that confessional programming works well. There is something to it when you explain the problem to someone out loud, as you truly must put it into words. This has every so often worked, and I should probably do it more regularly, even if it is to a rubber duck.

Solving the business status filter bug this week involved rubberducking as I explained my conundrum to a fellow developer. When I explained why the business status was being filtered out and that results in the filter defect, I realized that state handling was a possible solution. I put into words my theory of saving the business status filter value into a state before filtering it out and then reinstating it from that state back to the filter when navigating back to the dashboard (or any other page). Having a clear vision of the implementation, it was down to coding the solution. Solving this issue involved some of Humbles problem solving steps - understanding the problem, rubberducking and coding the possible solution.

In software testing it is beneficial to differentiate between found bugs in the software rather than using the terms as synonyms. This way it is clear to everyone what kind of discrepancy it is as well as its urgency. In this thesis we will use the International Software Testing Qualification Board (ISTQB) differentiation and definition. These are:

- **Error:** "A human action that results in a defect"
- **Defect:** "An imperfection or deficiency in a work product where it does not meet its requirements or specifications or impairs its intended use "
- **Failure:** "An event in which a component or system does not meet its requirements within specified limits"

(ISTQB Glossary n.d.a, ISTQB Glossary n.d.b; ISTQB Glossary n.d.c; Homes 2012, 4)

During these past two weeks we have encountered all three kinds of bugs. We encountered an error in the software when the business status was cleared when visiting the task page in editing mode. This was due to a logical mistake in the code, which resulted in a defect after the user action on the website. As the ISTQB definition implies, an error in the code results in a defect of the software product. Most bugs in the software are errors and defects, and it is probably almost impossible to write 100% bug free code as there is always a possibility that the code has unknown errors. These must simply be fixed along the software lifecycle whenever they are detected. Fixing failures is much

more critical, as that means that the software crashes. A fault can result in a failure, which we encountered in week one when the page crashed if the user filled in both a start date and an end date in the filter and then tried to remove one of the dates.

According to Geeks for Geeks (2024a) stress tests are in software development defined as a technique that “determines the robustness of software by testing beyond the limits of normal operation” and that stress tests “verifies the stability and reliability of the system”. Stress tests are done for all kinds of software but needs to be especially executed for business-critical software components. Stress tests are needed to ensure that

- the software can accommodate sudden increases in traffic,
- appropriate error messages are displayed in error scenarios,
- the software continues to function under abnormal conditions and
- there is a contingency plan in case of sudden failure

due to stress conditions. (GeeksforGeeks 2024a)

We haven’t done any stress tests in the Nebula project yet. It is crucial to get the product stress tested and inform what the max capacity is before the system starts to slow down or crashes completely. The stress tests to be done are for the REST API and backend requests. The stress test for the already launched data page is imminent as the test user group will be posting data to the database simultaneously, and that will create load on the AWS servers. It is important for us to know what the theoretical maximum load is. If we do not figure out before launch and communicate it to the board of directors, we will have to postpone the launch itself. That would have a major impact on the business, and we need to do everything in our power to have a successful launch in September and in October.

3.3 Observation week 3

This week the last MVP cards must be completed, as they have a deadline on Friday. Therefore, I will focus on my MVP cards this week, which includes resolving UI inconsistencies, improving table sorting functionality and adding an instruction link to the dashboard. As a team we will continue stress testing preparations and keep the customer informed about the development progress.

Monday

On Nebula project meetings, we discussed the DataSpace stress tests. We will stimulate heavy API load to the AWS database with a script that uses the Locust Python library. We also considered AWS pricing, as finances must be accounted for when executing stress tests. As the API Gateway has a limit of 20 billion requests before incurring extra costs, there was no immediate concern. Therefore, we are no ready to go and the backend programmer will proceed in creating the required script.

I had no time to work on the Nebula project today due to payroll-related responsibilities.

Tuesday

I completed the “air” issue in the task forms that I was working on the previous week. I resolved it by adding padding manually to the horizontal lines wherever they do exist. The horizontal lines are in every form as a divider between different sections of the form, so it seemed like a good place to add it. The issue with the horizontal lines only affected a handful of forms, and as far as I can see in the BPMN tool, they are missing there too. Therefore, I reassigned the issue to my colleague responsible for BPMN development.

Additionally, I merged the filter view changes to the testing environment, as my pull request had been approved. To ensure that the new filter feature still worked after integration, I needed to test it as the main codebase had evolved since my original implementation. I tested the feature by both inspecting the cookies in the developer tool and monitored how the filter behaved in the UI. Testing confirmed that the filter feature remained functional, with no issues detected.

Wednesday

I started working on the WorkSpace table sorting issue, specifically adding sorting to the payment date column in task page tables. The customer also requested that the sorting order be cached so users wouldn’t lose their preferences when navigating away. This is a similar request to the cache memory of the total amount of rows shown in the tables. Hence, the solution was simple. After enabling sorting to the payment date column, I created a new cookie, ‘Table_Sort_Prefs’, to store user preferences. However, I encountered a question – should columns shared between the

dashboard and task page maintain the same sorting, or be treated separately? Since this impacted the implementation, I had to wait for customer input. Unfortunately, they were occupied with an urgent issue. The payroll automation had stopped working due to Ui Path, our robots' engine, being offline. This meant that the customer had to keep the payroll consultants informed on the next steps and up to date. I checked Ui Path's website and observed that it was the document understanding that had stopped functioning. As document understanding is essential for our workflows, this disruption impacted our processes significantly. A screenshot from the Ui Path website is displayed in Figure 5 below.

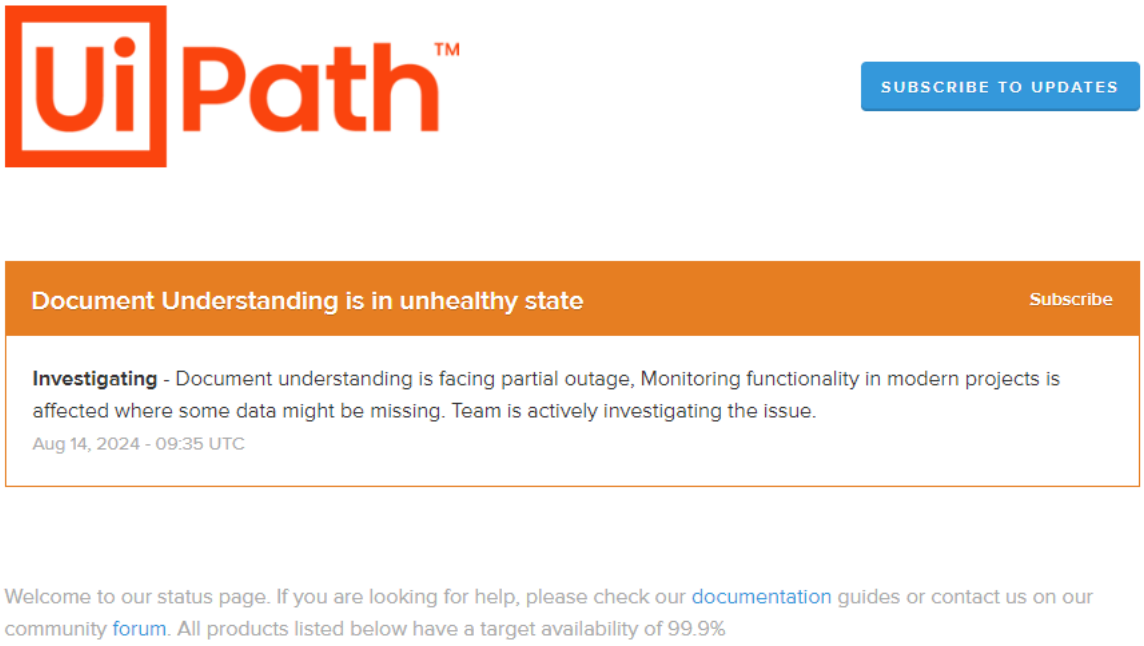

Figure 5 - Ui Path document processing outage

Thursday

In addition to payroll matters, I worked on a request to add an instruction link to the dashboard. Since the customer hadn't defined any specific UI design, I adjusted styling and hover effects until it felt natural with the existing design. It was very easy to add the instruction link to the bottom of the page, and I had a fun time implementing different UI designs to it. I only hope that the customer will be content with the final look.

Later, I met with a colleague on Teams to discuss a dashboard issue. The customer had discovered that the process statuses displayed on the dashboard weren't updated immediately after the user has completed a task. When a user completes a task of one part of the process, the status of that

part should be updated and possibly the next step in the process as well. The customer is concerned that the user will be confused whether they completed the task, if the status of the process doesn't emulate it. We agreed that I will investigate it further independently.

Friday

Today was frustrating. The dashboard status issue turned out to be an MVP card, meaning it should have been completed today. However, no ticket had existed for it until I created one the previous afternoon. Everyone assumed my colleague had it under control as he had reassured the customer it would be handled. Had I known earlier, I could have worked on it throughout the week. Ultimately, the issue was reclassified as an MVP Plus card, as there wasn't enough time for us and the customer to test it thoroughly.

Another ongoing frustration is the dependency on others for code integration. Every time I need to merge my branch into development, a colleague must approve the pull request. After that, a second pull request is required to merge development into testing. While I understand everyone has their own workload and this increases code quality, waiting for approvals prevents me from staying productive. With no other development tasks available, I spent some time assisting colleagues in payroll, which at least kept me occupied and productive.

Despite these frustrations, I resolved the dashboard status issue. I added a 2 second delay before redirecting the users from the task page to the dashboard, allowing enough time for the BPMN tool to process and return the updated status. Initially, I tried with a 1 second delay, but it wasn't sufficient. Additionally, I added a check for tasks without additional data, as only tasks with additional data were verified to see whether there are additional tasks to be completed. The reason behind this verification is that the user is redirected from the task page to the dashboard if there are no more tasks to complete.

Weekly analysis 3

This week is a classic example of failed communication. At the beginning of the week everyone seemed to be on the same page – complete all MVP issues and make progress on stress tests by the end of the week. Somewhere down the line we weren't following the project guidelines regarding communication and how new issues or features are added to the development backlog. The customer should have made a ticket of the dashboard status as an MVP issue. If it would have been on the backlog, anyone from the development team could have addressed it. Furthermore, it is questionable whether it is feasible to add a new MVP issue on the same week that the deadline is, as the development team hasn't allocated time in their schedule to complete it. We on the other hand should have created the issue ourselves, ask the customer to do it or communicated clearly that we do not accept new additions to this week. Now we assumed that one developer would complete it, and everyone had a different perception on how urgent the issue in question was. This resulted in a disappointed customer and chaos within the development team.

Failing to communicate is according to experts the largest threat to any project. Most problems within a project, such as unrealistic expectations and schedules, indicate communication problems. With clear and consistent communication, we can minimize conflicts, prevent misunderstandings, and improve overall project efficiency. In addition, it serves as a key factor for reducing risks, promoting innovation and ensuring projects are completed on time. Therefore, it is imperative that we prioritize and maintain communication throughout the project lifecycle to increase the probability of a successful project. (Schwalbe 2019, 426; Minois 20 October 2023)

Throughout the project we have faced adversity due to communication failures. The project is already behind its original schedule, partially due to unforeseen circumstances beyond our control, but mainly due to misunderstandings and unclear instructions. Maintaining the current project schedule is crucial for the business, so we must take all necessary measures to improve project communication. Whenever we fail to communicate, like we did this week, we jeopardize the project. It is important to keep expectations realistic to avoid disappointments and create a positive experience for the customers. We failed to manage the customers' expectations this week when we overpromised the delivery of the dashboard status issue.

There are many aspects to consider when communicating with stakeholders. Especially in IT projects there is a gap in knowledge and experience between developers and business professionals which can create misunderstandings. Any technical jargon needs to be simplified and rephrased into layman's language. (Schwalbe 2019, 426)

I believe that I am quite good at rephrasing and explaining technical aspects in simplified language. For instance, when a business stakeholder encounters an issue that would require me to see what they have in the website's developer tools, I create a step-by-step guide with pictures to guide them if a video call is not possible. Nevertheless, I need to keep this in mind when communicating with non-IT professionals.

We also need to focus and understand group- and individual communication needs. All individuals have different personality traits that ultimately impact their communication preferences. Therefore, we cannot apply the same communication methods for all. We should first understand our own communication styles and then acquire the ability to put ourselves in someone else's shoes. Only then do we have the tools to communicate properly. When a group consists of personality differences, such as introverted, intuitive, extroverted and sensation-oriented, there is a high risk of miscommunication. For instance, written guides by developers might lack the details most users need, who often prefer face-to-face interactions or videos. (Schwalbe 2019, 429)

I haven't thought much about my communication style before. At the workplace I prefer to have a meeting about a topic rather than reading a memo or instructions. If I need to do something that requires following a specific set of instructions, I'd rather follow them in written form than watching a video. I also tend to "wing it" and read the documentation if things don't work out. I must admit that whenever I feel overwhelmed by my workload, communication is the first thing I tend to withdraw from. This creates a vicious circle - the longer I wait, the harder it gets to reach out. I need to work on this as over-communicating is always better than under-communicating.

Another aspect we need to consider is that messages are rarely interpreted exactly as they were intended. Therefore, we need to use multiple communication methods, such as written text, videos, visuals and meetings, to create a clear and open communication climate. Instead of assuming that the receiver has understood the intent of the message, we can ensure it with a feedback loop. (Schwalbe 2019, 429)

We could have avoided a lot of headaches this week if we would have done this with the dashboard status issue. If the development team and the customer would have had a thorough discussion about the topic, we most likely would have classed the issue as MVP Plus rather than MVP from the start. This would have also ensured that everyone would agree on the issue priority and schedule. It is as important to ensure that the developer has understood the customer request as it was intended. It is difficult to describe requests comprehensively, and there is always a risk for different interpretations. Therefore, an open ongoing communication between the developer and the requestor is essential to avoid redundant work. This week, I made a good decision by asking the customer whether the sorting direction should be shared between the dashboard and task page, rather than making assumptions.

This question helped initiate a conversation and will likely lead to a meeting where I can better explain the possible solutions.

A project scope is the total amount of work that needs to be completed by the end of the project, broken down into tasks and deliverables. Defining the project scope is a crucial step in the planning process, as it guides project managers in team assembly, resource estimation and project plan, schedule and budget creation. However, in an agile (IT) project, the scope is a simple description of the end-product with an overall vision, as it's impossible to know every single detail from the start. The scope needs to be managed throughout the project lifecycle, as incoming project changes and adjustments need to be managed in a controlled manner. This means that the requesting party needs to submit a change request that describes needs and priorities, which is then examined by the project manager. After the change has been officially documented, will it undergo an approval process performed by senior management. Only then is the change added to the backlog and the project scope is revised. (Landau 30 June 2023; BVOP n.d.)

In the Nebula project we manage project scope using the Microsoft Planner, which organizes tasks into columns that visualize the project backlog and progress. When the customer discovers a new requirement, they create a card with a description and priority. The development team then reviews it, deciding to accept, reject, or request more details. Only once a card is accepted and fully understood, does it move to the backlog for development. This process works well when followed, as project meetings help clarify uncertainties. It is when we deviate from the process that we encounter problems. Clear communication and alignment minimize conflicts, prevent misunderstandings, and improve overall project efficiency

3.4 Observation week 4

This week the first stress tests must be completed, to allow time for adjustments in the case they fail. I aim to complete all my MVP Plus issues and see whether I could improve the websites performance somehow.

Monday

I started my day by merging the dashboard status issue into the testing environment, as my pull request had finally been approved. I tested it using the *Black Box* method and was happy to conclude that it worked as expected.

In meetings we scheduled a plan for the next four weeks in Microsoft Planner, assigning cards to specific weeks so they would be ready for approval testing the following week. Most tasks were for the BPMN payroll process, and some might need to be moved to version 1.0. After my colleague completes time estimations, she will discuss with the customer which tasks can be moved out of the MVP Plus scope. The UI tasks were minimal and all fit into this week. In addition, the training process needs to be completed today so that we may start testing it. My colleague informed that BPMN REST API stress test was completed with no bottlenecks found, but our backend and the BPMN provider stress tests remained. The business stakeholders have demanded that these are done by Wednesday.

In the afternoon I received a Teams message from my payroll colleague about a bug in the DataSpace client list. Its popup menu allowed selection between “Payroll”, “Mepco” and “All clients”, but it always defaulted to “All clients”. I investigated by checking the routing setup, which appeared correct. Adding console logs revealed that the function determining the selection alternated multiple times before settling on “All clients”. I didn’t have time to fully resolve the issue today, so I will have to continue another day.

Tuesday

Today, I fixed a bug that had been found in the DataSpace selection table cells. Occasionally, the selector would not open and flash instead. Investigating it further, I found that the problem stemmed from the focus state not always being correctly assigned. Since the selector’s open state relied on focus, it sometimes failed. I updated the logic to use the table cells active state instead, and the issue no longer occurred in local testing. After I had fixed and tested the bug, I added a tooltip text to the header on the customer’s request. This was simple as we already had a universal tooltip component.

In meetings the BPMN developer presented her time estimations, which left a comfortable margin. She determined six hours of work needs to be moved from the MVP Plus to version 1.0 and would discuss this with the customer. We also reviewed the stress tests, and evidently the BPMN provider has promised to deliver them today. Additionally, we debated whether the backend tests should be done locally or with Jenkins.

Wednesday

In today's morning meeting we discussed the BPMN tools app publish error: "*Database has reached size quota"*. It is only the payroll process that gives this error upon publish, the other ones work fine. We sent a ticket to the BPMN tool provider immediately, so that they may investigate the issue. There should not be any size restrictions in the processes, so we hope they can fix it as soon as possible.

The BPMN tool provider delivered the stress test results today. The heaviest process, which creates a new payroll process, had no issues with 300 simultaneous users, but displaying open processes encountered issues. While the system handled 1000 open processes with 300 users, it struggled with 3000 open processes, failing about 10% of calls. The provider will adjust the server settings and retest. The 300-users, 3000 open processes are our worst-case scenario, but it needs to be functional by October. Meanwhile, our backend developer tested our database servers load capacity with growing simultaneous users. He discovered that the maximum load that the servers can take is around 350 simultaneous users. He will try to make changes to the AWS servers to increase the load capacity.

We also discussed possible optimizations on our end to reduce server load. The main issue is that the JSON file is about 30MB, and with 300 users simultaneously fetching processes every 5 seconds, the system overloads. One option is filtering data in the BPMN database rather than in the UI. Another is replacing the 5-second refresh with a manual refresh button and a timestamp indicating when the processes were last updated. Additionally, we could pause requests when the user navigates away.

As the customer had confirmed that the sorting memory for the dashboard and task tables should be unified, I was able to finalize this ticket. This required refactoring the task page columns to ensure consistency, which I tested locally before submitting a pull request. Additionally. I made a hotfix to the team filter on the task page. A bug prevented rows from displaying when a team was added to the filter. The issue was the team's variable name that hadn't been updated after changing the team from a string to an object. Updating the ID reference in *types.tsx* and *constants.tsx*, resolved the issue. Lastly, I fixed a visibility issue in the payroll process selector. When the table had a few rows,

the dropdown wasn't visible due to lack of space within the table. I initially tried adjusting the z-index and container size but discovered that it was simply solved by removing *'disablePortal'* from the Autocomplete component.

Thursday

Today I worked on a bug affecting disabled button colors. Instead of being grey, they appeared violet – the color of an active button. This happened because I had previously had added *'!important'* to the violet shade, but had not done the same for the grey shade. After fixing this, I also replaced the basic HTML buttons with React's Button component for consistency. I darkened the grey color to ensure that the white text remained legible and changed the save button to green. I then sent a screenshot of the suggested UI changes to the customer for approval.

Friday

Today I worked on improving the termination process, which was too slow. Since terminations are rare, the issue hadn't been prioritized, but I had time to optimize it today. The process requires multiple steps and API calls: retrieving action IDs, assigning the correct one, opening the termination form, fetching form data, and finally submitting the termination request. One issue was that I previously opened the termination form before retrieving the action IDs. I updated the function to wait for the correct action ID before proceeding, reducing delays. I also added a break statement to exit the for loop once the correct action ID was found. During testing, I discovered that some test environment processes lacked action IDs. I didn't find the same issue in production but added error handling just in case. Now if a process is missing a termination action ID, a popup directs the user to contact support. Lastly, I removed a 5.5 second timeout from the termination form, which was previously used to delay the render of the dashboard as it takes time for the termination process to complete. Instead, the terminated process is filtered out, making the termination appear instant to users while processing the termination in the background.

Weekly analysis 4

I was able to complete all MVP Plus cards, fixing bugs and improving UI consistency. I even had time to optimize the termination process a bit, improving the user experience (UX) with reduced loading times. As I have a duplicate role, both that of a user and UI developer, I notice deficiencies in the UX that might otherwise go unnoticed. This is a valuable and rather rare advantage. I know from firsthand experience what is important to the user and what their work consists of. Usually there is a gap between the developer and the user that needs to be filled so that the developer produces the solution that serves the user best. This can be tricky as the developer may never completely understand all factors of the users work.

The ISO 9241-210 (2010) standard defines UX as a “person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service”. From this definition we can derive five key elements of UX: a user, a system, an interaction, user perception and user responses resulting from the use of the system. Understanding UX helps improve design by considering user goals (what they want to achieve) and tasks (how they achieve it), acknowledging individual predispositions (such as experience, skills, and expectations), and accounting for environmental and social factors that influence interaction. The technical and physical environment, along with social structures and organizational goals, further shape UX, making human-centered design a powerful tool beyond individual products, extending to systems and organizations. The extended model of UX is visualized in Figure 6 below. (de Voil 2020, Chapter 2)

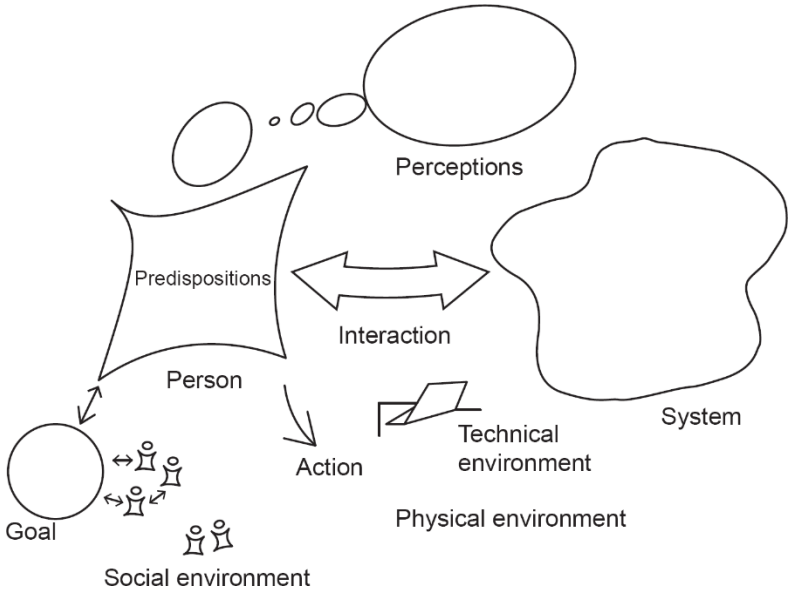

Figure 6 - Extended model of UX (de Voil 2020, Chapter 2)

The DataSpace table bug, where the select menu would briefly flash instead of opening as expected, resulted in a poor user experience. Users naturally expect a selection menu to open smoothly, and when it behaves unexpectedly, it causes frustration. Resolving this issue was important to ensure a seamless and intuitive experience. Fixing these types of problems feels somewhat ungratifying, as they involve meeting basic functionality expectations – users only notice when something is broken but rarely acknowledge when it works as intended.

The Nebula software solution aims to reduce manual labor while guiding the user to follow best practices and adhere to ISAE standards. It enhances quality with automated solutions and detecting errors before delivery to the customer, ensuring a reliable and efficient process. Since users are accustomed to performing work tasks in a specific way, the new system must be intuitive, easy to use, and offer a clear improvement over current method. Failing to fully understand the users or delivering a poor user interface is according to Forbes council member (2024) some of the reasons software projects fail. Therefore, we need to collect user feedback to detect and fix any deficiencies that may ultimately result in the user never using Nebula.

In my opinion, one of the most annoying features of an ill-manufactured software solution is long waiting times. Website responsiveness matters for two reasons: humans are limited especially regarding memory and attention and humans aspire to be in control rather than being at the mercy of a computer. According to Nielsen (2010) there are 3 response-time limits:

- **0.1 seconds** feels instantaneous, giving users the sense that they, rather than the computer, caused the outcome. This level of responsiveness is crucial for direct manipulation, a key technique that enhances engagement and user control.
- **1 second** maintains a seamless thought process. While users notice a slight delay and recognize that the computer is generating the response, they still feel in control and can navigate smoothly.
- **10 seconds** tests users' patience. Between 1–10 seconds, delays become frustrating but manageable. Beyond 10 seconds, users lose focus, making it difficult to re-engage once the system finally responds.

The termination process was frustratingly slow, taking several seconds to load the termination form and complete the process. Sometimes, the form didn’t open at all, leaving users waiting indefinitely. Given that opening a simple form should feel instant, improving API calls and code structure was essential for a smoother experience. That is why it felt important to improve the API calls and the structure of the code, to create a smooth and seamless UX. For some reason, loading the task forms occasionally takes several seconds, which is something we need to improve as well. The API calls

for the task forms are very straightforward though, so there must be something that can be improved in the connection or BPMN database.

Server load directly impacts website response times. This week, we discussed ways to speed up table row loading while reducing server strain. No system can handle 300 users simultaneously fetching 30 MB of data every 5 seconds without significant delays. In the worst case, requests take over 5 seconds to complete, creating a continuous data retrieval loop. Currently, the GET request crashes due to an unnecessary elastic search in the BPMN database, but there is no way to bypass it. To mitigate this, we could filter data directly in the BPMN tool database instead of on the UI, reducing payload size. The strain could also be reduced by only sending a new GET request if the previous one is successful. Then if the server crashes, we wouldn't strain it more with additional requests. Additionally, replacing the 5-second auto-refresh with a longer interval could help. We could also add a manual refresh button and display a "last updated" timestamp, giving users control over data retrieval. Furthermore, we could pause GET requests when the user navigates away from the page to optimize further. Personally, I find the 5-second refresh unnecessary and hope that the customer will approve the proposed changes. Implementing these four optimizations would significantly improve system capacity and performance.

3.5 Observation week 5

This week we will create a template of our UI description standard to ensure consistency across all future processes to prevent unnecessary rework and eliminate uncertainty about the customers' expectations. We should also receive the BPMN providers stress test report and aim to explore solutions for improving both their API connection and our database connections system performance, including database optimizations and request indexing. Personally, I aim to investigate the termination action ID issue and make UI improvements.

Monday

Today I had a team work session where we created a UI description for the new demo process *Employee onboarding*. The UI description for the payroll process hadn't been done according to the standard guidelines, which resulted in multiple versions due to inconsistencies in defining the process and UI requirements. To avoid unnecessary rework, we need to follow a standard model for all future processes, ensuring efficiency and clarity.

During meetings we discussed the stress test report that the BPMN provider should have delivered already. Evidently the report is ready but awaited manager approval. Our own previous stress tests revealed a pain limit of 350 simultaneous users, which falls short of the future 1000-user capacity. The primary issue lies in database capacity and connection limits. One potential solution involves modifying the database in AWS to handle higher loads. The stress test included GET, POST, PATCH and DELETE requests, and we agreed that testing each request type individually would provide better insights. Since GET requests are the most frequent, optimizing them by indexing the columns could significantly increase performance. To ensure transparency and continuity, we decided to document all stress test results in Confluence, allowing us to track changes over time and clearly communicate findings to the customer.

I also encountered an issue in PyCharm where multiple files appeared modified despite me not having edited them. My colleague explained that this was due to using both rebase and merge, which altered the hash and made it seem like changes had been made. The recommended solution was to save my changes elsewhere, roll back the branch, and create a new one from the development branch. I followed this approach and hope I preserved all necessary changes, but I will only know for sure once we have more open processes originating from the BPMN tool.

We merged the development branch with the testing branch today, allowing me to test my work. The tested cards included fixes for DataSpace tooltips, selector menu bugs, client menu issues, WorkSpace sorting, and various visibility bugs. All tests were conducted using *Black Box* testing, and no issues were found. The only uncertainty remains with the padding in the "air" section, which

may require further adjustment. If returned to development, I will seek a second opinion before finalizing it.

Tuesday

Today we received the BPMN providers stress test report, which assessed Fetch Case Instance API performance with 1 to 300 parallel users over five-second intervals. The platform scaled well with 1000 open processes but crashed with 3000. The server automatically restarts within two minutes when it fails. To address this, we will implement lazy loading and pagination to reduce request sizes, while the BPMN provider will increase server capacity. We also need confirmation on what happens to incomplete processes when the server crashes. Despite these concerns, we concluded that we are ready for launch and will document the necessary improvements for MVP Plus and version 1.0.

I investigated the missing termination action ID issue. Initially, I suspected that starting the process in the BPMN tool rather than Nebula UI caused the issue, but I had to rule that out. I checked the API response and documentation to determine when the termination action ID is assigned to the process and why the GET request doesn't always return it. I found that it was possible to filter the GET request, which returned only termination action IDs. The original unfiltered request returned all types of action IDs, which made finding the correct action ID that much harder. There are however "terminoi" and "terminate" action IDs, of which my colleague confirmed that "terminoi" is the correct one. This optimization significantly reduced the JSON response size, improving performance.

Wednesday

Today I finalized the termination process optimization by ensuring collapsed rows refresh every five seconds. It was as easy as changing the *'openProcess'* state to false. However, the termination is now so fast that users might doubt that the process is terminated. To improve user experience, I added a Snackbar notification that provides feedback on successful or failed terminations. I recorded a video of my screen and asked the customer and team for opinions. After committing the current version of the termination optimization branch and submitting its pull request, I made the color changes to the buttons for which I'd received feedback earlier. The customer had otherwise approved the changes but said that the save button cannot be green as the user then would be inclined to press it. It can be the lighter grey with black text, and the disabled button grey with white text. Changing the colors was a very simple task as it only required a few adjustments to the CSS. Lastly, I decreased load impact of the quality form by stopping table row get requests whenever the form is open.

Thursday

Having completed all my development tasks, I started writing a test plan for Nebula in Confluence. I applied the Haaga-Helia Software Testing courses guidelines for creating automated tests with Robot Framework. A well-made test procedure requires that you start with making a testing plan. The project manager suggested considering UI Path alongside Robot Framework. The next phase will determine which approach best fits our needs.

Friday

On Friday, my work on optimization was unexpectedly escalated by the customer contact person, who was concerned that these changes had not been formally requested. Although I had documented my work transparently in Microsoft Planner and limited it to the development branch, the customer felt that it added to her already high testing workload. While I was prepared to discuss these improvements in the next meeting, scheduling conflicts delayed this conversation until Monday. The situation was frustrating, but I decided to cut my workday short, recognizing the need to balance my workload with personal well-being.

Weekly analysis 5

Although the week ended on a low note, we accomplished a great deal. We established a standardized UI description that will be used in future software orders, ensuring consistency and efficiency. We also made progress in our stress tests and although the results revealed concerns, we know what steps to take to improve performance. I am particularly pleased with the optimization of the termination process, which was made possible by the insights I gained from my API documentation investigation. Since performance optimization of REST API requests and connections has been the key focus this week, we will now dive deeper into understanding their functionality and exploring optimization strategies.

REST API is a flexible and lightweight interface that integrates applications and connects components. The API is created by developers to allow other applications communicate with the application programmatically. It is the *rules* that must be followed to communicate with the application in question. The REST software architecture enforces the conditions on *how* an API should work. Together they enable performing standard database functions such as CRUD (create, read, update and delete) within a resource by communicating via HTTP requests. Any identifier information like metadata, authorization, caching etc. is passed in the REST API request headers and parameters. The API follows the architectural style of REST design principles, which are uniform interface, client-server decoupling, statelessness, cacheability, layered system architecture and optionally code on demand. (IBM n.d., AWS n.d.)

The **uniform interface principle** states that all same resource API requests should maintain the same structure, regardless of their origin. Additionally, one piece of data should only have one uniform resource identifier (URI). Resources should be compact yet comprehensive, including all the information a client might need without being excessively large. **Client-server decoupling** means that both server and client applications mustn't depend on each other. The client application should only know the requested resource's URI and the server application can only pass requested data via HTTP. (IBM n.d.) A **stateless** protocol does not preserve state information, which means that every connection starts from a clean slate (Geeks for Geeks 2024b). **Cacheable** means that the http response can be stored in a cache closer to the user, to be retrieved in later requests. (MDN n.d.a) The aim of the **layered system architecture** is for neither the server nor the client to know whether it communicates with and intermediary or the end application, that is the client, and the server aren't directly connected to each other. The optional **code on demand** principle states that when a HTTP response contains executable code, it should only run on demand. (IBM n.d.)

I find the BPMN REST API documentation difficult to navigate, often requiring me to consult their AI for guidance when I can't find necessary information. The distinction between Open Source and

Platform versions is unclear, making it difficult to determine which should be used in different scenarios. Additionally, some cases require multiple API requests to retrieve all relevant data, violating the uniform interface design principle by failing to provide complete information in a single response. Another issue is the presence of multiple identifiers for the same purpose, such as process identification and not retrieving all critical identifiers in one call. For instance, obtaining a termination action ID based on a known process case ID isn't possible. Instead, all termination action IDs must be requested to find the correct one. Furthermore, the stress tests on the BPMN API revealed performance bottlenecks under increased load, suggesting a need for further optimization and highlighting potential underlying inefficiencies.

It is important to optimize API's and API requests to for instance decrease latency, increase performance and reduce operational costs. An API can be optimized by enabling caching, implementing debouncing and throttling, and optimizing payload sizes. Caching reduces redundant calls by reusing previously requested data and thereby returning the response faster to the user. By debouncing and throttling user-triggered API calls, we can ensure that excessive user requests are optimized, safeguarding smoother interactions and faster rendering. It is important to reduce payload sizes as large API responses can degrade network performance and add processing strain on the client. This can be achieved with compressed responses, returning only the required fields in API responses and large object pagination. Additionally, sorting data on the server before sending it to the client also helps optimizing performance. (Pragada 26 November 2024)

Both our and the BPMN providers stress tests discovered that the API servers need to be optimized to be able to handle the future request load. The BPMN provider will increase server capacity, but hopefully they will investigate whether they can do other optimizations as well, such as caching, throttling and adjusting payload sizes. We will implement lazy loading and pagination to reduce request sizes and modify the database in AWS to handle higher loads. It is possible for the AWS server to handle higher loads by enabling data sorting, throttling, compressing responses and paginating large objects. It would also be beneficial to enable caching for frequently requested data, as that would minimize latency and increase performance.

It is possible to optimize API requests and thereby reduce payload size and minimize latency by using techniques such as pagination and filtering. Implementing pagination through parameters such as limit and offset allows retrieving only necessary data, while server-side filtering can further reduce the amount of data fetched. Additionally, batching multiple small API calls into a single request reduces HTTP connections and speeds up data retrieval. Prefetching data for components or screens that users are likely to visit next enhances perceived performance. Finally, ensuring robust error

handling and retry mechanisms for failed requests improves reliability and the overall user experience. (Pragada 26 November 2024)

Sending oversized API requests strains the API server unnecessarily, and therefore we need to make as precise API requests as possible. This means that we need to include pagination and filters in our API requests to only retrieve data we need. This reduces the response sizes, which in turn reduces the strain on API servers. If we would continue requesting unfiltered data from the BPMN API and filter all data in the frontend, the user would experience an unnecessary long response time. By filtering on the server side, we both reduce the response size and remove the need to filter the data in the frontend, cutting one step from the data loading process. When I added the filter “terminoi” to the termination form API request, I reduced the size of the API response. Before, the API request returned all action IDs, no matter the type. By adding the filter, only termination action IDs were returned, decreasing response size. The solution still requires looping through all action IDs to find the processes own termination action ID, but now the number of iterated rows have decreased significantly.

3.6 Observation week 6

On Friday we have the MVP Plus version deadline, which means that everything needs to be ready and tested. My goal is to complete my final issue regarding the disappearing horizontal lines, as well as test the whole MVP Plus version. It is important that multiple persons test the MVP Plus version to ensure that there are no critical issues. I hope that I will have time to continue writing the automated test plan, so that I may soon start coding the actual tests.

Monday

Today I investigated the disappearing form horizontal lines issue. As I couldn't find a form where this issue persisted, I requested the customer to link a form with a disappearing horizontal line. I then continued writing the test plan, transforming it into a Nebula documentation including a test plan for the different components. It is important to document the project components and functionalities to ensure contingency.

Lastly, I investigated a bug where the DataSpace tables returned incorrect data in the table rows upon filtering. I began by reviewing the components of the page, which features a table using React DataSheetGrid, with no standard mapping row in place. The filter function worked by comparing the user's input to the data in the various table cells, displaying rows where a match was found. The issue arose when a row contained an empty cell, other row's data would populate that empty cell after filtering. After some research, I discovered that the problem was related to the key configuration, The DataSheetGrid component lacked a key, so I assigned it to the length of the filtered rows. This ensured that each time the data was filtered, the table would re-render, correctly displaying the data from the database.

Tuesday

I began my day by testing the termination process in the testing environment. My colleague had encountered an issue where some processes couldn't be terminated, displaying an error message indicating that the required data was missing. This was related to the error handling I had implemented to prevent indefinite loading screen when no action ID for termination is found. I suspected there might be an issue with the error logic, so I reviewed the returned JSON. As it turns out, my error handling is working correctly, as there was indeed no termination action ID in those specific processes. To investigate further, I created a few processes of my own, all of which had a termination action ID assigned. I messaged the project's technical teams group chat to share my findings and asked if they knew the cause of the issue or if we should reach out to the BPMN tool provider.

Next, I tested the quality reporting function by using the Developer Tools and reviewing the Network tab. It performed as expected, with no processes fetched through the REST API when the quality form is open. I also tested the button colors by navigating through various task forms, during which I noticed that I was occasionally redirected to the dashboard, even with open tasks. Additionally, I found that the delay in opening the task form was due to the first GET request returning a 504 Gateway Timeout when it took too long to fetch form variables. Case instances were also being fetched at the same time, so this might be related. I plan to add a response check for the form GET Request after the variables are retrieved, so if the response isn't successful, it will retry the request.

During the project meeting it was agreed that no new MVP Plus cards can be added before the current ones are completed. While new requests can be evaluated on a case-by-case basis, they should generally be declined unless deemed critical. We also discussed the termination action ID issue, and it seems that my colleague had made some changes to the variables that might have contributed to the problem. We agreed that further testing by multiple team members is necessary to resolve the issue. Additionally, we discussed the differences between the testing and production environment. Since we can't make the data in the testing environment identical to the production, a third environment will need to be created.

Wednesday

Today I worked on the Nebula documentation. I am almost done writing the requirements and will soon be able to start designing the test cases. In addition, I did some non-project related work tasks.

Thursday

No software development work today.

Friday

Today I mainly worked on the documentation and test plan. I was able to complete most of the requirements and started to write the test plan and -case design for the filter. Next week I will be able to start creating the test implementations using Robot Framework. I reported my progress to the technical lead, and he agreed to help me get started on Monday.

Weekly analysis 6

I couldn’t complete my last MVP Plus card, where the task forms horizontal lines weren’t displayed in some task forms. It is quite impossible to repair an issue that you cannot replicate yourself. When debugging, you must first understand the problem. I do understand that the horizontal line should be displayed on the form, but I cannot find the reason behind the defect without knowing where it is missing. From my point of view there are horizontal lines displayed on the form, so I do not see why they wouldn’t be displayed on some, assuming they have been added to the form in the BPMN tool. Therefore, I will not be able to solve this issue without the customer defining the issue in detail.

We performed many tests this week, discovering new defects that need to be resolved. Luckily, I was able to solve the DataSpace table issue where incorrect data in the table rows was returned upon filtering. This was a critical issue, as the users couldn’t rely on the displayed data’s correctness. The users who have discovered this malfunction refuse to use the DataSpace table created for them and will only do so once the error is resolved in production. This will require more manual work from our side, as they manage their data in an excel, from which we update the data in the DataSpace table. This is a classic example of the impact of a defective application which was discussed in week 1.

When executing software tests, it is necessary to upkeep traceability. Traceability ensures clear tracking of progress, requirements coverage, and execution results, allowing teams to quickly assess progress and coverage. This can be accomplished by breaking down requirements, assigning test case references and mapping them in a table. By listing and linking requirements and test cases we can clearly identify dependencies between them. Figure 7 below shows an example of such traceability and coverage tables. (Homès 2024, Chapter 4)

List of requirements

Reference	Requirement
EX-001.01	Text of requirement 1.01
EX-001.02	Text of requirement 1.02
EX-001.03	Text of requirement 1.03
EX-002.01	Text of requirement 2.01
EX-002.02	Text of requirement 2.02
EX-003.01	...
...	...

Test cases and design progress

Reference	Test case	Designed
CT-0101.01	Description of test case 0101.01	OK
CT-0101.02	Description of test case 0101.02	OK
CT-0102.01	Description of test case 0102.01	OK
CT-0102.02	Description of test case 0102.02	In progress
CT-0102.03	Description of test case 0102.03	In progress
CT-0103.01	Description of test case 0102.03	--
...	...	...

Traceability and execution progress

Requirement reference	Test case reference	Execution process			
		Run1	Run2	Run3	...
EX-001.01	CT-0101.01	Fail	Pass		
EX-001.01	CT-0101.02	Pass	Pass		
EX-001.02	CT-0102.01	Fail	Fail		
EX-001.02	CT-0102.02	--	--		
EX-001.02	CT-0102.03	--	--		
EX-001.03	CT-0103.01	--	--		
EX-002.01	...	...	...		
EX-002.02	...	...	...		

Figure 7 - Traceability and coverage test table (Homès 2024, Chapter 4)

There are two types of testing techniques, static and dynamic. Static testing, or verification testing, is performed without executing the code to check and prevent defects. Informal reviews, code walkthroughs, technical reviews, code reviews and code inspection are all examples of static test techniques. In dynamic testing the code is executed to analyze whether an input produces the desired output, confirming that the software works as per business requirements. Dynamic testing includes techniques such as *White-, Black- and Grey Box testing.* The difference between White- and Black Box testing is that in White Box testing you inspect the internal workings of the code, whereas Black Box testing only inspects the applications functionalities. Grey Box testing is a combination of these two. (GeeksforGeeks 2024c, GeeksforGeeks 2024d)

I have performed a combination of static and dynamic tests throughout the project. When I am developing a new feature or fixing a bug, I conduct static testing by inspecting the code. At times I explain the code to my fellow colleague when I cannot find a solution to the problem at hand. We also conduct frequent code reviews, as all code needs to be approved in a pull request before merging. This week's topic has been dynamic testing though, as I have been creating the test plan including test cases. The test cases are based on business requirements, where an input involving a user action is producing an output. As we have had returning bugs in cleared browser cookies and filtering functionalities, I decided to write a test plan and test case design for those scenarios.

The primary purpose of a test plan is to document and communicate testing details to all stakeholders. A test plan is a document outlining the scope, resources, approach and schedule for planned testing activities. It defines key items such as test items, testable features, testing tasks, assigned responsibilities, tester independence, test environment, test design techniques, and the entry and exit criteria. Additionally, it explains the rationale behind these choices and highlights any risks requiring contingency plans. A test case on the other hand provides step-by-step instructions for testers to validate specific aspects of the application's functionality. (BrowserStack n.d.b)

My test plan - if you can even call it that - falls quite short from the above definition. It consists only of a few sentences outlining what is being tested and the number of required test cases. However, my focus is on designing test cases for automated tests, making it unnecessary to include everything described. I could provide a high-level overview of the Robot Framework tests, but since tests will include descriptions in the code, maintaining a separate, highly detailed list elsewhere could quickly become redundant and outdated. The test case designs on the other hand fill their purpose. First, I have drawn a use case diagram, visualizing the users process from start to end. Then I have added a table outlining the precondition, postcondition, normal flow, and alternative flows and exceptions. I have also included pictures of the specific buttons to illustrate the different kinds of filter states and corresponding buttons. Based on these I can write the Robot Framework test when I get that far.

3.7 Observation week 7

The team's goal for this week is to make progress in setting up the new development environment in AWS and prepare the MVP Plus version release. I will focus on performing regression testing, retesting and examine the web application UI to validate application quality. I aim to also complete the correct dashboard status issue that has been returned to development by the customer. The Robot Framework test environment should be configured this week as well, and hopefully I will have time to create the first automated tests.

Monday

During today's meetings we discussed management's decision regarding the MVP Plus release. Management decided that only the DataSpace customer list bug, the dashboard process status bug and the optimization of the API calls to the BPMN tool would be included from the original requirements. This means we need to test them in the testing environment along with the other modifications and cherry-pick relevant commits when publishing the next version. The remaining completed requirements will be released after the new environments are created in AWS and the BPMN tool. We have a meeting with AWS on Friday, where I will participate in a listening role. Additionally, we discussed how to handle and announce Release Notes. One option is to create a new dedicated channel in our intranet, but for now, we are tracking releases in a Teams chat. We also need to establish a weekly release schedule, such as Tuesday evenings, aligning with other integrations.

Since our Technical Lead was unavailable today, we couldn't setup Robot Framework together. Instead, I continued documenting the Nebula UI.

Tuesday

Today I didn't work directly on the project, as we had a workshop focused on defining the phases of a user interface, integration or RPA project. We combined previously built cases to map out these phases, outlining what each phase entails. The goal is to create a BPMN process that standardizes our approach, improves clarity of both our team and customers, and enhances communication and quality.

Wednesday

Today I tested the API changes implemented by my colleague. The REST API requests were modified to filter data directly in the BPMN tool instead of filtering it in the frontend. This reduced the JSON response size and improved the table row loading speed. To verify the changes, I monitored

Developer Tools Network calls, compared displayed rows to the BPMN tool data, and tested various filters. Although filtering is more complex in the BPMN tool, I am confident that the results are accurate.

Additionally, I worked on ensuring users are immediately redirected to the dashboard when no tasks remain, as this wasn't working consistently. I found that removing *"&& !tasks!"* from an if statement that checks whether there are any tasks improved reliability, but there is still a two-second delay before redirection. This delay was intentionally added earlier to ensure that resolved process stages have the correct status in the dashboard before redirecting. However, the customer manually navigated to the dashboard before the status was updated, which means that they delay in redirection doesn't solve the issue. Since the GET request retrieves the status directly from the BPMN API, the displayed status is always the most recent one from the request. To clarify this, I recorded a video demonstration showing the process status in the BPMN tool UI and Nebula UI simultaneously. The video clearly illustrates that the status doesn't update instantly in the BPMN tool UI either. Hopefully. this visual explanation helps the customer understand why an immediate update isn't technically possible.

Thursday

Today I focused on improving the forms button UI. While examining the web application, I noticed that buttons wrapped incorrectly within themselves instead of moving to the next line when space was insufficient. I also found buttons that weren't aligned with the theme. I fixed these by inspecting elements in Developer Tools and updating styles in the App.css file. Additionally, I found that text overlapped images when the screen width was reduced. To resolve this, I adjusted the image width to fit-content, ensuring that elements wrap correctly instead of overlapping. Since all forms are displayed with a single multi-component, I thoroughly tested these changes with all kinds of forms to ensure compatibility. Once satisfied, I ran the linter and submitted a pull request with a note that it should only be merged once we can make changes to the testing environment again. Currently, the customer has frozen changes, and we have already prepared cherry-picked updates for the release approved changes in the testing environment.

Friday

Today I installed Robot Framework in our project with support from the Technical Lead. We evaluated different options and concluded that the Selenium Library was the best choice. I was able to complete the installations without assistance, although it was nice to have the support. We also installed WebDriver Manager for cross-browser testing. The Chrome WebDriver installation was

successful, but we encountered an issue with Edge WebDriver. We resolved this by assigning a universal link path, ensuring accessibility once pushed to development branch.

After we were done with the installations I went ahead and did my first test. I wrote an automated login test using my own credentials. I plan to request robot-specific credentials on Monday and store them securely in the env file. But for now, my credentials will do as I keep the code local. To make the test work, I had to add IDs to input fields on the login page, as these identifiers are necessary for automation. Initially, I struggled to run the test in PyCharm, so I asked ChatGPT for guidance. It suggested creating a custom run configuration by setting the script path to the robot.bat file, specifying the Robot Framework test file path and defining an output folder for test results. After applying these settings, the test worked like a charm.

The second test I created was “Delete Cookies”. Our application has experienced some crashes due to missing null checks in cookies. By clearing cookies and reloading the dashboard, we can verify with this regression test that the application works correctly for first-time users.

In the afternoon we had a meeting with Amazon Web Services to discuss the new development environment. The setup is straightforward – it’s a copy of production and staging, but with no statistics database (only configuration database and pipeline). Furthermore, all developers should have full read and write permissions for the environment as it is used for development. Since the new BPMN environment hasn’t been created yet, we couldn’t complete setup. Once it’s ready, we will finalize the configurations. We also need to decide on the deployment strategy for development branches. Either we deploy every new branch created by developers, or we deploy only the common development branch where changes are merged. We need to evaluate cost implications, but deploying only the common development branch is likely the most cost-effective approach.

Weekly analysis 7

I successfully achieved all my goals for the week, aligning with the team's objectives of progressing in setting up the new development environment in AWS and preparing the MVP Plus version release. I performed regression testing after the REST API filtering changes and validated the application by retesting known bugs such as the redirection from the task page to the dashboard. However, it wasn't technically possible to ensure that the process phase status displayed correctly on the dashboard, after the user completed the last task in a phase. Explaining this to the customer has been challenging, as they remain adamant about resolving the issue. Hopefully the recorded video will finally help them understand why it isn't feasible.

As the new environments haven't been configured yet, management decided to release only three issues from the original scope. Therefore, we need to cherry-pick these changes, of which the dashboard process status bug won't even be completed as it isn't technically possible. A git cherry-pick selects Git commits by reference and appends them to the current working HEAD (Atlassian n.d.b). I hadn't heard about git cherry-picks before this project, but it sounds very demanding. All changes are interconnected, so how can we ensure that the cherry-picked version of the codebase works?

With Git cherry-pick specifics commits are selected (instead of whole branches) which are then integrated into a new branch before merging with the target branch. Thereby the cherry-picked version can be validated before merging into production. Git cherry-picks should be avoided and used only when absolutely necessary. Some of these scenarios are hotfixes, recovering lost commits from stale branches or releasing a specific feature from a shared codebase. (Atlassian n.d.b; Git-Tower n.d.; 314rate 8 October 2024)

In our case we needed to release specific features from our testing branch that consisted of multiple other features as well. Therefore, it wasn't possible to merge or rebase the testing and production branch, leaving only one option – Git cherry-pick. Even though I won't be the one executing the cherry-pick, but the Technical Lead, I have learned something new. Whenever I am in a similar scenario in the future, I will be able to suggest cherry-picking and possibly perform it with the support of git command documentation.

I am very happy that we found the time at the end of the week to configure the Robot Framework test environment. Although I probably could've performed the installation on my own, it was important to discuss what libraries and web drivers should be used in the project. Completing the installation allowed me to create my very first tests and getting acquainted with the framework. As I don't have much experience with Robot Framework, I am quite slow in creating the tests based on the test case

designs. I am however confident, that I will become more efficient over time. Later, I can also follow suit from previously created tests, which will quicken the process.

Robot Framework is an open-sourced, Python-based, keyword-driven test automation framework that is ideal for acceptance testing. The framework allows you to pinpoint failures, reduce errors and create a test audit trail while being fully automated and reusable. The Robot Framework ecosystem consists of multiple components. First the user configures tests and test data that are then run with Robot Framework. The Robot Framework communicates with the testing tool (actual testing software) through a testing tool driver. The testing tool connects with the end application and returns the test results through the testing tool driver back to Robot Framework and the user. The Robot Framework Ecosystem is illustrated in Figure 8 below. (Bisht 2013, Chapter 1; Robot Framework n.d.a)

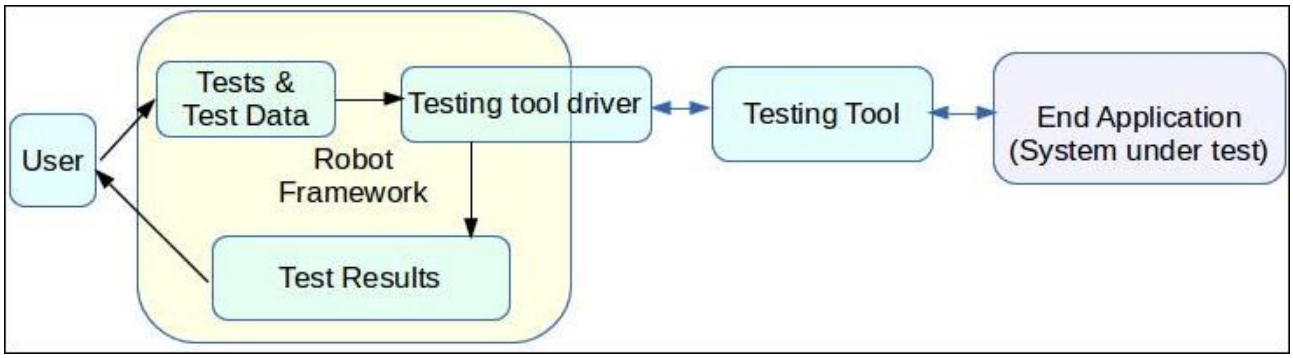

Figure 8 - Robot Framework Ecosystem (Bisht 2013, Chapter 1)

To run automated tests for Web Applications with Robot Framework, you must first have Python installed. Once Python is installed, Robot Framework can be installed with the command “pip install robotframework”. It is also recommended to manage dependencies with poetry, which lets you specify your projects dependencies and automatically handle their installation and updates. Additionally, Robot Framework automated tests require a library to run and which one you choose depends on your application and what you want to test. The steering of the UI in web applications can be tested with the Selenium- and Browser Library. A websites services and RESTful APIs can on the other can be tested with the Requests Library. (Robot Framework n.d.b; Robot Framework n.d.c)

As we are testing a Web Applications UI in a web browser, we were contemplating between the Selenium Library and Browser Library The differences between these two libraries are that Selenium drives on Selenium which is based on Java and Browser is based on Playwright which is based on NodeJS (Github n.d.a, Github n.d.b). Both have their pros and cons, but my senior colleague preferred Selenium, thus we chose that one. I was content with the choice as I have used the Selenium library during the Haaga-Helia Software Testing course.

Finally, the Web Application tests for UI in the browser require a WebDriver manager to work. With a Selenium WebDriver configured, it is possible to perform a wide range of tests, such as functional testing[1], cross-browser testing [2]and regression testing[3]. Currently the following browser WebDriver's are available in the Python WebDriver manager: Google Chrome, Mozilla Firefox, Chromium based Opera, Microsoft Edge, Edge Chromium and Internet Explorer. (BrowserStack n.d.a.; PyPi 2021)

We installed Google Chrome and Microsoft Edge web drivers, as these are most used in our company. It is important to test the web application UI in different browsers to ensure compatibility and consistent functionality across different rendering engines (Siddiqui 26 December 2023; Boamah 6 March 2024). Different browsers may interpret HTML, CSS, and JavaScript differently, which can lead to variations in layout, behavior, or performance (Siddiqui 26 December 2023; Boamah 6 March 2024). By testing in both Google Chrome and Microsoft edge, we can identify and resolve any browser-specific issues, ensuring a smooth user experience for all users.

[1] Functional testing validates the web application functionality with user interactions by verifying expected outcomes(BrowserStack n.d.a)

[2] Cross-Browser Testing validates a web applications consistency across multiple browsers and/or versions. (BrowserStack n.d.a)

[3] Regression Testing verifies existing web application functionality when a new feature is introduced. (BrowserStack n.d.a)

3.8 Observation week 8

I aim to focus on the Robot Framework automated tests this week. First, I need to acquire the test sign-in credentials, whereafter I can continue coding the next test cases. Hopefully we can integrate the automated tests to the development and testing branches soon to assist regression testing.

Monday

In today's meetings, I asked about the required sign-in credentials for the Robot Framework tests, and whether existing credentials were available. It turns out the robots' access Nebula via API and don't need separate login credentials. Nevertheless, the Project Manager recommended creating a dedicated user credential for this purpose, costing around 20 € / month. I was authorized to proceed with the order through IT. During the project meeting with the customer, we confirmed that the MVP Plus version will be launched today at 11:00. Everything is ready on both the BPMN and UI sides. Once live, the customer will terminate and reopen early-stage processes to avoid lingering bugs from the old BPMN version.

I continued working on Robot Framework tests, starting with securely storing my own credentials. After consulting ChatGPT, I created a new robot file for variables and added it to *.gitignore*. I faced multiple issues, including import errors (due to improper resource referencing), typos, and insufficient wait times before validation. Eventually I resolved them, and the test passed - though the password still appears in the test log. I will investigate that further tomorrow. My first test cases Valid Login and Delete cookies is illustrated in Figure 9 below.

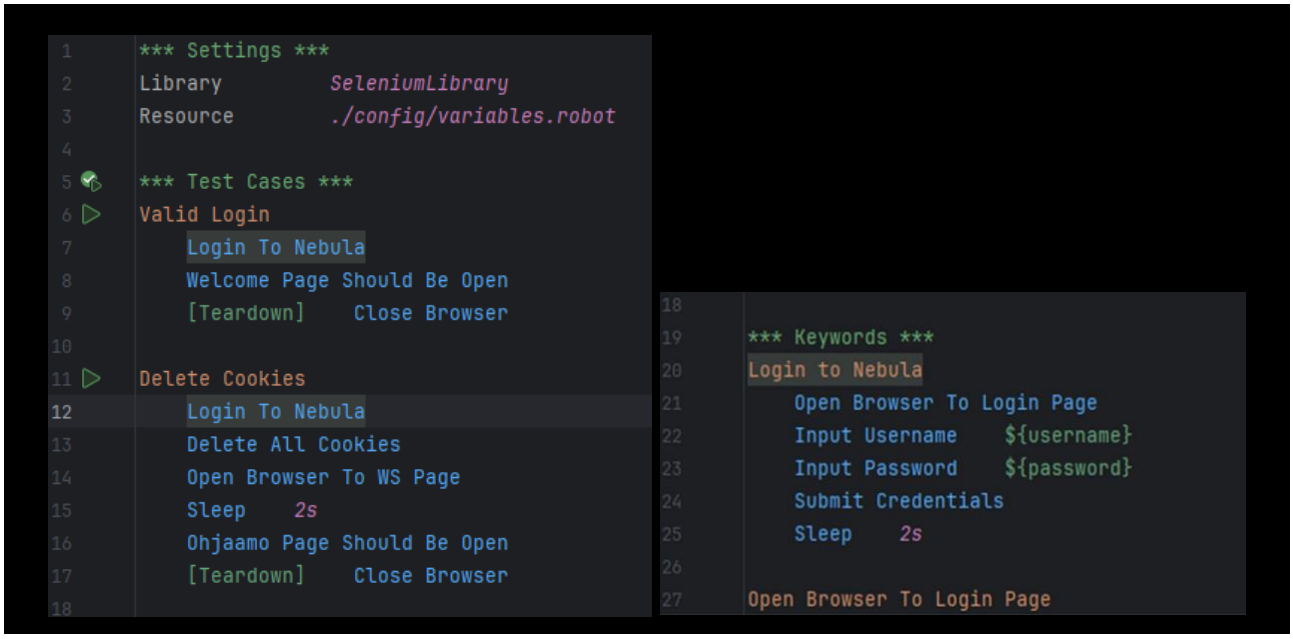

Figure 9 - Valid Login and Delete Cookies Robot Framework test cases

Tuesday

I planned to continue working on the Robot Framework tests today but had to shift focus to urgent customer requests for the Nebula UI. The customer reported an issue where filtering “Ready” processes didn’t display in the table. This occurs because these processes are filtered in the BPMN tool, unlike user and team filters that are stored in our database. One possible fix is adding a button to apply the new filters. The customer also encountered a DataSpace table issue, where users couldn’t save new rows, and adding a new row moved them to the top of the table instead keeping them in place. I investigated and found that a key added to React DataSheetGrid caused the scrolling bug. After confirming the issue, I rolled back the change, which unfortunately reinstated the bug where incorrect data appeared in table cells when filtering.

As for the bug row-saving issue, that was a bit more peculiar. I couldn’t reproduce the exact bug, but I did get a “Cannot read properties of undefined (reading ‘startsWith’)” error when selecting a previously entered text value. However, the customer’s video showed no confirmation that the save was successful. Additionally. I noticed that when saving a new row, the customer’s name (retrieved from our database), wasn’t immediately visible. I resolved this by adding a listener to a useEffect function, reordering the code and cleaning up the JSON by replacing undefined values with empty strings. This ensured controlled data processing, preventing cells from displaying incorrect values when filtering or adding new rows.

In the afternoon, the customer reported that training process tasks were appearing in the working version of WorkSpace (we maintain separate versions for training and actual work). Fixing this required an update in both the BPMN tool and UI, which I handled with a colleague. We changed the training process assignee in BPMN and added a condition in the code to check for ‘?Training=True’ in the search parameters. This ensures training processes and tasks appear only when explicitly requested. Once the fix was tested, I submitted a pull request to the development and testing branch. They were approved in time, and I confirmed the solution worked in testing. The customer will now decide on a release schedule.

Wednesday

Today I continued working on the DataSpace table to verify yesterday’s changes and uncover any remaining issues. I first ensured that the JSON cleanup and controlled state handling were working smoothly. Then, I investigated a right-click issue where both our custom menu and the browser’s default menu were appearing, sometimes making our menu inaccessible. I resolved this by adding an event listener to the tables div, preventing the browser menu from displaying when right-clicking within the table. In figure 10 below illustrates the solution.

```
<div onContextMenu={(event : MouseEvent<HTMLDivElement> ) : void  => {
    event.preventDefault()
}}>
```

Figure 10 - Event listener preventing browser menu

Next, I revisited the “Cannot read properties of undefined (reading ‘startsWith’)” error from yesterday. Like the JSON cleanup issue, this stemmed from new row cell values being undefined. When users selected a previously entered value, the transition from an uncontrolled to a controlled state triggered the error. I initially tried enforcing controlled states for new row columns, but since multiple tables use the same component, defining each would be tedious and problematic for some column configurations. Since the issue only occurred when selecting a past value, I opted for a simpler solution: disabling the browsers autocomplete using an onFocus handler. The solution is illustrated in Figure 11 below.

```
const handleFocus = (event) : void  => {
    if (event.target.tagName === 'INPUT') {
        event.target.setAttribute( qualifiedName: 'autocomplete',  value: 'off');
    }
};
```

Figure 11 - Disable autocomplete with onFocus handler

I also found that no error message appeared when the application encountered a connection issue. To address this, I added 404-error prompt instructing users to reload the page and check their VPN connection. While testing by disconnecting from the database, I noticed the error message was always the same. I discovered the issue stemmed from incorrect error handling – error comparison was based on an expected code, but the actual response returned text instead. I modified the createRecord POST method to return the correct JSON response rather than generating a new error. This allowed the frontend to properly handle errors and display the appropriate messages instead of a generic fallback.

Thursday

Today I completed and tested locally bugfixes and error handling of the DataSpace tables. At the end of the day, I had resolved these bugs:

1. *when filtering, incorrect data is shown on the fields*
2. *when saving a new row, the customer’s name is not displayed in its field right away*

3. *when adding a new row and writing in an optional field and the field above is empty, the new fields data is shown on the field that was and should be empty after it has been saved.*
4. *when saving an identical row, the page crashes and then returns the incorrect error message*
5. *when right clicking the table, the browsers own options menu is visible*
6. *when selecting a previously used input, the page crashes*
7. *when adding a new row, user is moved to the top of the table*

Friday

After verifying the DataSpace table bug fixes from yesterday, I submitted a pull request to the development branch. However, a bug remains in the hover fields, where typing isn't due to incorrect focus. My colleague noted that the focus is on the entire table instead of the cell and that the customer is already aware of the issue. I'll attempt to fix this, but if unsuccessful, I will leave it as it is.

Weekly analysis 8

This week didn't go as planned – I intended to focus on creating Robot Framework automated tests but shifted priorities to urgent bug fixes. I did submit the test login credentials request to IT and securely stored my own credentials in the *variables.robot* file, though the password still appears in the test logs. After the customer reported the issues with process filtering and the DataSpace table, I spent the rest of the week resolving those and other bugs I uncovered along the way. Addressing unexpected production issues quickly is crucial for maintaining customer satisfaction. Fixing bugs immediately not only prevents disruptions, but also helps identify root causes before they escalate, ensuring a more stable system in the long run.

It was concerning that the bug causing incorrect data to appear in table cells when filtering had to be reintroduced. As I have pointed out before, some users refuse using the DataSpace tables while the table might display incorrect data upon filtering. Therefore, it was imminent that I resolved this issue as quickly as possible. In the end, I found that the root cause was that the data had been saved with *null* values in the database. I solved this by "cleaning up" the data before displaying it, meaning that I transformed null values into empty strings. Now when the table is filtered, other rows data won't populate the cells with null values.

It is hard to account for everything when developing a new product. As we had only one process developed, we didn't consider how different versions should be processed in the Nebula UI. It is clear in the BPMN tool, as training and working are technically different processes. The Nebula UI however displays tasks based on the assignee, and when they were the same for training and working WorkSpace, both were incorrectly displayed. Now we know to create specific assignees for each future BPMN process to enable retrieving tasks from a specific process.

Event Listeners are JavaScript functions that as defined by AWS "waits for an event to occur then responds to it" (AWS n.d. b). I find Event Listeners both a blessing and a curse. They are a blessing in the sense that it's not necessary to develop every feature, as it's possible to profit from pre-built features. They are a curse when the application triggers a custom and pre-existing feature simultaneously, resulting in undesired application behavior. I experienced it this week, when I discovered the bug where both our custom-made menu and the browsers menu were displayed on top of each other when the user right clicked the table. The right-click triggers the event, and the menus are displayed due to the browser listening for user actions. By adding the *'preventDefault'* inside the table, it is possible to easily prevent a pre-existing default feature from being triggered. The Event preventDefault method ensures that the normal action won't be triggered, unless stated explicitly (MDN n.d.c).

It is important to display correct and descriptive error messages to inform the user what the issue is and/or guide what corrective actions can be taken. Luckily, I discovered and corrected the defective error handling we had in the web application this week. Different error codes imply different error causes, all represented by a number and a string. By utilizing the different returned error numbers, it is possible to customize the error messages displayed to the users. I for instance added the error message (translation from Finnish) “Identical row is already found. Customer number: X” when the returned error code was 409. X represents the customer code object, which informs the user which customers table row initiated the error.

A reliable and stable website can be ensured by implementing effective error handling. This entails identifying, analyzing, and fixing error situations to upkeep a smooth user experience. By being proactive in error situations, the probability for disruptions is minimized and user trust is strengthened. Users understand the error cause and how to recover from it when clear instructions in error messages are provided. This naturally improves user experience, as users don’t need to investigate what happened. Well-executed error handling can also safeguard users’ sensitive information from being exploited by attackers when it is ensured that error messages don’t expose vulnerabilities or reveal sensitive data. Lastly, error handling is a great tool for troubleshooting, as error logging can provide important insights in the errors root cause, which enables effective repairing of said error. (Webflow Team 1 May 2024; Moretti 16 June 2023)

I have implemented success popup messages to the website as well. I find it as important to inform the user of a successful save or form submission. Otherwise, the user might wonder whether their user action was recorded by the website and the desired result achieved. Furthermore, I believe informing users builds trust, removing the aspect of second-guessing actions. Implementing message popups is especially important when nothing but the Developer Tools Network settings would tell whether an API request was successful. It is not feasible to expect a user to keep Developer Tool open when browsing a website, nor is it to expect that they know what it is or how to open it.

In modern programming languages, such as JavaScript, *Try…Catch* blocks are the cornerstones of error handling implementation. They allow graceful management of exception interception, as any unsuccessful line of code beneath the *Try* block, will be caught by the *Catch* block which can trigger alternative execution logic, user notifications, or error logging. A *finally* block can also be added to the code, ensuring execution regardless of what happens in the Try…Catch block. (Chai and Coding 24 February 2024; MDN n.d.d)

3.9 Observation week 9

I plan to focus on debugging found defects as well as implement a new customer request for tax and social security (TAS) payment reference validation. Each issue will be tested both statically and dynamically after completion. As the DataSpace tables have quite many found defects, I will create a manual test case design for me and testers.

Monday

I started my day with tackling the hover field issue, where typing wasn't possible when navigating via the tab key. Adding *autofocus* worked but introduced a new bug where focus jumped to the last element after typing the first letter. Since the same issue exists in production without complaints, I rolled back all changes. Clicking and typing still works, so the field remains usable. After the roll back I merged my other DataSpace table bug fixes and requested testing support from the team. I also fixed a sorting bug by adding *'.toLowerCase()'* to ensure capital letters no longer appear before lowercase ones.

During testing, I found issues the MappinDatasheet component – modified cells remained highlighted after saving, incorrect data sometimes appeared, and timestamp updates weren't automatic. I started rearranging the code based on improvements made to the RobotDatasheet component but will need to continue tomorrow.

In project meetings, we discussed the new environments: a development environment for the UI and a testing environment for the BPMN tool. The UI environment is progressing with AWS, but no estimation is available for BPMN. In the afternoon we successfully published API changes to optimize the dashboard, with testing confirming no issues.

Tuesday

I spent the day investigating the DataSpace component MappingDatasheet issues. Oddly, rolling back to the test environment version still resulted in 500 Internal Server Error when adding new rows, though editing and deleting worked. According to the RFC 9110, this error suggests an unexpected server condition. Comparing test and local environments revealed data discrepancies – likely an ID conflict causing the failure. After confirming with the backend developer, we ran the prefill script to sync the backend, restoring row addition functionality. Tomorrow I'll continue fixing MappingDatasheet bugs.

Wednesday

With the data now synced, fixing MappingDatasheet was much smoother. I replicated improvements from the RobotDatasheet, adjusting error handling since MappingDatasheet lacked customer IDs, using the conflicting data instead. After thorough local testing across multiple sites, I submitted a pull request and created a manual testing guide for my colleague.

Next, I worked on a customer request for the DataSpace table used by our payment-processing robot, “Make”. The customer wanted to validate TAS payment reference numbers, ensuring they are always exactly 16 digits long. I proposed adding a validation that the reference starts with “RF”, which was approved. I implemented this using a function that verifies length and format via regex, integrating it into both update and create functions. I ensured users could edit rows after invalid entry but didn’t have time to test it thoroughly – I’ll finish that tomorrow.

Thursday

I began by thoroughly testing the Make table TAS reference validation for various cases: too short, too long and incorrect format. Everything worked as expected, including the row creation, edition and deletion. After final code review, I removed unused imports and out-commented code before submitting a pull request. My colleague suggested refactoring the function into an arrow function and replacing if statements with a switch case. While implementing this, I realized I hadn’t accounted for undefined values. The user should be able to save a row without a reference if customers handle payments themselves. I added it as the first case, returning false for validation errors. I also informed our tester, who will include it in tomorrow’s test cycle.

In meetings we discussed upcoming release cycles: major releases will occur on the 24th weekday of each month, and minor releases on the 7th. There’s also a request to decrease management involvement and centralize approvals in Jira instead of Microsoft Planner, but no final decision has been made yet.

Friday

I planned to merge the development branch into the testing branch but encountered persistent merge conflicts. I attempted to rebase by merging testing into development branch, with the aim to resolve the conflicts. As it wasn’t possible to directly merge changes into the development branch without a pull request, I pushed the rebased version into a new branch, which I merged into the development branch. As the conflicts remained, I had to give up and request my senior colleague to fix the diverges.

While waiting for pull requests approvals, I investigated a bug where filter fields became un-editable after cancelling a filter change. Debugging by selectively commenting out lines revealed that the issue stemmed from the *'editedFilter'* state. Despite being set to false, logging showed it remained true. Additionally, an incorrect state was passed in one component and unnecessarily redefined in a child component. Fixing these helped but didn't fix the bug entirely – I'll continue investigating next week.

Weekly analysis 9

Throughout this project I have spent much time debugging defects and failures, and this week was no different. We have especially discovered DataSpace table bugs lately, most likely due to me fixing previously found bugs and thereby conducting more explicit testing. The DataSpace tables are especially challenging to debug because they are huge code files (almost 1000 lines long) and are in desperate need for refactoring. Because everything is so heavily linked together in long functions and nestled if statements, it is sometimes hard to foresee what impacts one change in the code has. Here it is beneficial to have the bigger picture of user requirements, database architecture and robot functionality to understand the implications of changes. Sometimes I learn new dependencies after I have implemented a change and must correct it in hindsight.

Fowler defines refactoring as “the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure”. As refactoring doesn’t produce new features, but only cleans up code that already works, it can sometimes be challenging to convince management that it needs to be done. However, a poorly programmed software is difficult to modify because understanding what to change and predicting its impact on existing code is challenging, increasing the likelihood of mistakes and bugs. Whenever the code is poorly structured, it is better to refactor the code before adding new features. Debugging and adding new features is much easier when the program is constructed into easily understandable sets of functions and other system elements. However, if no changes need to ever be made to the code and nobody ever needs to understand it, then refactoring is unnecessary. (Fowler 1999, Preface & Chapter 1)

Our current codebase requires quite a lot of refactoring, especially in DataSpace. We haven’t done it yet due to time constraints and our plan to replace the React Datasheet Grid component (used in DataSpace tables) to the React MUI table component (used in WorkSpace tables). Therefore, it is a waste of time and energy to refactor it completely, as we will in any case rebuild it from scratch. I have done some refactoring when debugging the code, as I have encountered some duplicate functions across components. Whenever I have been programming the UI for WorkSpace (displays the BPMN processes) I have attempted to create smaller components to facilitate addition of new features. Even so, the index component has grown enormous and needs to be refactored before we can add new BPMN processes to WorkSpace.

According to Fowler it is possible to refactor the codebase successfully by first building a set of automated tests to avoid introducing new bugs. After each *small* change, the code is compiled and tested to easily track down bugs whenever a mistake is made. When searching for bits in the code to refactor, try to identify long statement methods and decompose them into smaller methods. It is also useful to ensure variable names are descriptive as good code can be understood by humans,

not only computers. Temporary variables are also often a problem, as they are only available inside their own routine, generating long and complex routines. Therefore, it might be essential to transform them into global variables or methods. Lastly, duplicates should be eliminated to ensure the code states everything once, the core of good design. (Fowler 1999, Chapter 1 & Chapter 2)

Fowler (1999) makes a great point by emphasizing that refactoring is done in small increments that are always verified with a set of automated tests. I've been too lazy to create the automated tests myself when adding a new feature or refactoring the code. Nevertheless, I do realize now the benefit from it. I could have saved myself from a lot of headaches had I built the automated tests early on. I also tend to refactor in quite big chunks before testing, mainly because I must test manually. If I could test the program with a click of a button, I would do so much more often. Thereby I would also find the cause to the bugs I might have introduced right away.

The point of refactoring is producing clean code. Clean code is readable, understandable, and maintainable, with a well-structured concise design that uses meaningful names, follows best practices and design patters. Clean code supports team collaboration as well as issue resolution and debugging. Furthermore, it prioritizes readability and behavior over performance and implementation details. Even so, writing clean code often results in better performance when bottlenecks are removed. The Pareto 20/80 rule states that by addressing 20% of critical bottlenecks, performance is increased with 80%. However, by improving 20% of bad performance, system speed is likely increased with 80%. Therefore, performance optimizations should be done only with hard proof evidence after refactoring. (Contieri 2023, Chapter 1; Codacy 19 December 2023)

When coding a new feature, I often focus on getting a working solution first, then refining it for readability and breaking it into smaller parts. While I often identify refactoring opportunities myself, sometimes a second pair of eyes helps. Like this week for instance, I implemented the TAS reference regex validation using an if statement solution, but during the pull request code review, my colleague suggested that a switch-case solution would be much smoother. I know I still have a lot to learn before I consistently write clean code. Tight project deadlines sometimes make me cut corners, but I recognize that writing clean code ultimately saves time when I or another developer needs to revisit it.

Writing clean code in TypeScript involves utilizing its powerful features to improve readability, maintainability, and prevent errors. *Type annotations* help catch errors early and enhance code clarity, while *enums* provide named constants that reduce magic numbers. *Optional chaining* and *nullish coalescing* both make the code more robust by safely handling undefined or null values. *Generics* promote reusability with different types, and *interfaces* enforce contracts for objects. classes and functions. *Destructuring* simplifies variable extraction, and *async/await* improves

asynchronous code readability. *Functional programming techniques,* such as higher-order functions, pure functions and immutability, generate readable, predictable and testable code. Lastly, utility types like *discriminated unions, pick-* or *omit helper* enhance type safety. By incorporating these practices, it is possible to create scalable, error-resistant, and well-structured TypeScript applications. (Gouvea 28 February 2023)

The Nebula UI is based on React, programmed with JavaScript and TypeScript. We started developing the UI with JavaScript but pivoted to TypeScript due to its many functionalities, which improved code maintainability and type safety. I hadn't coded with TypeScript before this change in the project and am still uncomfortable in applying many of the functionalities. I have implemented type annotations and interfaces, although only when refactoring, as well as async/await. We have had issues with undefined and null values, some of which could possibly be tackled with optional chaining or nullish coalescing. I'm also curious about implementing higher-order functions by passing a function to a function. I will return to Gouvea's article the next time I'm refactoring or building a new feature.

3.10 Observation week 10

The official launch of Nebula is on Tuesday. As no new UI version is to be released, we will focus on setting up the new environments, resolving outstanding bugs, and ensuring a smooth launch for the new Nebula WorkSpace users. I will continue working on the WorkSpace issue where filter fields are un-editable after cancelling a filter change among others.

Monday

I continued investigating the WorkSpace bug preventing text input into the filter fields. Initially, I copied the handleApprove functions' code into handleCancel, as handleApprove didn't trigger the issue. However, the bug persisted, so I rolled everything back. I reckoned it's linked to the return statement, and experimented with adding an extra approve button and adjusted button rendering positions. I discovered that the first button always worked correctly, while others caused the issue. The breakthrough came when I noticed a button's hover text lingering after clicking – text input remained disabled until I hovered over the button again. This revealed that the focus was stuck on the invisible hover text rather than shifting to the input field. I fixed this by ensuring the 'handlePopupClose' function was called in all button functions.

In project meetings we discussed two (2) current topics. First, AWS has completed the setup for the new environment, and we'll meet tomorrow to finalize deployment. This will allow more reliable testing. Second, we discussed the growing risk of merge conflicts due to many unpublished changes. Picking updates selectively might become problematic since both dev and test environments contain all changes. We considered rejecting cherry-picks for the next release, ensuring either all changes go live or none. With so many pending changes, code cleanup and new linter rules remain impractical.

Later, I resolved merge conflicts with my senior colleague, and the correct version was finally published to the testing environment. Testing confirmed that my fixes worked. However, our tester found a bug where adding an identical row triggered both error and success messages. I found the bug as well and began investigating it. I will have to continue tomorrow as I didn't find the root cause.

Tuesday

Today is the official go-live of Nebula WorkSpace. Since all requested features were already tested and deployed, this was more of a business milestone than a technical one. The client side remained quiet, suggesting a smooth rollout - no news is good news. The project has officially concluded, and development will continue as part of regular operations.

I continued debugging the DataSpace table issue where the users received conflicting success and error messages when adding duplicate rows. I traced the problem to the *'errorMessages[ ]'* state array, which was still empty when the function checked for errors. Since the array was empty, the function mistakenly proceeded as if there were none. I made significant progress on the bug fix but feel like the code could be optimized further.

Wednesday

No software development work today.

Thursday

Today I solved the DataSpace table issue where the users received conflicting success and error messages when adding duplicate rows. The root cause to the bug was that the row data was corrupted. The table row data structure is as in table 1 below, depending on whether it is a customer data table or a data mapping table.

Customer data table	**Data mapping table**
{	{
“id”: *number*	“id”: *number*
"customer_id": *number*,	"process_id": *number*,
“customer_data”:{},	"mapping": {},
"process_id": *number*,	"created_at": *date*
"config": {},	"updated_at": *date*
"created_at": *date*,	"updated_by": *"string"*
"updated_at": *date*	}
"updated_by": *"string"*	
}	

Table 1 - DataSpace table data structure

The row data is stored as key-value pairs in either *config* or *mapping* object. I discovered that *updated_at* and *updated_by* was for some rows mistakenly stored in the *config* or *mapping.* When a

seemingly identical row was added to the table, it triggered the error message. However, as there wasn't an identical row mapping stored in the database, the post request was successful, triggering the success message. Hence, the error message shouldn't have been triggered in the first place.

To avoid having conflicting success and error messages, I refactored the code to only trigger the error message if the database returned an error. After we have published all bugfixes to production, we need to clean the database to remove the possibility to have two "identical" rows.

Friday

No software development work today.

Weekly analysis 10

The Nebula project reached a milestone with its official release on Tuesday. Moving forward, development will continue as part of regular operations, with a planned release cycle of twice a month. This should keep the production and development branches closely aligned, reducing the risk of unforeseen merge conflicts and development issues. Unfortunately, this release cycle won't be implemented right away, as business has requested that nothing is released to production before the new environments have been setup. They are concerned that the test results aren't valid if the testing- and production environments aren't identical. It is partially a valid concern, as we have encountered bugs due to these differences. This doesn't however apply to every aspect of the application, which is why it could be negotiable to release some bug fixes. Hopefully, we will have the environments setup and ready soon, to avoid merge conflicts and development issues.

In the weekly analysis 1, a project was defined as having three main characteristics: temporary, unique and progressively elaborate (Brewer & Dittman 2023, 13). The Nebula projects objectives were achieved on the official launch, where payroll professionals implemented the WorkSpace payroll process into their work. Further development is considered as regular operations as the definition of a project is no longer met. Even though Nebula will be developed in steps and increments, it is done with no unique deliverable until the end of its lifecycle.

In the project closing phase the project manager ensures everything planned is completed and objectives met. Here the final product is delivered to the customer, documentation is updated, a final report is written with lessons learned, and organizational process assets are updated. Successful project finalization requires engaging with stakeholders preferably in a meeting. The final meeting should according to the Project Management Institute (2017, 127) take place "to confirm that the deliverables have been accepted, to validate that the exit criteria have been met, formalize the completion of the contracts, to evaluate the satisfaction of the stakeholders, to gather lessons learned, to transfer knowledge and information from the project, and to celebrate success". How do we know whether a project is successful? Traditionally, project success has been measured in time, cost, scope, and quality. These days achieving project objectives are considered as important. Therefore, it is important to document measurable objectives at the start of the project. (Project Management Institute, Inc. 2017, 34 & 121-127)

Although we ultimately reached our objective, I would not consider this a successful project. The original project schedule targeted a spring release, which we didn't meet. Additionally, the product quality could be improved by addressing the lingering bugs in production. One of the main challenges was the poorly defined scope and requirements. New requirements were introduced along the way, and unclear requirements often resulted in rework. Furthermore, we barely followed project

management standards, resulting in inefficiencies and disorganization. We did improve towards the end of the project with a stronger emphasis on communication and the introduction to new tools to better manage and visualize progress.

Sometimes spotting the root cause to a defect comes down to sheer luck. I have been puzzled for a long time by the bug preventing text input into the filter fields and had almost accepted it as an unfortunate feature. Most likely my previous investigations, where I had found focus related issues, contributed to my final breakthrough when I discovered that the focus lingered on a buttons hover text instead of the input field. The more experience I gain in debugging and developing new features, the better I become at delivering high quality products. Each challenge expands my knowledge base, deepens my understanding of different aspects of development, and sharpens my ability to anticipate and account for potential issues.

When looking to improve coding skills it is important to be open to learning new technologies and innovations. As this is a rapidly changing industry, it is important to stay up to date. Additionally, it is beneficial to master one core language while understanding the underlying programming concepts. Once you have a solid grasp of fundamentals in one language, transitioning to other languages and tools is much easier. When learning a new language, try different learning methods and learn the fundamentals incrementally to internalize the concept at hand. Furthermore, get into the habit of reviewing the code of others, asking questions, refactoring, continuous practice and requesting feedback from others. Doing so not only improves your coding skills but also helps recognizing different coding styles, identify best practices, and learn more efficient ways to solve problems. (Indeed Editorial Team 4 March 2025; Gaël 27 November 2019)

I am particularly pleased to have solved the DataSpace table issue where the users received conflicting success and error messages when adding duplicate rows. Fortunately, the bug that initially caused the corrupted rows of data has already been addressed. The longer this error remained in production, the more widespread the corrupted rows became, resulting in increased defective behavior in the application. Once we’re able to release all the bug fixes for the DataSpace table, we must clean the database of any corrupted data. This will ensure that duplicate rows are handled correctly, with the row data adhering strictly to the table’s original specifications, free of any inconsistencies.

4 Discussion

At the beginning of this thesis project, I defined objectives that I aimed to develop during the diary observation weeks. I wanted to refine my skills in TypeScript further by defining missing types and writing clean code according to high quality coding standards. I planned writing tests with Robot Framework; to utilize the skills I learned during a testing course last spring. Furthermore, I predicted that I would develop my debugging skills, as malicious testing usually discovers hidden bugs. I also wanted to develop my skills in drawing new processes with the BPMN tool and improve project management skills.

I somewhat achieved my TypeScript goal as I defined types while coding in TypeScript, but I could've made more of an effort. There is much more to writing clean code in TypeScript besides type annotation, creating interfaces and implementing async functions, as we learned in week 9. It is hard to balance time schedule and take the time to improve your coding skills and the code itself. Incorporating a programming language features into the existing codebase takes time as you must internalize it and recognize use cases. Our React application would benefit from incorporating chaining or nullish coalescing to tackle the adversity we've faced with null values.

I have practiced my clean coding skills when I've refactored the codebase. Refactoring enhances readability and the internal structure of the code, simplifying code modification and decreasing the number of bugs. One could question whether it would be easier to write clean code from the very start. As nice as it sounds, writing perfect code from the beginning is very challenging, unless it's a very simple function. This is because we cannot foresee how the product will evolve and what features will be required. I also find it easier to first get a working structure and then examine how functions could be chopped into smaller logical entities that are easily modified and shared between components. I also ensure that the code logic is commented in the code to ensure clarity for the next developer that revisits it.

As anticipated, I had a lot of opportunities to develop my debugging skills. I find it harder to debug than to code a new component. When coding a new component, you get to start from a clean slate, whereas debugging involves finding the cause from a large, possibly complex codebase. Fortunately, it is possible to identify the source of the bug by investigating the browsers Developer Tools, which narrows down the number of files that need to be examined. This is however not possible when debugging a UX error, as that doesn't necessarily show on the Developer Tools error console log.

Whichever the problem, it is beneficial to take the four (4) debugging steps: understand the problem, gather information, produce potential solutions and start coding the preferred solution. When you

understand what the code is doing as opposed to what it is doing, it is possible to come up with potential solutions. Sometimes, it is beneficial to gather information about the codebase or related documentation. As I was very familiar with the codebase as one of the original programmers, I mostly skipped this step during the observation weeks. I also didn't take the time to come up with and iterate possible solutions. I went with the first idea that came to my mind, which in my defense usually worked. I think that this advice is beneficial for very complex errors, where the solution isn't evident. I also think that the final solution can in some cases reveal itself by working on the solution and logging states and variable values along the way. Most errors during these observation weeks have been due to a state having the incorrect value, which meant that it wasn't set to the correct value somewhere in the code. For example, the pages have an is loading state, and if that state isn't ever set to false, the page will load indefinitely.

When developing a website—especially one that plays a central role in employees' daily work—it's crucial to understand and apply principles of UX design. Although UX wasn't initially a specific goal during the observation weeks, I had to account for it while programming the user interface. Since our project lacks a dedicated UX designer, the customer and I have been responsible for creating the designs ourselves. I believe I have a good eye for effective UX, and whenever something feels off, but I can't quite identify the issue, I seek feedback from a colleague. This often leads to valuable, constructive input.

One of the most challenging aspects of UX is ensuring website responsiveness. This is important not only for the technical performance but also for the user's psychological experience. Long waiting times can be especially frustrating, as users expect a sense of control and immediate feedback from the system. The three (3) key response-time thresholds according to Nielsen's 1993 research are 0.1 seconds (instantaneous), 1 seconds (noticeable, but seamless interaction) and 10 seconds (risk losing the user's attention entirely). The complexity of website responsiveness arises from the many potential causes of lag and how they interact. It's critical to implement proper error handling to prevent excessive server load, which could ultimately lead to crashes. Additionally, it's important to optimize REST API calls by filtering and minimizing the returned data. Smaller payloads and efficient requests result in faster, more responsive performance.

The second objective for the observation weeks was to improve my Robot Framework skills. I wrote a test plan and test case designs for the automated tests. I used the skills I'd learned during a testing course, visualizing the users action options. Although the test plans turned out thin when compared to literature, the test case designs were sufficient. The website features were clarified with use case diagrams and test case tables that outlined the precondition, post condition, normal flow, and alternative flows. These were helpful in test case implementation as I only had to follow the steps that I'd

already defined by translating them into code. Additionally, incorporating the test case designs into the documentation, as well as producing a documentation, visualizes the website functionalities to someone who isn't familiar with it. It will support the orientation of new developers and serve as a dictionary when creating new sites on the website. It is much easier to see what functionalities exists and how their components have been defined from documentation than searching in the codebase.

Although automated tests are important to simplify regression testing and reduce reintroduced bugs, they require a lot of time and effort in the beginning. Therefore, I had to take the time to prioritize and create them. As the project was in the testing and monitoring phase of its lifecycle, it was natural that I developed my manual testing skills. I performed static (verification) testing by inspecting my code and reviewing others code in pull requests. Most performed tests were dynamic, as it was essential to verify user inputs produced desired outputs. I refined my manual test case designing skills, as it is important to have features tested by testers, not only the developers themselves. This is because developers may have blind spots for their own code and might overlook potential issues that an independent tester could identify.

I was not able to develop my BPMN process drawing skills as planned. Although that was part of the original plan, I was direly needed in UI development, debugging and testing. In the end others in the team were allocated to BPMN process development. I don't mind, as I am more interested in programming.

The last objective was to improve my project management skills, which primarily involved improving communication. Successful communication is key to a successful project, and failing to do so often results in failure. We have been struggling with communication throughout the project and have recently made more of an effort to focus on good communication with documented instructions and discussions within the team. The communication improved significantly when we implemented Microsoft Planner to visualize project progress. It enabled us to keep ourselves and the customer up to date without spamming the project chat, whilst allowing developers to focus on programming a solution rather than updating the customer every step of the way.

I have improved my own communication skills as well. At the beginning, I responded quickly when noticing messages, but without giving the response too much thought. This contributed to misunderstandings and undesired results. Now I continue responding to messages as soon as I've noticed them and ensure that the written message is friendly and clear. By responding right away, I avoid my habit of having a hard time to respond when a lot of time has elapsed. Furthermore, I think that it is important to add positive emojis to the messages, as an-emoji free message can be interpreted as rude. Written communication lacks the possibility to interpret body language and tone of voice, which is why a positive emoji can transform a message into a friendly tone.

I have learned a great deal during this thesis project, both through hands-on experience and while exploring theoretical concepts for my weekly analyses. I made a lot of useful insights while writing the weekly analyses, many of which I will carry with me into the future. For instance, when debugging a piece of code, I will consciously apply the four-step process to improve efficiency and accuracy. When I have the time and opportunity to improve the codebase, I will revisit Gouvea's (2023) article *12 TypeScript tricks for Clean Code* as a practical reference. I've also learned the importance of refactoring code in small increments, which allows introduced bugs to surface and be addressed early. Additionally, I aim to also create automated tests from the start whenever possible.

Effective communication, I've realized, requires continuous effort and intentionality. The weekly analyses helped me identify some of my communication weaknesses and gave me strategies to avoid misunderstandings and ensure timely, clear responses.

In the future, I aim to further develop my skills in TypeScript, Robot Framework, and Python. Improving my TypeScript proficiency will involve exploring the language's unique features and strengths—otherwise, I might as well be coding in plain JavaScript. My experience with Robot Framework will naturally deepen as I continue writing new automated tests. This will become especially important when we begin developing new WorkSpaces for the upcoming BPMN processes, as automated regression tests will help ensure we don't introduce new bugs.

As mentioned at the beginning of this thesis, I will also be working on independent Python-based software integration projects soon. Although I have previously completed Python courses, I plan to take a refresher course to rebuild confidence and ensure I'm fully prepared before diving in. The Python software integration projects will also introduce me to Jenkins, of which I have no prior experience.

This thesis project has been a valuable and enriching experience, contributing significantly to my professional growth as a programmer. Writing a diary and analyzing the entries through the lens of theory, has encouraged me to reflect more deeply on myself and my work. I've realized that taking the time to pause and analyze events – rather than constantly pushing forward – is an essential part of personal growth. Without reflection, it's easy to miss out on important lessons or moments of insight. I discovered many useful perspectives in the theoretical material, some of which didn't make it into this thesis but will certainly be revisited in the future.

References

314rate (2024) *Mastering Git Cherry-Pick: Advanced Guide with Real-World Examples | by 314rate | Medium*. Available at: https://medium.com/@314rate/mastering-git-cherry-pick-advanced-guide-with-real-world-examples-3df3d9f284f5 (Accessed: March 13, 2025).

Atlassian (no date) a *Get Started with Jira - Comprehensive Beginner's Guide*. Available at: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software (Accessed: March 27, 2025).

Atlassian (no date) b *Git Cherry Pick. Atlassian Git Tutorial*. Available at: https://www.atlassian.com/git/tutorials/cherry-pick (Accessed: March 11, 2025).

AWS (no date) a *What is Cloud Computing? - Cloud Computing Services, Benefits, and Types*. Available at: https://aws.amazon.com/what-is-cloud-computing/ (Accessed: March 27, 2025).

AWS (no date) b *What is an Event Listener? - Javascript Event Listener Explained*. Available at: https://aws.amazon.com/what-is/event-listener/ (Accessed: March 13, 2025).

AWS Marketplace (no date) *SaaS-based products in AWS Marketplace*. Available at: https://docs.aws.amazon.com/marketplace/latest/userguide/saas-products.html (Accessed: January 11, 2025).

Bisht, Sumit. (2013) *Robot framework test automation*. Packt Publishing.

Boamah, O. (2024) *What is Cross-Browser Compatibility? How to Build Websites that Work Everywhere*. Available at: https://www.freecodecamp.org/news/what-is-cross-browser-compatibility/ (Accessed: March 13, 2025).

Brewer, J.L. and Dittman, K.C. (2023) *Methods of IT Project Management, Fourth Edition*. Purdue University Press.

BrowserStack (no date) a *Selenium Testing: Detailed Guide*. Available at: https://www.browserstack.com/selenium (Accessed: February 27, 2025).

BrowserStack (no date) b *Test Plan vs Test Case: Core Differences*. Available at: https://www.browserstack.com/guide/test-plan-vs-test-case (Accessed: January 20, 2025).

BVOP (no date) *What is Scope Management in Agile Project Management*. Available at: https://bvop.org/learn/scopemanagement/ (Accessed: February 9, 2025).

Chacon, S. and Straub, B. (2014) *Pro Git*. 2nd Edition. Apress. Available at: https://git-scm.com/book/en/v2 (Accessed: March 27, 2025).

Chai and Coding (2024) *Comprehensive Guide to Error Handling in Web Applications. Medium*. Available at: https://medium.com/coffee-coders/comprehensive-guide-to-error-handling-in-web-applications-977e4a2f7f0a (Accessed: March 13, 2025).

Codacy (no date) *What Is Clean Code? A Guide to Principles and Best Practices*. Available at: https://blog.codacy.com/what-is-clean-code (Accessed: March 13, 2025).

Contieri, M. (2023) *Clean Code Cookbook*. O'Reilly Media, Inc.

Dustin, E., Rashka, J. and Paul, J. (1999) *Automated Software Testing: Introduction, Management, and Performance*, *Addison-Wesley Professional*. Addison-Wesley Professional.

Eskerod, Pernille. and Jepsen, A.Lund. (2016) *Project Stakeholder Management*. Routledge.

Fielding, R., Nottingham, M. and Reschke, J. (2022) *RFC 9110: HTTP Semantics*. Available at: https://www.rfc-editor.org/rfc/rfc9110#status.500 (Accessed: September 24, 2024).

Fowler, M. and Beck, K. (1999) *Refactoring : improving the design of existing code*, *Improving the Design of Existing Code*. Reading (MA) : Addison-Wesley.

Gaël, T. (2019) *How to Improve Your Programming Skills*. Available at: https://www.freecodecamp.org/news/how-to-improve-your-programming-skills/ (Accessed: March 13, 2025).

GeeksforGeeks (2024) a *What is Stress Testing in Software Testing?* Available at: https://www.geeksforgeeks.org/stress-testing-software-testing/ (Accessed: January 15, 2025).

GeeksforGeeks (2024) b *Difference Between Stateless and Stateful Protocol*. Available at: https://www.geeksforgeeks.org/difference-between-stateless-and-stateful-protocol/ (Accessed: February 22, 2025).

GeeksforGeeks (2024) c *Difference between Static and Dynamic Testing*. Available at: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-testing/ (Accessed: February 25, 2025).

GeeksforGeeks (2024) d *Difference between Black Box and White and Grey Box Testing*. Available at: https://www.geeksforgeeks.org/difference-between-black-box-vs-white-vs-grey-box-testing/ (Accessed: February 25, 2025).

GitHub (no date) a *MarketSquare/robotframework-browser: Robot Framework Browser library powered by Playwright.* Available at: https://github.com/MarketSquare/robotframework-browser (Accessed: February 27, 2025).

GitHub (no date) b *robotframework/SeleniumLibrary: Web testing library for Robot Framework*. Available at: https://github.com/robotframework/SeleniumLibrary (Accessed: February 27, 2025).

Git-Tower (no date) *Git Cherry Pick - How to use the "cherry-pick" command in Git. Learn Version Control with Git*. Available at: https://www.git-tower.com/learn/git/faq/cherry-pick (Accessed: March 13, 2025).

Gouvea, M.V. (2023) *12 TypeScript tricks for Clean Code. Medium*, *Medium*. Available at: https://medium.com/@mvsg/12-typescript-tricks-for-clean-code-b23651dd0430 (Accessed: March 6, 2025).

Hashemi-Pour, C. and Churchville, F. (2024) *What Is a User Interface (UI)? TechTarget*. Available at: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI (Accessed: April 17, 2025).

Hegde, P. (2019) *How to Develop a Test Automation Strategy*. Available at: https://www.logigear.com/blogs/test-automation/How-to-Develop-a-Test-Automation-Strategy (Accessed: September 8, 2024).

Homes, B. (2012) *Fundamentals of Software Testing*. London: Wiley-ISTE. Available at: http://www.amazon.com/Fundamentals-Software-Testing-Bernard-egrave/dp/1848213247 (Accessed: March 13, 2025).

Humble, Charles. (2024) *Improve Your Problem-Solving Skills*. O'Reilly Media, Inc.

IBM (no date) *What is a REST API?* Available at: https://www.ibm.com/think/topics/rest-apis (Accessed: February 17, 2025).

Indeed Editorial Team (2025) *How To Improve Coding Skills (With Practical Tips And Advice)*. Available at: https://in.indeed.com/career-advice/career-development/how-to-improve-coding-skills (Accessed: March 8, 2025).

Institute, P.M. (2017) *A guide to the project management body of knowledge (PMBOK guide)*. Sixth edition. Newtown Square, Pennsylvania, USA : Project Management Institute (PMBOK guide).

ISO (2010) *ISO 9241-210:2010(en), Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*. Available at: https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-1:v1:en (Accessed: February 15, 2025).

ISTQB Glossary a (no date) *error*. Available at: https://glossary.istqb.org/en_US/term/error (Accessed: January 18, 2025).

ISTQB Glossary b (no date) *defect*. Available at: https://glossary.istqb.org/en_US/term/defect (Accessed: January 18, 2025).

ISTQB Glossary c (no date) *failure*. Available at: https://glossary.istqb.org/en_US/term/failure (Accessed: January 18, 2025).

JSON (no date) *Introducing JSON*. Available at: https://www.json.org/json-en.html (Accessed: April 27, 2025).

Kvalnes, Øyvin. (2023) *Communication Climate at Work*. Springer International Publishing.

Landau, P. (2023) *What Is Project Scope? Scope Management Steps, Tips & Tools*. Available at: https://www.projectmanager.com/blog/project-scope (Accessed: February 9, 2025).

Martech Zone (no date) *What Is ISAE? International Standards On Assurance Engagements. Martech Zone Acronyms*. Available at: https://martech.zone/acronym/isae/ (Accessed: April 17, 2025).

MDN (no date) a *Cacheable - MDN Web Docs Glossary: Definitions of Web-related terms*. Available at: https://developer.mozilla.org/en-US/docs/Glossary/Cacheable (Accessed: February 22, 2025).

MDN (no date) b *CSS: Cascading Style Sheets*. Available at: https://developer.mozilla.org/en-US/docs/Web/CSS (Accessed: March 27, 2025).

MDN (no date) c *Event: preventDefault() method - Web APIs*. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault (Accessed: March 13, 2025).

MDN (no date) d *try...catch - JavaScript*. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch (Accessed: March 13, 2025).

Microsoft Azure (no date) *What is a VPN?* . Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-vpn (Accessed: April 27, 2025).

Minois, N. (2023) *The Importance of Communication in Project Management. IPM*. Available at: https://instituteprojectmanagement.com/blog/the-crucial-role-of-communication-in-project-management/ (Accessed: February 9, 2025).

Moretti. Francisco (2023) *Proper Error Handling in Web Development - Best Practices*. Available at: https://www.franciscomoretti.com/blog/avoiding-common-mistakes-proper-error-handling-in-web-development (Accessed: March 13, 2025).

Nidhi, R. (no date) *Minimum Viable Product (MVP): What is it & Why it Matters. Atlassian.* Available at: https://www.atlassian.com/agile/product-management/minimum-viable-product (Accessed: April 17, 2025).

Nielsen, J. (1999) “User interface directions for the web,” *Communications of the ACM*, 42(1), pp. 65–72. Available at: https://doi.org/10.1145/291469.291470/ASSET/F6117D35-D885-4960-A613-8AF99AA17536/ASSETS/291469.291470.FP.PNG.

Pragada, A. (2024) *API Optimization for React Apps: Minimizing Data Fetching Overhead. Medium*, *Medium*. Available at: https://medium.com/@abhi.venkata54/api-optimization-for-react-apps-minimizing-data-fetching-overhead-634c81b43808 (Accessed: February 22, 2025).

PyPi (2021) *webdrivermanager*. Available at: https://pypi.org/project/webdrivermanager/ (Accessed: February 27, 2025).

React (no date) *React*. Available at: https://react.dev/ (Accessed: March 27, 2025).

Robot Framework (no date) a *Robot Framework*. Available at: https://robotframework.org/ (Accessed: February 27, 2025).

Robot Framework (no date) b *I’m looking for testing*. Available at: https://docs.robotframework.org/docs/getting_started/testing (Accessed: February 27, 2025).

Robot Framework (no date) c *How to find the right library*. Available at: https://docs.robotframework.org/docs/different_libraries/how_to_find_library (Accessed: February 27, 2025).

Schwalbe, Kathy. (2019) *Information technology project management*. Ninth edition. Boston, MA : Cengage Learning.

Siddiqui, A. (2023) *Cross-Browser Compatibility: Ensuring Consistency in JavaScript Development | Code by Zeba Academy*. Available at: https://code.zeba.academy/cross-browser-compatibility-javascript-development/ (Accessed: March 13, 2025).

Sommerville, Ian. (2016) *Software engineering*. Boston : Pearson (Always learning).

TypeScript (no date) *What is TypeScript?* Available at: https://www.typescriptlang.org/ (Accessed: March 27, 2025).

Visual Paradigm (no date) *What is BPMN?* Available at: https://www.visual-paradigm.com/guide/bpmn/what-is-bpmn/ (Accessed: April 17, 2025).

de Voil, N. (2020) *User Experience Foundations*. BCS, The Chartered Institute for IT.

Webflow Team (2024) *Error handling & exceptions: The ultimate guide*. Available at: https://webflow.com/blog/error-handling (Accessed: March 13, 2025).