Bachelor’s thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering

Staverløkk, Trine Merete

Web-based prototype for simplified bank reconciliation in accounting software

Bachelor’s thesis in Computer Science
Supervisor: Anniken Karlsen
January 2024

NTNU
Norwegian University of
Science and Technology

Staverløkk, Trine Merete

Web-based prototype for simplified bank reconciliation in accounting software

Bachelor’s thesis in Computer Science
Supervisor: Anniken Karlsen
January 2024

Norwegian University of Science and Technology
Faculty of Engineering

Staverløkk, Trine Merete

Web-based prototype for simplified bank reconciliation in accounting software

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.	☒
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver	☒
5.	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31	☒
6.	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider	☒

Preface

This thesis stands as a testament to my hard work to achieve knowledge within the field of Computer Science at NTNU, of which I am very proud, and the generosity of those who invested time and knowledge in my academic pursuits, of which I am very grateful.

Thank you to NTNU for providing me with the knowledge and opportunity to take on this bachelor thesis, the most significant milestone in my academic journey.

I wish to thank my dedicated supervisor, Anniken Karlsen. Her support and constructive feedback greatly contributed to the development and completion of this work. I also wish to thank Girts Strazdins for his help and guidance.

My sincere appreciation to the client, Tritt and Conta, whose collaboration made this project possible. Their real-world challenges enriched my learning experience and added practical relevance to the theoretical concepts explored throughout this project.

I also wish to thank the team, Sølve Monterio, Regine Giskegjerde Urtegård, Simen Stokkeland Fuglseth and Nina Vinding Blindheim for their help and support. Special thanks to Sølve Monteiro for his cooperation and mentorship.

Sammendrag

I regnskapsbransjen er manuell avstemming av transaksjoner mellom regnskapsprogrammer og bankaktiviteter en tidkrevende prosess som kan inneholde feil. Kunden har identifisert en mulighet til å effektivisere denne bankavstemmingsprosessen. Oppgaven tar sikte på å undersøke og utvikle en prototypefunksjonalitet som sammenligner regnskapstransaksjoner med faktiske banktransaksjoner. Problemstillingen som skal besvares i oppgaven er derfor: "Hvordan kan man demonstrere forenkling av bankavstemmingsprosessen i et regnskapssystem gjennom utvikling av en prototype?" Prosjektet er strukturert i tre hoveddeler: en mulighetsstudie, utvikling av en nettbasert prototype og sluttarbeid med testing med ekte regnskapsførere.

Prosjektet bruker agile metoder med en solo-utviklertilnærming, støttet av et team fra kunden. Teknologier som er brukt i utviklingen av prototypen er Docker, Gradle, Micronaut, Vue3, Java, TypeScript, HTML, Groovy, Cypress, Postgres, IntelliJ and Git/GitHub for versjonskontroll.

Prosjektet leverer en web-basert proof-of-concept av en sammenlignings-motor som tar for seg et sett av bank- og regnskapstransaksjoner. Prototypen inkluderer funksjoner som en periodevelger, selve sammenligningen med fire ulike scenarioer, lagring av sammenlignede perioder i en database og enkel navigering mellom sidene via en sidemeny.

Det konkluderes med at vitenskapelige funn indikerer en vilje til å ta i bruk strømlinjeformede prosesser innen regnskap, og det finnes I dag løsninger hos konkurrenter som har automatisk bankavstemming. Prototypen som er utviklet I dette prosjektet støtter opp om funnene i den vitenskapelige delen av oppgaven, og den oppfyller også kundens krav til prosjektet. Prosjektets suksess tilskrives samarbeid med et støttende team, overholdelse av smidige metoder og bruk av relevante teknologier. Ytterligere arbeid innebærer å forbedre prototypen for økt pålitelighet og hastighet, test-dekning og tilgjengelighet til alle brukere.

Abstract

In the accounting industry, manual reconciliation of transactions between accounting programs and banking activities is a time-consuming prosess that can contain errors. The customer identified an opportunity to streamline this bank reconciliation prosess. The project aims to investigate and develop a prototype functionality that compares accounting transactions with actual banking transactions. The problem formulation which this project aims to answer is therefore: " How can one demonstrate simplification of the bank reconciliation prosess in an accounting system through the development of a prototype?” To solve this problem, the project has been structured in three main parts: a scientific study, development of a web-based prototype and final work with testing with real accountants and concluding in a report.

The project uses agile methods with a solo-development approach, supported by a team from the customer. Technologies used in the development of the prototype are docker, Gradle, Micronaut, Vue3 (compositions API), Java, TypeScript, HTML, Groovy, Cypress, Postgres, IntelliJ and Git/GitHub for version control.

The project delivers a web-based proof-of-concept of a comparison engine that deals with a set of banking and accounting transactions. The prototype includes functions such as a period selector, the actual comparison with four different scenarios, storage of compared periods in a database and simple navigation between the pages via a page menu.

Findings in literature show a willingness to adopt streamlined prosesses in accounting, access to technology and there are existing solutions used in other accounting solutions that have implemented streamlined or even automatic bank reconciliation. The prototype developed in this project supports the findings in the scientific part of the thesis, and the prototype meets the customer's requirements for the project. The project's success is attributed to collaboration with a supportive team, adherence to agile methods and the use of relevant technologies. Future work beyond the scope of this thesis involves improving the prototype for increased

reliability and speed, greater test-coverage, accessibility to all users and then implementation into the clients accounting program.

Background

This chapter will introduce the client and the requirement the client have set for this project.

The client

Tritt AS is a software development company, which was founded in 2016. Tritt specializes in different software solutions in accounting, and is particularly focused on developing software within the realms of salary management, accounting, invoicing, and annual financial statements. Tritt AS merged with Conta in 2023, which delivers high quality accounting-software solutions.

Requirement specification

The client has requested me as a part of a feasibility study, to design and develop a prototype of a bank reconcilliation system for their accounting program. The prototype is required to be a differensial engine (diff-engine) with basic functionality, including comparing two sets of transactions (bank transactions and accounting transactions).
The displayed results generated by the diff-engine should include various possible outcomes, including:

- **Match**: Identifying instances where bank-transactions matches with accounting-transactions.
- **Partial Match**: This happens if there is a partial correspondence between bank-transactions and accounting-transactions.
- **Missing Bank-Transactions**: Happens if transactions are present in the accounting records but are absent from the bank records.
- **Missing Accounting Transactions**: If transactions are recorded by the bank but are not reflected in the accounting records.

Other wanted functionality is a possibility to store the compared periods in a different view, and intuitive navigation between the pages.

This diff-engine-prototype will function as a first iteration of a bank-reconciliation prosess feature in the client accounting system.

Thesis structure

Chapter 1: Introduction – Includes glossary, acronyms, background/motivation, problem formulation, objective, requirements, limitations/boundaries and a list of relevant subject areas.

Chapter 2: Theory - Presents the theory this thesis is based on. Design, technology, product development, documentation and testing.

Chapter 3: Materials and methods – Presents the materials and methods used to gather information, develop the product, testing and writing the report.

Chapter 4: Results – Presents the results of the project. Includes scientific results, engineering and administrative results.

Chapter 5: Discussion – Discusses and reflects upon the results of the project.

Chapter 6: Conclusion and further work – Conclude the results and discussion of the project, societal impact and describes further work.

Table of contents

Preface ***3***

Sammendrag ***4***

Abstract ***5***

Background ***7***

The client **7**

Requirement specification **7**

Thesis structure **9**

Table of contents ***10***

1 Introduction ***17***

1.1. Glossary **17**

1.2. Acronyms **18**

1.3. Background/motivation **19**

1.4. Problem formulation **19**

1.5. Objectives **20**

1.6. Requirements **20**

1.7. Limitations **22**

1.8. Subject areas **23**

2 Theory ***24***

2.1. Theory spesific to domain **24**

2.2. System Development **24**

2.2.1. Agile development 25

2.2.2. Prototyping 25

2.2.3. Model based systems engineering 26

2.2.4. Domain driven design (DDD) 27

2.3. Design 27
2.3.1. User interface (UI) 27
2.3.2. Model-View-Controller pattern (MVC) 28
2.3.3. Availability and accessibility 29
2.4. Software developing tools and technologies 30
2.4.1. Micronaut 30
2.4.1.1. Bean-classes 30
2.4.2. Gradle 31
2.4.3. PostgreSQL 31
2.4.4. Groovy 31
2.4.5. Cypress 31
2.4.6. VUE.js 32
2.4.7. Version-control 32
2.4.8. Containerization in Docker 32
2.5. Quality assurance 33
2.5.1. Testing in computer science 33
2.5.1.1. Agile testing 33
2.5.1.2. Unit testing 34
2.5.2. Usability testing 34
2.5.3. Code quality 34
2.5.3.1. Cohesion and coupling 34
2.5.3.2. SOLID 35
2.5.3.3. Code-quality plugins for IntelliJ 35
3 Materials and methods 36
3.1. Methodology 36
3.2. Project management 36
3.2.1. The team 36
3.2.2. Meetings 36
3.2.3. Communication 37
3.3. Scientific approach 37
3.4. Technologies used 38
3.4.1. Modelling 38

3.4.2. Prototyping ... 38
3.4.3. Code quality ... 38

3.5. Testing ... 39
3.5.1. Acquiring test data ... 39
3.5.2. Software testing ... 40
3.5.3. Usability testing ... 40

4 Results ... 41

4.1. Features ... 41

4.2. Theoretical contributions ... 45
4.2.1. Theoretical findings ... 45
4.2.2. Exisiting solutions ... 47

4.3. Engineering results ... 47
4.3.1. Architecture ... 47
4.3.2. Domain model ... 50
4.3.3. Modelled prototypes ... 51
4.3.3.1. Low-fidelity prototype ... 51
4.3.3.2. High fidelity prototype ... 52
4.3.4. Project hierarchy ... 53
4.3.4.1. Frontend project hierarchy ... 53
4.3.5. Backend project hierarchy ... 54
4.3.6. Persistence ... 55
4.3.7. Business logic ... 56
4.3.7.1. Backlog ... 56
4.3.8. Vue-components ... 57
4.3.9. Frontend services ... 59
4.3.10. Views ... 59

4.4. Quality assurance results ... 60
4.4.1. Unit testing ... 60
4.4.2. E2E-testing ... 62
4.4.3. Usability testing ... 71
4.4.4. Documentation ... 73
4.4.4.1. README ... 74

4.5. Administrative results and development prosess 74
4.5.1. Time-management 74
4.5.2. Meetings 74
4.5.3. Project management 75

5 Discussion 77

5.1. Theoretical discussion 77
5.1.1. Methods for data collection 77
5.1.2. Result reflection 78

5.2. Engineering discussion 78
5.2.1. Design 78
5.2.1.1. Prototyping 78
5.2.1.2. Design material and methodology 79
5.2.2. Development 79
5.2.2.1. Project structure 79
5.2.2.2. Comparison results explained 80
5.2.2.3. Entities 81
5.2.2.4. Controllers 81
5.2.2.5. Views and components in frontend 83
5.2.3. Code quality 84
5.2.3.1. Checkstyle 85
5.2.3.2. SonarLint 85
5.2.3.3. Robustness 85
5.2.3.4. Transaction Service-class 87
5.2.3.5. Bugsolving 90
5.2.4. Usability 90
5.2.5. Accessibility 91
5.2.6. Software testing 92
5.2.6.1. Component testing in backend 92
5.2.6.2. E2E-testing in frontend 93

5.3. Administrative discussion 94
5.3.1. The team 94
5.3.2. Project management 94
5.3.2.1. Issue-tracking 94

5.3.3. Methodology/structuring development prosess 95

6 Conclusion 96

6.1. General conclusion 96

6.1.1. Scientific conclusion 96

6.1.2. Engineering conclusion 96

6.1.3. Administrative conclusion 96

6.2. Community impact 97

6.3. Further work 97

Works Cited 98

Appendix 1 – AI declaration 105

Appendix 2 - Usability test report first iteration 106

Appendix 3 - Usability test report second iteration 110

Figure 1 - Prosess diagram of bank reconciliation prosess in Conta before implementing project feature 21
Figure 2 - Prosess diagram of bank reconciliation after implementing of project-features 22
Figure 3 - MVC 29
Figure 4 - Home view 42
Figure 5 - diff-engine view 42
Figure 6 - Period picker 43
Figure 7 - modal overview 43
Figure 8 - diff-engine result when not accepting reconciliation 44
Figure 9 - Results with icons 44
Figure 10 - Compared periods view 45
Figure 11 – Architecture 47
Figure 12 - Use case diagram 48
Figure 13 - Sequens diagram 49
Figure 14 - domain model planned 50
Figure 15 - Domain model rendered from IntelliJ 50
Figure 16 - Low fidelity prototype 51
Figure 17 - High fidelity prototype 52
Figure 18 - Frontend project hierarchy 53
Figure 19 - Backend project hierarchy 54
Figure 20 - Persistence hieracrhy 55
Figure 21 – TransactionService 56
Figure 22 - Backlog 57
Figure 23 - Alert box 57
Figure 24 - Card from Period Comparison 58
Figure 25 - Comparison Results 58
Figure 26 - PeriodPicker 59
Figure 27 - Project hierarchy in unit testing 60
Figure 28 - mock setup 61

Figure 29 - Overview tests 61
Figure 30 - Unit test example TransactionService-class 62
Figure 31 - Overview of E2E-tests 63
Figure 32 - Testing sidebar-navigation 64
Figure 33 - Displaying diff-engine in Cypress 65
Figure 34 - Opening period-picker in Cypress 66
Figure 35 - Choosing period in Cypress 67
Figure 36 - Verifying chosen period contains expected value 68
Figure 37 - Displaying alert box in Cypress 69
Figure 38 - Cypress tests 70
Figure 39 - first version of high-fidelity prototype ´ 71
Figure 40 - Second iteration of high-fidelity prototype 72
Figure 41 - Example on code documentation Java 73
Figure 42 - Example on code documentation TypeScript 73
Figure 43 - README 74
Figure 44 - issues 75
Figure 45 - Example of commit-indexing in GitHub 76
Figure 46 - Example of commits in Git 76
Figure 47 - Several partial matches 81
Figure 48 - Initial controller structure 82
Figure 49 - Latest controller 82
Figure 50 - Checkstyle example 85
Figure 51 - null-check example 86
Figure 52 - compareTransactions-method 87
Figure 53 - one of the extracted methods from compareTransaction-method 89
Figure 54 - Omitted methods from tests 93
Figure 55 - Example of commit-indexing in GitHub 95

1 Introduction

The main objective of the bachelor thesis has been to look into the possibilities of developing a more efficient accounting prosess. This includes developing a prototype that will demonstrate a possibility for a more efficient bank reconciliation prosess, by comparing transactions from the bank with transactions from the client accounting system. A crucial component of financial management is bank reconciliation, enabling businesses to ensure that their financial records align with their bank statements. This prototype is designed to compare transactions from the bank and transactions from an accounting program, thereby finding errors before the reconciliation. The development of a web-based prototype of such a system is a significant task, given the complexity and critical role of bank reconciliation in financial operations.

The prototype developed during this bachelor matches transactions from bank statements and internal accounting systems. It aims to ensure data accuracy and completeness, thereby minimizing the risk of errors in the finished reconciliation. The system also includes displaying discrepancies periodically, minimizing the need for manual intervention. Also, the prototype provides a Proof of Concept (POC), showing one way of standardizing the bank reconciliation prosess, contributing to time savings, error reductions and a higher level of regulatory compliance. The system is designed to be user-friendly and serve as a building block for a desired feature in the existing accounting program.

This thesis report presents the work done as a bachelor thesis in the autumn of 2023. The

1.1. Glossary

Agile development - methodology

Backend – The server-side of the project. Persists data and includes business-logic.

Bug – An error, flaw, or fault in the system. Non-expected behavior.

Diff-engine – The engine that will show the differences found when matching transactions.

Docker Compose – A tool for running multi-container Docker applications.

Frontend – The visual side of the product, includes what the user sees.

GitHub – A platform for version-control and collaboration (in this project used mainly for version-control).

RecSys – The name of the project. Abbreviation for Reconciliation System.

1.2. Acronyms

API - Application Programming Interface

DDD – Domain-Driven Design

DB – Database

HTML - Hyper Text Markup Language

IDE - Integrated Development Environment

JVM - Java Virtual Machine

MBSE - Model-Based Systems Engineering

NTNU - Norwegian University of Science and Technology

OOP – Object-Oriented Programming

POC – Proof of Concept

RPA - Robotic Prosess Automation

SDLC - System Development Life Cycle

TDD – Test-Driven Development

UI – User Interface

UX – User Experience

1.3. Background/motivation

As an accountant a substantial part of daily tasks is to manually keep an overview of transactions using an accounting program. These transactions must then be matched with the actual activity on a company's bank account. This is called bank reconciliation, and is an important control mechanism for detecting errors, deficiencies, or fraud (Conta, 2023). This can be a long prosess, with room for improvements in efficiency.
The client has identified an opportunity in streamlining the bank reconciliation prosess which is desirable to investigate further. As the first step in this prosess, it is desirable to create a functionality that compares the accounting transactions with actual transactions in the bank. After discussions with the client, we therefore concluded that developing a prototype with this functionality is both exciting and highly relevant for future development. It is particularly motivating that the client wants to implement and continue this functionality in their accounting program in the future. In addition to the above, I believe that it is a realistic task for me to carry out, while it also poses a significant technical challenge, giving ample learning opportunities.

1.4. Problem formulation

The overall question, initially asked and guiding this thesis, was formulated as: How can one demonstrate simplification of the bank reconciliation prosess in an accounting system through the development of a prototype?

To answer this question, the project was divided into three main parts:

1. Investigate how bank reconciliation in an accounting system can be simplified (a feasibility study).

   a. Carry out a literature review to identify existing solutions for bank reconciliation in accounting systems.
   b. Explore any existing solutions and your functions.
   c. Identify the most important functions that the prototype should contain and plan the technologies to be used. This comes from the client.
2. Develop a web-based prototype that demonstrates differences and shows discrepancies.
3. Final work, including testing of the solution with real accountants and completion of the report with results from parts 1 and 2.

1.5. Objectives

The client needed a solution for simplifying and streamlining the bank reconciliation prosess in an accounting program. The purpose of this project was to develop a web-based prototype, that completes this task with basic functionality that is expandable in the future.

The prototype will function as a diff engine, a display that compares differences between actual bank transactions and the manually conducted transactions and will be based on research that will be carried out in the first part of the project. It is desirable from the client to have a display that gives a clear overview of differences/deviations, which clearly highlights these on the manually entered transactions.

1.6. Requirements

Create a web-based POC of a diff-engine that compares transactions that is made by an accountant in the program, and actual bank-transactions.

Features:

- Prototype of a diff-engine – will complete matches and display results
- Period picker – possibility for the user to compare transactions within a period.
- Store compared periods in DB

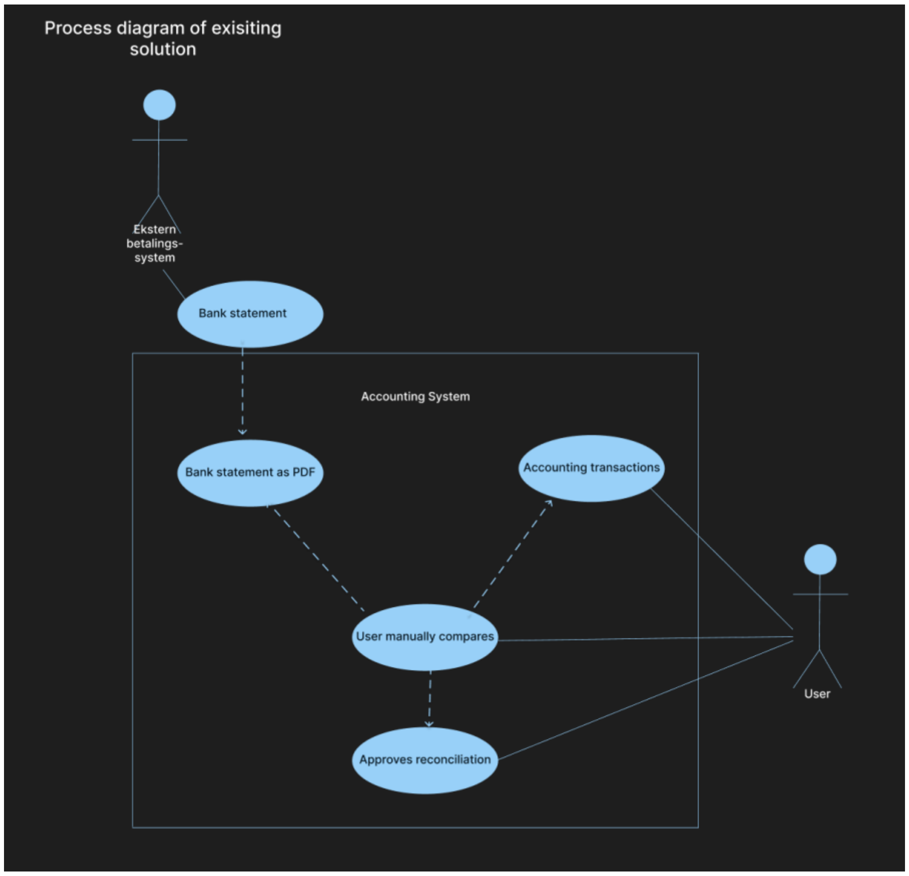


Figure 1 - Prosess diagram of bank reconciliation prosess in Conta before implementing project feature

Figure 1 shows a prosess diagram of the bank reconciliation prosess as it is today. The client wanted to streamline this project, shown in figure 2.

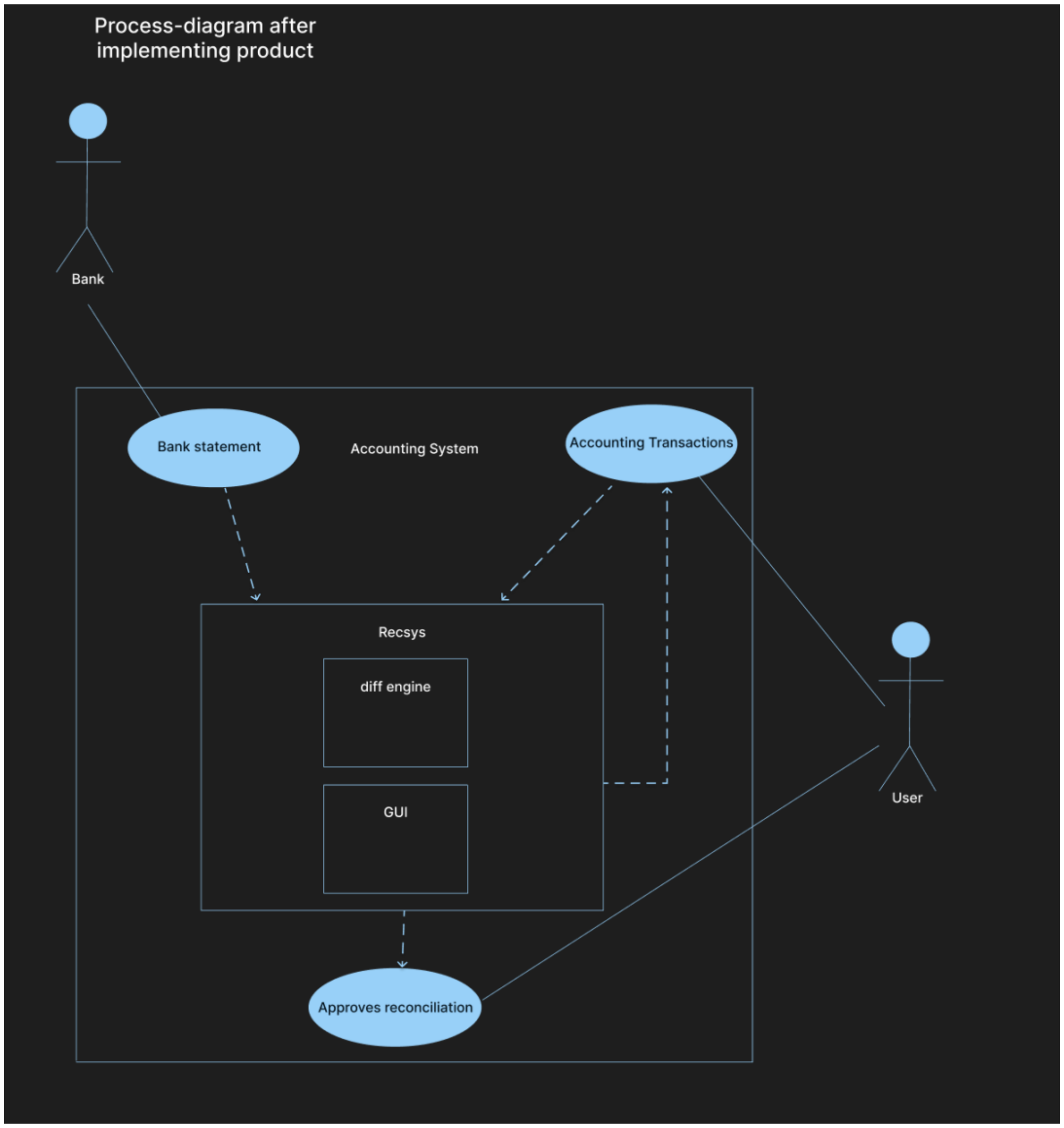


Figure 2 - Prosess diagram of bank reconciliation after implementing of project-features

1.7. Limitations

In a project that is a feature for an exisiting client system, there are some limitations when it comes to technology to use and the visual. If the client is going to benefit from this feature, the technology needs to fit with exisiting technology, so it easyely can be implemented into their exisiting solution. The client also expected the project to follow some visual guiedelines for a more esaily implementation into their exisiting solution.

There was also some limitations in the data:

- Data is real, from a real customer, but has been anonymized. These were picked because they represent real life and is close to real data.

There has been some omitted work, because of the scope of the project. This includes:

  - Possibility to make changes to accounting-transactions if result is not match
  - Sorting-options
  - Security, authentication, authorization

These are omitted on the grounds that the finished product will serve as a prototype, and these features are already a part of the clients program where this prototype will be implemented.

1.8. Subject areas

- DB - Database
- Frontend development (Vue 3 compositions API, tailwind, CSS, HTML, typescript)
- Backend development (Micronaut, Java)
- Agile development methodology
- DDD – domain driven design
- Testing – unit-testing, E2E-testing and usability test
- Documentation – source code comments and writing reports
- UX-design

2 Theory

This chapter will present some relevant theory to the project.

2.1. Theory spesific to domain

Bank reconciliation is a prosess in accounting that involves comparing the carried-out transactions in the accounting program with the corresponding transaction on its bank statement (CFI Team, 2023). Regular bank reconciliations are essential for detecting fraud and any cash manipulations, and they also help keep track of accounts payable and receivables of the business (Freshbooks, 2023) (QuickBooks, 2023).

2.2. System Development

System development is the prosess of designing, implementing, testing and maintaining a new product in computer science (Computer Hope, 2023).

System Development Life Cycle (SDLC) is a well-defined prosess in which one gives structure to the prosess where an application is planned, developed, implemented and maintained. The phases of the system development life cycle provide a basis for management and control because they define segments of the workflow that can be identified for management purposes and specify the documents or other deliverables to be produced in each phase (FCA, 2007).

The first phase in SDLC is the initiation phase, in which conceptualization and planning is carried out, which involves identifying a problem/possibility. This phase is about understanding the requirements from the client and setting the goals for the system. It includes defining the scope of the project, opportunities and setting a timeline for the project. Second phase is the feasibility phase, where the initial investigation is carried out to determine whether the product could be made or should be pursued. If the feasability phase proves the products possibility, the next phase is the requirements analysis phase, where developers and other team members analyse and determine the needs of end users based on research conducted. Based on this the team develop the requirement specifications. Designing the product is then the next phase of SDLC.

This phase involves creating a more logical structure from the requirements specification which can later be implemented in a programming language and designing the product. Plans are drawn up so developers know what they need to do through iteration of the project. After this the actual development of the product is started in the development phase. Thi sis where the programming, testing and necessary adjustemnts are made. After the product developed, tested and accepted by the client, the product it installed to support the intended business functions in the implement phase. Even after implementation the system developing is ongoing in the operations and maintenance phase. This is where the product is monitored for continued performance in accordance with user requirements and needs (Michigan Tech, 2023).

As we can see, the system development prosess is not only about writing code, but also about understanding user needs, translating the needs into detailed specifications, implementing the product and finally maintaining it (FCA, 2007).

2.2.1. Agile development

Agile development is an umbrella term for a particular methodical and iterative approach that is often used in software development projects. The fundamental idea of an agile development prosess is to work on delivering work in small, incremental units (Agile Alliance, 2023). One therefore often talk about prosess tools such as Scrums, pair programming, standups, sprints and sprint planning. Engaging in agile development promotes an increased flexibility and adaptability in the work, allowing the team to face changes in product requirements and deal with unforeseen issues that may arise during the project. Working a solo-project this has still been a big focus-area in my product development, as I will further discuss in chapter 5.3.

2.2.2. Prototyping

Creating a prototype means to create a preliminary model or design of a software to test its functionality and/or viability (Wikipedia, the free encyclopedia, 2023). It functions as a simulation of the final product, allowing developers and users to interact with the system and provide feedback before actually developing the product.

A prototype in computer science is the prosess of creating an initial design or model of a software to test its functionality. It acts as a simulation of the final product, allowing developers and users to interact with the system early in the system development prosess, usually in the design phase. This allows for providing feedback before spending resources on developing the product (Wikipedia, the free encyclopedia, 2023).

There are two common ways to prototype a model/design, which is low- and high fidelity prototyping. Low fidelity prototyping often consist of a basic drawing or first concept of the product or feature. Can be hand drawn on paper or basic wireframes without colour or content (UXPin, 2023).

The high fidelity prototype is more advanced than the low-fidelity prototype using mock-ups with colour and content to achieve a more accurate representation of the final product (UXPin, 2023).

2.2.3. Model based systems engineering

Model-Based Systems Engineering (MBSE) is a methodology used to develop products in, where models are at the center of system design (McGrath & Jonker, 2023).

MBSE is the use of modeling systems to explore and document system properties. Models provide an efficient and clear way to communicate system aspects to customers. MBSE uses a modeling language, in this project UML is used, to describe the problem that the designed system solves and the design itself (the solution) (McGrath & Jonker, 2023). The model describes both the problem side (meaning the operational point of view, which represents business prosesses, goals, organizational structure, use cases and information flow), and the pay side, (meaning the system point of view, which describes the system's behavior, structure, data flow between components and allocation of functionality) (Wikipedia, the free encyclopedia, 2023).

2.2.4. Domain driven design (DDD)

Domain-Driven Design (DDD) is a software development approach that focuses on the importance of understanding the business domain, and from that understanding models the software. DDD advocates for the use of a rich, expressive language that is used by all team members to connect the software with the business domain (Penchikala, 2008). MSBE can be regarded as a framework that provides the techniques to put DDD in to practice (Cabot, 2023).

2.3. Design

Considering design is a big part of developing my product development. Pertaining to this central the product needs to be well thought-out and well tested. Designing software involves creating a plan for a system that defines the responsibilities of its components and how they interact (Alam, 2023). This prosess is crucial in making future changes easier to implement. It has been important to look into UX design when developing this product.

2.3.1. User interface (UI)

The interface that enables a user to communicate with machines (Geeks for Geeks, 2023). The program becomes more popular if its UI is simple to use, easy to understand, attractive and responsive and consistent on all screens (Geeks for Geeks, 2023).

User experience (UX) design is the prosess of creating products that provide meaningful experiences for users. How the user perceives using the product, often with a special emphasis on the emotions the system envokes in the user (Stevens, 2022).
Nielsen (Rahdan, 2020) has identified 10 heuristics for user interface design:

- Visibility of system status

- Match between system and the real world
- User control and freedom
- Consistensy and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetics and minimalistisc design
- Help users recognise, diagnose and recover from errors
- Help and documentation

These heuristics area set of good rules of thumb, not specific usability guidelines (Rahdan, 2020).

2.3.2. Model-View-Controller pattern (MVC)

The Model-View-Controller (MVC) pattern is a design pattern commonly used to develop web applications (Wikipedia, the free encyclopedia, 2023). MVC divides the software into three main parts: the model, the view, and the controller. Each component has a particular function, which simplifies the overall design and improves the readability and scalability of the application.
The model is responsible for handling the data and dynamic data structure. This means the model interacts with the DB and handles all logic and manipulation of the data. The controller should never directly interact with the data logic (this should go through the model). This means the controller never has to worry about the data it sends and receive, instead only need to tell the model what to do in respond based on on what the model returns. This also means the model never needs to worry about handling user request or what to do upon failure or success. That is handled by the controller. After the model sends its response back to the controller, the controller then needs to interact with the view to render the data to the user. The view is only concerned about how to present the information the controller sends. The view will then send final presentation back to the controller, and the controller sends it back to the user. The model

and view never interact with each other, this always happens through the controller (Wikipedia, the free encyclopedia, 2023).

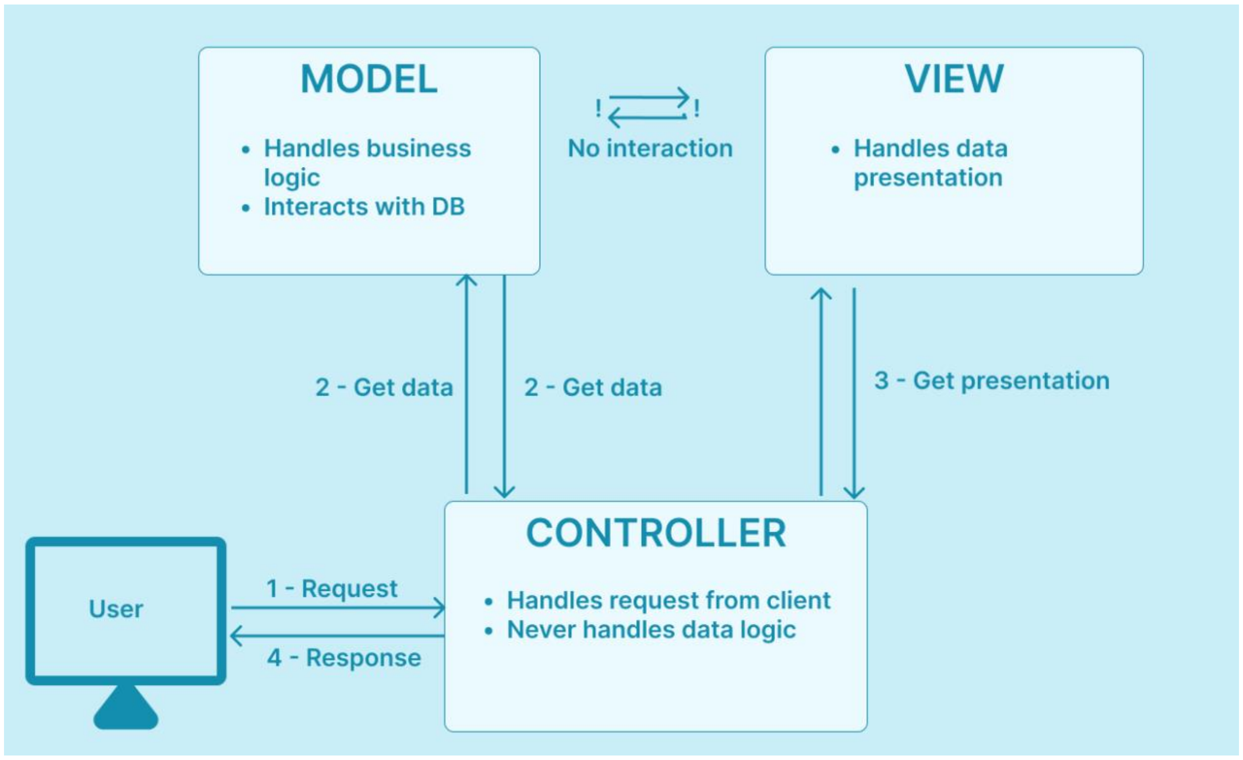


Figure 3 - MVC

The MVC pattern allows a clear separation of responsibilities, making the application easier to develop (Wikipedia, the free encyclopedia, 2023).

2.3.3. Availability and accessibility

Universal design is about creating buildings, products or environments that are accessible to people of all ages and abilities. The purpose of implementing universal design for websites is to offer inclusive, simple and accessible solutions for users with disabilities, thereby promoting equal democratic rights, facilitating access to information and promoting opportunities for all (Wikipedia, the free encyclopedia, 2023).

Availability, in the context of software design, refers to the degree to which a system is accessible to users. It's an important aspect of user experience (UX) design and is often evaluated through the lens of accessibility. Accessibility is about making a product usable by as many people as possible, taking those with disabilities into consideration (Wikipedia, the free encyclopedia, 2023).

There are many aspects to consider, colour-contrast, keyboard navigation, including text size, timing, and more. By incorporating these principles into the design, the system is accessible to a wider range of users.

2.4. Software developing tools and technologies

Technologies in software developing aften refers to programming languages, frameworks, databases, and other tools, methods and practices to develop a software (Bhatt, 2023). This chapter will provide som theoretical knowlegde on some relevant technologies for this project.

2.4.1. Micronaut

Micronaut is a JVM-based, full-stack framework designed for building JVM applications that are modular and serverless and easily testable. Micronaut is designed to make it seamless to create microservices, which this bachelor project could be characterized as. Micronaut is designed to avoid reflection, thereby reducing memory consumption and improving startup times. Functions that would typically be implemented at runtime are instead precomputed at compile time (Micronaut, 2023).

2.4.1.1. Bean-classes

A bean is a Java class that follows a certain convention regarding property (all properties must be private, but with a public no-argument constructor) and implement serializable (can therefore be written to streams, object databases, etc). They follow this convention so a lot of libraries and framework can depend on them, including Micronaut (Chandan, 2023).

2.4.2. Gradle

Gradle is a tool used in software development for automation build. It automates the prosess of compiling source code to runnable files by controlling the development prosess, handling tasks from compilation and packaging to testing, distribution and publishing (Wikipedia, the free encyclopedia, 2023).

2.4.3. PostgreSQL

PostgreSQL, or Postgres, is an open-source object-oriented database system. It uses and extends the SQL language, and runs on all major OSes making it very widespread. PostgreSQL is highly extensible, so you can define your own data types, build custom functions and write code from different programming languages without compiling the database (PostgreSQL, 2023).

2.4.4. Groovy

Groovy is a dynamic, OOP language that runs on the JVM. In the context of software testing, Groovy can be used to create readable test scripts and test cases, execute powerful assertions, and generate orderly test reports (Bhutada, 2023). Groovy is known for its simplicity and readability, which makes it particularly useful for writing concise and readable tests. The syntax integrates well with testing frameworks such as Spock, which makes Groovy a tool for both unit testing and integration testing in software development projects (Apache Groovy, 2023).

2.4.5. Cypress

Cypress is a frontend testing tool designed for web applications. It was initially designed for end-to-end (E2E) testing, which is how it is also used in this project. A typical E2E-test visits the page in a browser, and performs action in the UI just like a real user. Cypress also provides steps with snippets of the completed tests. This makes the test easy to debug (Cypress, 2023).

2.4.6. VUE.js

Vue.js is a framework used to build user interfaces, built on standard HTML, CSS and JavaScript (or typescript). Vue provides a component-based programming model that helps you develop simple or complex user interfaces (Monocubed, 2023).

2.4.7. Version-control

Version control is a system that save and manage changes to software code. It is an important tool, as it helps in maintaining a detailed overview of every modification made to the code, ensuring that these changes are both trackable and reversible. It also makes it possible to collaborate effectively. The most widely used version control system in the world is Git, which is an open source and actively maintained project, developed by Linus Torvalds in 2005 (Atlassian, 2023).

GitHub is a widely used platform that provides hosting for software development and version control using Git. It offers functionality like access control and project management tools, such as collaboration features, issue-tracking, UI of branches and commits and issueboards (Rao, 2023).

2.4.8. Containerization in Docker

To enable packaging applications into containers I used Docker, which is excellent for this purpose. In Docker, containers are isolated environments that contain everything needed to run an application, ensuring it works consistently across different systems (Baeldung, 2023). Docker images are the basis for these containers. An image is a standalone and executable software package that includes all needed to run a software, including the code, a runtime, libraries, environment variables, and config files (Simic, 2022). When a container is run from an image, it includes the application and its dependencies. This allows for efficient, portable, and scalable deployments of applications. Docker's use of containers and images simplifies the software development and deployment prosess, making it easier to manage and scale applications (Docker, 2023).

2.5. Quality assurance

Software testing involves inspecting the elements and actions of software through validation and verification. Additionally, it offers an impartial and objective perspective on the software, enabling developers to comprehend and assess the risks associated with software implementation (Wikipedia, the free encyclopedia, 2023).

Software testing involves three stages. First setting up application state. Then you take som action to change the application state, followed by a check to see the resulting application state. This is called state transition testing (Geeks for Geeks, 2023)

2.5.1. Testing in computer science

Tests in software engineering is used to validate that the system respond as expected on all kinds of input, don't have bugs, performs well enough (or as expected), is user friendly, meets clients expectations, can be installed in correct environment and that it doesn't break when changing something (Yasar, 2023).

2.5.1.1. Agile testing

Is the opposite of traditional tesing, when a team of testers tests the whole product or feature before major release. Agile testing focus on testing after each implementation is completed. Makes testing the whole teams responsibility, instead of one team for development, and one for testing (GeeksforGeeks, 2023).

2.5.1.2. Unit testing

A unit test is a software test, that test that each units of a code works as expected. A unit can be a function, method, module, object, or other entity in an application's source code (Moradov, 2023).

Mocking data in software testing is a technique to test in a controlled environment by using fake objects known as mock objects (CodiumAI Team, 2023).

2.5.2. Usability testing

A usability test simulates a real situation, includes concrete tasks, is a test where you observe the user and is used to evaluate the usability of the system (Maze, 2023).

2.5.3. Code quality

Code of high quality refers to a systems robustness (system’s ability to cope with errors (Wikipedia, the free encyclopedia, 2023)), reusability (the capability to re-use pre-exisiting code), reliability (the probability of the systm performing its intended functions without failure), and understandability (when a system allows an engineer to easily comprehend it, readability) (Indeed Editorial Team, 2023).

2.5.3.1. Cohesion and coupling

Coupling and Cohesion are two important aspects in all object-oriented programming languages, as high cohesion and loose coupling will lead to a system that is easier to understand, maintain and modify, and generally higher quality code (Wikipedia, the free encyclopedia, 2023).

Coupling is defined by the degree of which different components are dependent on one another. It describes relationships between components. Loose coupling means that each component is not dependant on each other. Integrated products such as most phones are examples of tight coupling. If the battery dies, one might as well buy a whole new phone, as the battery is very expensive to replace. Loose coupled code works the same way, it facilitates extension, replacement and it more readable (Wikipedia, the free encyclopedia, 2023).

Cohesion is defined by the degree of which elements of a component are functionally related. It describes relationships within components. High cohesion is associated with attributes such as robustness, reusability, reliability, and readability (Wikipedia, the free encyclopedia, 2023).

2.5.3.2. SOLID

Solid is an acronym in computer science that includes five design principles for enhanced code quality:

- Single responsible principle: Every class should have one responsibility.
- Open-closed principle: Software entities needs to be open for expansion, but closed for alteration.
- Liskov substitution principle (design by contract): Functions that work with base classes using pointers or references should be able to handle objects from derived classes without needing to know the specific derived class.
- Interface principle: If a software-entity does not use an interface, it shouldn't be forced to be dependent on it.
- Dependency principle: Software-entities should depend upon abstracts and not concretes.

(Wikipedia, the free encyclopedia, 2023)

2.5.3.3. Code-quality plugins for IntelliJ

CheckStyle is a tool for developers to help write Java code that adheres to a specific code standard (checkstyle, 2023).
SonarLint is a tool that identifies code smells, and catches issues immidiatly right in the IDE (SonarSource, 2023).

3 Materials and methods

This chapter will present working prosesses and tools used during project development.

3.1. Methodology

The project uses agile methods with a solo-development approach, supported by a team from the client. This chapter will present the methodology in which this project has been conducted.

3.2. Project management

3.2.1. The team

This project was done as a bachelor thesis by Trine Staverløkk, a third year Computer Science student at NTNU. The work was done at the client facilities, in Fremmerholen, Ålesund. This gave raise to the project being done in a real life professional work setting, wherein scrums, standups, mob-sessions and generally contribution with knowlegde and support. The team was a composite of colleagues having backgrounds as both junior and senior developers. Because of this team, it was possible to follow an agile methodology in my project. This is discussed further in chapter 5.3.1.

3.2.2. Meetings

The project was divided into sprints, lasting about two weeks each time. After each sprint there was a retrospective with the client and supervisor, discussing progress, results from the sprint and planning of next sprint. This made progress more feasible, helped maintaining structure and made the whole project less overwhelming for a solo-developer.

In the mornings we also did stand-ups, where I collected feedbacks and agreement on what I would be working on in that day.

3.2.3. Communication

Communication between developer, supervisor and client have mainly been through channels like email. Document-sharing have been enabled through Teams, where supervisor and client both had the opportunity to view and leave comments on the document.

Meeting coordination and communication between developer, supervisor and client involved a combination of oral communication during regular sprint retrospects and the use of email and outlook for written invitation afterwards, ensuring clarity and formality in scheduling.

3.3. Scientific approach

Scientific research to gain theoretical knowledge has been explored through various platforms Google scholar have been mostly used, and additionally NTNU open together with the school library. Example of search words for digital science include “computer science”, “accounting”,” reconciliation”, “bank integration”, “bank account reconciliation”, “RPA” and every search on google scholar has been filtered to after 2019. See appendix (…) for full overview of search and search results. In addition to this, I have also investigated citations from the articles found, and through these found more relevant science.

To search for existing solutions, I went to their respective websites and looked for functionalities they provided their clients.

In addition to google scholar I used AI-tools as a search engine. Leveraging AI as a search engine enhanced information retrieval. Phind.com provides the sources, which made it easy to navigate and obtain knowledge from the original source where Phind.com found its information. This approach of utilizing AI as search engine expedited research but also broadened the scope.

3.4. Technologies used

The developing technologies were chosen based on existing solutions in Tritt. This includes Java, Typescript as programming languages, Docker desktop for containerizing, gradle as build-system, groovy and cypress for testing, postgres as DB and IntelliJ as IDE. For backend framework I used micronaut, and for frontend framework I used Vue3. The only different technology is the remote version control platform, which I chose based on what we mostly have used at NTNU. GitHub was used for project management, including version control. One repository was created for the whole project.

3.4.1. Modelling

For modelling I used the web-based modelling program diagrams.com and Figma. Both are open-source web-based programs suitable for modeling and prototyping. Figma has been utilized for planning, prototyping and modelling the entities and DB. Since this is open source and web-based it was easy to share and use everywhere if needed. I also used diagrams.net for the modelling.

3.4.2. Prototyping

In the start of the project, I worked together with to employees to create a low-fidelity prototype describing the basic workflow. Based on this, I developed a high-fidelity prototype in Figma.

3.4.3. Code quality

Quality code is code that is efficient, readable and usable (Codegrip, 2023). I have utilized various ways to ensure and enhance code quality. Most important have been to leverage theoretical knowledge foundation for high code quality aquired during my education at NTNU. This together with theoretical knowledge aquired while working on this project have been useful to write quality code. Additionally, I have maintained open communication with the client

and the team, where I have asked for peer review, to ensure that my coding practices aligns with their expectations for quality.
Lastly I used the plugins SonarLint and Checkstyle to enforce and enhance code quality and identify code smells.

3.5. Testing

3.5.1. Acquiring test data

I was provided with a XML-file with 53705 lines, from an actual, real company. After removing unwanted data (other datas than amout, date and description), we ended up having around 5000 lines of data for testing (552 objects of type transactions). This XML file where converted into a JSON-file, using the IDEs "replace all"-functionality, and flattened it. After this we anonymized the data by creating a naming generator for names of customers (Keith Armstrong, 2023). Used axios.get(URL (randomUser)) for this. We also tweaked the amounts, so its not possible to trace to a real customer. To change the amounts, we used (Math.random()* 200-100) so the amount to add is centered around 0, and range from –100 to 100, evening it out. After this we removed some transactions to 100 transactions.
This file was duplicated, one to represent the transactions from the bank-statement, and one to represent the accounting transactions, namely BankTransactions.json and AccountingTransactions.json. We then changed the AccountingTransactions-file to have these scenarios:

- Some missing transactions
- Some extra transactions
- Some transactions with wrong amount
- Some transaction with changed descriptions
- Some transactions with wrong date

3.5.2. Software testing

This project has conducted unit-testing in Groovy, and e2e-tests in Cypress. The methodology for Groovy-testing has consisted of a combination of strategies, some tests are written before the code, some during and some after. It has depended on the functionality and the complexity of the code. Cypress-tests are written after coding.

To mock data in the Groovy tests, the mocking framework Spock was used.

3.5.3. Usability testing

I conducted usability testing by developing a questionnaire to be used by accountants working in Conta. The tests were initially conducted after making a prototype of the basic functionality, and then after most of the development was done. Both the tests were the same, but with some adjusted questions and tasks after some of the functionality were changed. The questions for both tests were both specific questions to the design and UX, and some more general and open questions.
Materials for the first test was the test-subjects work-computers, where they were provided with a link to the Figma-project. They were handed out a questionnaire, with first a set of tasks to complete, and then a set of questions to answer. The second test was almost identical, except some of the questions which were altered to fit the design, and the the PC, which was the developers'.

4 Results

This chapter will provide the results from the feasibility-, design- and development stage in the SDLC. The feasibility phase includes the theoretical methods and findings, the design phase is the prototyping, modelling and planning part of the project, and the development phase is the engineering results. These will all be discussed in this chapter. This chapter will also show results from the administrative part of the project. All these will be discussed further in chapter 5.0, Discussions.

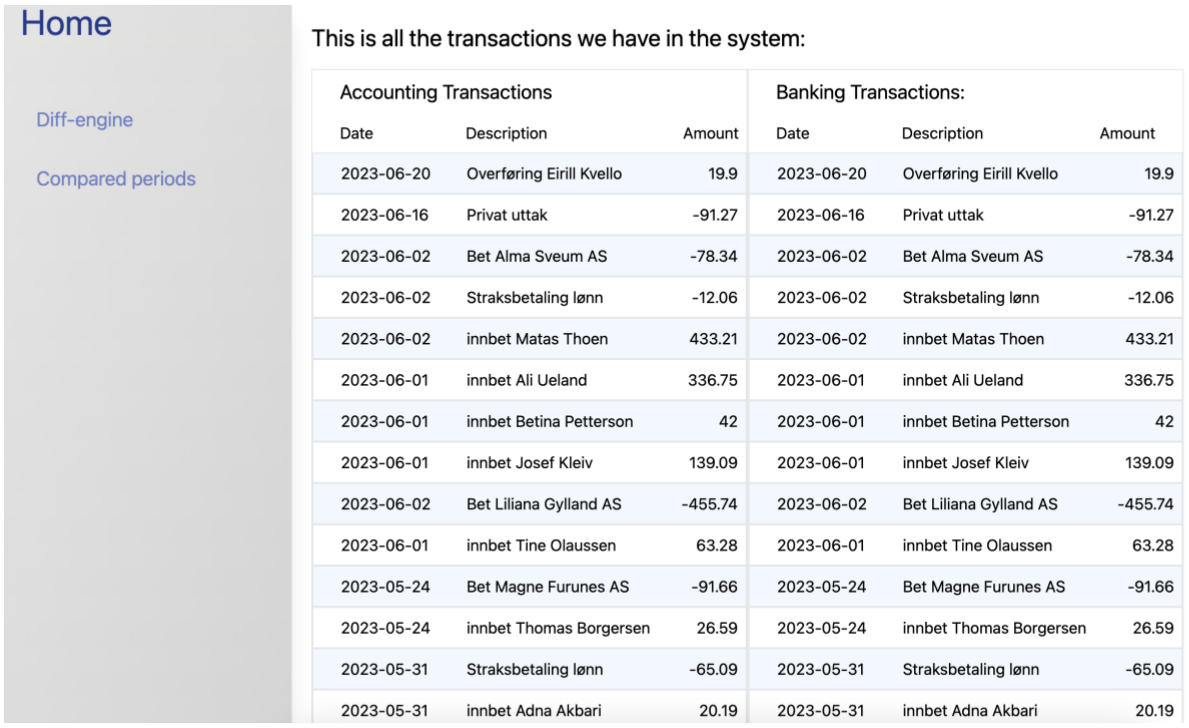
4.1. Features

These are the finished features of the prototype.

- Prototype of a diff-engine – completes comparisons of sets of bank- and accounting transactions and display results
- Period picker that provides two calendars for the user
- Store compared periods in DB
- Navigation between the sites

The features provided in the finished product aligns with the requirement from the client.

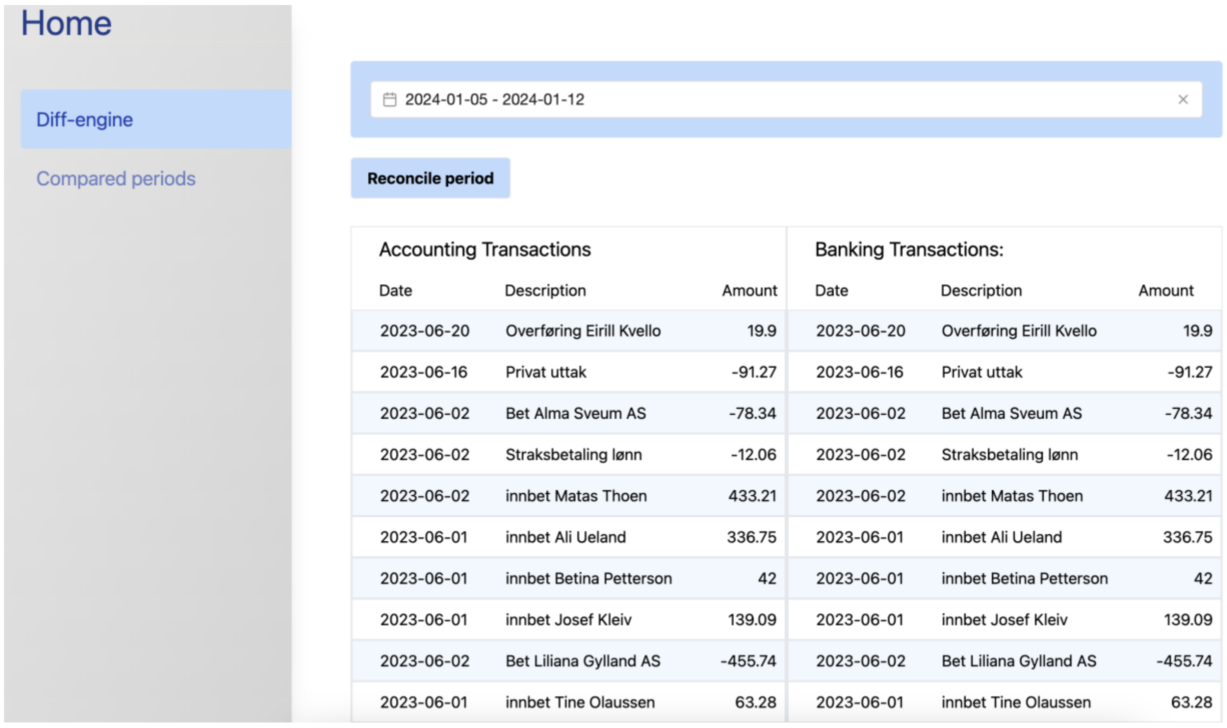
The home page is the first view of the product. This view is used for listing the transactions that are in the system. In the sidebar the user can navigate to either “Diff-engine” or “Compared periods”. The “Diff engine” (figure 5) is where the comparison-prosess happens and “Compared periods” (figure 10) is where the accepted comparisons are saved.



This is all the transactions we have in the system:

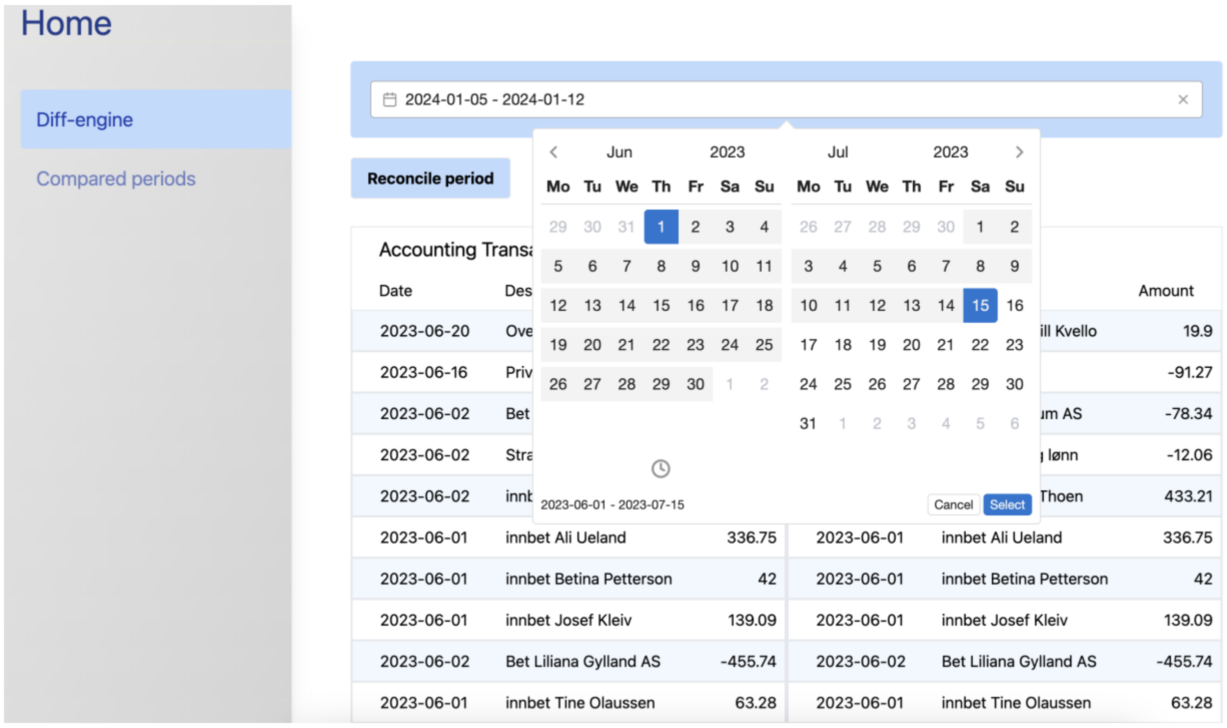
Accounting Transactions			Banking Transactions:		
Date	Description	Amount	Date	Description	Amount
2023-06-20	Overføring Eirill Kvello	19.9	2023-06-20	Overføring Eirill Kvello	19.9
2023-06-16	Privat uttak	-91.27	2023-06-16	Privat uttak	-91.27
2023-06-02	Bet Alma Sveum AS	-78.34	2023-06-02	Bet Alma Sveum AS	-78.34
2023-06-02	Straksbetaling lønn	-12.06	2023-06-02	Straksbetaling lønn	-12.06
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09
2023-06-02	Bet Liliana Gylland AS	-455.74	2023-06-02	Bet Liliana Gylland AS	-455.74
2023-06-01	innbet Tine Olaussen	63.28	2023-06-01	innbet Tine Olaussen	63.28
2023-05-24	Bet Magne Furunes AS	-91.66	2023-05-24	Bet Magne Furunes AS	-91.66
2023-05-24	innbet Thomas Borgersen	26.59	2023-05-24	innbet Thomas Borgersen	26.59
2023-05-31	Straksbetaling lønn	-65.09	2023-05-31	Straksbetaling lønn	-65.09
2023-05-31	innbet Adna Akbari	20.19	2023-05-31	innbet Adna Akbari	20.19

Figure 4 - Home view



Accounting Transactions			Banking Transactions:		
Date	Description	Amount	Date	Description	Amount
2023-06-20	Overføring Eirill Kvello	19.9	2023-06-20	Overføring Eirill Kvello	19.9
2023-06-16	Privat uttak	-91.27	2023-06-16	Privat uttak	-91.27
2023-06-02	Bet Alma Sveum AS	-78.34	2023-06-02	Bet Alma Sveum AS	-78.34
2023-06-02	Straksbetaling lønn	-12.06	2023-06-02	Straksbetaling lønn	-12.06
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09
2023-06-02	Bet Liliana Gylland AS	-455.74	2023-06-02	Bet Liliana Gylland AS	-455.74
2023-06-01	innbet Tine Olaussen	63.28	2023-06-01	innbet Tine Olaussen	63.28

Figure 5 - diff-engine view



2024-01-05 - 2024-01-12

Jun 2023						
Mo	Tu	We	Th	Fr	Sa	Su
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2

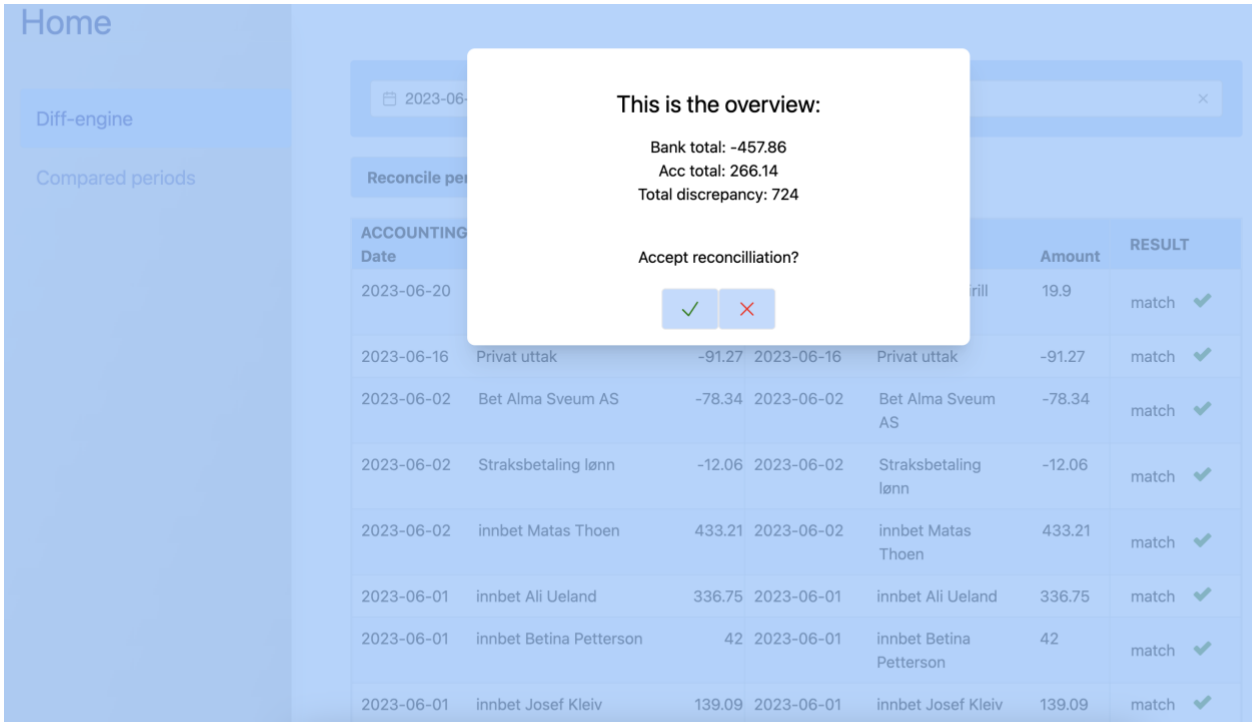
Jul 2023						
Mo	Tu	We	Th	Fr	Sa	Su
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

2023-06-01 - 2023-07-15

Accounting Transa

Date	Des				Amount
2023-06-20	Ove			ill Kvello	19.9
2023-06-16	Priv				-91.27
2023-06-02	Bet			um AS	-78.34
2023-06-02	Stra			lønn	-12.06
2023-06-02	innb			Thoen	433.21
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09
2023-06-02	Bet Liliana Gylland AS	-455.74	2023-06-02	Bet Liliana Gylland AS	-455.74
2023-06-01	innbet Tine Olaussen	63.28	2023-06-01	innbet Tine Olaussen	63.28

Figure 6 - Period picker



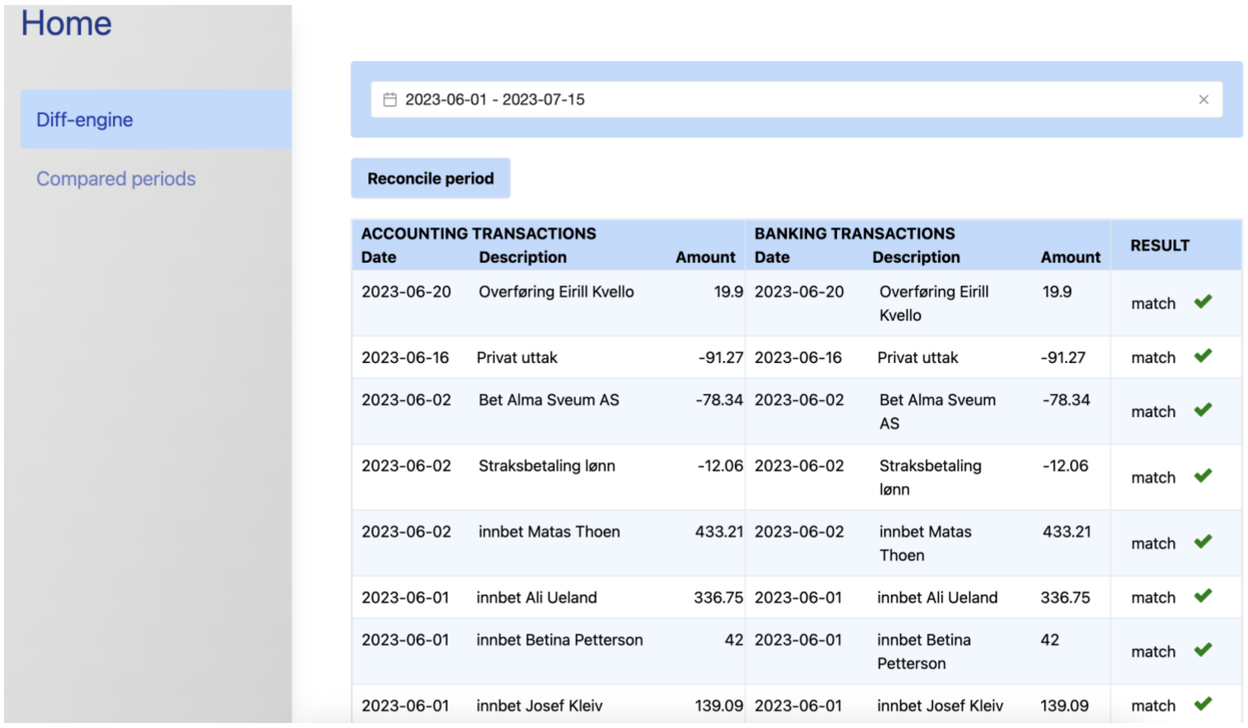
This is the overview:

Bank total: -457.86
Acc total: 266.14
Total discrepancy: 724

Accept reconcilliation?

ACCOUNTING Date					Amount	RESULT
2023-06-20				irill	19.9	match ✓
2023-06-16	Privat uttak	-91.27	2023-06-16	Privat uttak	-91.27	match ✓
2023-06-02	Bet Alma Sveum AS	-78.34	2023-06-02	Bet Alma Sveum AS	-78.34	match ✓
2023-06-02	Straksbetaling lønn	-12.06	2023-06-02	Straksbetaling lønn	-12.06	match ✓
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21	match ✓
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75	match ✓
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42	match ✓
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09	match ✓

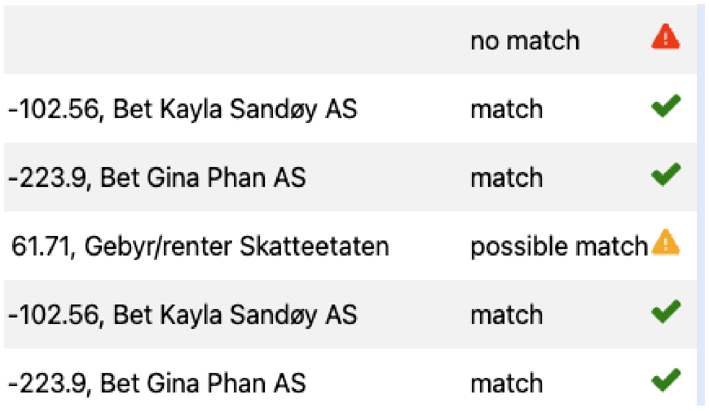
Figure 7 - modal overview



ACCOUNTING TRANSACTIONS			BANKING TRANSACTIONS			RESULT
Date	Description	Amount	Date	Description	Amount	
2023-06-20	Overføring Eirill Kvello	19.9	2023-06-20	Overføring Eirill Kvello	19.9	match ✓
2023-06-16	Privat uttak	-91.27	2023-06-16	Privat uttak	-91.27	match ✓
2023-06-02	Bet Alma Sveum AS	-78.34	2023-06-02	Bet Alma Sveum AS	-78.34	match ✓
2023-06-02	Straksbetaling lønn	-12.06	2023-06-02	Straksbetaling lønn	-12.06	match ✓
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21	match ✓
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75	match ✓
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42	match ✓
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09	match ✓

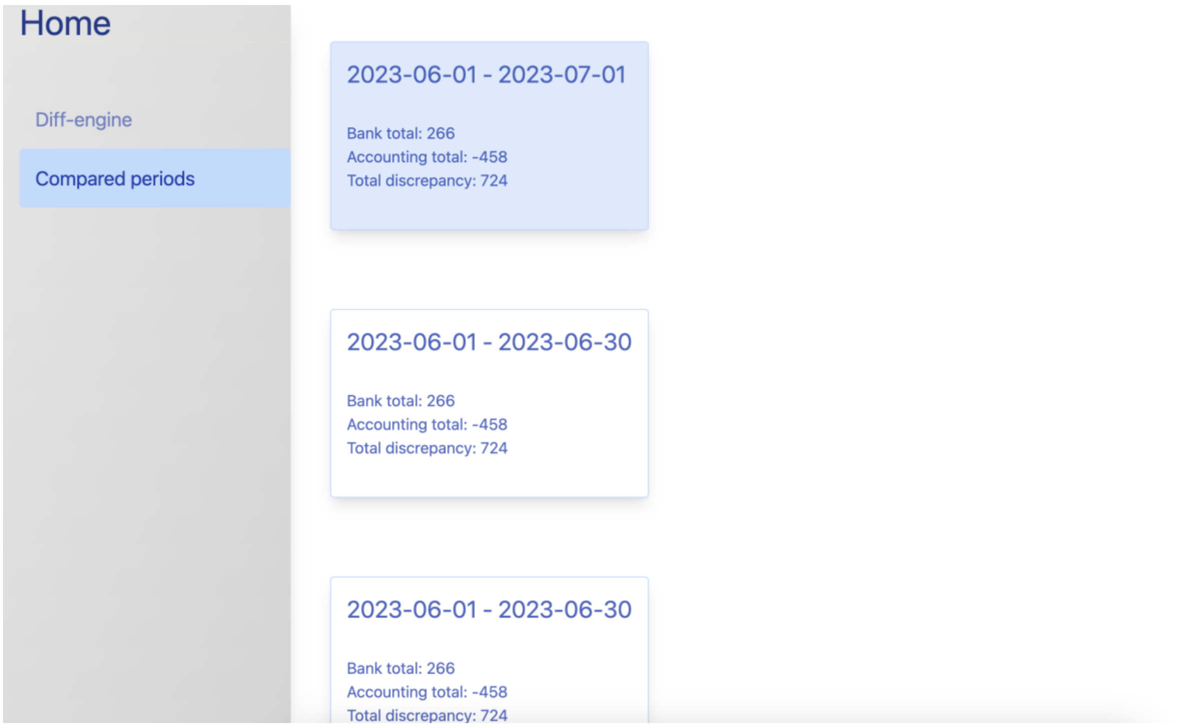
Figure 8 - diff-engine result when not accepting reconciliation

Matches/non-matches as presented to the user as shown in in figure 9.



	no match	⚠
-102.56, Bet Kayla Sandøy AS	match	✓
-223.9, Bet Gina Phan AS	match	✓
61.71, Gebyr/renter Skatteetaten	possible match	⚠
-102.56, Bet Kayla Sandøy AS	match	✓
-223.9, Bet Gina Phan AS	match	✓

Figure 9 - Results with icons



2023-06-01 - 2023-07-01

Bank total: 266
Accounting total: -458
Total discrepancy: 724

2023-06-01 - 2023-06-30

Bank total: 266
Accounting total: -458
Total discrepancy: 724

2023-06-01 - 2023-06-30

Bank total: 266
Accounting total: -458
Total discrepancy: 724

Figure 10 - Compared periods view

4.2. Theoretical contributions

This chapter will present the findings from the feasability stage in the SDLC, which is to determine whether the product could be made or should be pursued. In the theoretical part of the project I collected data regarding bank-reconcilliation, looked into

4.2.1. Theoretical findings

Knudsen-Braas concludes that through technology and integration throughout systems streamlining the accounting business is a possibility. The technology is under constant development, and there is a willingness to change. Opportunities associated with digitization include efficiency, quality, faster access to information, the ability to report in new ways, and knowledge sharing (Knudsen-Baas, 2023).

Prosess automation leverages robotic and cognitive technologies to streamline and mechanize manually executed and standardized tasks in the accounting business. The robotic software plays a pivotal role in reducing human involvement, liberating employees from repetitive tasks and enabling them to concentrate on more important business goals and operations that computers cannot handle as well. Automation offers a number of benefits to the workplace, including cost reduction, higher efficiency, advanced analytics, performance and quality improvement (Can Tansel Kaya, 2020).

Advantages of Robotic Prosess Automation (RPA) according to Convedo:

- Cost Reduction
- Enhanced Speed
- Enhanced scalability
- Enhanced precision and Accuracy

Compliance Improvement: In sectors like financial services, where adherence to regulations such as the Sarbanes-Oxley Act is crucial, RPA facilitates easier documentation and logging of prosesses, ensuring compliance is demonstrable (Convedo, 2019).

Ramona Lacurezeanu says Information Technology (IT) has become an integral component of nearly every business. Given the distinctive nature of their operations and their wide-reaching impact, professional accounting and auditing services stand to enhance their efficiency through the implementation of RPA. Additionally, RPA holds the potential to bolster the credibility of the accounting profession while optimizing operations to align with professional standards, all at significantly reduced costs. This research, grounded in a comprehensive literature review and adopting an exploratory approach, initiates a discourse on the concept of RPA. It delves into customizing RPA in the realm of professional accounting services, scrutinizing robotics models tailored specifically for accounting and audit functions (Ramona Lacurezeanu, 2020).

4.2.2. Exisiting solutions

Tripletex offers their customers automatic bank-reconciliation. Fiken and Sticos offers bank integration and a streamlined reconciliation prosess, but they do not have an automatic bank reconciliation-prosess (Fiken, 2023) (STICOS, 2023). EAccoounting have a diff-engine, with a comparison-model (eAccounting, 2023). Centiga matches transactions from the bank automatic (Centiga, 2023).

4.3. Engineering results

This chapter will present the results from the development phase of the project.

4.3.1. Architecture

Figure 11 shows the system architecture, where it follows a MVC-architecture.

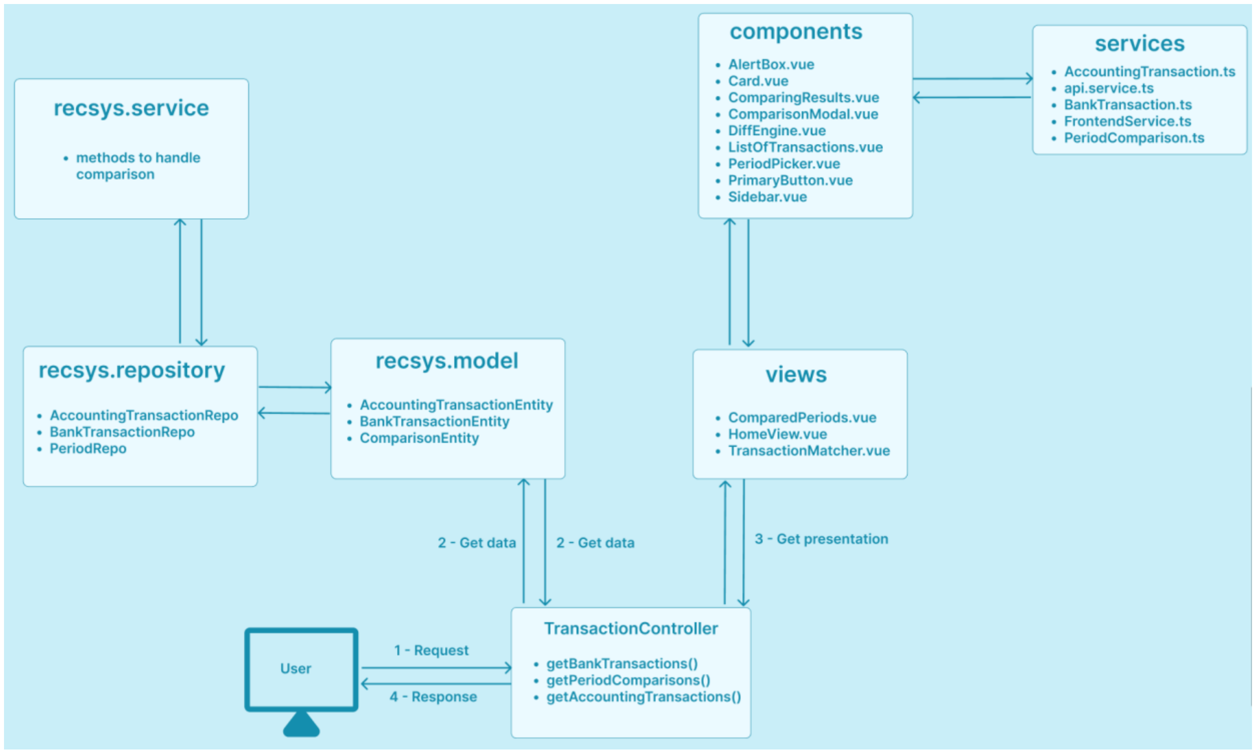


Figure 11 – Architecture

The user sends a request to the controller, which sends requests to the model. The model then manipulates data, and send it to the controller. The controller sends this to the view, which displays it to the user.

As is shown in figure 12 the accounting system collects transactions from users. Recsys then compares the sets of transactions, by retrieving transactions, viewing matching details and completes match.

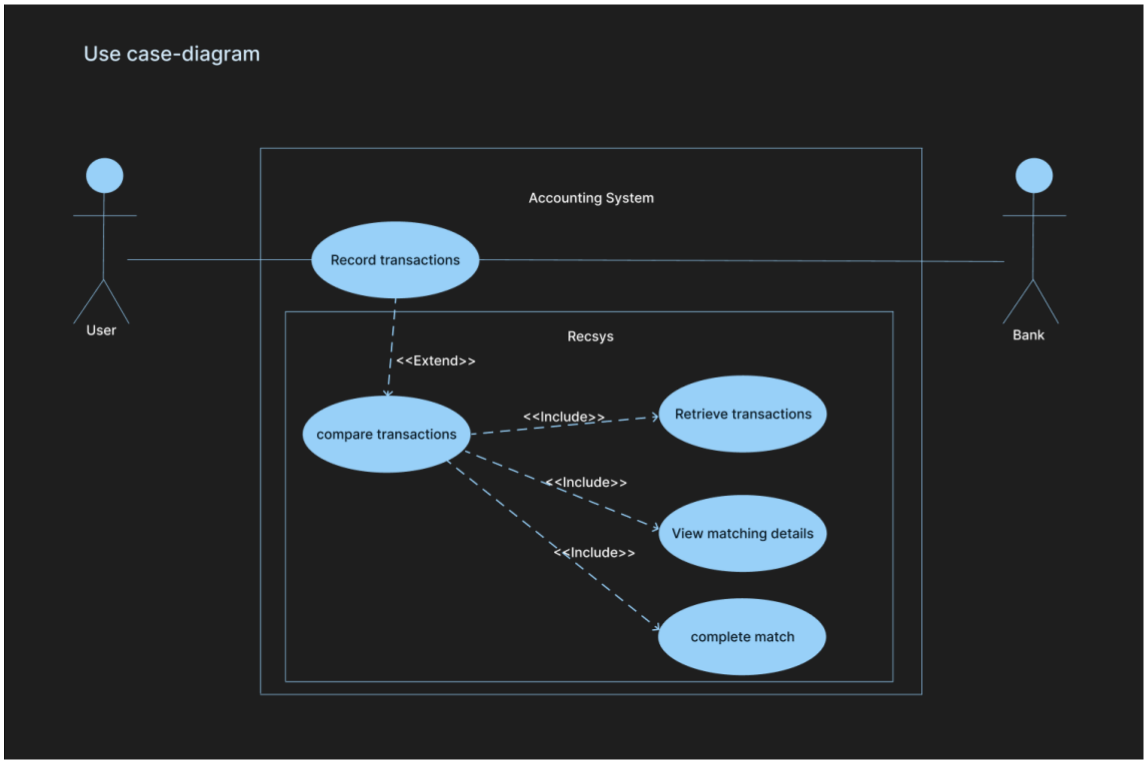


Figure 12 - Use case diagram

This figure is a use-case diagram showing what the finished product includes, which aligns with the clients requirements.

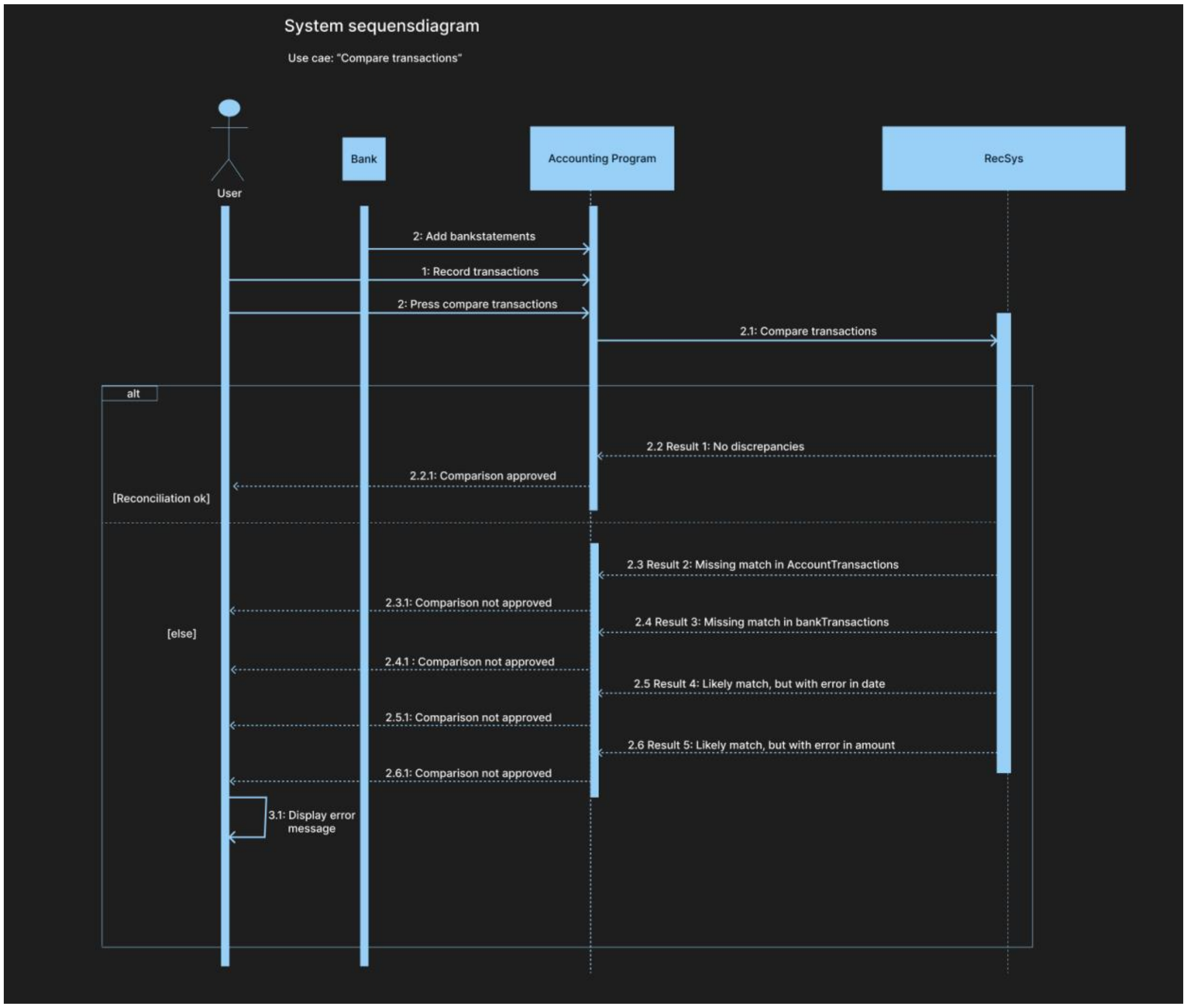

System sequensdiagram
Use cae: "Compare transactions"
User
Bank
Accounting Program
RecSys
2: Add bankstatements
1: Record transactions
2: Press compare transactions
2.1: Compare transactions
alt
2.2 Result 1: No discrepancies
2.2.1: Comparison approved
[Reconciliation ok]
2.3 Result 2: Missing match in AccountTransactions
2.3.1: Comparison not approved
2.4 Result 3: Missing match in bankTransactions
[else]
2.4.1 : Comparison not approved
2.5 Result 4: Likely match, but with error in date
2.5.1: Comparison not approved
2.6 Result 5: Likely match, but with error in amount
2.6.1: Comparison not approved
3.1: Display error message

Figure 13 - Sequens diagram

Figure 13 shows the system sequens diagram. A user record transaction, add bankstatements and press compare-button. The accounting receives these, and RecSys (feature in the accounting system) compares the transactions, and sends results back to the accounting-program. Depending on the results, the user accepts or rejects the reconciliation.

4.3.2. Domain model

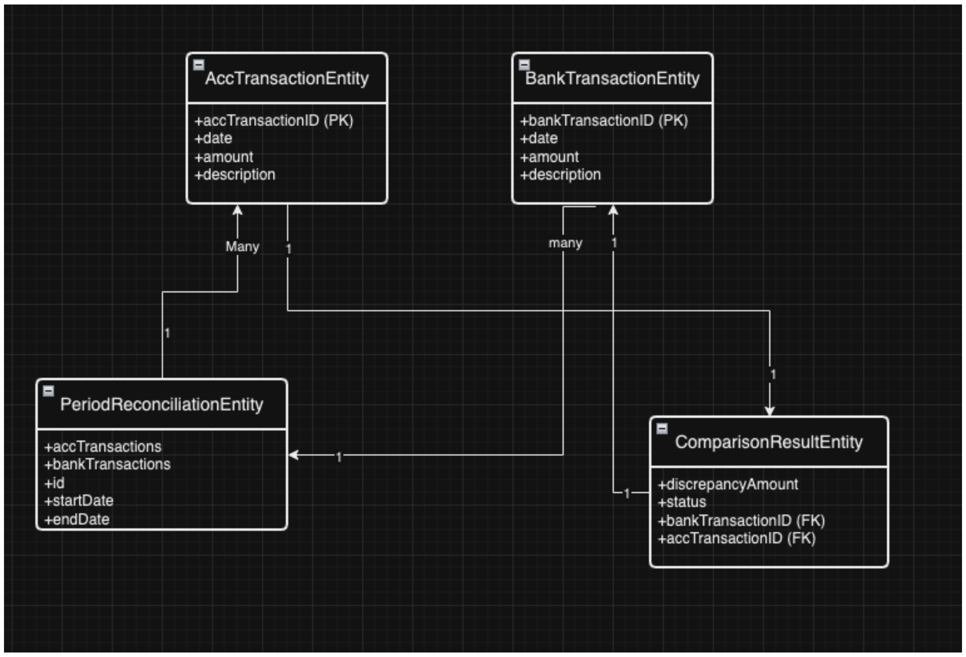


Figure 14 - domain model planned

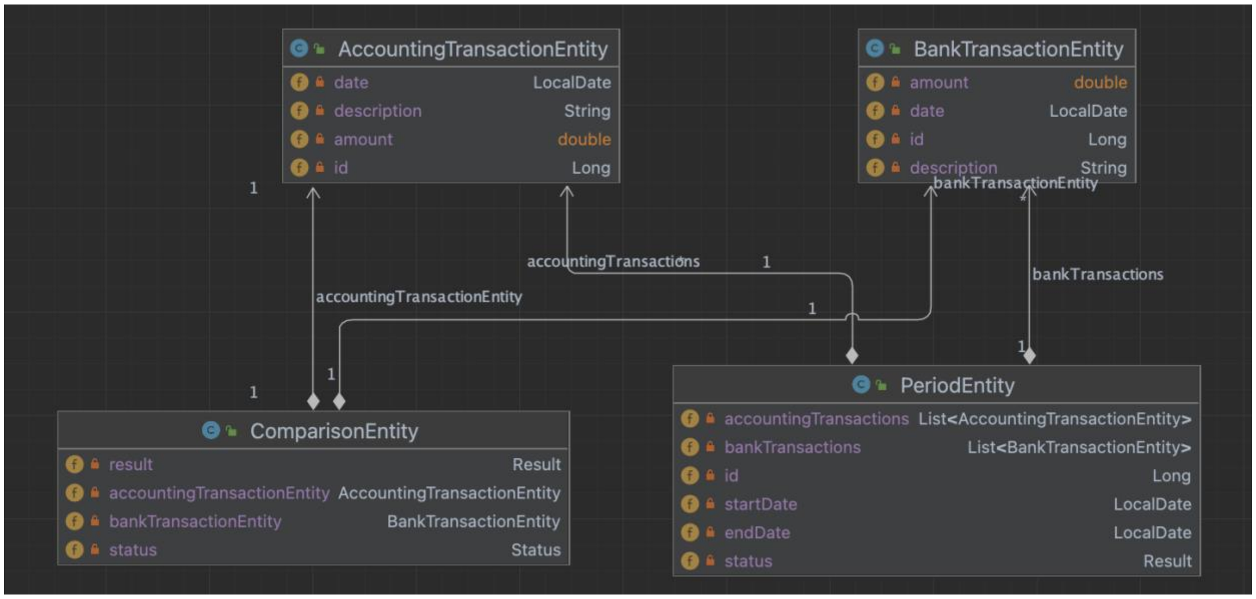


Figure 15 - Domain model rendered from IntelliJ

AccountingTransactionEntity: represents the transactions from the accounbting software, that the accountains have posted for a company.

BankTransactionEntity: Represents the transactions that have actually happened, collected from the bank.
ComparisonEntity: A run-time entity that stores the informatino about the comparison, to be presented for the user.
PeriodEntity: Saves information about the comparison, within the timeframe desired from the user.

4.3.3. Modelled prototypes

4.3.3.1. Low-fidelity prototype

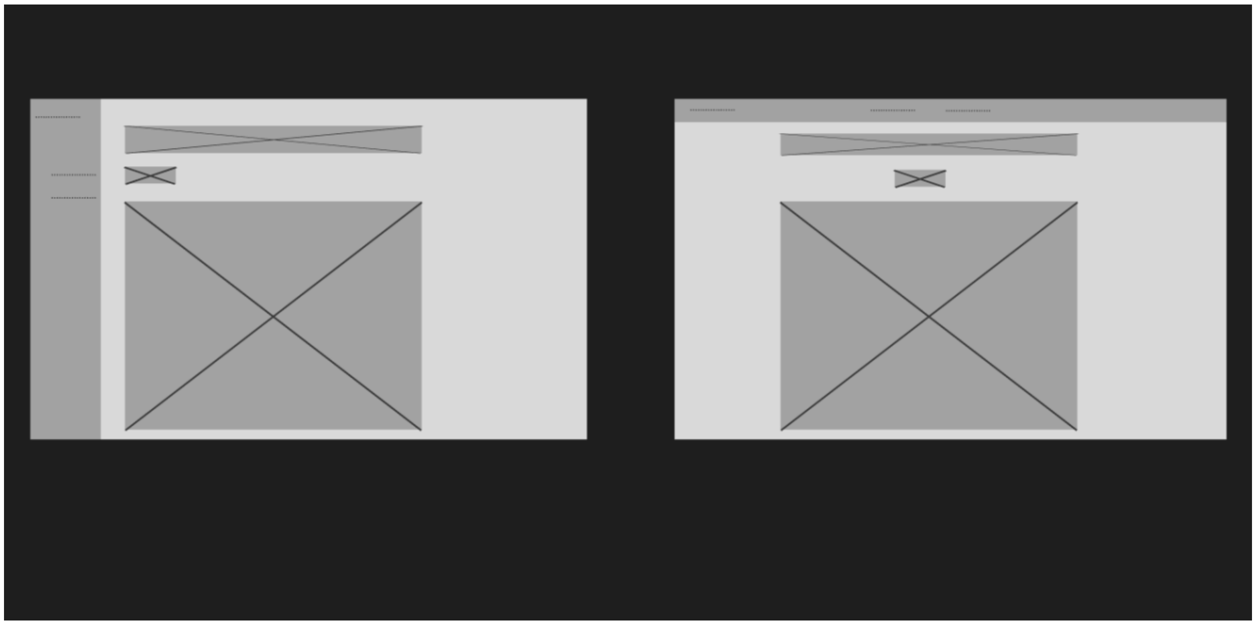


Figure 16 - Low fidelity prototype

The low fidelity prototype shows basic design, as shown in figure 16.

4.3.3.2. High fidelity prototype

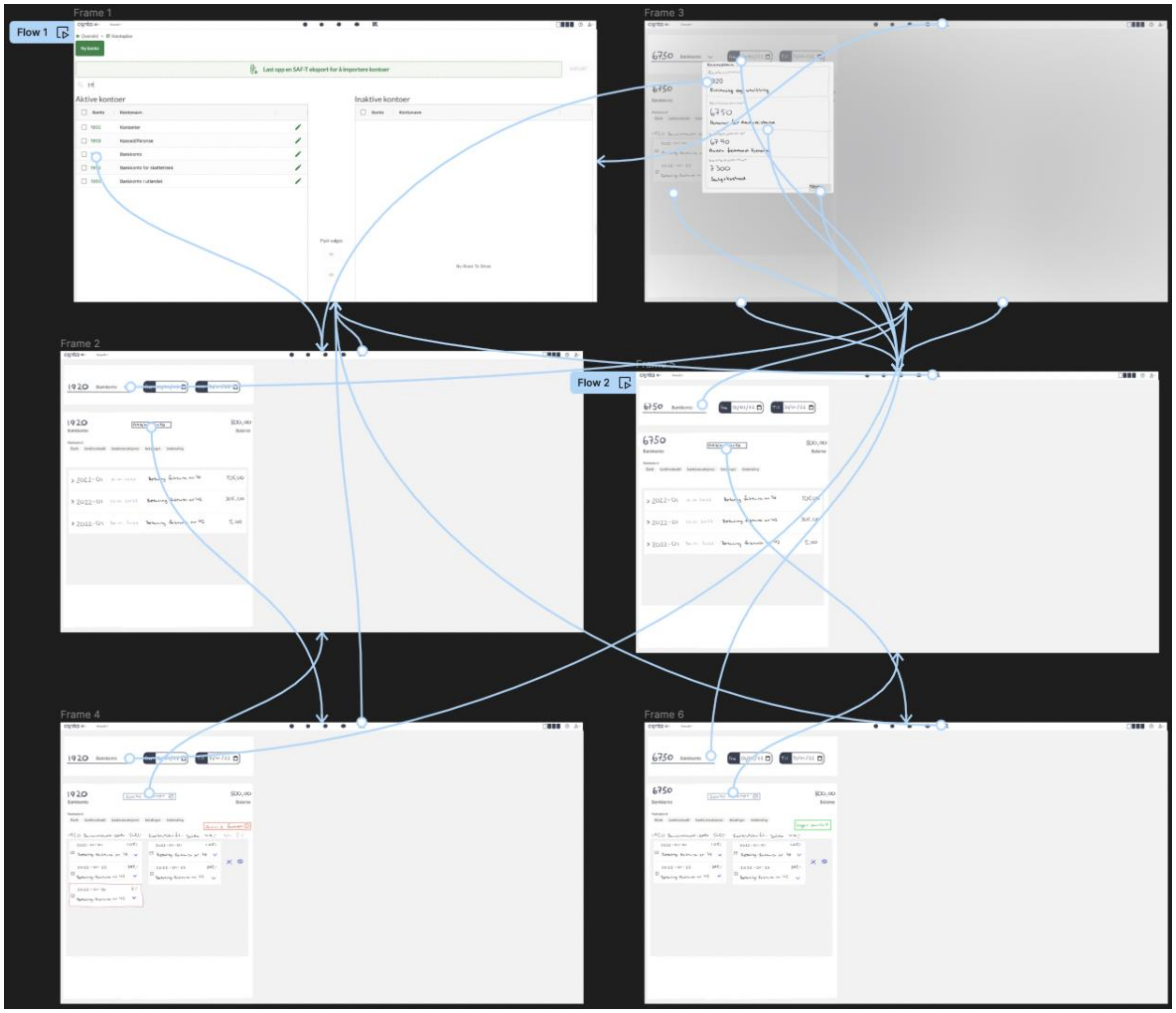
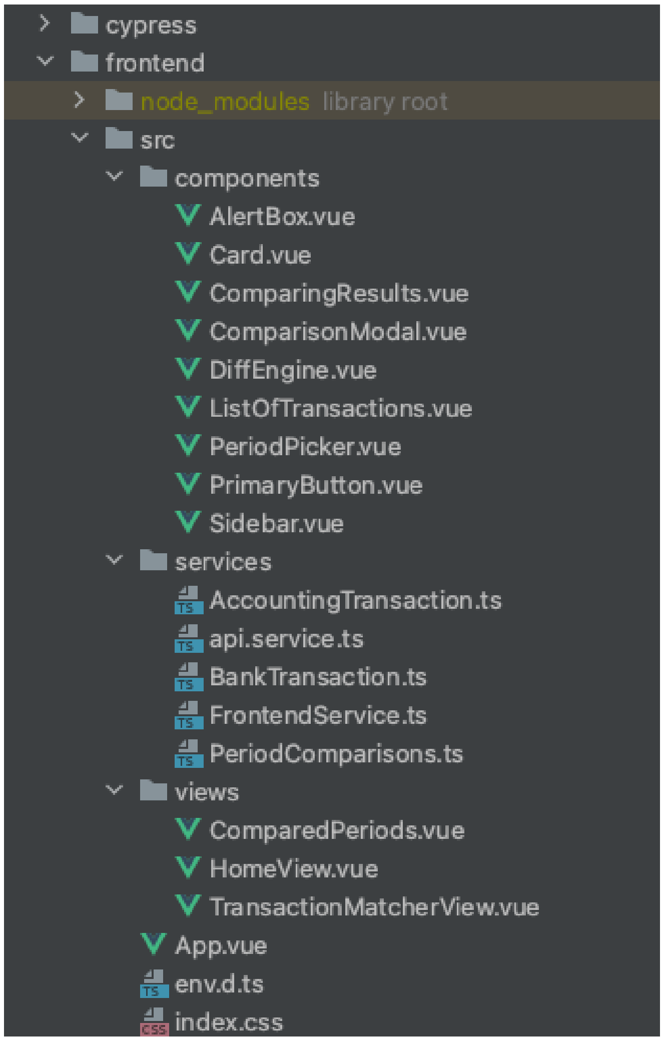


Figure 17 - High fidelity prototype

The high fidelity prototype includes links and pointers, so when the test-subject pressed the correct button they would be sent to the next page. The arrows in figure 17 shows the routing of the prototype.

4.3.4. Project hierarchy

4.3.4.1. Frontend project hierarchy

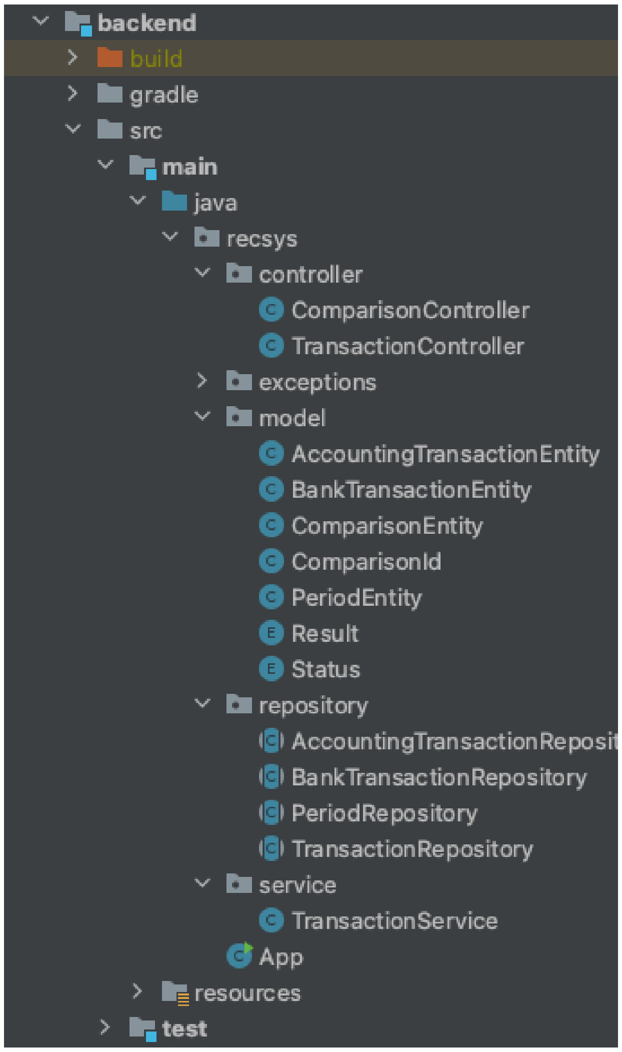


```
> cypress
v frontend
  > node_modules  library root
  v src
    v components
        AlertBox.vue
        Card.vue
        ComparingResults.vue
        ComparisonModal.vue
        DiffEngine.vue
        ListOfTransactions.vue
        PeriodPicker.vue
        PrimaryButton.vue
        Sidebar.vue
    v services
        AccountingTransaction.ts
        api.service.ts
        BankTransaction.ts
        FrontendService.ts
        PeriodComparisons.ts
    v views
        ComparedPeriods.vue
        HomeView.vue
        TransactionMatcherView.vue
      App.vue
      env.d.ts
      index.css
```

Figure 18 - Frontend project hierarchy

The frontend is divided into these packages: components, services and views, where components are the components that makes the web-site, the services are the TypeScript-logic and the views are the pages that contains all the components (As seen in figure 18).

4.3.5. Backend project hierarchy

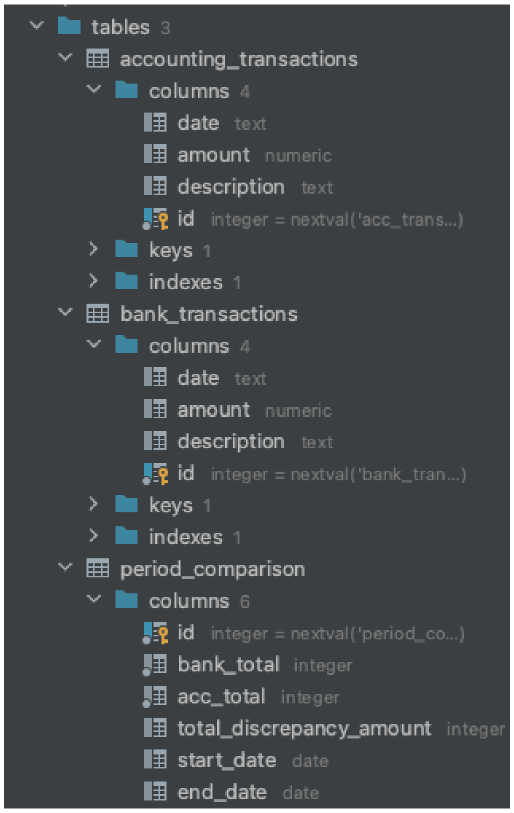


```
backend
  build
  gradle
  src
    main
      java
        recsys
          controller
            ComparisonController
            TransactionController
          exceptions
          model
            AccountingTransactionEntity
            BankTransactionEntity
            ComparisonEntity
            ComparisonId
            PeriodEntity
            Result
            Status
          repository
            AccountingTransactionReposit
            BankTransactionRepository
            PeriodRepository
            TransactionRepository
          service
            TransactionService
          App
      resources
    test
```

Figure 19 - Backend project hierarchy

As seen in figure 19, the backend hierarchy is divided into controllers, exceptions, models, repositories and service-packages, where controllers handle the communication with frontend, the model consists of the entities, repositories handle the method and info connected to entities, the service have the business-logic.

4.3.6. Persistence



```
tables 3
  accounting_transactions
    columns 4
      date text
      amount numeric
      description text
      id integer = nextval('acc_trans...)
    keys 1
    indexes 1
  bank_transactions
    columns 4
      date text
      amount numeric
      description text
      id integer = nextval('bank_tran...)
    keys 1
    indexes 1
  period_comparison
    columns 6
      id integer = nextval('period_co...)
      bank_total integer
      acc_total integer
      total_discrepancy_amount integer
      start_date date
      end_date date
```

Figure 20 - Persistence hieracrhy

There are three tables saved in the DB; accounting_transactions, bank_transactions and period_comparison.

Accounting transaction-table consists of data representing bookkeeped transactions in an accounting program.

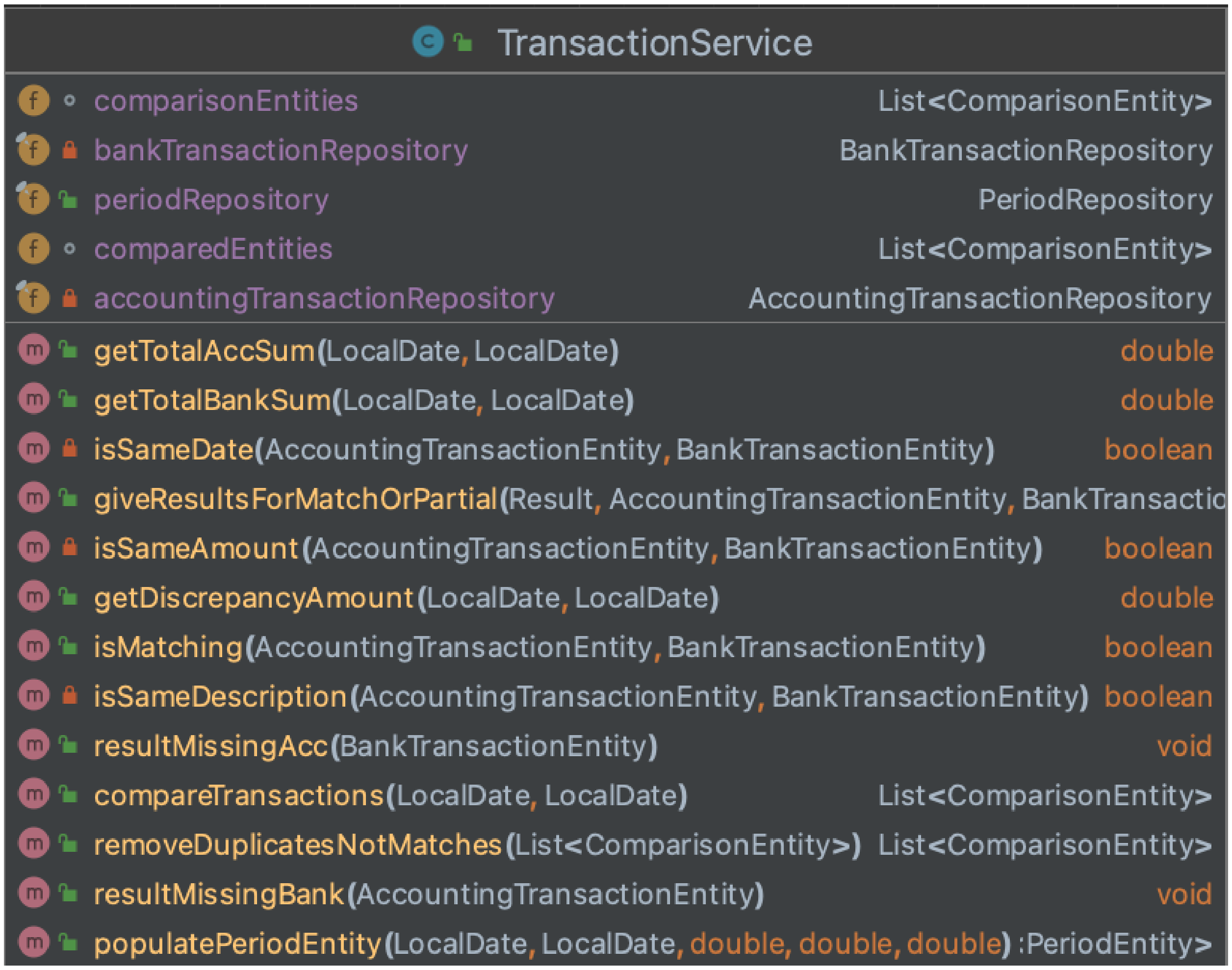
Bank transaction-table consists of data representing actual transaction a customer have made.

These two different sets of transactions are the transactions that needs to be compared to reconcile the period.

In addition there is a period_comparison-table, consisting of the results from the comparison and the time-period.

4.3.7. Business logic

Logic for comparing transactions exists in the class TransactionService.java. Figure 21 shows the fields and methods for the transactionService.class.



```
TransactionService
comparisonEntities                                   List<ComparisonEntity>
bankTransactionRepository                            BankTransactionRepository
periodRepository                                     PeriodRepository
comparedEntities                                     List<ComparisonEntity>
accountingTransactionRepository                      AccountingTransactionRepository
getTotalAccSum(LocalDate, LocalDate)                 double
getTotalBankSum(LocalDate, LocalDate)                double
isSameDate(AccountingTransactionEntity, BankTransactionEntity)          boolean
giveResultsForMatchOrPartial(Result, AccountingTransactionEntity, BankTransactio
isSameAmount(AccountingTransactionEntity, BankTransactionEntity)        boolean
getDiscrepancyAmount(LocalDate, LocalDate)           double
isMatching(AccountingTransactionEntity, BankTransactionEntity)          boolean
isSameDescription(AccountingTransactionEntity, BankTransactionEntity)   boolean
resultMissingAcc(BankTransactionEntity)              void
compareTransactions(LocalDate, LocalDate)            List<ComparisonEntity>
removeDuplicatesNotMatches(List<ComparisonEntity>)   List<ComparisonEntity>
resultMissingBank(AccountingTransactionEntity)       void
populatePeriodEntity(LocalDate, LocalDate, double, double, double) :PeriodEntity>
```

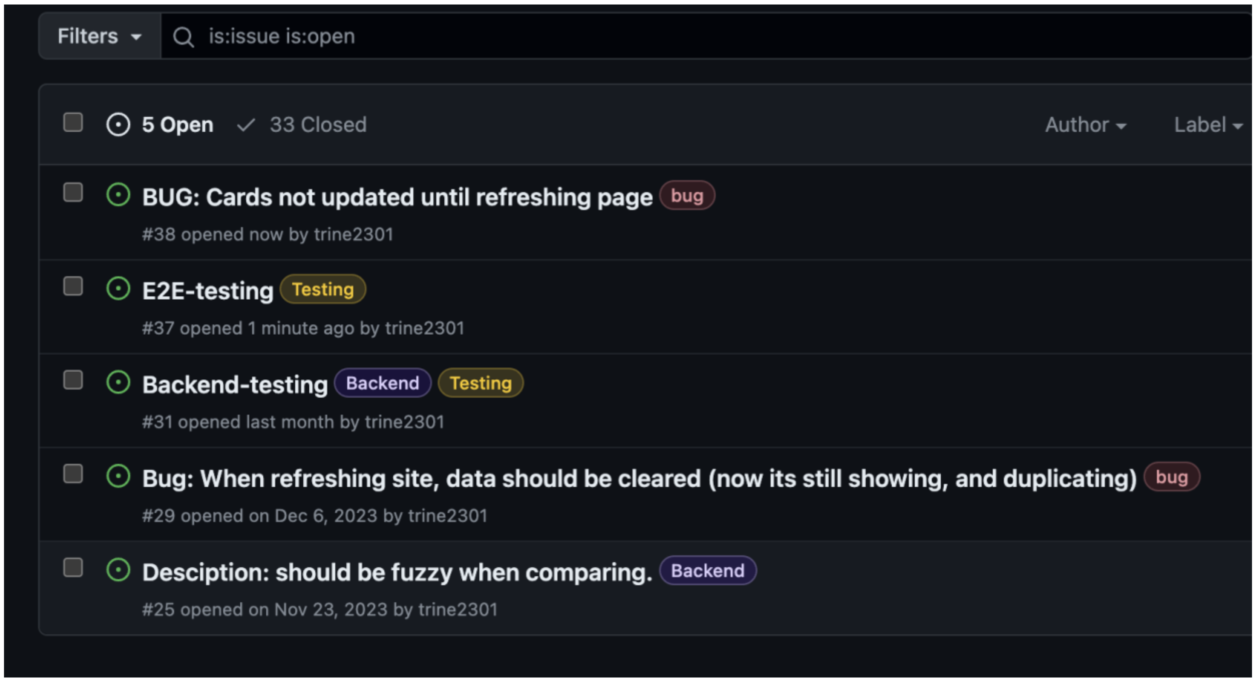
Figure 21 – TransactionService

4.3.7.1. Backlog

There are still some bugs in the system not fixed, these are:

- Cards not updated until page is refreshed
- Compared results in diff-engine is not cleared after comparing a new period

And issues not closed, these are presented in figure 22.



Filters ▾ is:issue is:open

5 Open ✓ 33 Closed Author ▾ Label ▾

BUG: Cards not updated until refreshing page bug
#38 opened now by trine2301

E2E-testing Testing
#37 opened 1 minute ago by trine2301

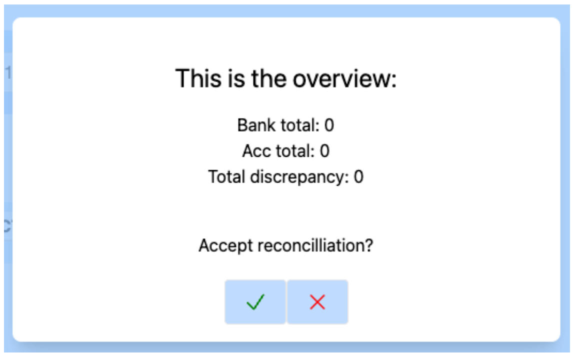
Backend-testing Backend Testing
#31 opened last month by trine2301

Bug: When refreshing site, data should be cleared (now its still showing, and duplicating) bug
#29 opened on Dec 6, 2023 by trine2301

Desciption: should be fuzzy when comparing. Backend
#25 opened on Nov 23, 2023 by trine2301

Figure 22 - Backlog

4.3.8. Vue-components

- AlertBox – When a comparison is made for the period, this is the dialogue that pops up, displaying the overall results from the comparison. The user can choose to accept or decline, and upon accepting the user is navigated to “See compared periods” and the comparison is saved.
  - 

This is the overview:

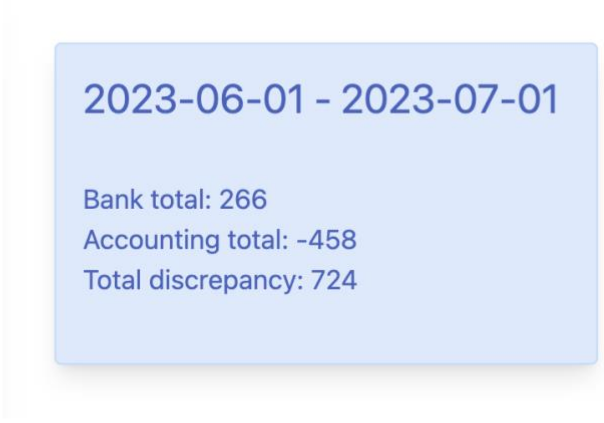
Bank total: 0
Acc total: 0
Total discrepancy: 0

Accept reconcilliation?

✓ ✕

Figure 23 - Alert box

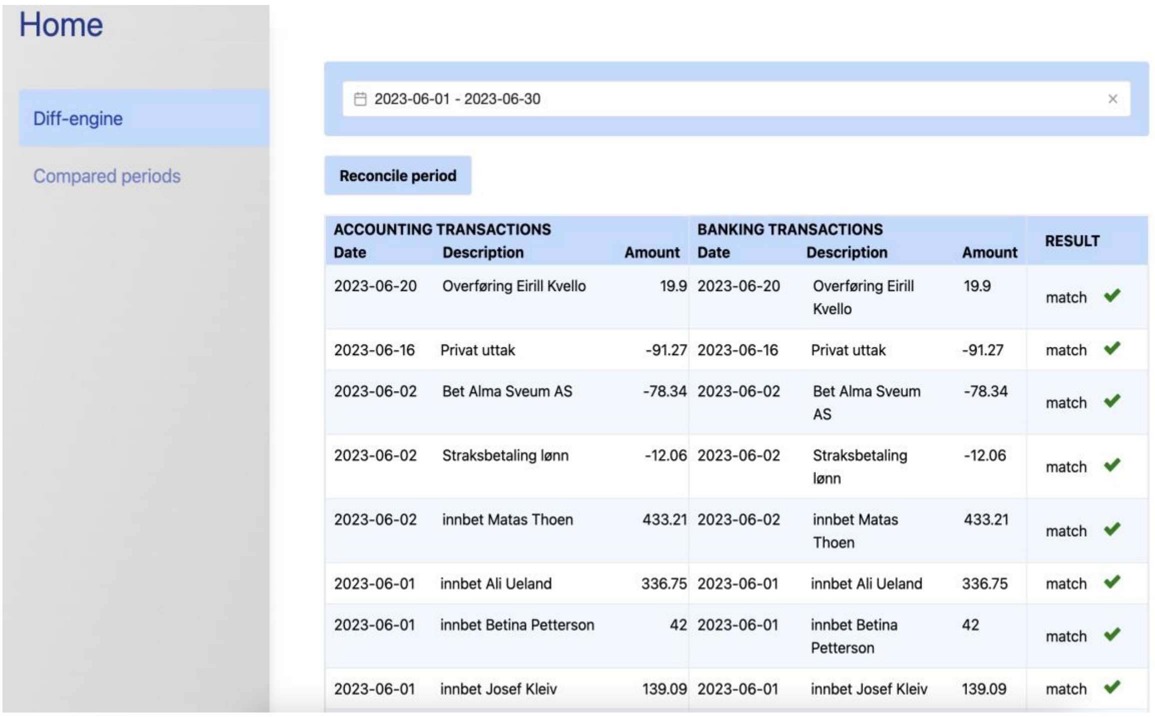
- Card – the card is the one the user sees after accepting the comparison and the comparison is saved.


2023-06-01 - 2023-07-01

Bank total: 266
Accounting total: -458
Total discrepancy: 724

Figure 24 - Card from Period Comparison

- ComparingResults – Will be shown in if there is discrepancy and the user does not accept the comparison. Lists transactions with the results displayed.

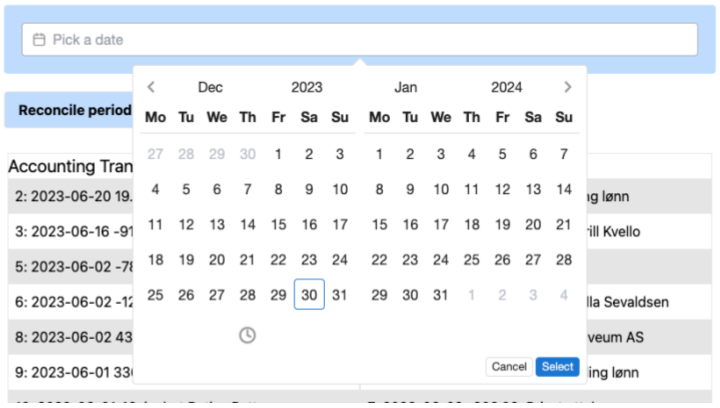


ACCOUNTING TRANSACTIONS			BANKING TRANSACTIONS			RESULT
Date	Description	Amount	Date	Description	Amount	
2023-06-20	Overføring Eirill Kvello	19.9	2023-06-20	Overføring Eirill Kvello	19.9	match ✓
2023-06-16	Privat uttak	-91.27	2023-06-16	Privat uttak	-91.27	match ✓
2023-06-02	Bet Alma Sveum AS	-78.34	2023-06-02	Bet Alma Sveum AS	-78.34	match ✓
2023-06-02	Straksbetaling lønn	-12.06	2023-06-02	Straksbetaling lønn	-12.06	match ✓
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21	match ✓
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75	match ✓
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42	match ✓
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09	match ✓

Figure 25 - Comparison Results

- DiffEngine – this component shows the periodPicker, button and comparingResults after user selects a period and presses button to compare them.
- ListOfTransactions – this component displays a list of transactions. Is used in diffEngine and HomeView.
- PrimaryButton – this component is a styled button.

- PeriodPicker – this component is made of two calendars, where the user can pick a period.



○

Figure 26 - PeriodPicker

- Sidebar – the navigation, set at a fixed width of 400 px. Styled a ready-tailwind/vue-sidebar

4.3.9. Frontend services

- AccountTransaction.ts – an AccountingTransactionType (id, date, amount and description). This class describes the shape of an object that represents an accounting transaction.
- api.service.ts – a plugin that uses the axios library to provide HTTP GET and POST methods to the project.
- BankTransation.ts – it's the same as AccoutTransaction
- FrontendService.ts – Contains methods to use in the components. Business logic.
- PeriodComparison.ts - This class describes the shape of an object that represents a period comparison.

4.3.10. Views

HomeView.vue– the startpage, it displays the transactions that is in the system. Has mainly been used to verify data and debugging purposes.

ComparedPeriods.vue – stores the periods that has been compared, with data about the comparison.

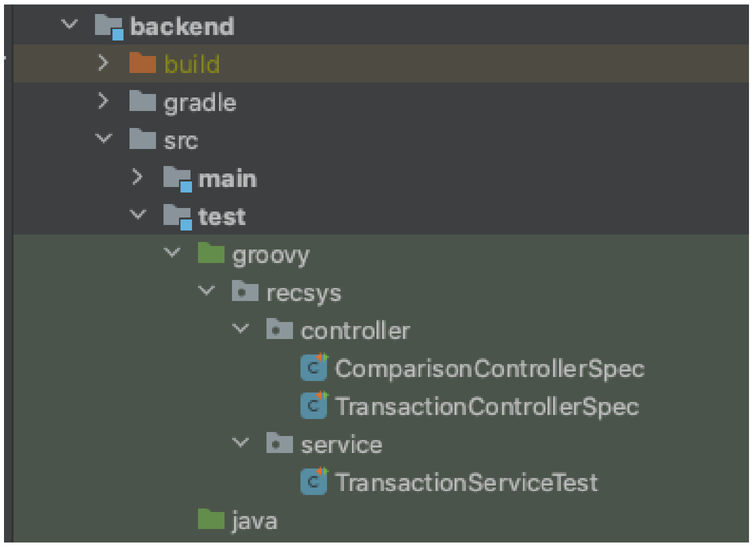
TransactionMatcherView.vue – the diff-engine that contains components that does the comparison for a desired timeframe.

4.4. Quality assurance results

I have conducted unit testing, e2e-tests and usability testing in this bachelor, and this chapter will provide the results of the testing with examples. They will further be discussed in chapter 5.2.13.

4.4.1. Unit testing

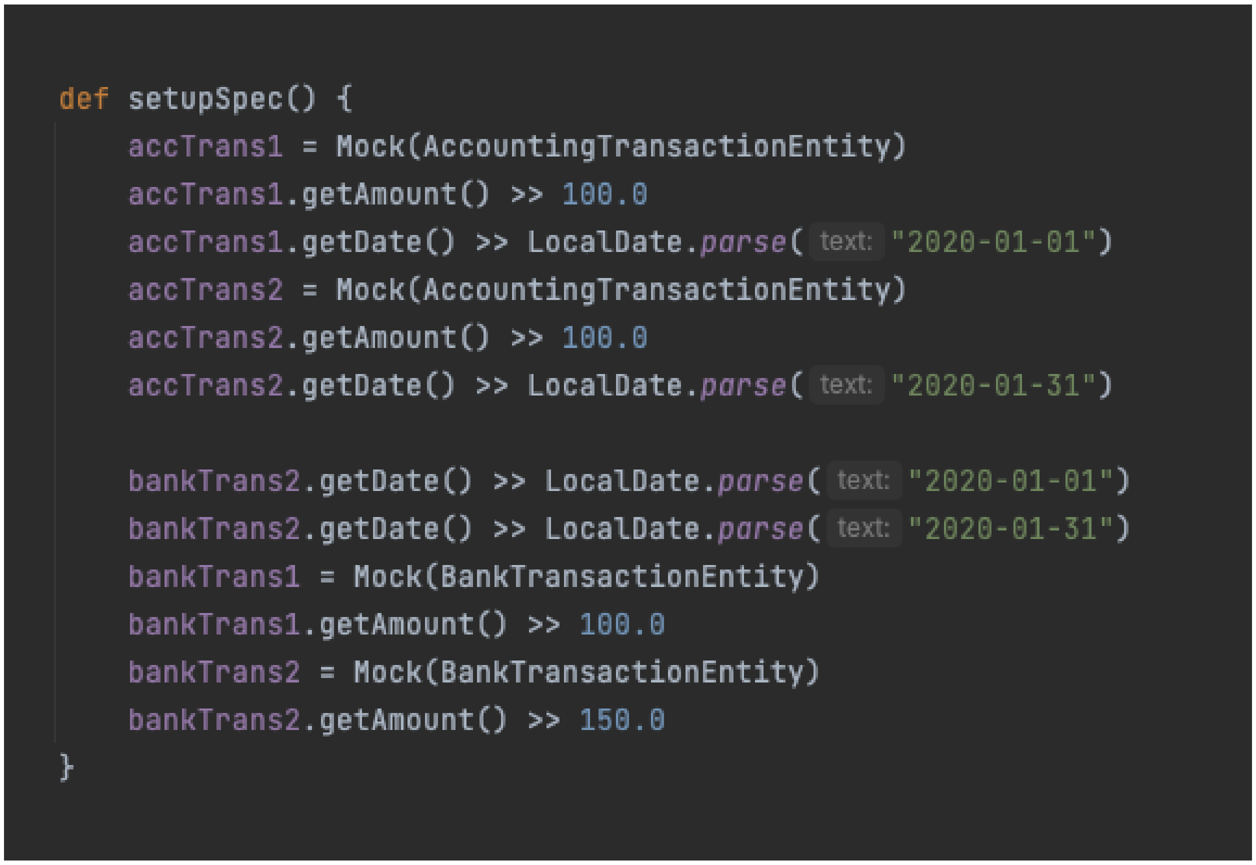
Done in groovy using Spock as mocking framework.



```
backend
  build
  gradle
  src
    main
    test
      groovy
        recsys
          controller
            ComparisonControllerSpec
            TransactionControllerSpec
          service
            TransactionServiceTest
      java
```

Figure 27 - Project hierarchy in unit testing

Figure 27 shows the test-classes shown in the hierarchy in the project. The TransactionServiceTest class provide tests for the business-logic in the project, and the controller-tests provides tests of the controllers.

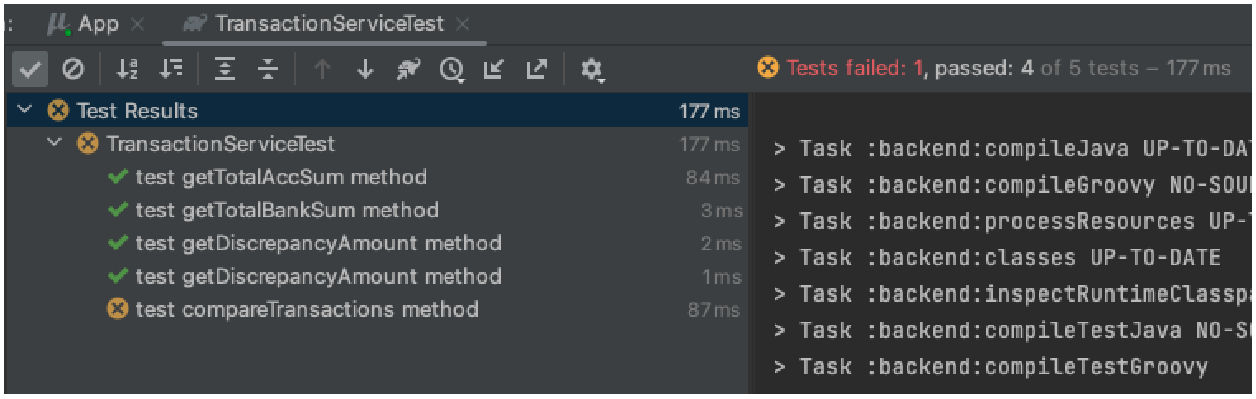


```
def setupSpec() {
    accTrans1 = Mock(AccountingTransactionEntity)
    accTrans1.getAmount() >> 100.0
    accTrans1.getDate() >> LocalDate.parse( text: "2020-01-01")
    accTrans2 = Mock(AccountingTransactionEntity)
    accTrans2.getAmount() >> 100.0
    accTrans2.getDate() >> LocalDate.parse( text: "2020-01-31")

    bankTrans2.getDate() >> LocalDate.parse( text: "2020-01-01")
    bankTrans2.getDate() >> LocalDate.parse( text: "2020-01-31")
    bankTrans1 = Mock(BankTransactionEntity)
    bankTrans1.getAmount() >> 100.0
    bankTrans2 = Mock(BankTransactionEntity)
    bankTrans2.getAmount() >> 150.0
}
```

Figure 28 - mock setup

Figure 28 shows how the test-environment with spock was set up to properly mock two transactions of type AccountingTransactionsEntitiy and two transactions of type BankTransactionEntity. I gave them some amount and a date.



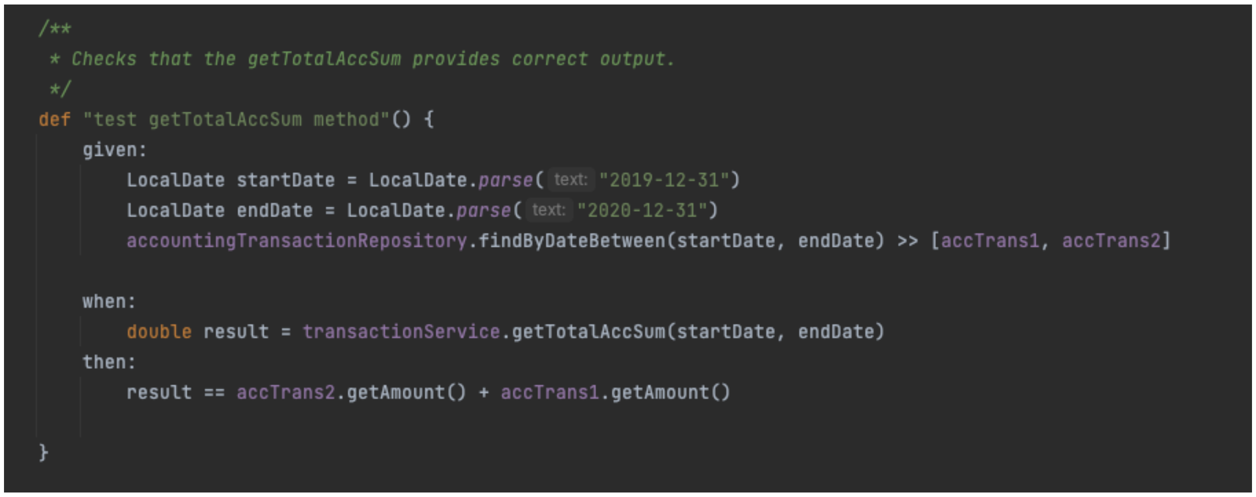
```
Tests failed: 1, passed: 4 of 5 tests – 177 ms

Test Results                                   177 ms
  TransactionServiceTest                       177 ms
    test getTotalAccSum method                  84 ms
    test getTotalBankSum method                  3 ms
    test getDiscrepancyAmount method             2 ms
    test getDiscrepancyAmount method             1 ms
    test compareTransactions method             87 ms

> Task :backend:compileJava UP-TO-DA
> Task :backend:compileGroovy NO-SOU
> Task :backend:processResources UP-
> Task :backend:classes UP-TO-DATE
> Task :backend:inspectRuntimeClassp
> Task :backend:compileTestJava NO-S
> Task :backend:compileTestGroovy
```

Figure 29 - Overview tests

Figure 29 shows an overview if the test results.



```
/**
 * Checks that the getTotalAccSum provides correct output.
 */
def "test getTotalAccSum method"() {
    given:
        LocalDate startDate = LocalDate.parse( text: "2019-12-31")
        LocalDate endDate = LocalDate.parse( text: "2020-12-31")
        accountingTransactionRepository.findByDateBetween(startDate, endDate) >> [accTrans1, accTrans2]

    when:
        double result = transactionService.getTotalAccSum(startDate, endDate)
    then:
        result == accTrans2.getAmount() + accTrans1.getAmount()

}
```

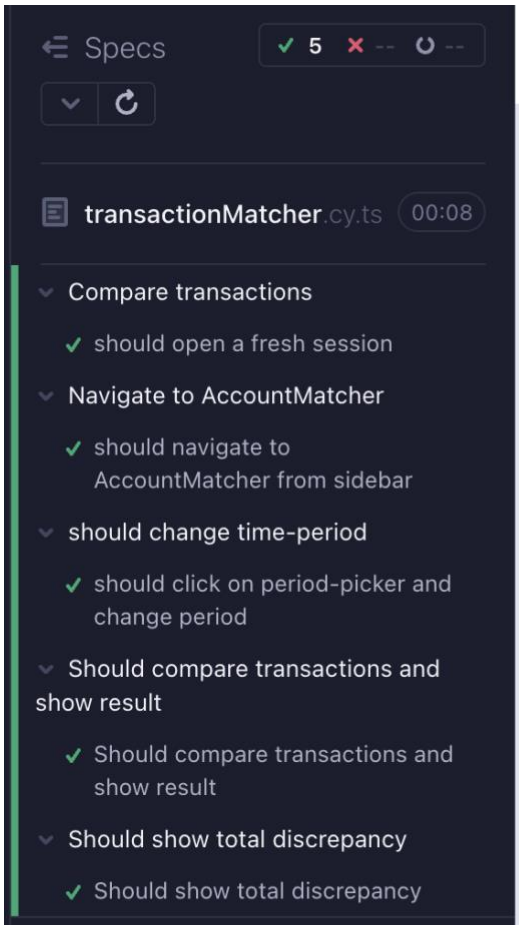
Figure 30 - Unit test example TransactionService-class

Figure 30 shows an example of how a Groovy test is carried out in this project.

4.4.2. E2E-testing

E2E-tests have been carried out in Cypress. Following components have been tested:

- Sidebar → navigating to diff-engine
- Period-picker
- Pressing «reconcile period»-button
- Displaying alert-box
- Pressing «decline»-icon in alert box
- Displaing requested and expected results



transactionMatcher.cy.ts 00:08

Compare transactions
✓ should open a fresh session

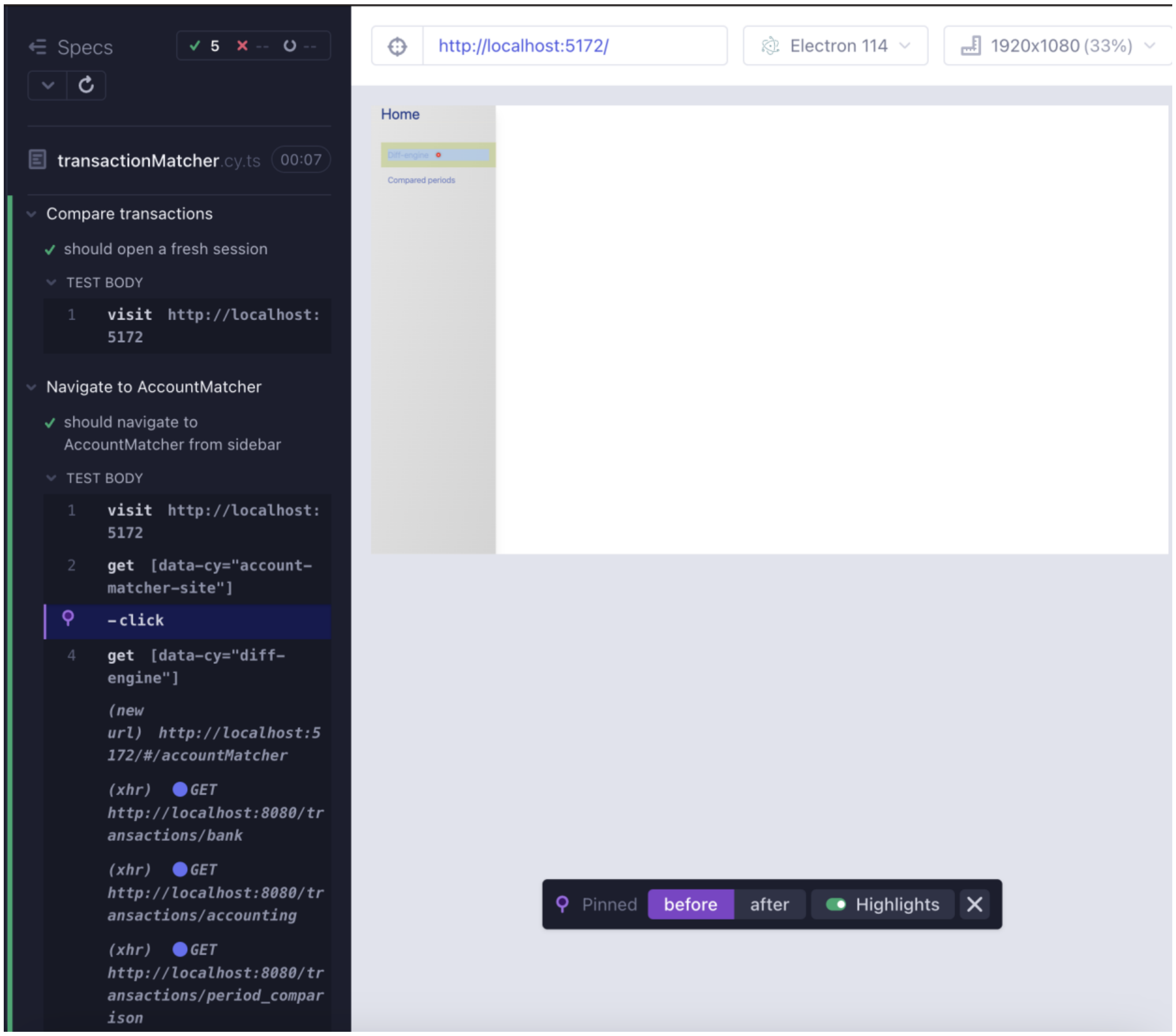
Navigate to AccountMatcher
✓ should navigate to AccountMatcher from sidebar

should change time-period
✓ should click on period-picker and change period

Should compare transactions and show result
✓ Should compare transactions and show result

Should show total discrepancy
✓ Should show total discrepancy

Figure 31 - Overview of E2E-tests



transactionMatcher.cy.ts 00:07

Compare transactions
✓ should open a fresh session
TEST BODY

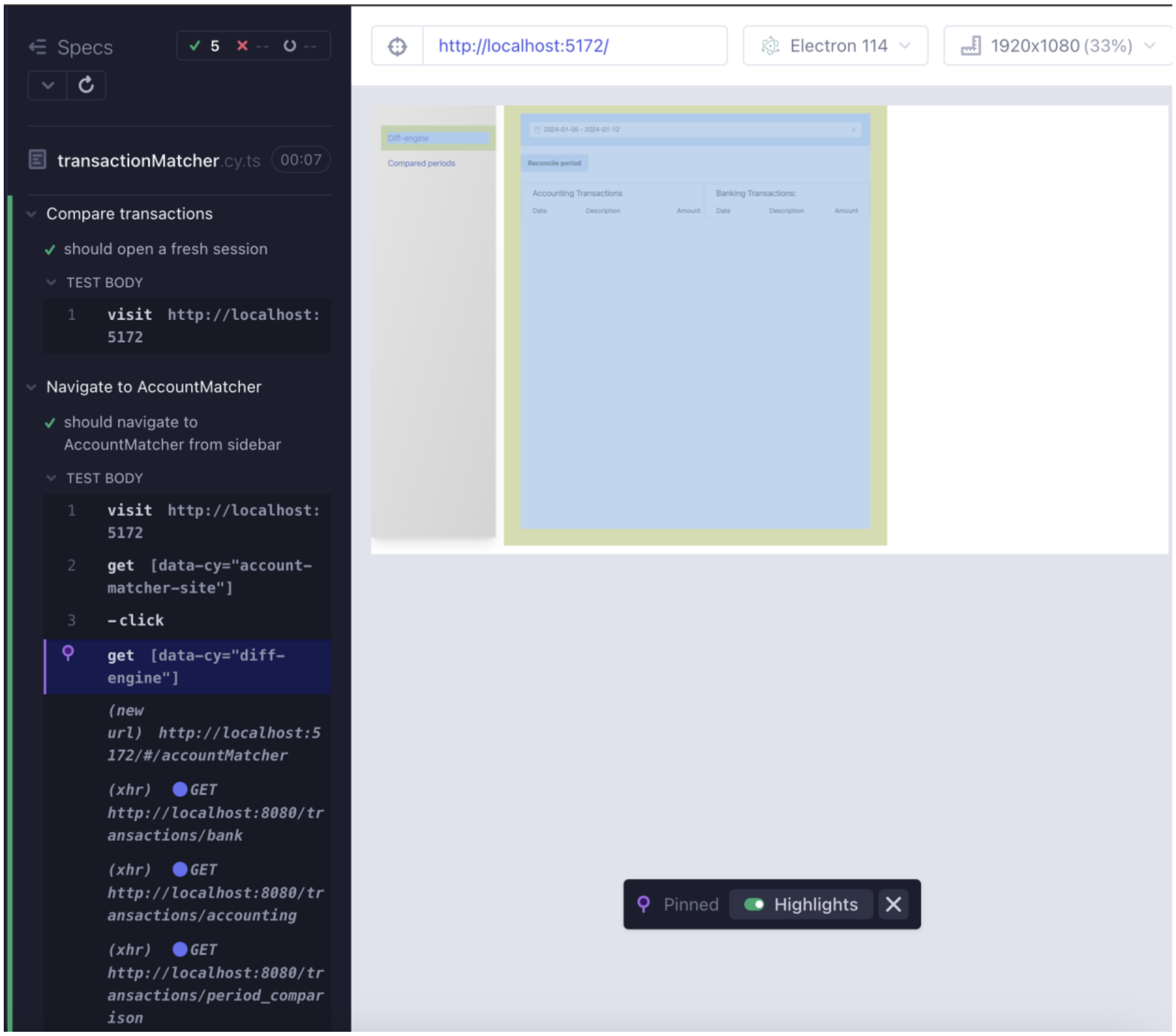
```
visit  http://localhost:5172
```

Navigate to AccountMatcher
✓ should navigate to AccountMatcher from sidebar
TEST BODY

```
visit  http://localhost:5172
get  [data-cy="account-matcher-site"]
-click
get  [data-cy="diff-engine"]
(new url)  http://localhost:5172/#/accountMatcher
(xhr)  ● GET http://localhost:8080/transactions/bank
(xhr)  ● GET http://localhost:8080/transactions/accounting
(xhr)  ● GET http://localhost:8080/transactions/period_comparison
```

Figure 32 - Testing sidebar-navigation

Figure 32 shows the steps the developer can click, and the display showing the website on the right.



transactionMatcher.cy.ts 00:07

Compare transactions

✓ should open a fresh session

TEST BODY

```
visit http://localhost:5172
```

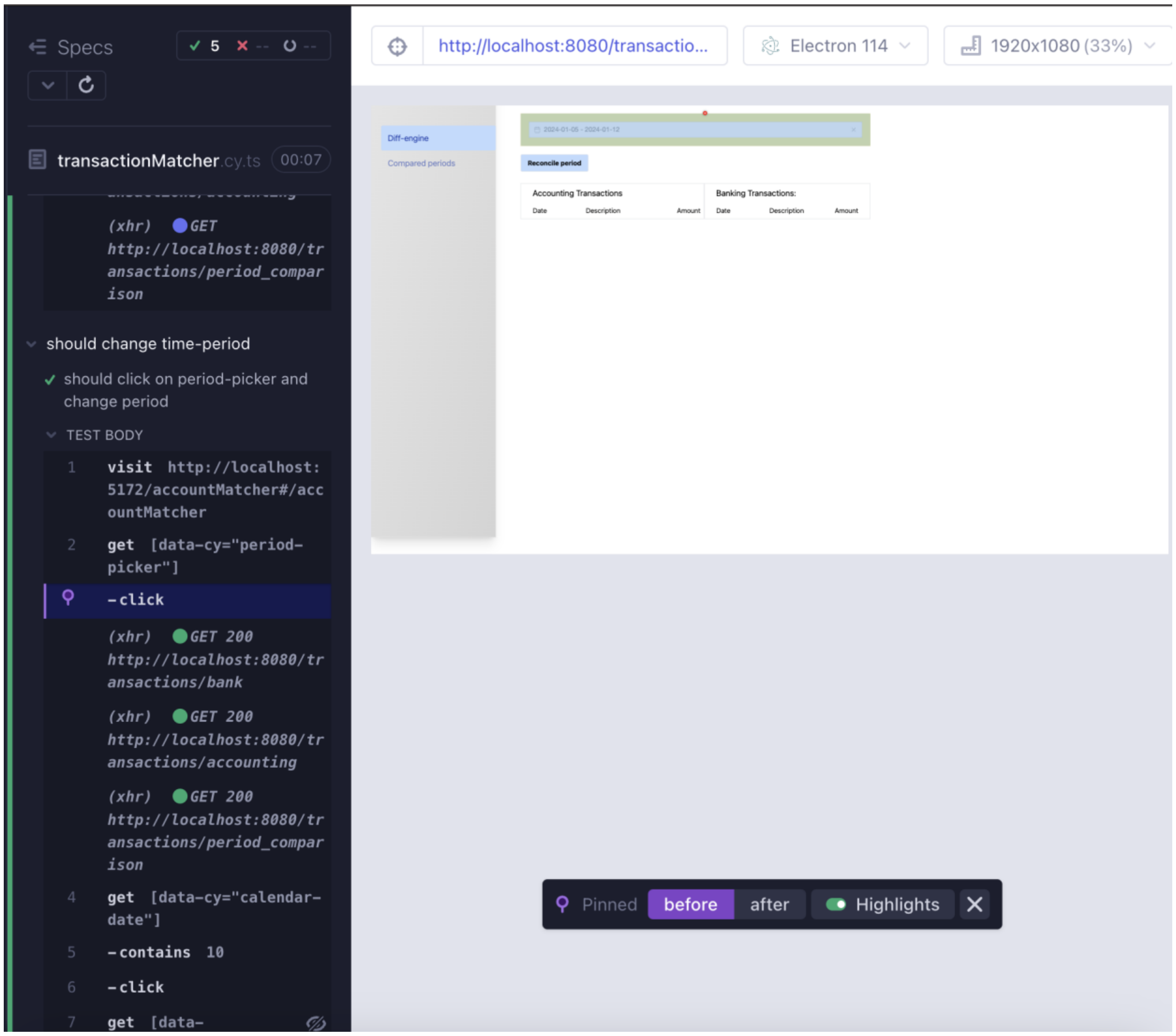
Navigate to AccountMatcher

✓ should navigate to AccountMatcher from sidebar

TEST BODY

```
visit  http://localhost:5172
get  [data-cy="account-matcher-site"]
-click
get  [data-cy="diff-engine"]
(new url)  http://localhost:5172/#/accountMatcher
(xhr)  ● GET  http://localhost:8080/transactions/bank
(xhr)  ● GET  http://localhost:8080/transactions/accounting
(xhr)  ● GET  http://localhost:8080/transactions/period_comparison
```

Figure 33 - Displaying diff-engine in Cypress

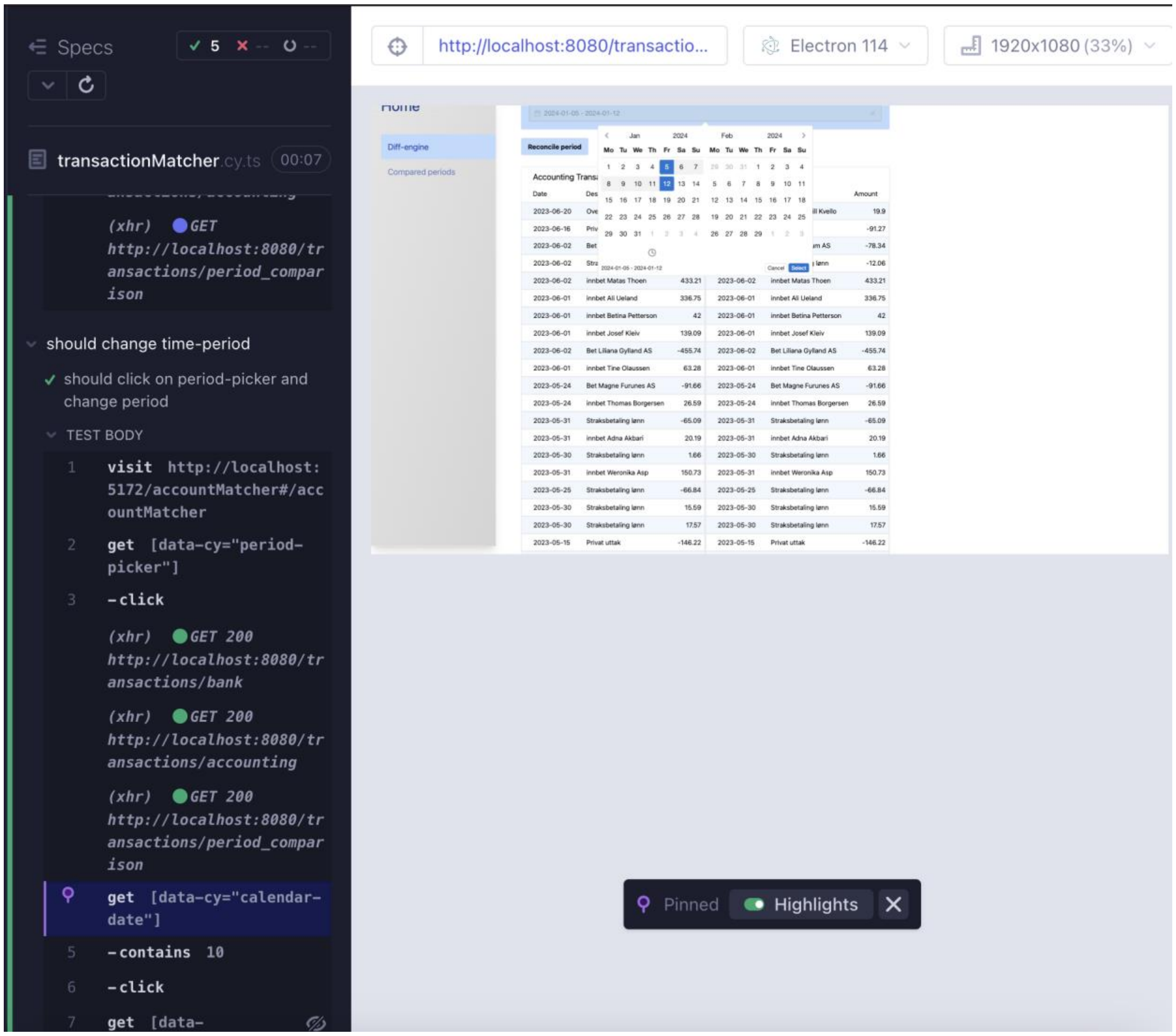


```
(xhr) ● GET http://localhost:8080/transactions/period_comparison

should change time-period
  ✓ should click on period-picker and change period
    TEST BODY
      visit http://localhost:5172/accountMatcher#/accountMatcher
      get [data-cy="period-picker"]
      -click
        (xhr) ● GET 200 http://localhost:8080/transactions/bank
        (xhr) ● GET 200 http://localhost:8080/transactions/accounting
        (xhr) ● GET 200 http://localhost:8080/transactions/period_comparison
      get [data-cy="calendar-date"]
      -contains 10
      -click
      get [data-
```

Accounting Transactions			Banking Transactions:		
Date	Description	Amount	Date	Description	Amount

Figure 34 - Opening period-picker in Cypress



```
Specs   ✓ 5  ✕ --  ○ --

transactionMatcher.cy.ts   00:07

    (xhr)  ● GET
    http://localhost:8080/transactions/period_comparison

should change time-period
  ✓ should click on period-picker and change period
    TEST BODY
    visit http://localhost:5172/accountMatcher#/accountMatcher
    get [data-cy="period-picker"]
    -click
    (xhr)  ● GET 200
    http://localhost:8080/transactions/bank
    (xhr)  ● GET 200
    http://localhost:8080/transactions/accounting
    (xhr)  ● GET 200
    http://localhost:8080/transactions/period_comparison
    get [data-cy="calendar-date"]
    -contains 10
    -click
    get [data-
```

http://localhost:8080/transactio... Electron 114 1920x1080 (33%)

Home

Diff-engine

Compared periods

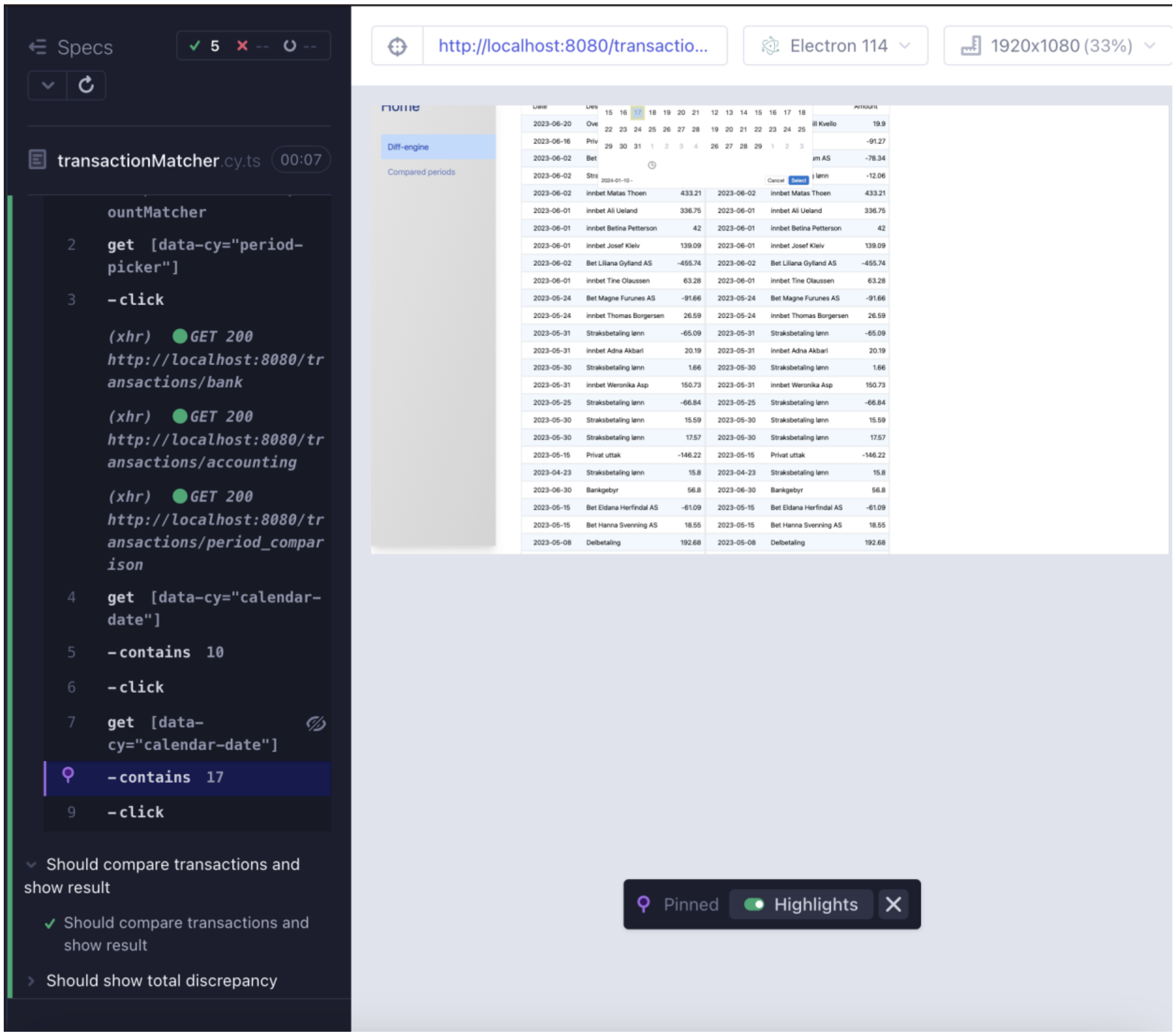
2024-01-05 - 2024-01-12

Reconcile period

Accounting Transa

Date	Des				Amount
2023-06-20	Ove			ill Kvello	19.9
2023-06-16	Priv				-91.27
2023-06-02	Bet			am AS	-78.34
2023-06-02	Stra			lønn	-12.06
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09
2023-06-02	Bet Liliana Gylland AS	-455.74	2023-06-02	Bet Liliana Gylland AS	-455.74
2023-06-01	innbet Tine Olaussen	63.28	2023-06-01	innbet Tine Olaussen	63.28
2023-05-24	Bet Magne Furunes AS	-91.66	2023-05-24	Bet Magne Furunes AS	-91.66
2023-05-24	innbet Thomas Borgersen	26.59	2023-05-24	innbet Thomas Borgersen	26.59
2023-05-31	Straksbetaling lønn	-65.09	2023-05-31	Straksbetaling lønn	-65.09
2023-05-31	innbet Adna Akbari	20.19	2023-05-31	innbet Adna Akbari	20.19
2023-05-30	Straksbetaling lønn	1.66	2023-05-30	Straksbetaling lønn	1.66
2023-05-31	innbet Weronika Asp	150.73	2023-05-31	innbet Weronika Asp	150.73
2023-05-25	Straksbetaling lønn	-66.84	2023-05-25	Straksbetaling lønn	-66.84
2023-05-30	Straksbetaling lønn	15.59	2023-05-30	Straksbetaling lønn	15.59
2023-05-30	Straksbetaling lønn	17.57	2023-05-30	Straksbetaling lønn	17.57
2023-05-15	Privat uttak	-146.22	2023-05-15	Privat uttak	-146.22

Figure 35 - Choosing period in Cypress

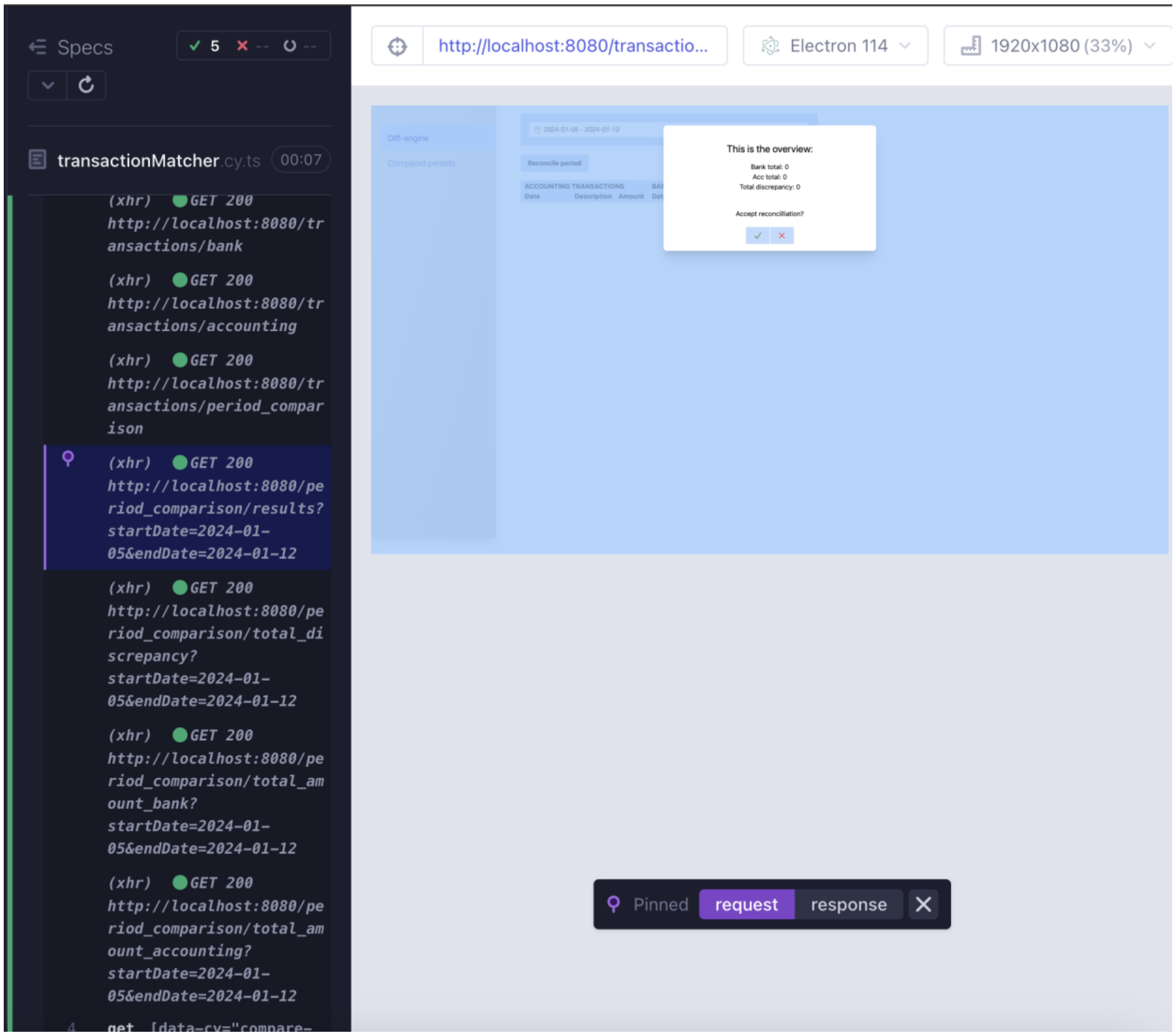


```
  ountMatcher
2 get [data-cy="period-picker"]
3 -click
  (xhr) GET 200 http://localhost:8080/transactions/bank
  (xhr) GET 200 http://localhost:8080/transactions/accounting
  (xhr) GET 200 http://localhost:8080/transactions/period_comparison
4 get [data-cy="calendar-date"]
5 -contains 10
6 -click
7 get [data-cy="calendar-date"]
  -contains 17
9 -click
```

Should compare transactions and show result
✓ Should compare transactions and show result
Should show total discrepancy

Date	Des				Amount
2023-06-20	Ove			ill Kvello	19.9
2023-06-16	Priv				-91.27
2023-06-02	Bet			um AS	-78.34
2023-06-02	Stra			lønn	-12.06
2023-06-02	innbet Matas Thoen	433.21	2023-06-02	innbet Matas Thoen	433.21
2023-06-01	innbet Ali Ueland	336.75	2023-06-01	innbet Ali Ueland	336.75
2023-06-01	innbet Betina Petterson	42	2023-06-01	innbet Betina Petterson	42
2023-06-01	innbet Josef Kleiv	139.09	2023-06-01	innbet Josef Kleiv	139.09
2023-06-02	Bet Liliana Gylland AS	-455.74	2023-06-02	Bet Liliana Gylland AS	-455.74
2023-06-01	innbet Tine Olaussen	63.28	2023-06-01	innbet Tine Olaussen	63.28
2023-05-24	Bet Magne Furunes AS	-91.66	2023-05-24	Bet Magne Furunes AS	-91.66
2023-05-24	innbet Thomas Borgersen	26.59	2023-05-24	innbet Thomas Borgersen	26.59
2023-05-31	Straksbetaling lønn	-65.09	2023-05-31	Straksbetaling lønn	-65.09
2023-05-31	innbet Adna Akbari	20.19	2023-05-31	innbet Adna Akbari	20.19
2023-05-30	Straksbetaling lønn	1.66	2023-05-30	Straksbetaling lønn	1.66
2023-05-31	innbet Weronika Asp	150.73	2023-05-31	innbet Weronika Asp	150.73
2023-05-25	Straksbetaling lønn	-66.84	2023-05-25	Straksbetaling lønn	-66.84
2023-05-30	Straksbetaling lønn	15.59	2023-05-30	Straksbetaling lønn	15.59
2023-05-30	Straksbetaling lønn	17.57	2023-05-30	Straksbetaling lønn	17.57
2023-05-15	Privat uttak	-146.22	2023-05-15	Privat uttak	-146.22
2023-04-23	Straksbetaling lønn	15.8	2023-04-23	Straksbetaling lønn	15.8
2023-06-30	Bankgebyr	56.8	2023-06-30	Bankgebyr	56.8
2023-05-15	Bet Eldana Herfindal AS	-61.09	2023-05-15	Bet Eldana Herfindal AS	-61.09
2023-05-15	Bet Hanna Svenning AS	18.55	2023-05-15	Bet Hanna Svenning AS	18.55
2023-05-08	Delbetaling	192.68	2023-05-08	Delbetaling	192.68

Figure 36 - Verifying chosen period contains expected value



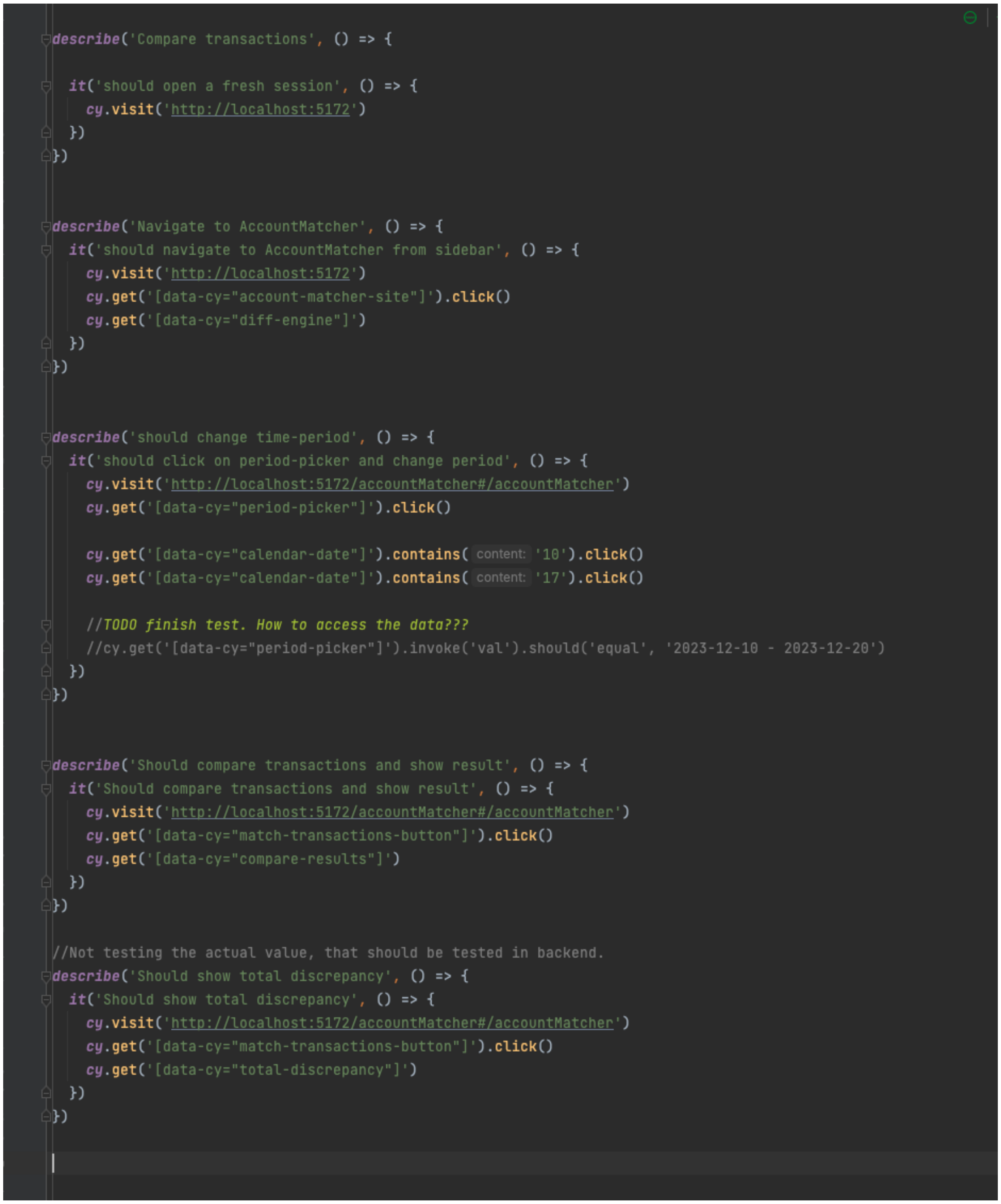
```
(xhr) ●GET 200
http://localhost:8080/transactions/bank
(xhr) ●GET 200
http://localhost:8080/transactions/accounting
(xhr) ●GET 200
http://localhost:8080/transactions/period_comparison
(xhr) ●GET 200
http://localhost:8080/period_comparison/results?startDate=2024-01-05&endDate=2024-01-12
(xhr) ●GET 200
http://localhost:8080/period_comparison/total_discrepancy?startDate=2024-01-05&endDate=2024-01-12
(xhr) ●GET 200
http://localhost:8080/period_comparison/total_amount_bank?startDate=2024-01-05&endDate=2024-01-12
(xhr) ●GET 200
http://localhost:8080/period_comparison/total_amount_accounting?startDate=2024-01-05&endDate=2024-01-12
get [data-cy="compare-
```

This is the overview:
Bank total: 0
Acc total: 0
Total discrepancy: 0

Accept reconcilliation?

✓ ✕

Figure 37 - Displaying alert box in Cypress



```
describe('Compare transactions', () => {

  it('should open a fresh session', () => {
    cy.visit('http://localhost:5172')
  })
})


describe('Navigate to AccountMatcher', () => {
  it('should navigate to AccountMatcher from sidebar', () => {
    cy.visit('http://localhost:5172')
    cy.get('[data-cy="account-matcher-site"]').click()
    cy.get('[data-cy="diff-engine"]')
  })
})


describe('should change time-period', () => {
  it('should click on period-picker and change period', () => {
    cy.visit('http://localhost:5172/accountMatcher#/accountMatcher')
    cy.get('[data-cy="period-picker"]').click()

    cy.get('[data-cy="calendar-date"]').contains( content: '10').click()
    cy.get('[data-cy="calendar-date"]').contains( content: '17').click()

    //TODO finish test. How to access the data???
    //cy.get('[data-cy="period-picker"]').invoke('val').should('equal', '2023-12-10 - 2023-12-20')
  })
})


describe('Should compare transactions and show result', () => {
  it('Should compare transactions and show result', () => {
    cy.visit('http://localhost:5172/accountMatcher#/accountMatcher')
    cy.get('[data-cy="match-transactions-button"]').click()
    cy.get('[data-cy="compare-results"]')
  })
})

//Not testing the actual value, that should be tested in backend.
describe('Should show total discrepancy', () => {
  it('Should show total discrepancy', () => {
    cy.visit('http://localhost:5172/accountMatcher#/accountMatcher')
    cy.get('[data-cy="match-transactions-button"]').click()
    cy.get('[data-cy="total-discrepancy"]')
  })
})
```

Figure 38 - Cypress tests

Figure 38 shows the test-code in Cypress.

4.4.3. Usability testing

As mentioned in chapter 3.5.3 the questions for both tests were both specific questions to the design and UX, and some more general and open questions.

This is the first version of the prototype:

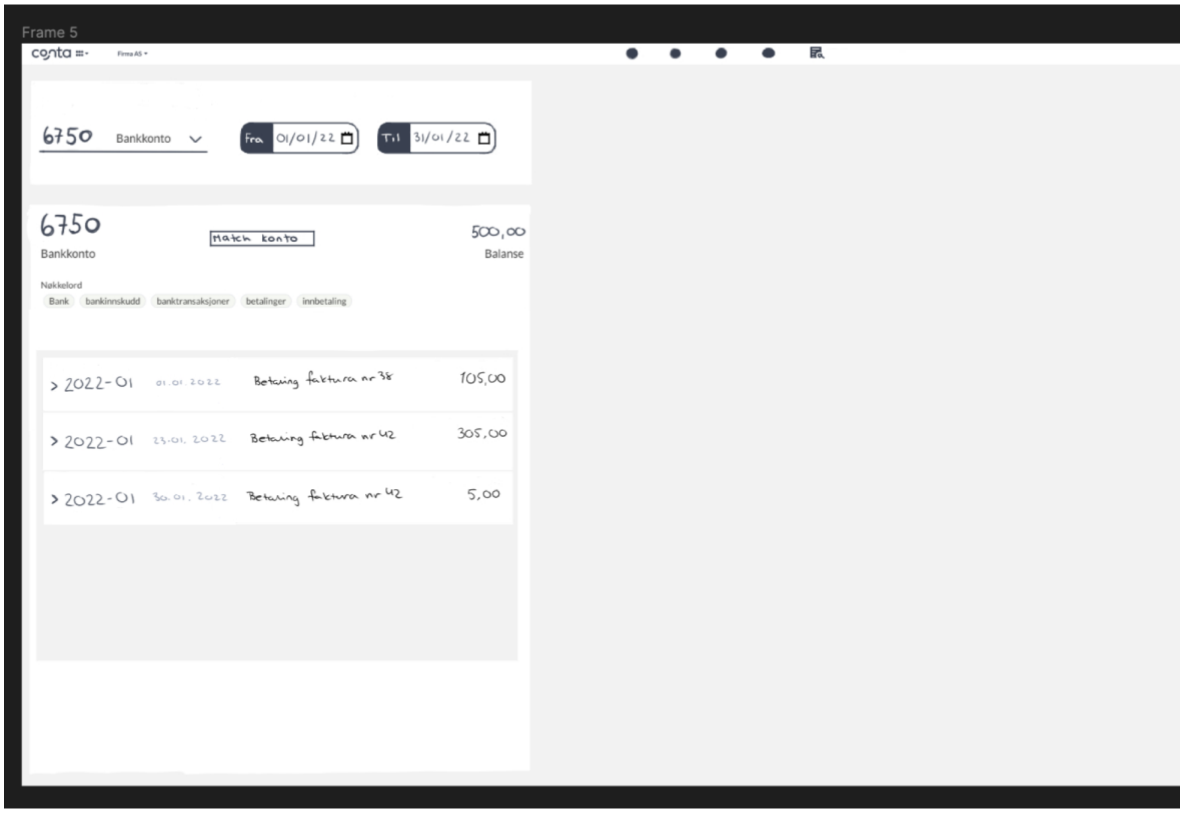


Figure 39 - first version of high-fidelity prototype ´

Results from first test-iteration:

- Changed colour and name to match-button, to make it more visible for the user. And the system is not matching accounts, its matching transactions, changed the name to match transactions/compare transactions.
- Removed functionality with accounts, only have transactions listed from accounting and bank.
- Added a homepage, page for compared periods and a page that does the comparing.

- Added a functionable sidebar.

This is how the prototype was modified based on the first user test:

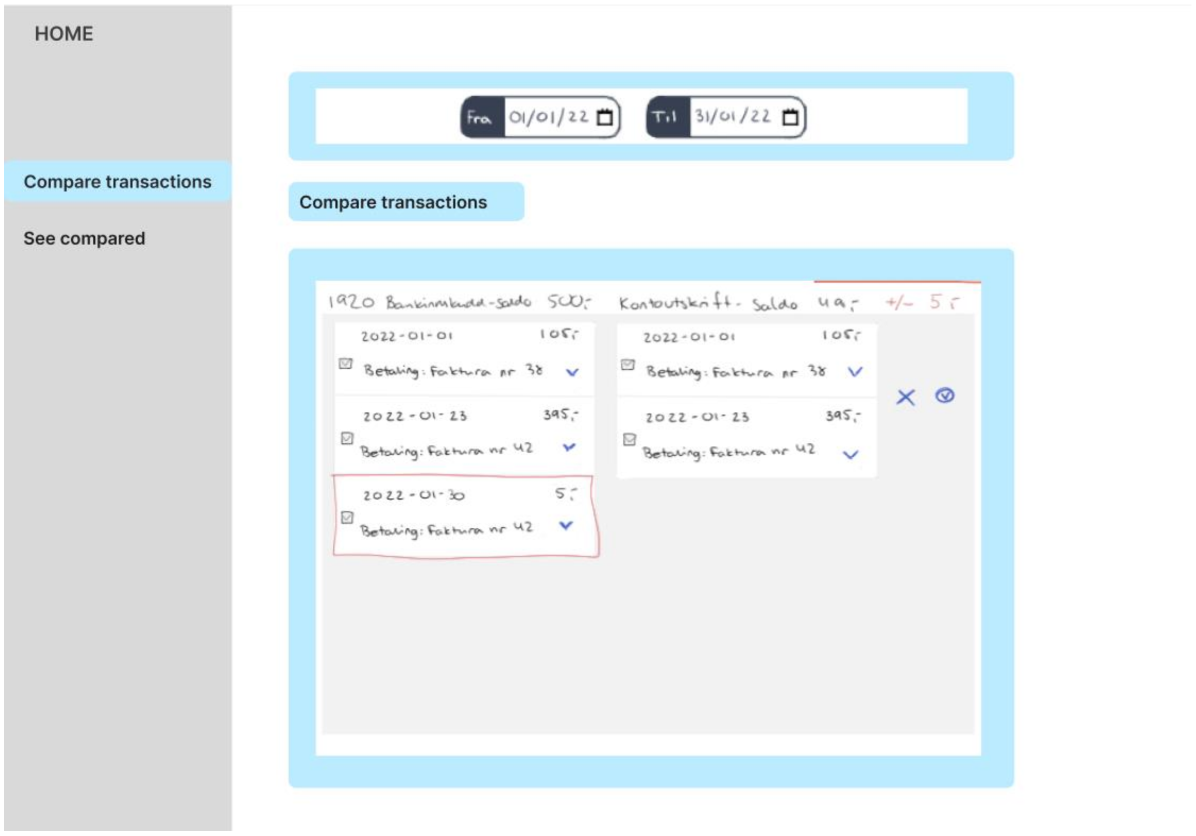
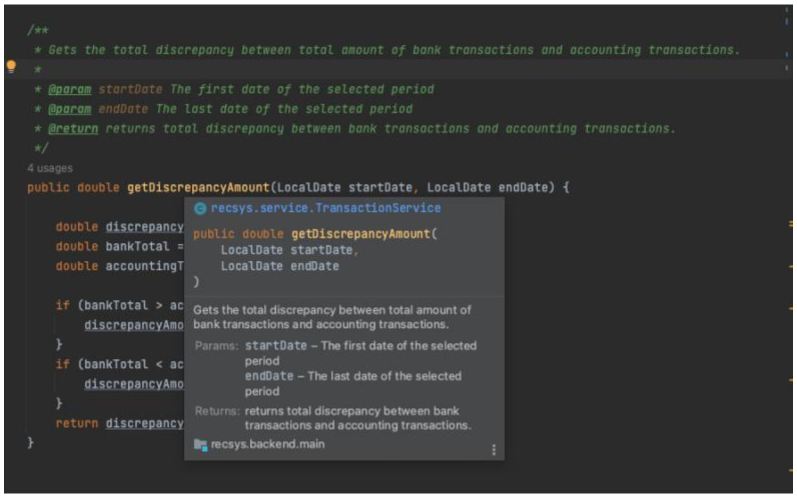


Figure 40 - Second iteration of high-fidelity prototype

Second iteration

- Included dialog that pops up when compared transactions is done if there is discrepancy, where you can accept or decline comparison.
- Navigate to saved when accepting.

4.4.4. Documentation



```
/**
 * Gets the total discrepancy between total amount of bank transactions and accounting transactions.
 *
 * @param startDate The first date of the selected period
 * @param endDate The last date of the selected period
 * @return returns total discrepancy between bank transactions and accounting transactions.
 */
4 usages
public double getDiscrepancyAmount(LocalDate startDate, LocalDate endDate) {

    double discrepancy
    double bankTotal =
    double accountingT

    if (bankTotal > ac
        discrepancyAmo
    }
    if (bankTotal < ac
        discrepancyAmo
    }
    return discrepancy
}
```

```
recsys.service.TransactionService
public double getDiscrepancyAmount(
    LocalDate startDate,
    LocalDate endDate
)
```

Gets the total discrepancy between total amount of bank transactions and accounting transactions.

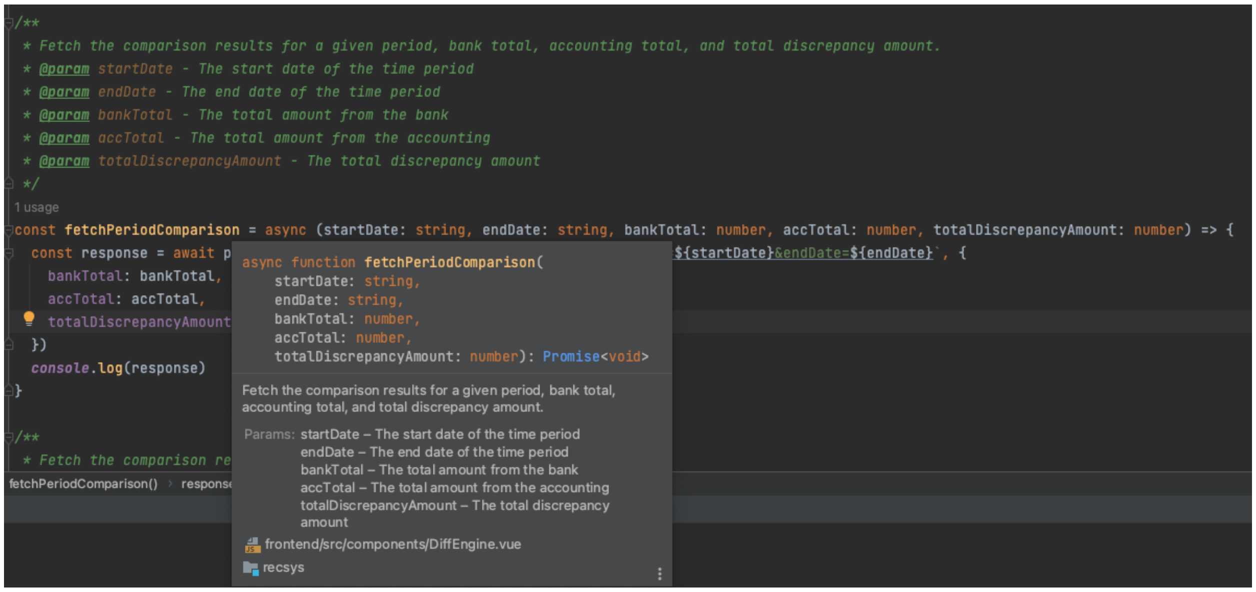
Params: startDate – The first date of the selected period
endDate – The last date of the selected period

Returns: returns total discrepancy between bank transactions and accounting transactions.

recsys.backend.main

Figure 41 - Example on code documentation Java

In this project methods in backend are documented with source code comments that follows the Java standard. See figure 41 for example.



```
/**
 * Fetch the comparison results for a given period, bank total, accounting total, and total discrepancy amount.
 * @param startDate - The start date of the time period
 * @param endDate - The end date of the time period
 * @param bankTotal - The total amount from the bank
 * @param accTotal - The total amount from the accounting
 * @param totalDiscrepancyAmount - The total discrepancy amount
 */
1 usage
const fetchPeriodComparison = async (startDate: string, endDate: string, bankTotal: number, accTotal: number, totalDiscrepancyAmount: number) => {
  const response = await p                                                    ${startDate}&endDate=${endDate}`, {
    bankTotal: bankTotal,
    accTotal: accTotal,
    totalDiscrepancyAmount
  })
  console.log(response)
}

/**
 * Fetch the comparison re
fetchPeriodComparison() › respons
```

```
async function fetchPeriodComparison(
    startDate: string,
    endDate: string,
    bankTotal: number,
    accTotal: number,
    totalDiscrepancyAmount: number): Promise<void>
```

Fetch the comparison results for a given period, bank total, accounting total, and total discrepancy amount.

Params: startDate – The start date of the time period
endDate – The end date of the time period
bankTotal – The total amount from the bank
accTotal – The total amount from the accounting
totalDiscrepancyAmount – The total discrepancy amount

frontend/src/components/DiffEngine.vue

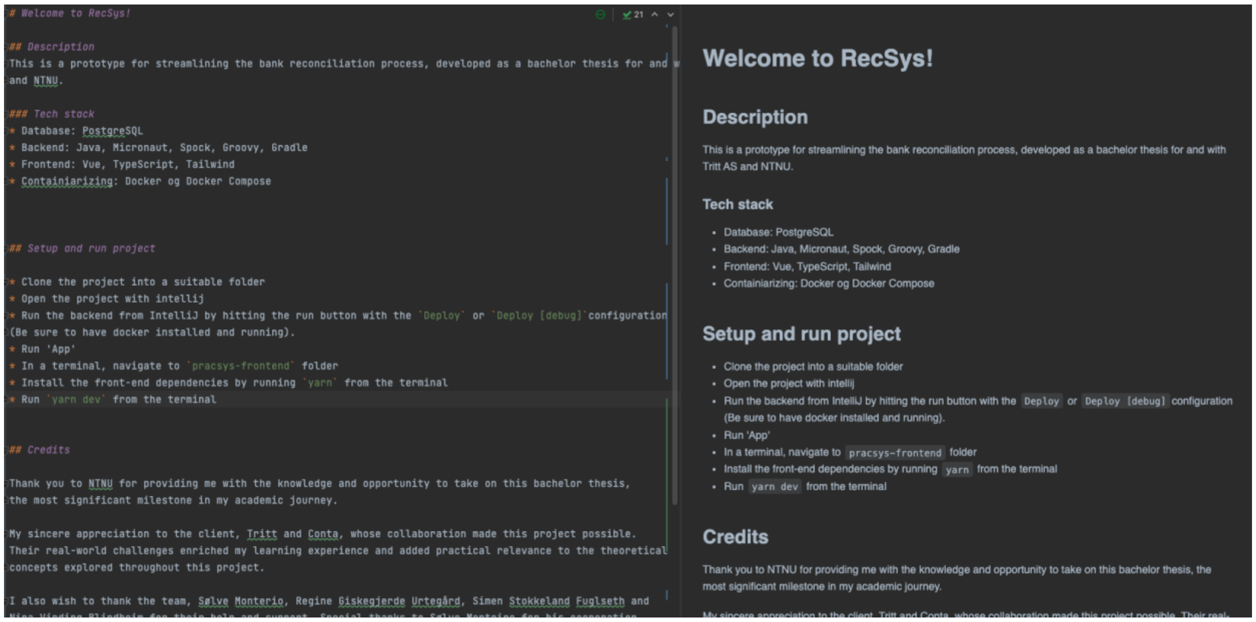
recsys

Figure 42 - Example on code documentation TypeScript

In frontend Typescript-methods are commented similar, using the JSDoc-standard, which is similar to Java (see figure 42).

4.4.4.1. README

The README-file provides information about running the application, using markup language. It is a guide for users or developers who interacts with the project.



```
# Welcome to RecSys!

## Description
This is a prototype for streamlining the bank reconciliation process, developed as a bachelor thesis for and w
and NTNU.

### Tech stack
* Database: PostgreSQL
* Backend: Java, Micronaut, Spock, Groovy, Gradle
* Frontend: Vue, TypeScript, Tailwind
* Containiarizing: Docker og Docker Compose



## Setup and run project

* Clone the project into a suitable folder
* Open the project with intellij
* Run the backend from IntelliJ by hitting the run button with the `Deploy` or `Deploy [debug]`configuration
(Be sure to have docker installed and running).
* Run 'App'
* In a terminal, navigate to `pracsys-frontend` folder
* Install the front-end dependencies by running `yarn` from the terminal
* Run `yarn dev` from the terminal


## Credits

Thank you to NTNU for providing me with the knowledge and opportunity to take on this bachelor thesis,
the most significant milestone in my academic journey.

My sincere appreciation to the client, Tritt and Conta, whose collaboration made this project possible.
Their real-world challenges enriched my learning experience and added practical relevance to the theoretical
concepts explored throughout this project.

I also wish to thank the team, Sølve Monterio, Regine Giskegjerde Urtegård, Simen Stokkeland Fuglseth and
```

Welcome to RecSys!

Description

This is a prototype for streamlining the bank reconciliation process, developed as a bachelor thesis for and with Tritt AS and NTNU.

Tech stack

- Database: PostgreSQL
- Backend: Java, Micronaut, Spock, Groovy, Gradle
- Frontend: Vue, TypeScript, Tailwind
- Containiarizing: Docker og Docker Compose

Setup and run project

- Clone the project into a suitable folder
- Open the project with intellij
- Run the backend from IntelliJ by hitting the run button with the `Deploy` or `Deploy [debug]` configuration (Be sure to have docker installed and running).
- Run 'App'
- In a terminal, navigate to `pracsys-frontend` folder
- Install the front-end dependencies by running `yarn` from the terminal
- Run `yarn dev` from the terminal

Credits

Thank you to NTNU for providing me with the knowledge and opportunity to take on this bachelor thesis, the most significant milestone in my academic journey.

Figure 43 - README

4.5. Administrative results and development prosess

4.5.1. Time-management

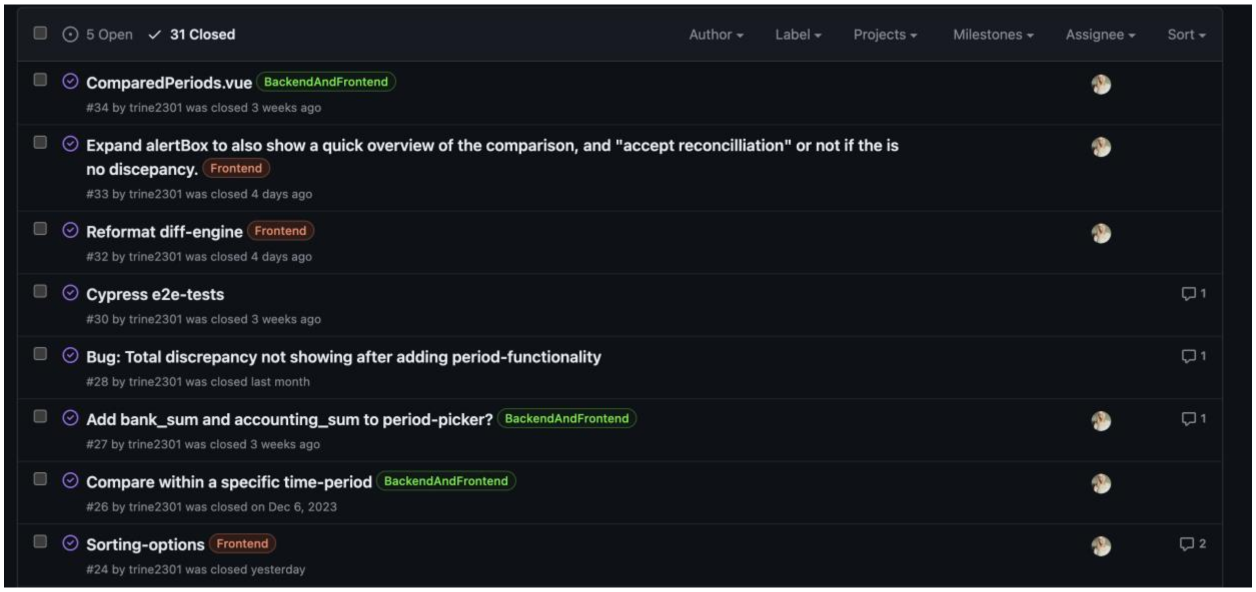
The project has been completed within the allocated time.

4.5.2. Meetings

Meetings have been consequent and frequent, and followed an agile standard. In the team we had Scrums, standups and retrospects, and together with client and supervisor I had sprint-retrospect every other week.

4.5.3. Project management

GitHub has been used as a tool in project management, specifically for sprint-tracking.



5 Open ✓ 31 Closed — Author Label Projects Milestones Assignee Sort

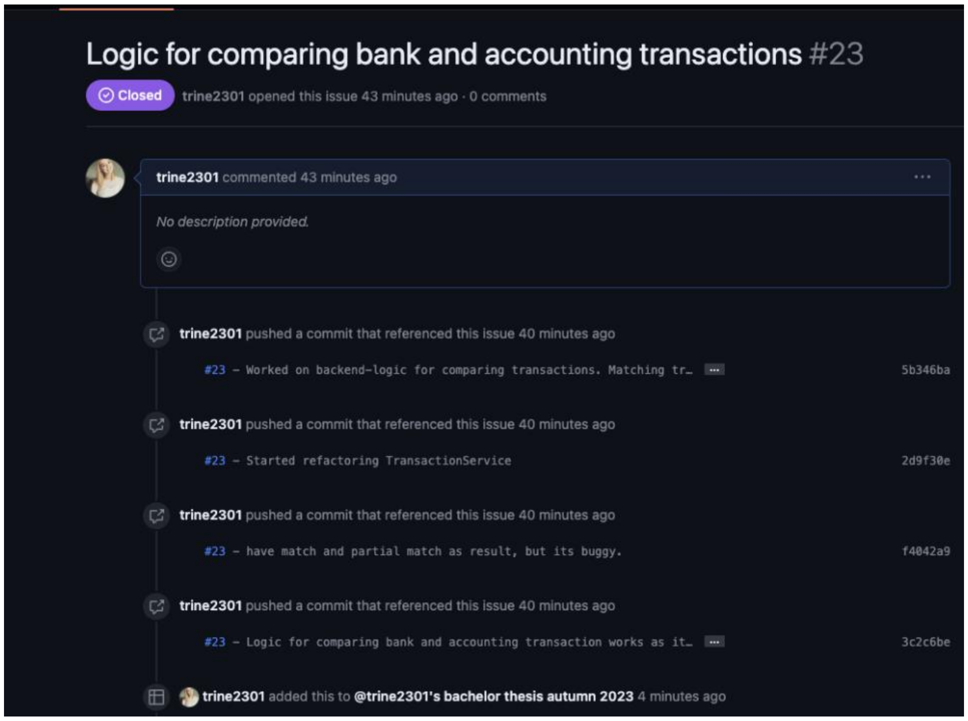
- ComparedPeriods.vue `BackendAndFrontend` — #34 by trine2301 was closed 3 weeks ago
- Expand alertBox to also show a quick overview of the comparison, and "accept reconcilliation" or not if the is no discepancy. `Frontend` — #33 by trine2301 was closed 4 days ago
- Reformat diff-engine `Frontend` — #32 by trine2301 was closed 4 days ago
- Cypress e2e-tests — #30 by trine2301 was closed 3 weeks ago — 1
- Bug: Total discrepancy not showing after adding period-functionality — #28 by trine2301 was closed last month — 1
- Add bank_sum and accounting_sum to period-picker? `BackendAndFrontend` — #27 by trine2301 was closed 3 weeks ago — 1
- Compare within a specific time-period `BackendAndFrontend` — #26 by trine2301 was closed on Dec 6, 2023
- Sorting-options `Frontend` — #24 by trine2301 was closed yesterday — 2

Figure 44 - issues

As seen in figure 44, each issue is linked to project, has a label and a sprint. This makes it easy to keep track of progress and keep a good overview over the project in general.

Upon committing, the commits have been linked to issues, by writing #<issuenumber> in the commit-messages. This has been done for a number of commits after implementing this method halfway through the project.

Started doing this throughout the project and found it to be very helpful. As shown in the figure 45 the issue has an issuenumber 23, and for each commit with the shown commit-message I could see in Github what had been done in that specific commit. This resulted in better overview and system of issues and commits that links to them.



Logic for comparing bank and accounting transactions #23

Closed trine2301 opened this issue 43 minutes ago · 0 comments

trine2301 commented 43 minutes ago

No description provided.

trine2301 pushed a commit that referenced this issue 40 minutes ago

#23 - Worked on backend-logic for comparing transactions. Matching tr… 5b346ba

trine2301 pushed a commit that referenced this issue 40 minutes ago

#23 - Started refactoring TransactionService 2d9f30e

trine2301 pushed a commit that referenced this issue 40 minutes ago

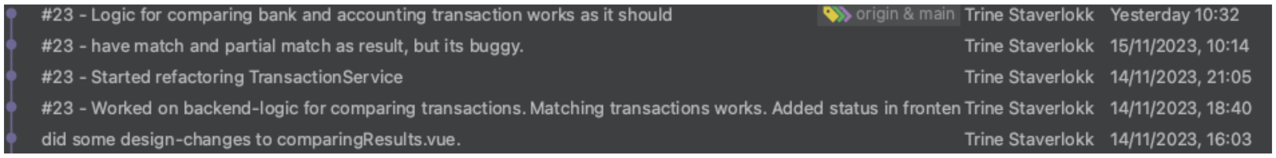
#23 - have match and partial match as result, but its buggy. f4042a9

trine2301 pushed a commit that referenced this issue 40 minutes ago

#23 - Logic for comparing bank and accounting transaction works as it… 3c2c6be

trine2301 added this to @trine2301's bachelor thesis autumn 2023 4 minutes ago

Figure 45 - Example of commit-indexing in GitHub



```
#23 - Logic for comparing bank and accounting transaction works as it should    origin & main  Trine Staverlokk  Yesterday 10:32
#23 - have match and partial match as result, but its buggy.                                   Trine Staverlokk  15/11/2023, 10:14
#23 - Started refactoring TransactionService                                                   Trine Staverlokk  14/11/2023, 21:05
#23 - Worked on backend-logic for comparing transactions. Matching transactions works. Added status in fronten Trine Staverlokk  14/11/2023, 18:40
did some design-changes to comparingResults.vue.                                               Trine Staverlokk  14/11/2023, 16:03
```

Figure 46 - Example of commits in Git

5 Discussion

The objective of this project was to develop a web-based prototype, as a POC of a streamlined bank reconciliation system, with basic functionality that is expandable in the future. As it is shown in chapter 4.0, Results, this has been achieved. This chapter will reflect upon and discuss the these results.

5.1. Theoretical discussion

This chapter will provide some discussion regarding the result of the feasability stage. In the theoretical part of the project I collected data regarding bank-reconcilliation and looked into exisiting solutions.

5.1.1. Methods for data collection

During the feasibility phase of the project, I used a combined practical- and theoretical approach to find results that would identify solutions for the problem defined.
Research was carried out through platforms such as Google Scholar, NTNU Open and the school library. The search terms and limitation set to 2019 yielded several relevant results. One can argue that there is relevant and still up-to-date research from before 2019, but since I got so many results, I consider more recent research to be preferable, especially the ones where technology was in focus.

In the search for existing solutions, websites with relevant features were examined for functionality. This gave me insight into the possibilities of bank reconciliation. Looking into existing solutions made me conclude that the feature the client has asked for was possible.

In addition to the theoretical approaches mentioned above, AI powered search engines were used for information retrieval. The use of AI as a search engine enabled fast navigation to relevant sources and therefore expanded the scope for this feasibility part of my project.

5.1.2. Result reflection

Research shows that there is a willingness to adapt to a more streamlined and possible automated way of doing accounting. Access to technology proves that streamlining accounting-prosesses are in prosess with multiple accounting businesses already.

With this I can conclude that there exists a promising opportunity to enhance efficiency and accuracy in bank reconciliations, meeting the evolving preferences of stakeholders for a more seamless and adaptive accounting framework.

5.2. Engineering discussion

This chapter will present the reflection and discussion around the finished product. This includes discussion regarding the design- and developing phases in SDLC.

5.2.1. Design

The design phase of this project has been invaluable for me as a student, where I learned a lot about communication with the client, understanding needs, using tools to show what I had i mind

5.2.1.1. Prototyping

The protyping has been extremely valuable for the prosess of developing the final product in terms of early user-testing and communication with the client. The high-fidelity prototype provided the client a sense of how the product would look and function, and it enabled good communication about their expectiation. The use of prototyping in the planning-phase proved highly beneficial, after user-study-testing (see appendix 2 and 3 for test results) where I was able to make asjustments to the product very early in the prosess.

5.2.1.2. Design material and methodology

The prosess of conceptualizing and planning the prototype for streamlining the bank reconciliation prosess was significantly enhanced using Figma for modelling and designing. Figma proved to be a a useful tool during the planning stage, offering an intuitive and collaborative environment that facilitated the visualization of elements for the finished product. This made it possible to share designs with the client regularly to get feedback consistently during the planning of the project. The platform's multifunctionality allowed for the creation of detailed prototypes, which was beneficial for planning the architecture and usability testing. Figma was overall a helpful tool in creating a system, following the MBSE-methodology.

To work based on a MBSE methodology provided a systematic approach to modelling the finished product. This helped ensure a comprehencive representation of the prototype structure and functionality early in the project and opened up for unambiguous communication between developer and client. The incorporation of Domain-Driven Design principles further refined the development prosess, aligning the prototype with the specific needs and intricacies of the bank reconciliation domain. By emphasizing a collaborative and iterative approach, DDD ensured that the final solution addressed not only the technical aspects but also the real-world challenges encountered in the bank reconciliation prosess.

5.2.2. Development

After the development phase the product is a working POC, with a robust, documented and tested code. This chapter will present the implementation of the diff-engine. This includes project structure, entities, example of classes and methods, discuss code quality, usability and testing.

5.2.2.1. Project structure

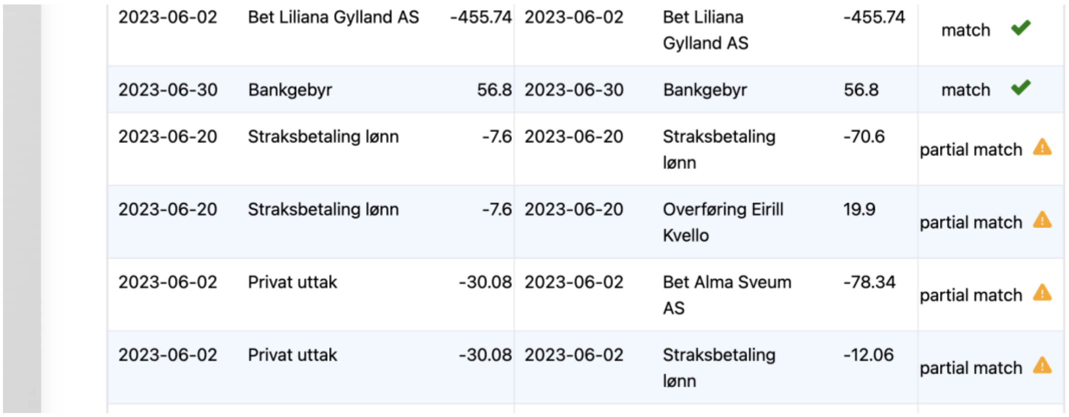
As seen in figure 11 (MVC Architecture in chapter 4.3.1) the hierarchy is divided into controllers, models, and views. This aligns with the vision to create a product following the MVC architecture to ensure a good architecture in the finished product. The use of MVC-pattern made it possible to create a product that is scalable and easy to debug, as the project structure was quite easy to navigate. MVC-architecture separates concerns, which allows for greater modularity and easier maintenance of the finished product, which the client appreciated. This architecture also made it easier to modify the product along the way, as a change in one section of the system did not affect the entire architecture. MVC also simplifies test-driven-development, which in retrospect would be a much better approach for testing (see chapter 5.2.6.1 for a reflection around TDD) (Interserver, 2023).

5.2.2.2. Comparison results explained

In "Diff-engine" the user can compare transaction within a desired time period.
The results are either match, partial match, missing bank transaction or missing accounting transaction.

- Match: when date and amount it the same.
- Partial-match: when date is the same, but amount is different.
- Missing bank transaction: when an accounting transaction cannot find a bank transaction with corresponding date or amount.
- Missing accounting transaction: when a bank transaction cannot find an accounting transaction with corresponding date or amount.

If there are several partial-matches, they are all listed (See figure 43), so the user can identify the most accurate match, and in the future edit transactions before accepting the reconciliation (This is not a part of this project scope, but this is part of the future features).



2023-06-02	Bet Liliana Gylland AS	-455.74	2023-06-02	Bet Liliana Gylland AS	-455.74	match ✓
2023-06-30	Bankgebyr	56.8	2023-06-30	Bankgebyr	56.8	match ✓
2023-06-20	Straksbetaling lønn	-7.6	2023-06-20	Straksbetaling lønn	-70.6	partial match ⚠
2023-06-20	Straksbetaling lønn	-7.6	2023-06-20	Overføring Eirill Kvello	19.9	partial match ⚠
2023-06-02	Privat uttak	-30.08	2023-06-02	Bet Alma Sveum AS	-78.34	partial match ⚠
2023-06-02	Privat uttak	-30.08	2023-06-02	Straksbetaling lønn	-12.06	partial match ⚠

Figure 47 - Several partial matches

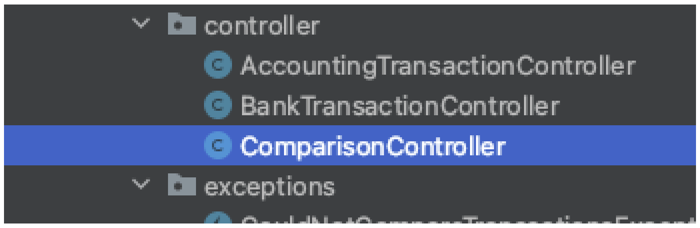
Description is a string, that describes the transaction. Should be fuzzy when comparing. When comparing two transactions the description uses a fuzzy-search library to search. That means that if there are some discrepancies it is not going to be a mismatch on the description alone. As the solution is today, the description is not considered.

5.2.2.3. Entities

Had two different entities for bank-transactions and accounting-transactions, even though they have almost identical fields. The reasoning for this is because this system represents a simplified version of a potential bigger system, which would have different fields, that would differentiate the entities in a more intuitive way. In addition, in accounting there exist strict rules when talking about modifying any persisted transaction. This means any accounting system needs to build strict protection regimes around protecting transactions from modification. This encourages a separate model for bank-entries that does not require the same restrictions.

5.2.2.4. Controllers

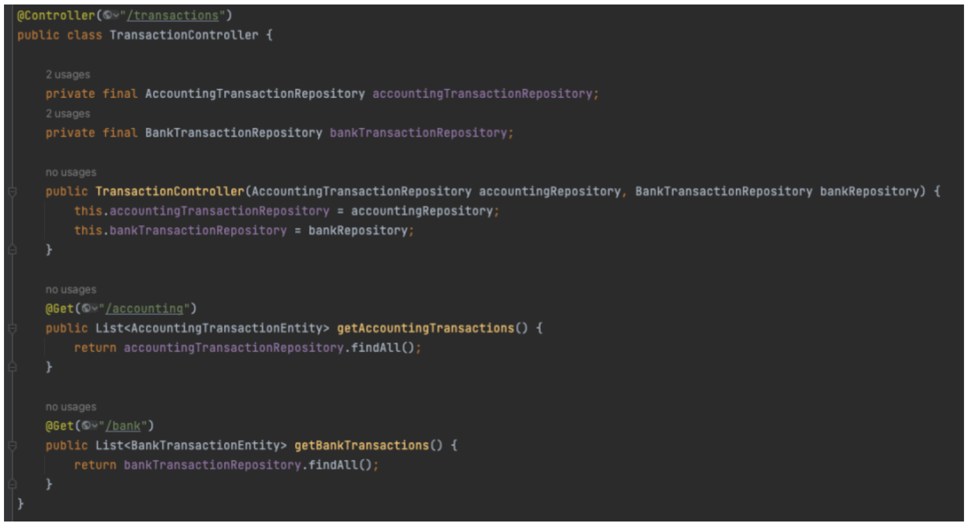
I chose initially to separate bank and accounting transactions into their own respective controller, even though they are reasonably alike. The reason for this was because it was thought to be easier to expand in the future if more complex transactions are added in the future. After a conversation with the client I learned that even though they could be more complex, the controllers would still be similar. This means that even though they are combined into one controller, the controller have a single purpose, and is therefore still a component of high cohesion when combined, and based on this it was worth considering merging them together. Figure 48 shows how the controllers initially was looking hierarchically.



```
controller
    AccountingTransactionController
    BankTransactionController
    ComparisonController
exceptions
```

Figure 48 - Initial controller structure

Figure 49 shows the code of the controller, showing separating them in the beginning was unnecessary.



```java
@Controller("/transactions")
public class TransactionController {

    2 usages
    private final AccountingTransactionRepository accountingTransactionRepository;
    2 usages
    private final BankTransactionRepository bankTransactionRepository;

    no usages
    public TransactionController(AccountingTransactionRepository accountingRepository, BankTransactionRepository bankRepository) {
        this.accountingTransactionRepository = accountingRepository;
        this.bankTransactionRepository = bankRepository;
    }

    no usages
    @Get("/accounting")
    public List<AccountingTransactionEntity> getAccountingTransactions() {
        return accountingTransactionRepository.findAll();
    }

    no usages
    @Get("/bank")
    public List<BankTransactionEntity> getBankTransactions() {
        return bankTransactionRepository.findAll();
    }
}
```

Figure 49 - Latest controller

The controller had initially the endpoints /accounting_transaction and /bank_transaction, so it is even more expandable with the endpoints /transaction/accounting and /transaction/bank.

This refactoring reduces the number of controllers from two to one, which can make the codebase easier to manage. However, it also means that the TransactionController class is now responsible for handling both types of transactions, which could make it more complex if more complex transactions are added in the future.

5.2.2.5. Views and components in frontend

In the frontend, there has been a real attempt at keeping components stupid, and having one responsibility each.

- AccountTransaction.ts – an AccountingTransactionType (id, date, amount and description). This class describes the shape of an object that represents an accounting transaction.
- api.service.ts – uses the axios library to provide HTTP GET and POST methods to the project.
- BankTransation.ts – a BankTransactionType (id, date, amount and description). This class describes the shape of an object that represents an accounting transaction.
- FrontendService.ts – contains methods to use in the components. Business logic.
- PeriodComparison.ts - This class describes the shape of an object that represents a period comparison.

Components serves a singular purpose, which is according to SOLID the first principle to make OOP designs more understandable, flexible, and maintainable. To partain to this principle, views' purpose is to display the components that the site is consisting of, and the components have one purpose each. Having views and components like this makes the site easier to expand in the future. One could argue that the diff-engine component has slightly too much responsibility, and could be divided into smaller components.
Initially, the majority of the frontend-logic was kept within the frontendService class, promoting a cleaner and more organized codebase. This proved to be challenging as the code grew more complex, and in the end much of the logic still resides inside its component.

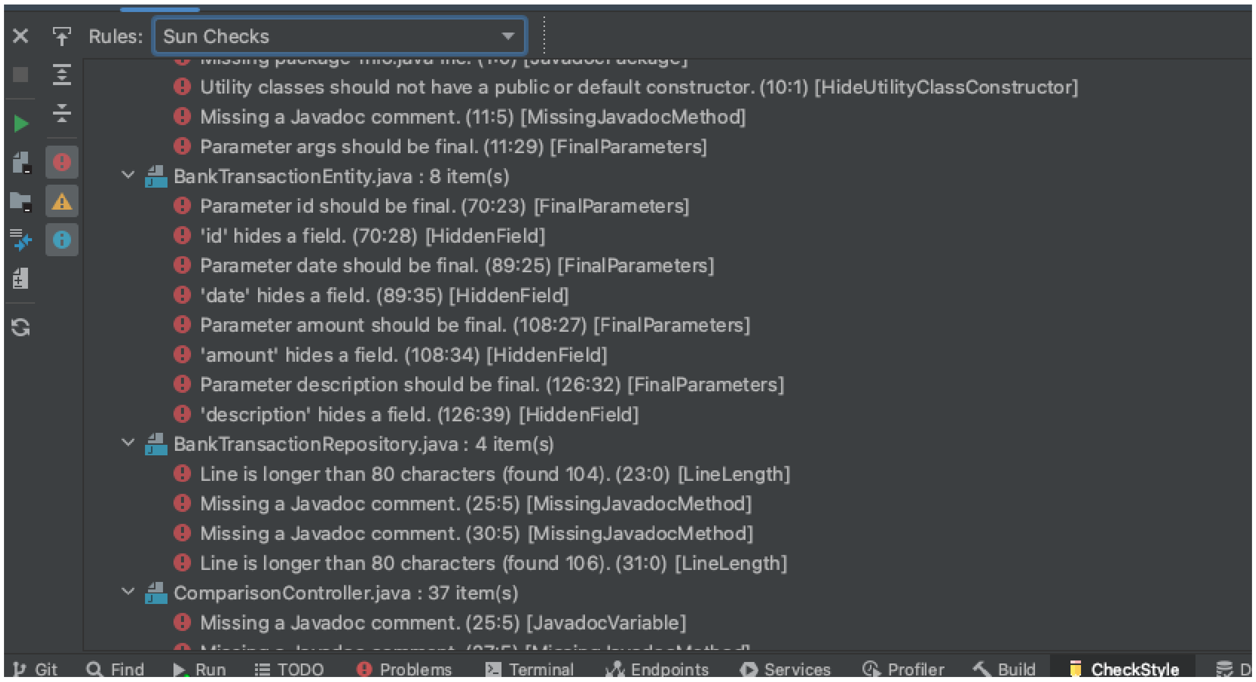
To make the code in frontend better, I would add some checks. Checks that probably should be added to period-cards: If the same period is already compared, should be a dialogue with the user that it is already saved. Ask to replace or abort. The date: should not exceeds todays date, and there should be some messages to the client if date is not set or is after today's date. Checks and comprehensive error-messages makes the whole site more robust, og easier to use for a user.

5.2.3. Code quality

Code of high quality refers to a systems robustness (system's ability to cope with errors (Wikipedia, 2024), reusability (the capability to re-use pre-exisiting code), reliability (the probability of the systm performing its intended functions without failure), and understandability (when a system allows an engineer to easily comprehend it, readability) (Wikipedia, 2024).

"Software code review, i.e., the practice of having other team members critique changes to a software system, is a well-established best practice in both open source and proprietary software domains." (McIntosh, Kamei, Adams, & Hassan, 2015)
Having the team around to do pair-coding and code review enhanced the code quality significant.

5.2.3.1. Checkstyle


Rules: Sun Checks

```
Utility classes should not have a public or default constructor. (10:1) [HideUtilityClassConstructor]
Missing a Javadoc comment. (11:5) [MissingJavadocMethod]
Parameter args should be final. (11:29) [FinalParameters]
BankTransactionEntity.java : 8 item(s)
    Parameter id should be final. (70:23) [FinalParameters]
    'id' hides a field. (70:28) [HiddenField]
    Parameter date should be final. (89:25) [FinalParameters]
    'date' hides a field. (89:35) [HiddenField]
    Parameter amount should be final. (108:27) [FinalParameters]
    'amount' hides a field. (108:34) [HiddenField]
    Parameter description should be final. (126:32) [FinalParameters]
    'description' hides a field. (126:39) [HiddenField]
BankTransactionRepository.java : 4 item(s)
    Line is longer than 80 characters (found 104). (23:0) [LineLength]
    Missing a Javadoc comment. (25:5) [MissingJavadocMethod]
    Missing a Javadoc comment. (30:5) [MissingJavadocMethod]
    Line is longer than 80 characters (found 106). (31:0) [LineLength]
ComparisonController.java : 37 item(s)
    Missing a Javadoc comment. (25:5) [JavadocVariable]
```

Figure 50 - Checkstyle example

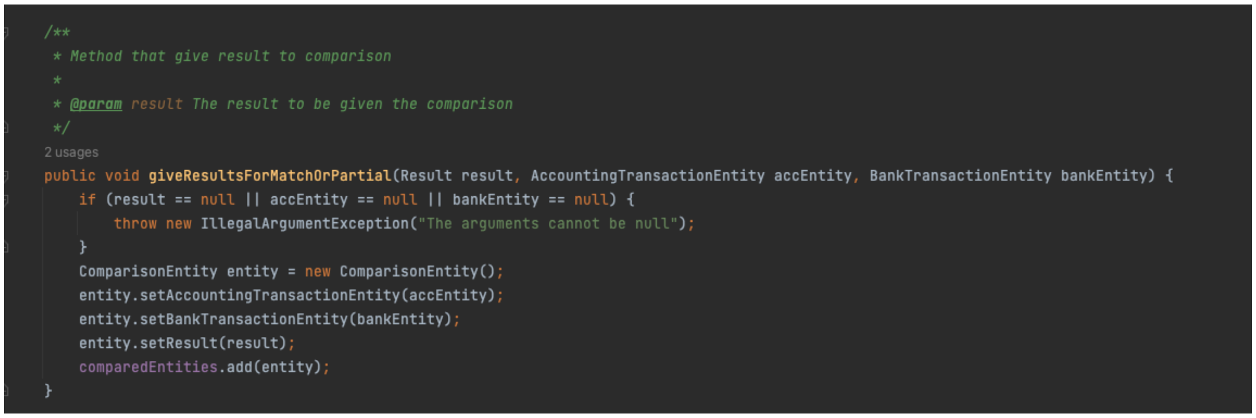
I did not consider all in sun checks or Google checks, but used the plugin to check that I documented all of the code and didn't have any big quality-issues. As seen in figure 50, it is easy to read the issues, and make a decision to either ignore it or go directly to the code and fix it.

5.2.3.2. SonarLint

Open-source IDE-extension that identifies and assists in fixing code quality issues as they are written. SonarLint focuses more on "code smells" and will underline problems in the IDE. In the project there is no SonarLint errors (SonarLint, u.d.).

5.2.3.3. Robustness

Exceptions and exception handling will handle errors at runtime (GeeksForGeeks, 2023). Having a good way of handling errors makes a code more robust.

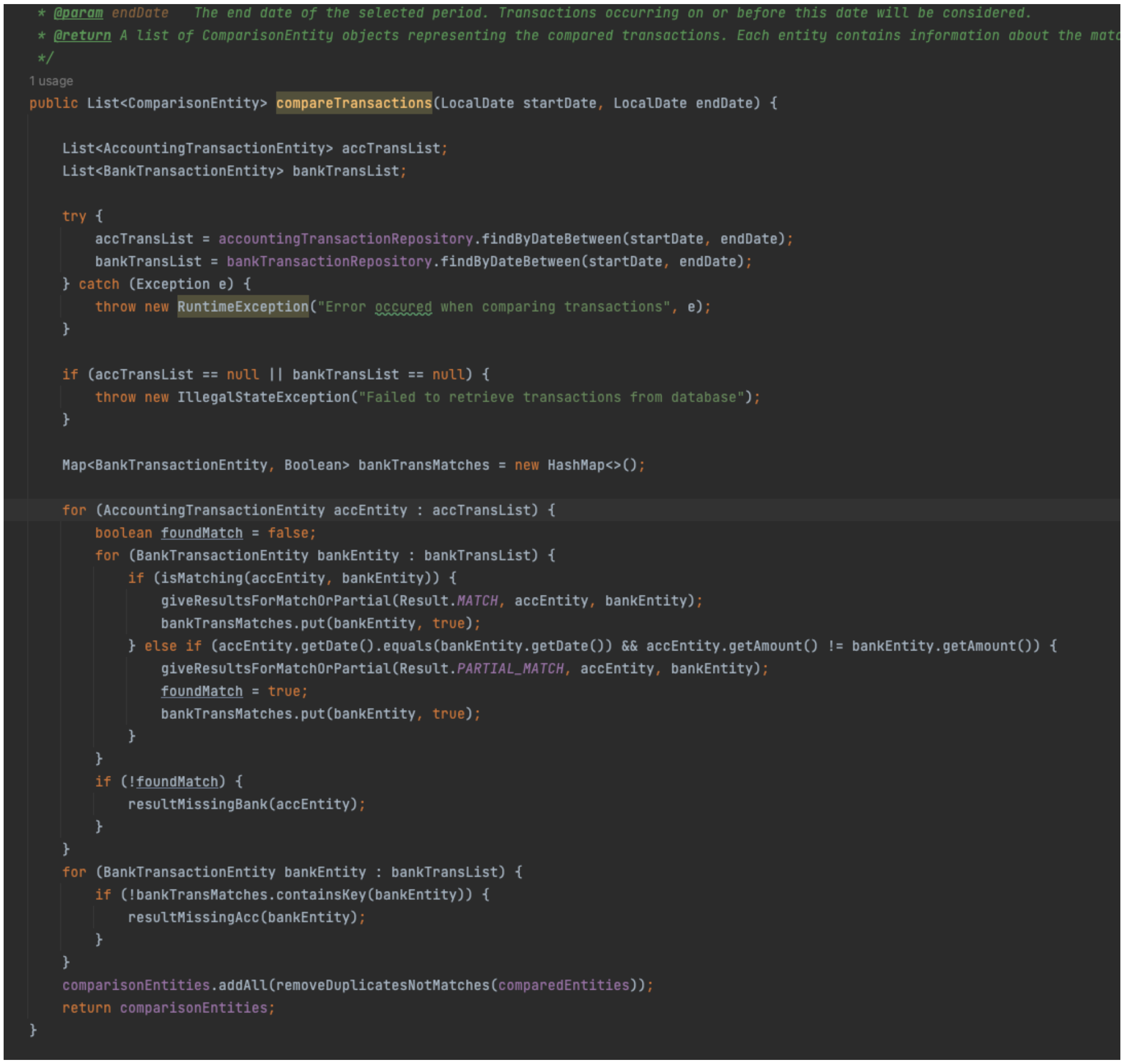


```java
/**
 * Method that give result to comparison
 *
 * @param result The result to be given the comparison
 */
2 usages
public void giveResultsForMatchOrPartial(Result result, AccountingTransactionEntity accEntity, BankTransactionEntity bankEntity) {
    if (result == null || accEntity == null || bankEntity == null) {
        throw new IllegalArgumentException("The arguments cannot be null");
    }
    ComparisonEntity entity = new ComparisonEntity();
    entity.setAccountingTransactionEntity(accEntity);
    entity.setBankTransactionEntity(bankEntity);
    entity.setResult(result);
    comparedEntities.add(entity);
}
```

Figure 51 - null-check example

In my project I have attempted to write robust code. A part of this is to focus on handling unexpected termination and unexpected actions. I have explored and attempted to adhere to the "Paranoia"-principle for writing robust code, which mean I assume the code may fail or work incorrectly, either because of my programming or because of the user will break the code (Wikipedia, 2024). To handle these possible fails, I have written code that throws exceptions and implemented try/catch-block with non-ambiguous error-messages. An example of this is from figure 51, where the code runs a try/catch sequence to check the parameters are not null, which would break the code.

5.2.3.4. Transaction Service-class



```
 * @param endDate   The end date of the selected period. Transactions occurring on or before this date will be considered.
 * @return A list of ComparisonEntity objects representing the compared transactions. Each entity contains information about the matc
 */
1 usage
public List<ComparisonEntity> compareTransactions(LocalDate startDate, LocalDate endDate) {

    List<AccountingTransactionEntity> accTransList;
    List<BankTransactionEntity> bankTransList;

    try {
        accTransList = accountingTransactionRepository.findByDateBetween(startDate, endDate);
        bankTransList = bankTransactionRepository.findByDateBetween(startDate, endDate);
    } catch (Exception e) {
        throw new RuntimeException("Error occured when comparing transactions", e);
    }

    if (accTransList == null || bankTransList == null) {
        throw new IllegalStateException("Failed to retrieve transactions from database");
    }

    Map<BankTransactionEntity, Boolean> bankTransMatches = new HashMap<>();

    for (AccountingTransactionEntity accEntity : accTransList) {
        boolean foundMatch = false;
        for (BankTransactionEntity bankEntity : bankTransList) {
            if (isMatching(accEntity, bankEntity)) {
                giveResultsForMatchOrPartial(Result.MATCH, accEntity, bankEntity);
                bankTransMatches.put(bankEntity, true);
            } else if (accEntity.getDate().equals(bankEntity.getDate()) && accEntity.getAmount() != bankEntity.getAmount()) {
                giveResultsForMatchOrPartial(Result.PARTIAL_MATCH, accEntity, bankEntity);
                foundMatch = true;
                bankTransMatches.put(bankEntity, true);
            }
        }
        if (!foundMatch) {
            resultMissingBank(accEntity);
        }
    }
    for (BankTransactionEntity bankEntity : bankTransList) {
        if (!bankTransMatches.containsKey(bankEntity)) {
            resultMissingAcc(bankEntity);
        }
    }
    comparisonEntities.addAll(removeDuplicatesNotMatches(comparedEntities));
    return comparisonEntities;
}
```

Figure 52 - compareTransactions-method

The compareTransaction method (As seen in figure 49) is the most important method in the whole system. This is where the comparison happens. The method takes in a startDate and an endDate as parameters and compares all transactions from their respective repositories in that given time-frame. The focus in this method have been to provide a robust, easy to understand and error-handling code.

This method was initially much bigger. When first written, it included the logic to give the results for all outcomes and remove duplicates that are not matches. This means a number if if-blocks for each result, which can be very hard to read. These if-statement blocks where extracted into smaller methods (see example in figure 53), which made this comparing-method much easier to read, understand and maintain. One could argue it could still be divided into smaller components, for example with smaller methods to find and compare matching transactions/partial matching transactions and missing transactions. These could then be used in the comparing matches method. This would make the method more readable, and even easier to expand in the future.

When it comes to error-handling, the try/catch block that will find an exception regarding the repositories of transactions at runtime if there are any and display an error-message. There will also be thrown an exception if accTransList or bankTransList is null.

To theoretically calculate performance, we need to look into several operations, including fetching transactions from the repositories, comparing transactions, and storing results. The time and space complexity of this method can be analysed based on these operations. The time complexity of fetching transactions from the repositories is likely to be O(N), where N is the number of transactions. This is because the findByDateBetween method iterates over all transactions to filter those within the specified date range. Next step is comparing the transactions. The nested loop structure used to do this has a time complexity of O(N^2), where N is the maximum number of transactions in either the accounting or bank transaction lists. For each accounting transaction, it checks against every bank transaction. After iterating over the transaction and comparing them, the method also stores the transactions into a list. Adding items to the comparedEntities list has a time complexity of O(1) per item.
So, considering these three operations, the overall time complexity of the compareTransactions method is O(N + N^2). However, in Big O notation, its normal to keep highest order term, so the time complexity would be O(N^2).

When calculating space complexity, we evaluate the same steps regarding storage. The space required to store the transactions fetched from the repositories is O(N), where N is the number of transactions. Storage of comparison results: The space required to store the comparison results is also O(N), where N is the number of compared transactions. So, the overall space complexity of the compareTransactions method is O(N + N) = O(2N). Again, in Big O notation, we keep the highest order term, so the space complexity would be O(N). Which means the space required grows linearly with the number of transactions.

Based on the analysis and calculations above of the performance, it is evident that the method could be improved when speaking of time-complexity (even though the complexity could vary, depending on different factors I did not consider). A time complexity of O(N^2) is not ideal when having higher number of transactions, which would be the case if the client wanted to implement this product into their existing software, where performance is critical, and the number of transactions can be quite large (Geeks for Geeks, 2024).

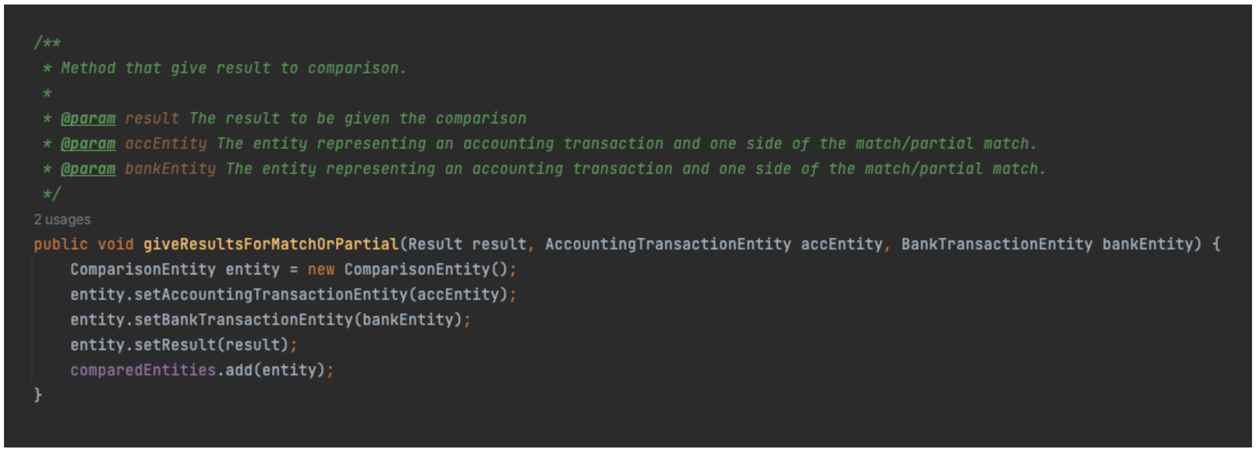

```java
/**
 * Method that give result to comparison.
 *
 * @param result The result to be given the comparison
 * @param accEntity The entity representing an accounting transaction and one side of the match/partial match.
 * @param bankEntity The entity representing an accounting transaction and one side of the match/partial match.
 */
2 usages
public void giveResultsForMatchOrPartial(Result result, AccountingTransactionEntity accEntity, BankTransactionEntity bankEntity) {
    ComparisonEntity entity = new ComparisonEntity();
    entity.setAccountingTransactionEntity(accEntity);
    entity.setBankTransactionEntity(bankEntity);
    entity.setResult(result);
    comparedEntities.add(entity);
}
```

Figure 53 - one of the extracted methods from compareTransaction-method

5.2.3.5. Bugsolving

There is one identified remaining bug, which is issue #29. When comparing transactions, a second time (same or different period), the previous comparison is not removed in the view. In this case, the total discrepancy and overview-amounts are not incorrect, and can be successfully saved, even though the view shows transactions from previous comparison. This is why this particular bug was handled using the Ostrich algorithm, which means to strategically ignore a problem based on the assumption that they may be exceedingly rare, or not worth the effort to fix (Baeldung, 2023). The assumption here is that when the prototype would be worked on further and implemented as a feature in the accounting software, this bug would be solved. Having this bug in the prototype will not corrupt the POC, and since it was discovered when it was, it will be left as it is.

5.2.4. Usability

Jacob Nielsen's usability heuristics for usability design are good rules of thumb when creating software and have been a receiving some emphasis during development of the finished product (Nielsen & Jacob, 2020).

The design consists of words, phrases, and concepts familiar to the user, ensuring similarity between the system and the real world. It has a logical order and follows real-world conventions. The system is also consistent, meaning same design throughout the whole product. I would also argue that the user has control and freedom when navigating the various views and components. It is easy for the user to backtrack and find whatever they are searching for. This is easy since the prototype is quite small still, but it is an important note still. The system also minimizes the user's memory load by making options and actions visible, and the aesthetics follows a minimalistic design.

Despite these considerations there are still some of the heretics that have not been receiving the same emphasis as the ones mentioned. To increase usability even more, there are several adjustments I would make. Firstly, I would make the system give more feedback to the user about what is happening. This would include a loading-icon for example. I would have included error-messages (as I previously mentioned), so the user always knows what went wrong if it did. Help and documentation could be improved by having hovering-effects with information and other helping on the site. There are, however, several of Nielsen's heuristics that have been achieved in the finished product.

Since this is a prototype, some of the heuristics are not as relevant. These includes the use of accelerators to speed up the interaction for the expert user. The product is too small for this to be a consideration in this version.

5.2.5. Accessibility

The finished product provides some support for users with dyslexia or poor vision. This involves the use of sufficient colour contrast throughout the product, a chosen font that is easy to see and read and the use of icons that is easily distinguishable and have a clear meaning.

This prototype will be used by professionals, so I have not considered the use of plain language to enhance understanding.

Plans for improvement includes implementing hovering and Helpers. I would also like to include tooltips or helper text to guide users through the interface as the prototype will expand in features. Another thing that would increase accessibility is key-Binding. Key binding can be useful for users who may have difficulty with traditional mouse-based navigation. Might also be a good idea to also have adjustable font sizes for future versions.

The finished product complies with recognized accessibility standards (WCAG) (Henry, 2023).

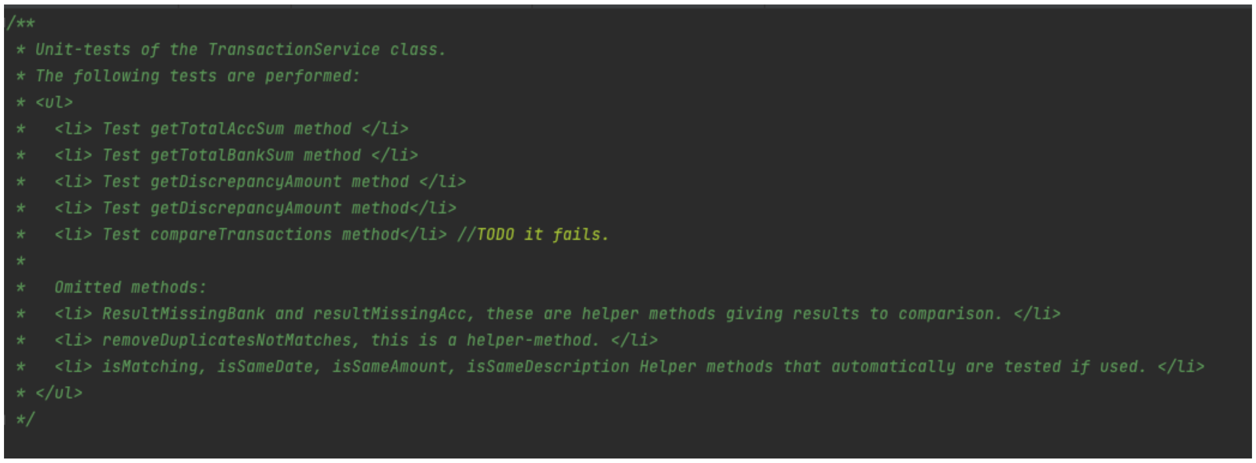
5.2.6. Software testing

This chapter will discuss the testing concluded in the project. Both E2E, unit-testing and user-testing.

5.2.6.1. Component testing in backend

This project have tested backend-code in Groovy, and frontend-code in Cypress. The methodology for Groovy-testing has consisted of a combination of strategies, some tests are written before the code, some during and some after. It has depended on the functionality and the complexity of the code. In retrospect I argue it would have been better to have one strategy during developing. And according to the science a test-first-approach increases productivity and minimizing error. Test-driven-development (TDD) have been well documented with a variety of positive outcomes. In retrospect I would argument that this approach would have suited my project well. Instead I went with a more flexible approach, where some tests were written before code, some during and some after. This also suited my project well, but having a more consistent flow.

The Testing-First approach, often associated with Test-Driven Development (TDD), is a methodology in software development where tests are written before the actual code. This approach is characterized by a cycle of writing a test that fails, making the test pass by writing the code, and then refactoring the code (Packt, 2023), (ISTQB Glossary, 2023).



```
/**
 * Unit-tests of the TransactionService class.
 * The following tests are performed:
 * <ul>
 *   <li> Test getTotalAccSum method </li>
 *   <li> Test getTotalBankSum method </li>
 *   <li> Test getDiscrepancyAmount method </li>
 *   <li> Test getDiscrepancyAmount method</li>
 *   <li> Test compareTransactions method</li> //TODO it fails.
 *
 *   Omitted methods:
 *   <li> ResultMissingBank and resultMissingAcc, these are helper methods giving results to comparison. </li>
 *   <li> removeDuplicatesNotMatches, this is a helper-method. </li>
 *   <li> isMatching, isSameDate, isSameAmount, isSameDescription Helper methods that automatically are tested if used. </li>
 * </ul>
 */
```

Figure 54 - Omitted methods from tests

Omitted methods are shown in figure 54. Some of these were omitted on the basis that they are "helper-methods", which means they are made to be used by another method for various reasons. If the parent-method is tested, the helper methods are also tested through that. In retrospect I see that this project would benefit from a bigger test-coverage, as the compareMethod-test fails, these are not tested at all.

5.2.6.2. E2E-testing in frontend

To setup test environment, Cypress templates and configurations from the client were used. Cypress-tests have been written after coding, as it proved to be the most straightforward prosess. The resulted in minimal time to finish it, and it is still not done.

The developer is pleased with the choice of E2E-testing framework and tool, as it proved to be simple to set up, straightforward to use with an intuitive UI, uncomplicated to debug, provided good feedbacks and easy navigation on failed tests. (Zanini, 2023), (TypeScript, 2023).

Looking back, I would start with E2E-testing sooner in the product development, rather than postponing it until the end of the project. I aim to achieve a more broader test coverage, and although I am generally satisfied with the E2E testing, I recognize the value of having a more extensive scope. Overall, testing in Cypress has been a good learning opportunity.

5.3. Administrative discussion

5.3.1. The team

It has been a pleasure to be working with a competent team of junior and senior developers when doing my bachelor thesis. Due to this I got hands-on, real-world experiences in a software company delivering services to the accounting industry. While I previously have been part of doing scrums in a classroom setting, it has been especially interesting to see it used in real world environment with stand-up meetings, mob-coding sessions and other agile developing methods.

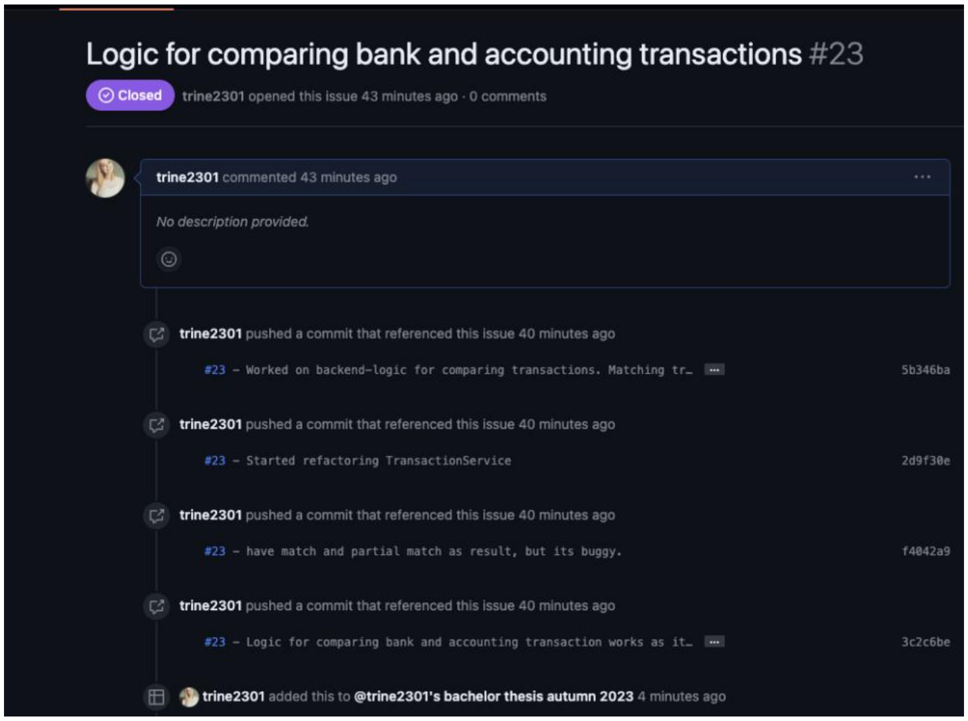
I experienced it was a good fit in what I have learned in school, and what is done in practice.

They have consistently provided me with a great learning environment, offering guidance and support to help with anything. Whether it was to uphold an agile and professional workflow by participating in scrums, stand-ups, or engaging in mob-coding sessions to solve some of the challenges I faced, their involvement has contributed not only to the tangible results I've achieved but also to an enriching learning journey within the field of system development.

5.3.2. Project management

5.3.2.1. Issue-tracking

As mentioned in chapter 4.5.3, the commits have been linked to issues. After I started doing this throughout the project I found it to be highly beneficial. As shown in the example below, the issue has an issuenumber 23, and for each commit with the shown commit-message I could see in Github what had been done in that specific commit. This resulted in better overview and system of issues and commits that links to them for easier navigation between commits and issues.



Logic for comparing bank and accounting transactions #23

Closed trine2301 opened this issue 43 minutes ago · 0 comments

trine2301 commented 43 minutes ago

No description provided.

trine2301 pushed a commit that referenced this issue 40 minutes ago

```
#23 - Worked on backend-logic for comparing transactions. Matching tr… 5b346ba
```

trine2301 pushed a commit that referenced this issue 40 minutes ago

```
#23 - Started refactoring TransactionService 2d9f30e
```

trine2301 pushed a commit that referenced this issue 40 minutes ago

```
#23 - have match and partial match as result, but its buggy. f4042a9
```

trine2301 pushed a commit that referenced this issue 40 minutes ago

```
#23 - Logic for comparing bank and accounting transaction works as it… 3c2c6be
```

trine2301 added this to @trine2301's bachelor thesis autumn 2023 4 minutes ago

Figure 55 - Example of commit-indexing in GitHub

5.3.3. Methodology/structuring development prosess

I have been privileged to have my team of engineers at work to help me follow a theoretical methodology for best result, which have been helpful. Being a solo-project did however introduce some challenges, despite the groups efforts to make at as professional and collaborative as possible. Following a certain methodology did prove to be difficult. There were days when the workload extended beyond working hours, resulting in varying levels of productivity over different periods.

Reflecting on the experience, had this project consisted of multiple members, the workflow could be more consequent. In addition, meetings, standups, scrums and other collaborative prosesses could have held more significant value than makeshift alternatives that we resorted to.

6 Conclusion

The client requirements were to develop a prototype as a POC for streamlining of the bank reconciliation prosess, using technology that makes merging with existing solutions easy. This chapter will conclude if this have been accomplished during developing the system. This chapter also addresses further work and community impact.

6.1. General conclusion

The developer is satisfied with having followed an agile methodology. This contributed with reaching the results that this project did. Both the theoretical- and engineering results were rewarding in terms of learning experience.

6.1.1. Scientific conclusion

The scientific results show that their technology available and a willingness to adapt to a more streamlined prosess of reconciliation. Results also show that some accounting programs already provides this functionality, proving it is a possibility for Tritt as well.

6.1.2. Engineering conclusion

This web based POC of a diff-engine for streamlined bank reconciliation meets requirements set by both the client and developer. The system is robust, well documented, accessible, and highly expendable, which have been a big fucus. However, it still has bugs, and there is room for improvement in terms of making the system more reliable, fast and fault tolerant. The realization that aiming for a higher test coverage could elevate the project results.

6.1.3. Administrative conclusion

The project was successfully finished within the allocated time. Using GitHub for tracking issues have been beneficial. Meetings have been productive and pleasant, and working solo have been a great learning experience. Especially lucky have I been to be able to follow an agile workflow thanks to the collaborative efforts of the team.

6.2. Community impact

I envision that this finished system can be leveraged further to integrate with features and functionalities in Tritt as a functionality Tritt can offer its clients.

The essence of project relates to streamlining and optimize manual work-prosesses. This means that upon successfully integrated with the accounting system that Tritt offers clients, workload will be lessened for accountants. In addition, errors will decrease, which is beneficial for both accountant that bookkeep transaction and their customers.

6.3. Further work

In order to enhance the system further, and making it ready to integrate with the exisiting Accounting software at tritt I aim to

- Fix failing tests, and add a more comprehensive test coverage
- As the system increases in complexity, user base and size, the need to expand on accessibility-features will be greater. This includes for example Documentation for Accessibility Features: Consider providing documentation or tooltips within the system that explain the accessibility features available. This helps users, including those with disabilities, to understand and utilize the functionalities. ADD Key-Binding: key-bindings will probably be essential functions, as this is already a big part of the solution that Tritt offers its future clients. This could be useful for users who may have difficulty with traditional mouse-based navigation.
- Enhancing speed, robustness and general code quality

Works Cited

CFI Team. (2015). *Bank Reconciliation*. Retrieved 04.09.23 from corporatefinanceinstitute: https://corporatefinanceinstitute.com/resources/accounting/bank-reconciliation/

Freshbooks. (2023). *How to Do Bank Reconciliation*. Retrieved 04.09.23 from freshbooks: https://www.freshbooks.com/hub/accounting/do-bank-reconciliation

QuickBooks. (2022). *Bank Reconciliation: Purpose, Example, and Prosess*. Retrieved 04.09.23 from QuickBooks: https://quickbooks.intuit.com/global/resources/financial-reports/bank-reconciliation-prosess/

TypeScript. (2020). *Cypress*. Retrieved 10.11.23 from TypeScript Deep Dive: https://basarat.gitbook.io/typescript/intro-1/cypress

FCA. (2007). *FCA Essential Practices for Information Technology Examinaton Manual IT Section.* Retrieved 10.11.23 from FCA: https://www.fca.gov/template-fca/download/ITManual/itsystemsdevelopment.pdf

Wikipedia, the free encyclopedia. (2023). *Software testing*. Retrieved 07.09.23 from Wikipedia: https://en.wikipedia.org/wiki/Software_testing

Computer Hope. (2022). *System development*. Retrieved 07.09.23 from Computer Hope: https://www.computerhope.com/jargon/s/systdeve.htm

Michigan Tech. (2016). *System Development Lifecycle (SDLC)*. Retrieved 07.09.23 from mtu: https://www.mtu.edu/it/security/policies-procedures-guidelines/information-security-program/system-development-lifecycle/

Wikipedia, the free encyclopedia. (2023). *Web Content Accessibility Guidelines*. Retrieved 10.11.23 from Wikipedia: https://en.wikipedia.org/wiki/Web_Content_Accessibility_Guidelines

Agile Alliance. (2022). *What is Agile Software Development?* Retrieved 07.09.23 from Agile Alliance: https://www.agilealliance.org/agile101/

Wikipedia, the free encyclopedia. (2023). *Software prototyping*. Retrieved 07.09.23 from Wikipedia: https://en.wikipedia.org/wiki/Software_prototyping

ISTQB Glossary. (2018). *Test-First Approach*. Retrieved 10.11.23 from ISTQB Glossary page: https://istqb-glossary.page/test-first-approach/

UXPin. (2020). *High-Fidelity Prototyping vs Low-Fidelity Prototypes: Which to Choose When?* Retrieved 07.09.23 from uxpin: https://www.uxpin.com/studio/blog/high-fidelity-prototyping-low-fidelity-difference/

Wikipedia, the free encyclopedia. (2023). *Model-based systems engineering*. Retrieved 07.09.23 from Wikipedia: https://en.wikipedia.org/wiki/Model-based_systems_engineering

Interserver. (2022). *What is MVC? Advantages and Disadvantages of MVC*. Retrieved 07.09.23 from InterServer: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/

Geeks for Geeks. (n.d.). *User Interface Design – Software Engineering*. Retrieved 10.12.23 from Geeks for Geeks: https://www.geeksforgeeks.org/software-engineering-user-interface-design/

Centiga. (n.d.). *Automatisk bankavstemming*. Retrieved 04.09.23 from Centiga: https://centiga.no/bankavstemming/

eAccounting. (n.d.). *Noen av funksjonene som gjør regnskapet enkelt*. Retrieved 04.09.23 from eAccounting: https://www.eaccounting.no/funksjoner/

Wikipedia, the free encyclopedia. (2021). *Universal design*. Retrieved 02.10.23 from Wikipedia: https://en.wikipedia.org/w/index.php?title=Universal_design&oldid=1017733026

STICOS. (n.d.). *Det eneste regnskapsfører trenger for effektiv oppdragsstyring og kvalitetssikring*. Retrieved from Retrieved 04.09.23: https://www.sticos.no/produkter/sticos-oversikt?utm_term=automatisk%20avstemming&utm_campaign=05+%C3%98konomi&utm_source=adwords&utm_medium=ppc&hsa_acc=4724287588&hsa_cam=6468739083&hsa_grp=92509880810&hsa_ad=421855298318&hsa_src=g&hsa_tgt=kwd-16499479796

Wikipedia, the free encyclopedia. (2022). *Model–view–controller*. Retrieved 07.09.23 from Wikipedia: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Fiken. (n.d.). *Bankavstemming*. Retrieved 04.09.23 from Fiken: https://hjelp.fiken.no/bankavstemming?gclid=CjwKCAjwjaWoBhAmEiwAXz8DBYHF3_ngbCyHq9fjJ93VYg9EQzEGg1acmfhyHSWOmA5h9oOj8o3P3BoCXy4QAvD_BwE

Tripletex. (n.d.). *Hvordan fungerer automatisk bankavstemming med bankintegrasjon?* Retrieved 04.09.23 from Tripletex: https://hjelp.tripletex.no/hc/no/articles/4416268743185-Hvordan-fungerer-automatisk-bankavstemming-med-bankintegrasjon-

Codegrip. (n.d.). *What is Code Quality & why is it Important?* Retrieved 10.11.23 from Codegrip: https://www.codegrip.tech/productivity/what-is-code-quality-why-is-it-important/

SonarSource. (n.d.). *Catching Issues in the IDE with SonarLint*. Retrieved 10.11.23 from SonarCloud: https://docs.sonarsource.com/sonarcloud/

checkstyle. (2023). *checkstyle*. Retrieved 10.11.23 from checkstyle: https://checkstyle.org

Wikipedia, the free encyclopedia. (2023). *SOLID*. Retrieved 07.09.23 from Wikipedia: https://en.wikipedia.org/wiki/SOLID

Wikipedia, the free encyclopedia. (2023). *Cohesion (computer science)*. Retrieved 15.10.23 from Wikipedia: https://en.wikipedia.org/wiki/Cohesion_(computer_science)

Indeed Editorial Team. (2023). *Code Quality: What It Is and How To Measure It (With Tips)*. Retrieved 15.10.23 from Indeed: https://www.indeed.com/career-advice/career-development/what-is-code-quality

Wikipedia, the free encyclopedia. (2023). *Robustness (computer science)*. Retrieved 15.10.23 from Wikipedia: https://en.wikipedia.org/wiki/Robustness_(computer_science)

CodiumAI Team. (2023). *Mock Testing: Understanding the Benefits and Best Practices*. Retrieved 10.11.23 from Codium: https://www.codium.ai/blog/mock-testing/

GeeksforGeeks. (n.d.). *Agile Software Testing*. Retrieved 10.11.23 from GeeksforGeeks: https://www.geeksforgeeks.org/agile-software-testing/

Micronaut. (2023). *A MODERN, JVM-BASED, FULL-STACK FRAMEWORK FOR BUILDING MODULAR, EASILY TESTABLE MICROSERVICE AND SERVERLESS APPLICATIONS*. Retrieved from Micronaut: https://micronaut.io

GeeksforGeeks. (n.d.). *Usability Testing*. Retrieved 10.11.23 from GeeksforGeeks: https://www.geeksforgeeks.org/usability-testing/

Chandan. (2023). *Micronaut Beans*. Retrieved 15.11.23 from Medium: https://medium.com/@chandanjena706/beans-dbe536910685

Wikipedia, the free encyclopedia. (2023). *Gradle*. Retrieved 10.12.23 from Wikipedia: https://en.wikipedia.org/wiki/Gradle

PostgreSQL. (n.d.). *PostgreSQL: The World's Most Advanced Open Source Relational Database*. Retrieved 07.09.23 from PostgreSQL: https://www.postgresql.org

Apache Groovy. (n.d.). *Apache Groovy*. Retrieved 10.11.23 from Apache Groovy: https://groovy-lang.org/testing.html

Cypress. (2023). *Introduction to Cypress*. Retrieved 10.11.23 from Cypress: https://docs.cypress.io/guides/core-concepts/introduction-to-cypress

Baeldung. (2023). *Difference Between Docker Images and Containers*. Retrieved 07.09.23 from Baeldung: https://www.baeldung.com/ops/docker-images-vs-containers

Docker. (2023). *What is a container?* Retrieved 07.09.23 from Docker: https://docs.docker.com/guides/walkthroughs/what-is-a-container/

Wikipedia (2023). *Version control*. Retrieved 15.11.23 from Wikipedia: https://en.wikipedia.org/wiki/Version_control

Atlassian. (n.d.). *What is version control?* Retrieved 15.11.23 from Atlassian: https://www.atlassian.com/git/tutorials/what-is-version-control

Cabot, J. (2017). *Comparing Domain-Driven Design with Model-Driven Engineering*. Retrieved 15.09.23 from modeling-languages: https://modeling-languages.com/comparing-domain-driven-design-model-driven-engineering/

Zanini, A. (2023). *How to Set Up a Cypress TypeScript Project*. Retrieved 10.11.23 from codemotion: https://www.codemotion.com/magazine/frontend/web-developer/how-to-set-up-a-cypress-typescript-project/

Alam, A. (2019). *Importance of Software Design*. Retrieved 07.09.23 from Medium: https://medium.com/swlh/importance-of-software-design-7ffea48ede17

Bhatt, T. (2023). *Software Development Technologies: A Brief Explanation With Examples*. Retrieved 07.09.23 from Intelivita: https://www.intelivita.com/blog/software-development-technologies/

Keith Armstrong, A. H. (2023). *Random user generator*. Retrieved from Randomuser.me: https://randomuser.me

Yasar, K. (2022). *software testing*. Retrieved 10.11.23 from TechTarget: https://www.techtarget.com/whatis/definition/software-testing

Moradov, O. (2023). *Unit Testing: Definition, Examples, and Critical Best Practices*. Retrieved 10.11.23 from Brightsec: https://brightsec.com/blog/unit-testing/

Monocubed. (2021). *Advantages of Vue js*. Retrieved from What are the Advantages of Vue js Framework in Web Development?: https://www.monocubed.com/blog/advantages-of-vue-js/

Geeks for Geeks. (n.d.). *State Transition Testing*. Retrieved 10.11.23 from State Transition Testing: https://www.geeksforgeeks.org/state-transition-testing/

Maze. (n.d.). *Maze*. Retrieved 15.10.23 from 5 Real-life usability testing examples & approaches to apply: https://maze.co/guides/usability-testing/examples/

Wikipedia. (2023). *Robustness (computer science)*. Retrieved 15.10.23 from Robustness (computer science): https://en.wikipedia.org/wiki/Robustness_(computer_science)

McIntosh, S., Kamei, Y., Adams, B., & Hassan, E. A. (2015, April 25). An empirical study of the impact of modern code review practices on software quality. *Springer Link, 21*, 2146-2189. Retrieved from https://link.springer.com/article/10.1007/s10664-015-9381-9

SonarLint. (n.d.). *our IDE and programming language. covered*. Retrieved October 2023, from https://www.sonarsource.com/products/sonarlint/

GeeksForGeeks. (n.d.). *Exceptions in Java*. Retrieved 22.12.23 from https://www.geeksforgeeks.org/exceptions-in-java/

Wikipedia. (2023). *Robustness (computer science)*. Retrieved 15.10.23 from https://en.wikipedia.org/wiki/Robustness_(computer_science)

Geeks for Geeks. (n.d.). *Time Complexity and Space Complexity*. Retrieved 02.01.24 from https://www.geeksforgeeks.org/time-complexity-and-space-complexity/

Baeldung. (2023). Retrieved 26.12.23 from https://www.baeldung.com/cs/ostrich-algorithm

Penchikala, S. (2008). *Domain Driven Design and Development In Practice*. Retrieved October 2023, from InfoQ: https://www.infoq.com/articles/ddd-in-practice/

Stevens, E. (2022). *7 fundamental UX design principles all designers should know*. Retrieved 07.09.23, from UX Design Institute: https://www.uxdesigninstitute.com/blog/ux-design-principles/

Henry, S. L. (2023). *WCAG 2 Overview*. Retrieved October 2023, from w3c: https://www.w3.org/WAI/standards-guidelines/wcag/

Ramona Lacurezeanu, A. T.-T. (2020). *Automatizarea proceselor prin robotizare in audit si contabilitate.* Retrieved october 2023, from CEEOL: https://www.ceeol.com/search/article-detail?id=906637

McGrath, A., & Jonker, A. (2023). *IBM*. Retrieved December 2023, from What is model-based systems engineering (MBSE)?: https://www.ibm.com/topics/model-based-systems-engineering

Nielsen, & Jacob. (2020). *10 Usability Heuristics for User Interface Design*. Retrieved September 2023, from Nielsen Norman Group: https://www.nngroup.com/articles/ten-usability-heuristics/

Convedo. (2019). *The Benefits of Robotics in Financial Services*. Retrieved September 2023, from Convedo: https://info.convedo.com/the-benefits-of-robotics-in-financial-services

Wikipedia. (2023). *SOLID*. Retrieved October 2023, from Wikipedia: https://en.wikipedia.org/wiki/SOLID

Rahdan, A. (2020). *Jakob Nielsen's 10 heuristics for user interface design with practical examples*. Retrieved September 2023, from UX Design: https://uxdesign.cc/jakob-nielsens-10-heuristics-for-user-interface-design-3fe09af5fd99

Can Tansel Kaya, M. T. (2020). *RPA Teknolojilerinin Muhasebe Sistemleri Üzerindeki Etkisi.* Retrieved August 2023, from Researchgate: https://www.researchgate.net/publication/340367501_The_Future_of_Robotic_Prosess_Automation_RPA_in_the_Banking_Sector_for_Better_Customer_Experience

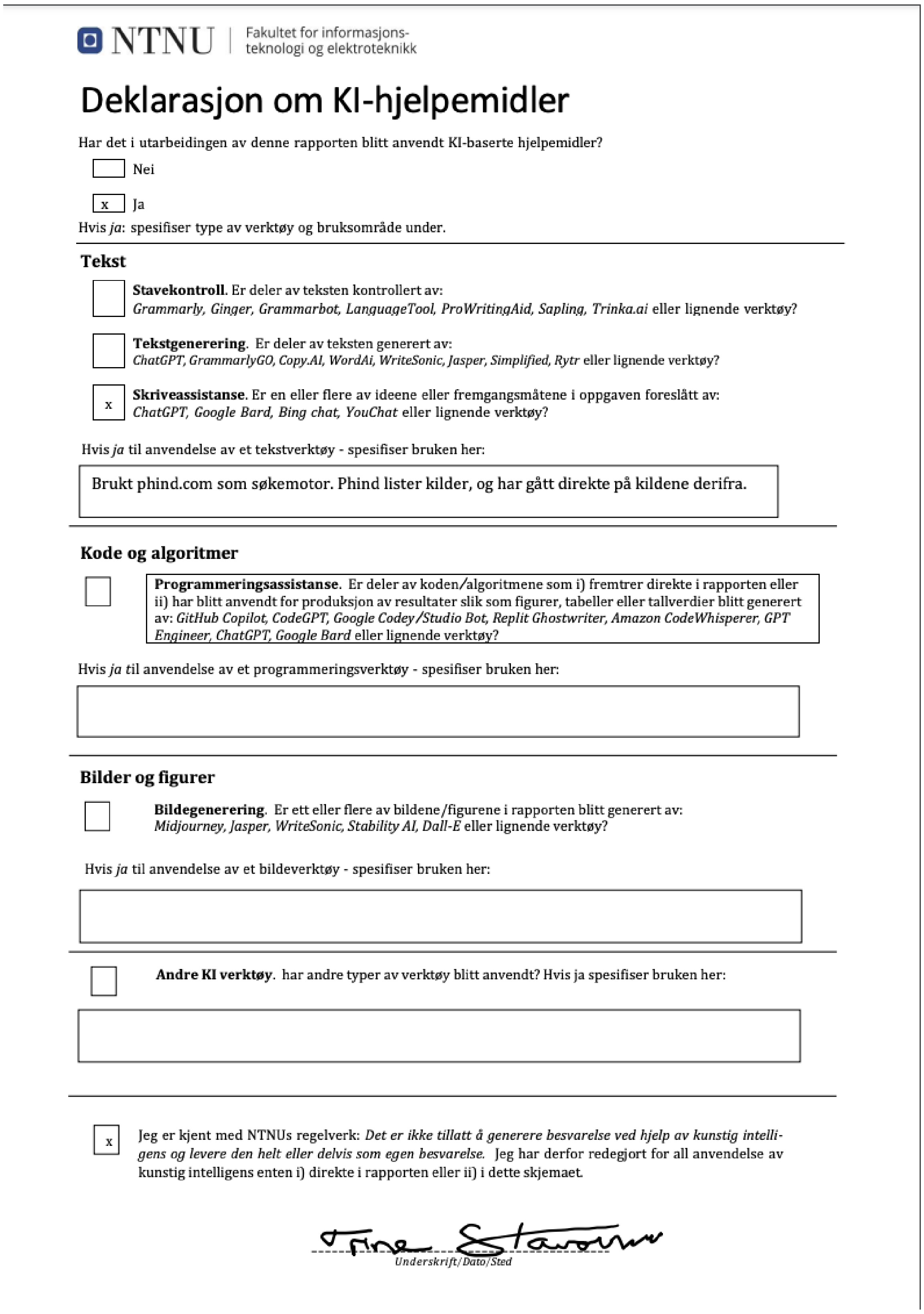
Knudsen-Baas, A. C. (2023). *Digitalisering i regnskapsavdelingen. En kvalitativ studie av et børstnotert selskap.* Retrieved August 2023, from UIA: https://uia.brage.unit.no/uia-xmlui/handle/11250/3083646

Simic, S. (2022). *Docker Image vs Container: The Major Differences*. Retrieved October 2023, from PhoenixNap: https://phoenixnap.com/kb/docker-image-vs-container

C. Vijai, S. S. (2020). *The Future of Robotic Prosess Automation (RPA) in the Banking Sector for Better Customer Experience.* Retrieved 04.09.23, from Researchgate: https://www.researchgate.net/profile/C-Vijai-2/publication/340367501_The_Future_of_Robotic_Prosess_Automation_RPA_in_the_Banking_Sector_for_Better_Customer_Experience/links/5ea93eba299bf18b9584643c/The-Future-of-Robotic-Prosess-Automation-RPA-in-the-Banki

Bhutada, T. (2023). *7 Real-time Use Cases of Groovy Scripting*. Retrieved November 2023, from Stackify: https://stackify.com/7-real-time-use-cases-of-groovy-scripting/

Appendix 1 – AI declaration



NTNU | Fakultet for informasjons- teknologi og elektroteknikk

Deklarasjon om KI-hjelpemidler

Har det i utarbeidingen av denne rapporten blitt anvendt KI-baserte hjelpemidler?

[ ] Nei

[x] Ja

Hvis *ja*: spesifiser type av verktøy og bruksområde under.

Tekst

[ ] **Stavekontroll**. Er deler av teksten kontrollert av: *Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinka.ai* eller lignende verktøy?

[ ] **Tekstgenerering**. Er deler av teksten generert av: *ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr* eller lignende verktøy?

[x] **Skriveassistanse**. Er en eller flere av ideene eller fremgangsmåtene i oppgaven foreslått av: *ChatGPT, Google Bard, Bing chat, YouChat* eller lignende verktøy?

Hvis *ja* til anvendelse av et tekstverktøy - spesifiser bruken her:

Brukt phind.com som søkemotor. Phind lister kilder, og har gått direkte på kildene derifra.

Kode og algoritmer

[ ] **Programmeringsassistanse**. Er deler av koden/algoritmene som i) fremtrer direkte i rapporten eller ii) har blitt anvendt for produksjon av resultater slik som figurer, tabeller eller tallverdier blitt generert av: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* eller lignende verktøy?

Hvis *ja* til anvendelse av et programmeringsverktøy - spesifiser bruken her:

Bilder og figurer

[ ] **Bildegenerering**. Er ett eller flere av bildene/figurene i rapporten blitt generert av: *Midjourney, Jasper, WriteSonic, Stability AI, Dall-E* eller lignende verktøy?

Hvis *ja* til anvendelse av et bildeverktøy - spesifiser bruken her:

[ ] **Andre KI verktøy**. har andre typer av verktøy blitt anvendt? Hvis ja spesifiser bruken her:

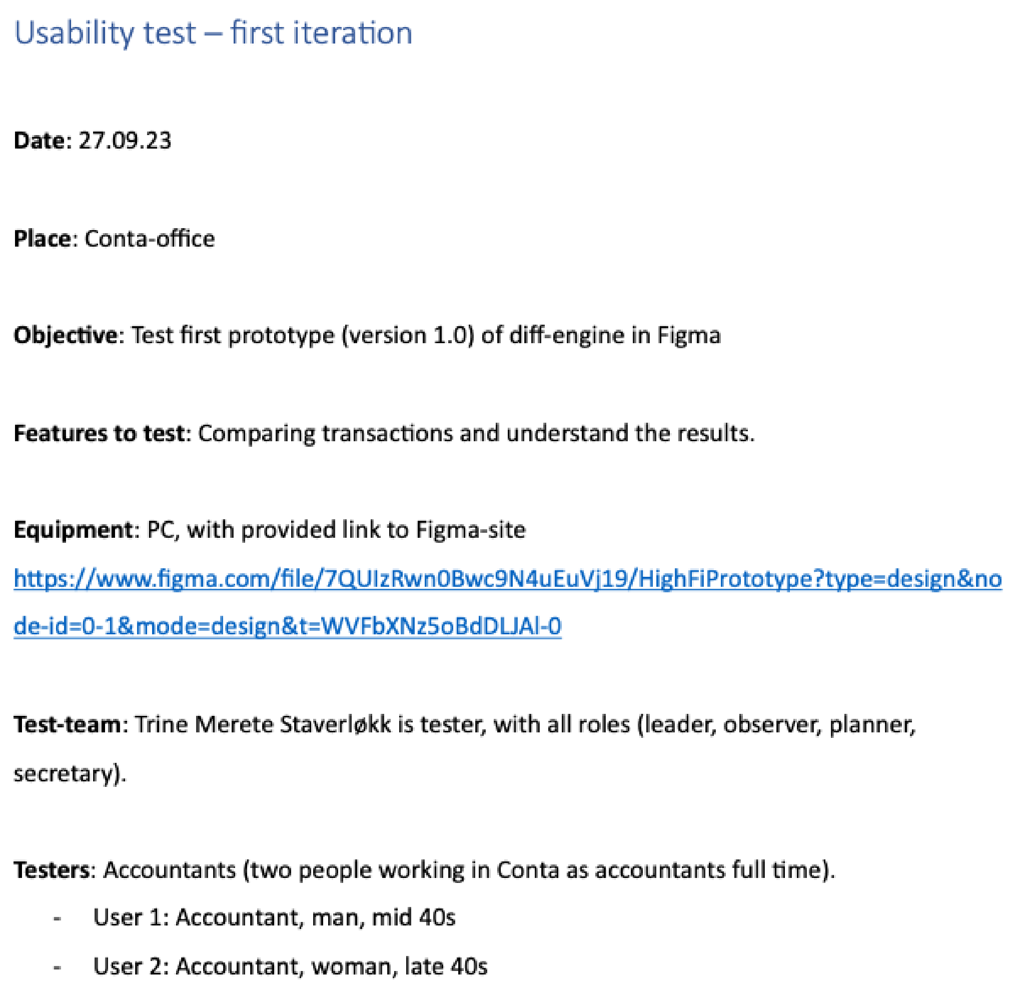
[x] Jeg er kjent med NTNUs regelverk: *Det er ikke tillatt å generere besvarelse ved hjelp av kunstig intelligens og levere den helt eller delvis som egen besvarelse.* Jeg har derfor redegjort for all anvendelse av kunstig intelligens enten i) direkte i rapporten eller ii) i dette skjemaet.

[illegible]

Underskrift/Dato/Sted

Appendix 2 - Usability test report first iteration

First test, 27.09.23



Usability test – first iteration

Date: 27.09.23

Place: Conta-office

Objective: Test first prototype (version 1.0) of diff-engine in Figma

Features to test: Comparing transactions and understand the results.

Equipment: PC, with provided link to Figma-site https://www.figma.com/file/7QUIzRwn0Bwc9N4uEuVj19/HighFiPrototype?type=design&node-id=0-1&mode=design&t=WVFbXNz5oBdDLJAl-0

Test-team: Trine Merete Staverløkk is tester, with all roles (leader, observer, planner, secretary).

Testers: Accountants (two people working in Conta as accountants full time).

- User 1: Accountant, man, mid 40s
- User 2: Accountant, woman, late 40s

Implementation

Flow – make user navigate to diff-engine and use it:

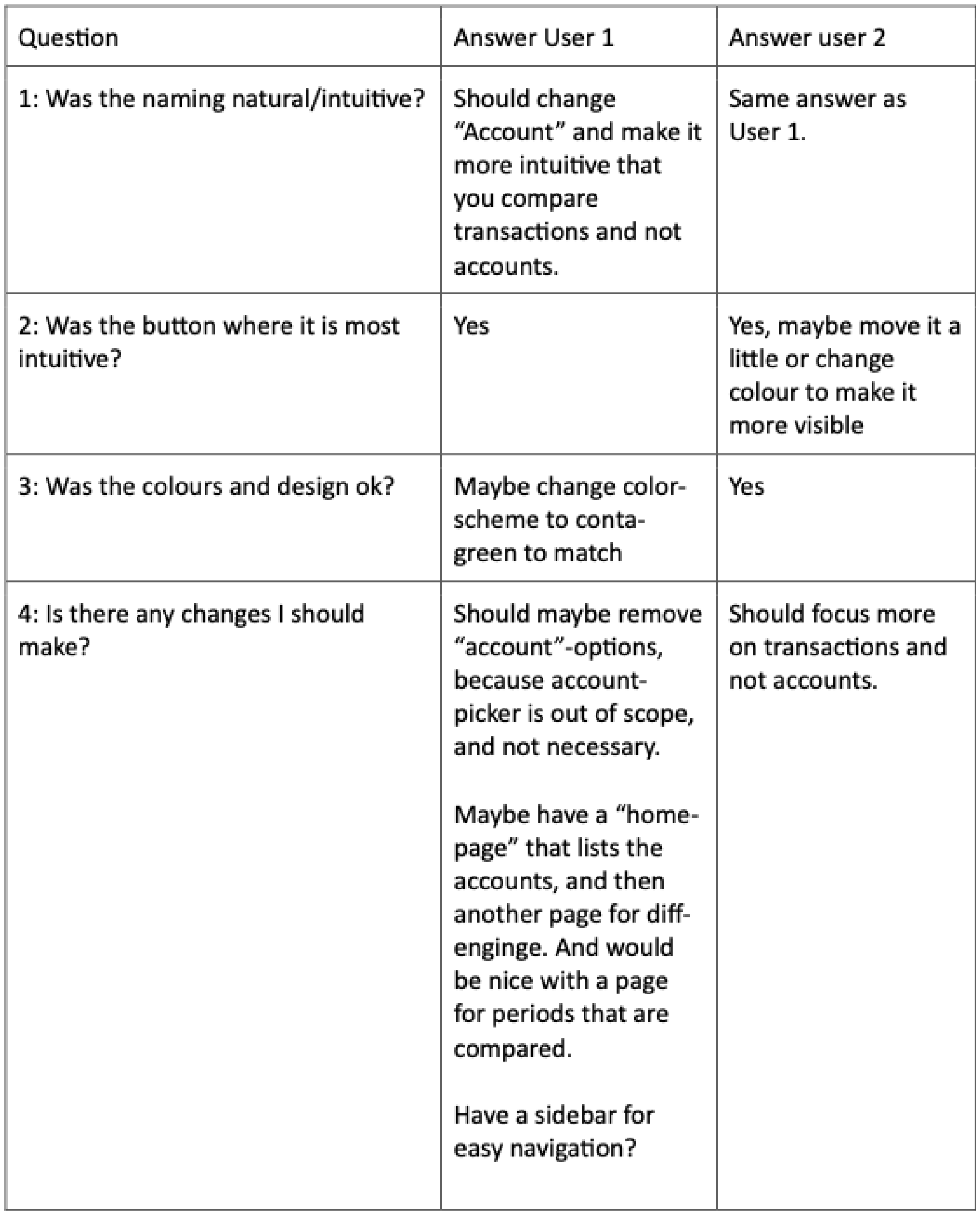
1. Click on account 1920 (Bankkonto)
2. Click on button "Match konto"
3. See results
4. Change account
5. Match again
6. See results

Questions:

- Was the naming natural/intuitive?
- Was the button where it is most intuitive?
- Was the colours and design ok?
- Is there any changes I should make?

Results

Task	Result
1: Click on account 1920 (Bankkonto)	No problem user 1, user 2 did not understand fully. Started clicking first account, but after tester explain better there was no problem.
2: Click on button "Match konto"	No problem both users
3: See results	No problem both users
4: Change account	No problem both users
5: Match again:	No problem both users
6: See results	No problem both users



Question	Answer User 1	Answer user 2
1: Was the naming natural/intuitive?	Should change "Account" and make it more intuitive that you compare transactions and not accounts.	Same answer as User 1.
2: Was the button where it is most intuitive?	Yes	Yes, maybe move it a little or change colour to make it more visible
3: Was the colours and design ok?	Maybe change color-scheme to conta-green to match	Yes
4: Is there any changes I should make?	Should maybe remove "account"-options, because account-picker is out of scope, and not necessary. Maybe have a "home-page" that lists the accounts, and then another page for diff-enginge. And would be nice with a page for periods that are compared. Have a sidebar for easy navigation?	Should focus more on transactions and not accounts.

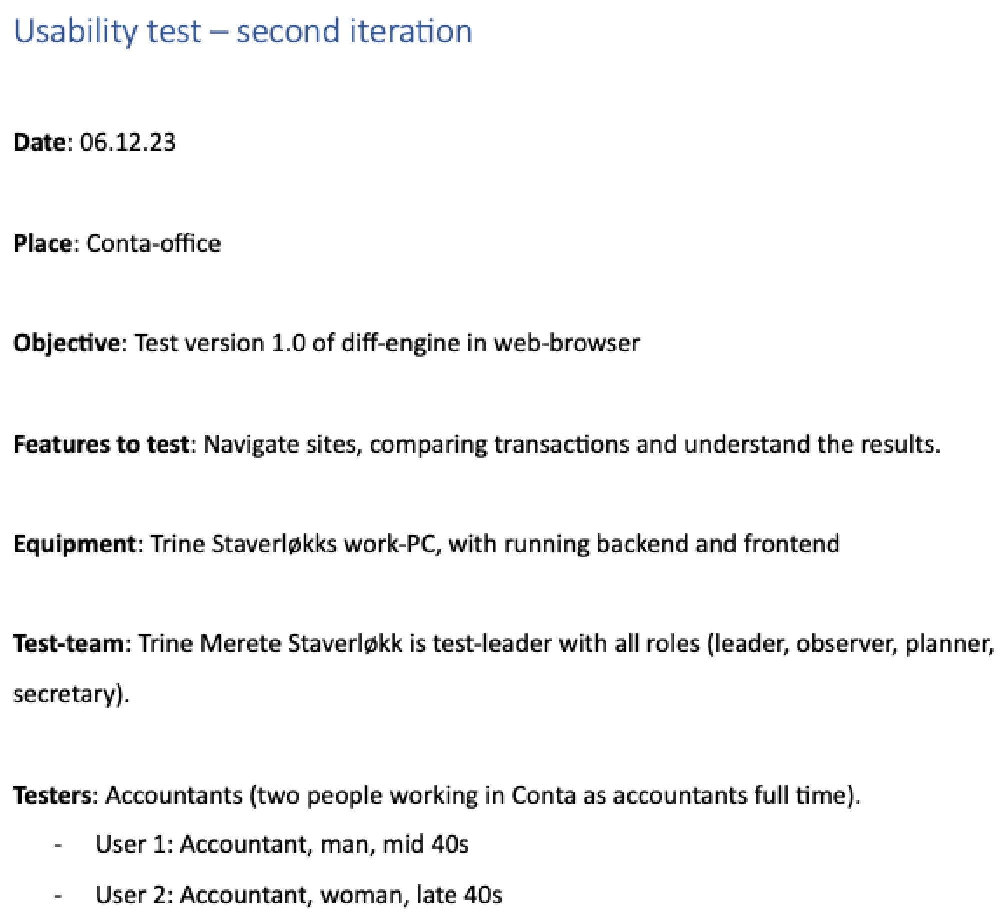
What to change when starting developing

- Change color and name to match-button. Make it more visible. And the system is not matching accounts, its matching transactions.
- Remove functionality with accounts, only have transactions listed from accounting and bank.
- Add a home-page, page for compared periods and a page that does the comparing.
- Add a sidebar or topbar. The one already on the prototype is just a placeholder for now, but it needs to be used more when having more pages.

What not to change:

- Colour to Conta-green. Reason: Conta-green will probably not be like that in the future in the accounting-system that Conta is making today. It is planned to be more blue, which is why the choice is blue in the prototype as well.

Appendix 3 - Usability test report second iteration



Usability test – second iteration

Date: 06.12.23

Place: Conta-office

Objective: Test version 1.0 of diff-engine in web-browser

Features to test: Navigate sites, comparing transactions and understand the results.

Equipment: Trine Staverløkks work-PC, with running backend and frontend

Test-team: Trine Merete Staverløkk is test-leader with all roles (leader, observer, planner, secretary).

Testers: Accountants (two people working in Conta as accountants full time).

- User 1: Accountant, man, mid 40s
- User 2: Accountant, woman, late 40s

Implementation

Flow – make user navigate to diff-engine and use it:

1. Find page for diff-engine
2. Match transactions in June
3. See results
4. Navigate to saved periods
5. See results

Questions:

1. Was the naming natural/intuitive?
2. Was the button where it is most intuitive?
3. Was the navigation intuitive and clear?
4. Is there any changes I should make?

Results

Task	Result
1: Find page for diff-engine	No problem both users
2: match transactions in June	No problem both users
3: See results	No problem both users
4: Navigate to saved-periods	No problem both users
5: See and understand results	No problem both users

Question	Answer User 1	Answer user 2
1: Was the naming natural/intuitive?	Yes, but maybe "reconcile period" is more professional.	Yes
2: Was the button where it is most intuitive?	Yes	Yes,
3: Was the navigation intuitive and clear?	Yes	Yes
4: Is there any changes I should make?	Maybe include a dialog that makes user agree to the comparison before saving it?	Maybe navigate directly to saved periods, or have some confirmation that its saved somewhere.

Further work

- Include dialog of period-change, where you can accept or decline comparison.
- Navigate to saved when accepting for a better flow.