

## ABSTRACT

Project development estimation is a challenging and critical aspect of project management. Accurate estimations of a project's duration and cost allow for proper planning, resource allocation, and budgeting. On the other hand, inaccurate estimates can lead to budget overruns, project cancellation, or losing clients. Project estimation is also a complex field with numerous random elements that can be difficult to plan for. In this thesis, we have been tasked by the software company Axbit with standardizing their estimation method and building the demonstrator for a tool to assist them with project estimation.

The team used the Agile development methodology to keep the project moving at a steady pace and in the right direction throughout the development. User stories aided us in aligning our requirements and expectations. This made setting up issues to work on easier and more manageable. Having weekly status meetings helped us get a good end and start on each iteration in the development.

During the project, we utilized modern frameworks and tools, such as SvelteKit, Web Workers, and Docker to create an application with modern features, among which are server-side rendering and multithreading.

The result of our project is a simulation-based method built on estimation theories like PERT and the Monte Carlo method. The method utilizes normalized units of estimation and takes into account the degree of uncertainty of each task's duration and the availability and efficiency of the members of the project's development team. As a demonstrator for a tool to assist with this method, we also made a web application with a complete CI/CD pipeline. The application implements the method and produces a visualization of the estimations, including an optimized project development plan.

## SAMMENDRAG

Prosjektutviklingsestimering er en utfordrende og kritisk del av prosjektledelse. Nøyaktige estimater av et prosjekts varighet og kostnad tillater riktig planlegging, ressursallokering og budsjettering. På den annen side kan unøyaktige estimater føre til budsjettoverløp, prosjektavbestilling eller tap av kunder. Prosjektberegning er også et komplekst felt med mange tilfeldige elementer som kan være vanskelig å planlegge for. I denne avhandlingen har vi blitt bedt av programvareselskapet Axbit om å standardisere deres estimeringsmetode og bygge en demonstrator for et verktøy for å hjelpe dem med prosjektberegning.

Teamet brukte Smidige utviklingsmetoder for å holde prosjektet i gang, i et jevnt tempo og i riktig retning gjennom hele utviklingen. User stories hjalp oss med å justere våre krav og forventninger. Dette gjorde det enklere og mer håndterlig å sette opp saker å jobbe med. Ukentlige statusmøter hjalp oss med å få en god slutt og start på hver iterasjon i utviklingen.

I løpet av prosjektet benyttet vi moderne rammeverk og verktøy, som SvelteKit, Web Workers og Docker for å skape en applikasjon med moderne funksjoner, deriblant server-side rendering og multithreading.

Resultatet av prosjektet vårt er en simuleringsbasert metode bygget på estimeringsteorier som PERT og Monte Carlo-metoden. Metoden benytter normaliserte enheter av estimater og tar hensyn til graden av usikkerhet for hver oppgaves varighet og tilgjengeligheten og effektiviteten til medlemmene i prosjektets utviklingsteam. Som en demonstrator for et verktøy til å assistere med denne metoden, lagde vi også en webapplikasjon med en komplett CICD-pipeline. Applikasjonen implementerer metoden og produserer en visualisering av estimatene, inkludert en optimalisert prosjektutviklingsplan.

## PREFACE

This thesis is written by Espen Otlo, Janita Røyseth, and Sakarias Sæterstøl and is the final assignment of their computer science study program at NTNU Ålesund. It describes a project conducted in cooperation with the software company Axbit.

The project has provided multiple challenging tasks and experiences in several fields, some worthy mentions being the interpretation of customer requirements, project planning, simulation, multithreading, and full-stack development. This thesis describes the theoretical foundations and methods utilized and the results that were achieved in the project.

### Acknowledgements

We would like to express our sincere gratitude to the following individuals for their invaluable contribution and support during our project:

Firstly, we want to extend our heartfelt thanks to Anniken Karlsen, who served as our supervisor throughout the project. Her guidance, encouragement, and constructive feedback helped us to stay on track and deliver a successful outcome.

We would also like to acknowledge Franck Chauvel, who represented Axbit and served as the project's stakeholder representative, whose collaboration and teamwork were instrumental in the project's success. His expertise and willingness to share his knowledge were truly appreciated.

Our sincere thanks also go to Girts Strazdins for the support he has provided throughout our three years of undergraduate studies, as well as for the commitment he has shown not only as a lecturer and program coordinator but also towards the subject matter and helping us to complete our studies successfully.

Finally, we express our gratitude to Arne Styve for his unwavering dedication to our education and for inspiring us to pursue our academic goals. His teachings and mentorship provided us with a solid foundation for our bachelor's degree and equipped us with the skills and knowledge to succeed in our future endeavors.

Once again, thank you to all these individuals for their contributions, without which the project would not have been possible.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Sammendrag</b>	<b>ii</b>
<b>Preface</b>	<b>iii</b>
<b>Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>Abbreviations</b>	<b>xii</b>
<b>Glossary</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Formulation . . . . .	1
1.3 Objectives . . . . .	1
1.4 Limitations . . . . .	2
1.5 Thesis Structure . . . . .	2
<b>2 Theory</b>	<b>3</b>
2.1 Agile Development . . . . .	3
2.1.1 User Requirements . . . . .	3
2.1.2 User Stories . . . . .	4
2.2 Challenges of Estimation . . . . .	4
2.2.1 Problem Scope . . . . .	4
2.2.2 Project Size . . . . .	4
2.2.3 Unknown Elements . . . . .	5
2.2.4 Efficiency . . . . .	5

2.2.5	Units of Estimation . . . . .	5
2.3	Estimation Methods . . . . .	6
2.3.1	Project Evaluation and Review Technique . . . . .	6
2.3.2	Critical Path Method . . . . .	7
2.3.3	COCOMO . . . . .	7
2.3.4	Planning Poker . . . . .	7
2.3.5	Estimation of project duration for Scrum team with differentiated specializations . . . . .	7
2.3.6	Monte Carlo Simulation . . . . .	8
2.4	Object-Oriented Programming . . . . .	8
2.4.1	Cohesion and Coupling . . . . .	8
2.5	System Documentation . . . . .	9
2.5.1	Source Code Comments . . . . .	9
2.5.2	API Documentation . . . . .	9
2.5.3	UML Diagram . . . . .	9
2.5.4	README . . . . .	9
2.5.5	Wireframes . . . . .	9
2.6	Testing . . . . .	10
2.6.1	Unit Tests . . . . .	10
2.6.2	Integration Tests . . . . .	10
2.6.3	End-to-end Tests . . . . .	10
2.6.4	Test Pyramid . . . . .	10
2.6.5	Usability Testing . . . . .	11
2.7	Continuous Integration, Continuous Deployment, and Continuous Delivery	13
2.7.1	Continuous Integration . . . . .	13
2.7.2	Continuous Delivery . . . . .	13
2.7.3	Continuous Deployment . . . . .	13
2.8	Universal Design . . . . .	13
2.8.1	Web Content Accessibility Guidelines . . . . .	13
2.8.2	Accessible Rich Internet Applications . . . . .	14
2.8.3	The Authority for Universal Design of ICT . . . . .	14
2.9	Design . . . . .	14
2.9.1	Design Principles . . . . .	14
2.9.2	Iterative Design Process . . . . .	15
2.9.3	Design Guidelines . . . . .	16
2.9.4	Wireframes . . . . .	16

2.10	Technologies . . . . .	16
2.10.1	Programming Languages . . . . .	16
2.10.2	Code Version Control . . . . .	17
2.10.3	Command Languages . . . . .	18
2.11	Web Application . . . . .	18
2.11.1	Roles of HTML, CSS, and JS . . . . .	18
2.11.2	Single-page Application . . . . .	18
2.11.3	Server-side Rendering . . . . .	18
<b>3</b>	<b>Methods</b>	<b>19</b>
3.1	Organization . . . . .	19
3.1.1	Team . . . . .	19
3.1.2	Supervisor . . . . .	19
3.1.3	Client & Product Owner . . . . .	19
3.2	Project Planning . . . . .	20
3.2.1	Preliminary Report . . . . .	20
3.2.2	Gannt Diagram . . . . .	20
3.2.3	Work Contract . . . . .	20
3.2.4	User Requirements . . . . .	20
3.2.5	User Stories . . . . .	21
3.3	Management Tools . . . . .	22
3.3.1	Teams . . . . .	22
3.3.2	Confluence . . . . .	22
3.3.3	Overleaf . . . . .	22
3.3.4	GitHub . . . . .	23
3.4	Developmental Tools and Applications . . . . .	23
3.4.1	Figma . . . . .	23
3.4.2	Lighthouse . . . . .	23
3.4.3	VS Code . . . . .	23
3.4.4	Docker . . . . .	23
3.4.5	pgAdmin . . . . .	23
3.4.6	Traefik . . . . .	24
3.4.7	Make . . . . .	24
3.4.8	Vite . . . . .	24
3.4.9	Vitest . . . . .	24
3.4.10	Storybook . . . . .	24

3.4.11	Playwright . . . . .	24
3.4.12	Package Manager . . . . .	24
3.4.13	GitHub Actions . . . . .	25
3.5	Framework, Libraries, Programming- and Scripting Languages . . . . .	25
3.5.1	Svelte . . . . .	25
3.5.2	SvelteKit . . . . .	25
3.5.3	Typescript . . . . .	25
3.5.4	SCSS . . . . .	26
3.5.5	Carbon Design Systems . . . . .	26
3.5.6	PostgreSQL . . . . .	26
3.6	Data and Configuration . . . . .	26
3.6.1	JSON . . . . .	26
3.6.2	YAML . . . . .	26
3.6.3	DTO . . . . .	26
3.7	System Documentation . . . . .	27
3.7.1	ER Diagram . . . . .	27
3.7.2	Class Diagram . . . . .	27
3.7.3	Source Code Documentation . . . . .	27
3.7.4	Svelte Component Documentation . . . . .	27
3.8	Design . . . . .	28
3.8.1	Wireframing . . . . .	28
3.8.2	Design Guidelines . . . . .	28
3.8.3	Accessibility . . . . .	28
3.9	Testing Methodology . . . . .	28
3.9.1	Unit Tests . . . . .	28
3.9.2	Integration Tests . . . . .	29
3.9.3	End-to-end Tests . . . . .	29
3.9.4	Usability Tests . . . . .	29
<b>4</b>	<b>Results</b>	<b>30</b>
4.1	Method of Estimation . . . . .	30
4.2	Continuous Integration and Delivery . . . . .	31
4.2.1	CI/CD Process Overview . . . . .	31
4.2.2	Workflows . . . . .	31
4.3	Environment Files . . . . .	33
4.4	Documentation . . . . .	33

4.4.1	Code Documentation . . . . .	33
4.4.2	Component Documentation . . . . .	35
4.5	Backend . . . . .	38
4.5.1	Entity Classes . . . . .	38
4.5.2	Simulation . . . . .	39
4.5.3	Database . . . . .	44
4.5.4	Server . . . . .	44
4.5.5	Docker . . . . .	45
4.5.6	Postgres . . . . .	46
4.6	Front-end . . . . .	47
4.6.1	Design . . . . .	47
4.6.2	Components . . . . .	49
4.6.3	Routing . . . . .	49
4.6.4	State Management . . . . .	50
4.6.5	User Interface . . . . .	52
4.7	Service Classes . . . . .	61
4.7.1	CalendarService . . . . .	62
4.7.2	EstimatableService . . . . .	62
4.7.3	ModalService . . . . .	63
4.7.4	RandomService . . . . .	64
4.7.5	SimulationService . . . . .	64
4.8	Test and Quality Assurance . . . . .	66
4.8.1	Unit Tests . . . . .	66
4.8.2	Integration Tests . . . . .	66
4.8.3	End-to-end Tests . . . . .	68
4.8.4	Usability Tests . . . . .	69
4.8.5	Code Format and Lint . . . . .	70
<b>5</b>	<b>Discussion</b>	<b>71</b>
5.1	Estimation Method . . . . .	71
5.1.1	Basic Unit . . . . .	71
5.1.2	Three-point Estimates . . . . .	71
5.1.3	Employee Efficiency . . . . .	71
5.1.4	Method Steps . . . . .	72
5.2	Framework . . . . .	72
5.3	Security . . . . .	72



5.4	UI/UX . . . . .	72
5.5	Testing . . . . .	73
5.6	Group Dynamic and Methodology . . . . .	74
5.6.1	Methodology . . . . .	74
5.6.2	Group Dynamic . . . . .	74
<b>6</b>	<b>Conclusions</b>	<b>75</b>
6.1	Conclusions . . . . .	75
6.2	Future work . . . . .	76
6.2.1	Validation . . . . .	76
6.2.2	Integration . . . . .	76
6.2.3	Historical data . . . . .	76
6.2.4	MTBF & MTTR . . . . .	76
	<b>References</b>	<b>76</b>
	<b>Appendices:</b>	<b>83</b>
	<b>A - Preliminary Project Plan</b>	<b>84</b>
	<b>B - First Draft Wireframe</b>	<b>94</b>
	<b>C - Second Draft Wireframe</b>	<b>96</b>
	<b>D - Final Draft Wireframe</b>	<b>98</b>
	<b>E - Design Guidelines</b>	<b>100</b>
	<b>F - Test Template</b>	<b>104</b>
	<b>G - 1st Iteration User Tests - Test Plan</b>	<b>106</b>
	<b>H - 1st Iteration User Tests - User A</b>	<b>109</b>
	<b>I - 1st Iteration User Tests - User B</b>	<b>115</b>
	<b>J - 2nd Iteration User Tests - Test Plan</b>	<b>121</b>
	<b>K - 2nd Iteration User Tests - User A</b>	<b>124</b>
	<b>L - 2nd Iteration User Tests - User B</b>	<b>129</b>

# List of Figures

2.2.1 Project Size . . . . .	5
2.6.1 Test pyramid . . . . .	11
4.2.1 Github action workflow . . . . .	31
4.2.2 CI . . . . .	32
4.2.3 CD to development environment . . . . .	33
4.3.1 .env.example file . . . . .	33
4.4.1 Code example from <code>SortableList</code> documentation in code . . . . .	34
4.4.2 Code example from <code>SortableList</code> documentation in code . . . . .	35
4.4.3 Code example from <code>SortableList</code> documentation in code . . . . .	36
4.4.4 Documentation displayed from <code>SortableList</code> component documentation . . . . .	36
4.4.5 Storybook UI button . . . . .	37
4.4.6 Storybook UI button edited properties . . . . .	37
4.5.1 Entities UML diagram . . . . .	38
4.5.2 Simulation Project Level . . . . .	40
4.5.3 Code Example - Project Level . . . . .	41
4.5.4 Simulation Assignee Level . . . . .	42
4.5.5 Code Example - Assignee Level . . . . .	43
4.5.6 ER Diagram . . . . .	44
4.5.7 Dockerfile . . . . .	45
4.5.8 Sequence code . . . . .	46
4.6.1 Final iteration of wireframes . . . . .	48
4.6.2 Color scheme . . . . .	48
4.6.3 Components folder . . . . .	49
4.6.4 Routes . . . . .	50
4.6.5 Code example from the <code>\$projects</code> store . . . . .	51
4.6.6 Login page . . . . .	52
4.6.7 Projects page . . . . .	52
4.6.8 Code projects layout . . . . .	53
4.6.9 Search projects modal . . . . .	54
4.6.10 Screenshot of the create project modal . . . . .	55
4.6.11 Project overview . . . . .	55
4.6.12 Edit project modal . . . . .	56
4.6.13 Delete project modal . . . . .	56
4.6.14 Features page . . . . .	57
4.6.15 Feature modal . . . . .	57
4.6.16 Creating and editing features in the application . . . . .	58
4.6.17 Delete feature modal . . . . .	58
4.6.18 Tasks page . . . . .	59
4.6.19 Task modal . . . . .	59

4.6.20	Modal in Create and Edit modes . . . . .	60
4.6.21	Delete task modal . . . . .	60
4.6.22	Tasks page . . . . .	61
4.7.1	Services UML diagram . . . . .	62
4.7.2	Code example from the calendar service . . . . .	62
4.7.3	Code example from the estimatable service . . . . .	63
4.7.4	Code example from the modal service . . . . .	63
4.7.5	Code example from the modal service . . . . .	64
4.7.6	Code example from the random service . . . . .	64
4.7.7	Code example from the simulation service . . . . .	65
4.8.1	Code Example - Assignee Level . . . . .	66
4.8.2	Storybook Configuration . . . . .	67
4.8.3	Storybook interaction test . . . . .	67
4.8.4	Integration test coverage . . . . .	68
4.8.5	End-2-end tests results . . . . .	69

## ABBREVIATIONS

- API** Application Package Interface. 62
- ARIA** Accessible Rich Internet Applications. 14, 28
- CDel** Continuous Delivery. xiv, 13
- CDep** Continuous Deployment. xiv, 13
- CI** Continuous Integration. xiv, 13, 70
- CICD** Continuous Integration and Continuous Delivery. i, xv, 25, 31, 45
- CLI** Command Line Interface. 18
- COCOMO** Constructive Cost Model. xiv, xv, 6, 7
- CORS** Cross-origin resource sharing. 72
- CPM** Critical path method. xv, 6
- CSS** Cascading Style Sheets. vi, 17, 18, 26, 73
- DOM** Document Object Model. 17, 18
- DTO** Data Transfer Object. 26
- E2E** End-to-End. 10, 24, 29, 68, 73
- ERD** Entity Relation Diagram. 44
- ERP** Enterprise Resource Planning. 76
- ESM** Enterprise Systems Management. 24
- FP** Function Points. 6
- GHA** GitHub Actions. 25, 31
- HTML** HyperText Markup Language. vi, 17, 18, 27
- HTTPS** Hypertext transfer protocol secure. 72

- JS** JavaScript. vi, 16–18, 25
- JSON** JavaScript Object Notation. 26
- LoC** Lines of Code. 6
- MVP** Minimum Viable Product. 29, 73
- OOP** Object-Oriented Programming. 1, 8, 16
- PERT** Project evaluation and review technique. i, xiv–xvi, 6, 30
- PR** Pull Request. 23, 25, 31
- SCSS** Sassy CSS. 26
- SQL** Structured Query Language. 17, 26
- SSR** Server-side rendering. 18
- TPE** Three-point estimate. 71
- TS** TypeScript. 17
- UI** User Interface. 1, 10, 14, 15, 73, 74
- UML** Unified Modeling Language. 9, 27
- UX** User Experience. 1, 14, 15, 73, 74
- VM** Virtual Machine. 17
- W3C** World Wide Web Consortium. 13
- WAI** Web Accessibility Initiative. 13
- WAI-ARIA** Web Accessibility Initiative - Accessible Rich Internet Applications. 14
- WBS** Work breakdown structure. 6, 7
- WCAG** Web Content Accessibility Guidelines. 13, 14, 28, 73
- YAML** YAML Ain't Markup Language. 25

## GLOSSARY

**80-hour rule** Stems from PERT and states that no single activities, or group of activities a the lowest level of a work breakdown structure should be more than 80 hours of effort. 30

**Agile development** An iterative and flexible approach to software development that emphasizes collaboration, adaptability and continuous improvement. Further described in section 2.1. i, 3, 4, 74, 75

**average software developer** Is a term defined by Teslyuk et al [1] to be used as a normalized metric for efficiency and refers to a software engineer with mid-level competency. xv

**Continuous Delivery** Continuous Delivery (CDel) is a practice where the code changes are automatically built, tested and pushed to a non-production environment, testing or staging. xii, xiv, 31

**Continuous Deployment** Continuous Deployment (CDep) is a practice where the code changes are automatically tested, build and released into production. xii, xiv

**Continuous Integration** Continuous Integration (CI) is a practice where developers merge their code changes regularly into a central repository. xii, xiv, 31

**cross-origin resource sharing** a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin (domain) permission to access selected resources from a server at a different origin. xii

**Docker** Docker is a platform for developing, shipping and running applications. Further described in section 3.4.4. i, 33

**Enterprise Resource Planning** A type of software that organizations use to manage day-to-day business activities such as accounting, procurement, project management, employee management and supply chain operations. . xii

**Extreme Programming** An agile software development methodology, advocating continuous code reviews in the form of pair programming. 7

**Function Points** A basic unit for measuring the size of a software system in the project estimation method COCOMO. xii, 6

- GitHub Actions** An automation tool for CICD provided by GitHub.. xii, 25, 31
- hypertext transfer protocol secure** An internet communication protocol that protects the integrity and confidentiality of data sent between a user's computer and a website. xii
- ideal hours** Refers to the required number of undisturbed hours spent by an average software developer working solely on the implementation of a task. 20–22, 39, 71, 75
- Lines of Code** A unit of measurement in the COCOMO method, that is used to measure the scope of a project. xiii, 6, 7
- linking table** A table in a database that exist solely to represent relationships between entities, especially many-to-many relationships.. 44
- man-hour** A unit used to measure the effort of a task in various project estimation methods, such as PERT and CPM. 6
- Monte Carlo method** A mathematical technique that simulates the range of possible outcomes for an uncertain event. Further described in section 2.3.6. i, 30
- multithreading** A feature that allows multiple lines of code to execute concurrently, thereby increasing the performance and responsiveness of a program.. i, 72
- object-oriented programming** A programming paradigm that organizes code into objects. Further explained in section 2.4. 25
- Planning Poker** A concensus-based technique for estimating. Further described in section 2.3.4. 30
- Product Owner** Serving as the 'voice of the customer' the Product Owner is the person or part of the team responsible for guiding the project toward the desired result. 2.3.1. 19–21, 25, 27, 74
- project evaluation and review technique** An expertise-based method of project estimation. Further described in section 2.3.1. xiii, xvi, 6
- scrum** An agile development framework, designed for small teams who structure their work into time-boxed iterations called sprints, and conduct daily status meetings of up to 15 minutes. 4, 7
- server-side rendering** A technique in web development where a web page is generated on the server upon each request and sent to the client, instead of being rendered in the client's browser. Further described in section 2.11.3. i
- story point** Used in Agile development as a measure for expressing an estimate of the effort required to implement a user story, task or other work item. 21, 22
- SvelteKit** A framework for creating Single-page applications with Server-side rendering. Further described in section 3.5.2. i, 72

**t-shirt size** Used in Agile development as a rough estimation technique to estimate the relative effort of user stories, tasks or other work items. The t-shirt sizes are typically represented by small, medium and large, similar to the sizes of t-shirts. 21, 22

**three-point estimate** Stems from Project evaluation and review technique and refers to the process of making one estimation for the number of hours required for the completion of a task for each of the best case scenario, the most likely scenario and the worst case scenario. xiii, 30, 39

**user story** Used in Agile development, a user story is an informal explanation of a software feature written from the perspective of the end user or customer. Further described in section 3.2.5. i, xv, xvi, 21, 75

**Web Worker** A feature of JavaScript that allows scripts to be executed in the background, on a separate thread from the main application. This feature allows for running multithreaded applications in the browser. i, 72

**work breakdown structure** A concept developed with the project evaluation and review technique, and is described as a hierarchical and incremental decomposition of a project into phases, deliverable and work packages. Further described in section 2.3.1. xiii, xiv, 6, 30, 72





## INTRODUCTION

This chapter will introduce the motivation for the project, the problem formulation and the objectives and limitations for the project, followed by a deliberation of the structure of this thesis.

### 1.1 Motivation

We have selected this task because it challenges all that we have learned through our study program at NTNU, touching all of UI/UX design, DB- and server management, Object-Oriented Programming, statistics and application development. Another reason for choosing this task was that through our projects in the study program, we have experienced first-hand the importance and the challenges of the cost and time estimation involved in project management. While extensive research has been conducted in this field, the complex and ever-evolving nature of project management presents a vast and multi-faceted problem that may not have a definitive and foolproof solution. Nevertheless, we believe further research into this area may bring valuable insights, enhance our understanding of the problem, and enable us to make more informed decisions and estimations with greater confidence in the future.

### 1.2 Problem Formulation

Axbit currently does not have a standardized method of formulating project descriptions and constraints, and so the aim of our thesis is two-fold; Develop a method for project planning and estimation to be used by project managers in their interactions with clients, and create a demonstrator for a tool to assist project managers with this task.

### 1.3 Objectives

The objectives for the thesis are as follows:

- Research project estimation theory and existing methods of operation in regards to project estimation used by Axbit's project managers.
- Develop a standardized method for project estimation that can be integrated into the business practices of Axbit.

- Develop a software application to serve as a demonstrator for a tool that can be used by the Axbit's project managers to provide estimates with low effort from the user.

## 1.4 Limitations

Axbit expressed an intention of furthering the development of the demonstrator, and therefore gave us the following requirements:

- The method must be suitable for agile teams working iteratively.
- The method must be usable both at the preliminary planning stage and during development.
- The tool must provide a project plan, including cost and time.

## 1.5 Thesis Structure

The remaining parts of the thesis are divided into theory, methods, results, discussion, and conclusion. In the theory chapter, we delve into the theoretical foundation and concepts used to solve our objectives in the thesis. The method chapter discusses the methods and materials used throughout the project's development. In the results chapter, we discuss and showcase the results we achieved. In the discussion chapter, we discuss our process and the results of our work. Lastly is the conclusion, where we summarize and conclude our findings throughout this thesis.

In this chapter, we explore the theories that are relevant to our project. We discuss both the theories of the technologies we utilized and the theories that guided our decision-making process throughout the project.

## 2.1 Agile Development

Agile development is an iterative and flexible approach to software development that emphasizes collaboration, adaptability, and continuous improvement. It focuses on delivering value to customers by frequently delivering working software increments.

The Agile Manifesto [2] was created in 2001 by a group of software developers and outlines the core principles of Agile development. These principles promote short and iterative development cycles, maintaining good communication and close cooperation, and using working software as the primary measure of progress.

The usage of the Agile development method has increased rapidly since it was introduced. As of 2022, the adoption of Agile has increased from 37% to 86% for software development teams [3], thereby becoming the leading method for software development.

### 2.1.1 User Requirements

User requirements describe what a user expects a system or product to do to fulfill their needs or goals. They focus on the functionality, features, and characteristics that are desired from the user's perspective. User requirements might include:

- **Functional requirements:** These specify what the system or product should do.
- **Performance requirements:** These dictate the quantified level of performance the system or application must achieve.
- **Interface requirements:** These refer to how the user will interact with the system or product.
- **Data requirements:** These detail the specific data formats, or how data is to be entered or displayed.

- Security requirements: These outline any needs related to data privacy, authentication or authorization.
- Usability requirements: These specify any user needs related to the usability of the system or product, such as ease of use or accessibility.

### 2.1.2 User Stories

User stories are a technique used in Agile development to serve as informal, general explanations of a software feature written from the perspective of the end user or customer. User stories focus on the value or benefit the user expects to derive from the software. They serve as a communication tool to ensure that the development team builds what the users truly need, thus enhancing the chances of product success.

## 2.2 Challenges of Estimation

Project development estimation is a critical aspect of project management. Accurate estimations allow for proper planning, resource allocation, and budgeting. However, there are several challenges and problems associated with reaching this accuracy. This section will cover the scope of the problem and present some of the larger contributing factors.

### 2.2.1 Problem Scope

The Standish Group in their CHAOS report from 2015 [4] found that only 40% of projects were completed on time, and 44% were completed within budget. The global consulting firm McKinsey also found in their study [5] that software development projects have an average cost overrun of 66% and a timetable delay of 33%. This great problem of inaccuracy stems from numerous factors, and mapping all of them in detail would be a study of its own. Instead, we focused on key factors relevant to Axbit, which operates with scrum teams with differentiating specializations. These factors include project size, unknown elements, employee efficiency and units of estimation.

### 2.2.2 Project Size

According to the CHAOS Report [4], the largest contributing factor to the project success rate is the scope of the project. Small projects showed a 61% success rate, whereas grand projects had a success rate of 6%. The decrease in accuracy of estimations introduced by an increase in project scope correlates with the "Cone of Uncertainty" [6], which states that the more unknown variables exist within a project, the higher the uncertainty of its estimations will be.

PROJECT SIZE BY CHAOS RESOLUTION				
	SUCCESSFUL	CHALLENGED	FAILED	TOTAL
Grand	6%	51%	43%	100%
Large	11%	59%	30%	100%
Medium	12%	62%	26%	100%
Moderate	24%	64%	12%	100%
Small	61%	32%	7%	100%

*The size of software projects by the Modern Resolution definition from FY2011-2015 within the new CHAOS database.*

Figure 2.2.1: The success rates of projects by project size [4].

### 2.2.3 Unknown Elements

Another factor closely linked with project scope is the number of unknown elements in the project. Larger projects tend to include more unknown elements, and these are particularly subjected to optimism bias and pressure from clients, which again leads to inaccurate estimates. This 'unknown' factor can to some degree be mitigated by team experience and/or altering the project scope to utilize familiar methods and technologies. The Chaos Report from 2015 [4] showed that the project success rate of experienced teams was more than twice that of inexperienced teams.

### 2.2.4 Efficiency

A misconception often made, especially by new or inexperienced teams, is mistaking working hours for time actually spent working. A survey from Vouchercloud [7] showed that the average time an office employee actually spends working each day is 2 hours and 53 minutes. Another survey from Zippia [8] found that the average time American workers spend actually working each day is 4 hours and 12 minutes. Because of this difference between time spent at work, and time spent actually working, it is paramount in the estimation process to clearly differentiate these two in order to avoid communication errors and misconceptions.

### 2.2.5 Units of Estimation

When it comes to project estimation, two main types of basic units of measurement exist - parametric and relative.

- Parametric units are absolute, and remain the same regardless of any other variable. Some examples of parametric units are days, meters and pixels.
- Relative units are relative to another value, and examples include percentages, story points and function points, the two latter being units most used in Agile development.

The usage of these types vary among the different well established estimation methods. For instance, COCOMO [6] uses the relative units Lines of Code (LoC) and Function Points (FP), while other methods, such as PERT [9] and CPM [10] use the parametric unit man-hour.

Mike Cohn, a prominent figure in the Agile community, argues that relative units of measurement require lower effort and they allow team members with different skill levels to communicate about and agree on an estimate [11]. However, a survey on project teams by A. Zarour and S. Zein [12] showed that 84% of the participants used expertise-based models, and a vast majority used time-based units for effort estimation, making parametric units the most popular in project estimation.

## 2.3 Estimation Methods

The field of project estimation is both sizeable and complex, and several different methods of estimation have been developed in efforts to simplify or improve the accuracy of project estimations. The following sections contain summaries of some of these methods.

### 2.3.1 Project Evaluation and Review Technique

Project evaluation and review technique (PERT) is a project management technique used to analyze and evaluate the time required to complete specific tasks within a project. PERT is a probabilistic method that takes into account the uncertainties and variations in task durations to estimate the project's overall duration [9].

PERT introduces the concept of a Work breakdown structure (WBS), which is a tool used to represent the scope of a project. It involves the decomposition of a project into smaller, manageable parts. This allows for more accurate planning, scheduling and tracking. PERT lists the following principles to follow when creating a WBS:

1. **100% Rule:** The WBS should include all work defined by the scope, and only the work required to complete the project. This means that every level of the WBS should account for 100% of the work of its parent level.
2. **Mutually Exclusive Elements:** Each element of the WBS should be distinct with no overlap in scope with any other elements. This ensures clear assignments and avoids confusion or duplication of work.
3. **Outcome-Focused:** The WBS should focus on deliverables or outcomes, not actions. Each element represents a tangible output or result, not the work required to produce it.

4. **80 Hour Rule:** The lowest level of the WBS, often called work packages, should be detailed enough that they can be scheduled, estimated, monitored, and controlled. As a rule of thumb, a work package should not require more than 80 hours of work.
5. **Coding Scheme:** WBS elements are usually numbered in decimal sequence from top to bottom. With such numbering, it is easier to identify the level of the task that the element represents when referring to it from outside of the context of the WBS chart.

### 2.3.2 Critical Path Method

The Critical Path Method (CPM) is a project management technique used to identify the sequence of activities that must be completed on time in order to complete a project within its allotted schedule. It helps project managers determine the longest path of dependent activities and identifies the tasks that have the most impact on the project's overall duration [10].

### 2.3.3 COCOMO

The Constructive Cost Model (COCOMO) is a method for estimation of the cost, effort and schedule of a software project. It was developed by Barry Boehm in 1981. The model calculates the effort and cost associated with a software development project, based on the size of the software, and a set of cost drivers. The size of the software is usually measured in Lines of Code, or function points. These are both relative units that are easy to measure with, but due to their abstract nature they are not perfect indicators of true system complexity or functionality.

### 2.3.4 Planning Poker

Planning Poker is a consensus-based agile estimation technique used in software development projects, particularly in Scrum and Extreme Programming. It is a collaborative approach that allows team members to collectively estimate the effort or size of user stories, tasks, or backlog items [13].

The primary goal of Planning Poker is to achieve more accurate and unbiased estimates by involving the entire team in the estimation process. It helps to avoid individual biases and promotes collective decision-making.

### 2.3.5 Estimation of project duration for Scrum team with differentiated specializations

Estimation of project duration for Scrum team with differentiated specializations is a method developed as part of a study by Teslyuk et al [1]. This method aims at software development projects implemented by Scrum teams with differentiated specializations and is based on the authors' system of working-time balance equations and the approach to measuring project scope with normalized time-based units.



### 2.3.6 Monte Carlo Simulation

Also known as the Monte Carlo method, Monte Carlo simulation is a mathematical technique, which is used to estimate the possible outcomes of an uncertain event. It was invented by John von Neumann and Stanislaw Ulam during World War II to improve decision-making under uncertain conditions. This method predicts a set of outcomes based on an estimated range of values versus a set of fixed input values [14].

## 2.4 Object-Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which are instances of classes that encapsulate data and behavior. It focuses on objects, as opposed to functional programming, which focuses on action or logic [15]

The key principles of OOP include the following:

- **Encapsulation:** Objects encapsulate data and methods, hiding internal details and providing an interface to interact with the object. This improves code modularity and reusability.
- **Inheritance:** Classes can inherit properties and methods from other classes, forming a hierarchy of classes. This also improves reusability.
- **Polymorphism:** This allows objects of different classes to be treated as objects of a common parent class. It enables methods to be defined in a generic class and their functionality overridden in more specialized child classes.
- **Abstraction:** This means creating a simplified representation of complex real-world objects or systems. It focuses on essential characteristics while hiding unnecessary details.

These principles mean that OOP provides increased maintainability, modularity, and reusability in software applications [15].

### 2.4.1 Cohesion and Coupling

In the context of OOP, cohesion, and coupling are two important concepts that describe the relationships and dependencies between classes and objects in a system [15].

Cohesion refers to the degree of which members of a class are related and work together to achieve a single, well-defined purpose or responsibility. High cohesion leads to a more modular and maintainable code [15].

Coupling refers to the connectivity between classes and is measured by how much a class relies on or knows about other classes. Tight coupling means that a class is heavily dependent on one or more other classes, reducing the class's modularity. Loose coupling means that the class is mostly independent of other classes and their internal properties. Having classes with loose coupling increases the modularity and maintainability and makes it more flexible and resilient to changes in other parts of the code [15].

## 2.5 System Documentation

Documentation in a software project includes all written materials that provide information about the software system. It serves to facilitate understanding of the system, for developers, stakeholders and users. Following is a more in-depth explanation of the various types of documentation that a software product may contain.

### 2.5.1 Source Code Comments

Source code documentation is the explanatory text, comments and annotations within the source code of an application, and serves to provide a clear understanding of the functionality of the code to a reader. The purpose of code documentation is to increase the maintainability of the code, by making it easier for a developer or maintainer to comprehend the code. In some programming languages, such comments are used to automatically generate API documentation.

### 2.5.2 API Documentation

API documentation serves as a technical guide that explains how to effectively use and integrate with an Application Programming Interface (API). APIs are sets of rules and protocols that determine how different software applications should interact, and their documentation is the map that guides developers through these rules. API documentation covers the functionalities of the API, explaining the available functions, including their parameters, response formats and potential error messages.

### 2.5.3 UML Diagram

Unified Modeling Language (UML) is a standardized general-purpose modeling language. It includes a set of graphical notation techniques to create visual models of object-oriented software systems. UML diagrams provide an abstracted view of the system, that hides the implementation details, making it easier for diverse stakeholders to understand the system and communicate about it. UML diagrams can be used to present two different views of a system model:

- Static view: Class diagrams, package diagrams and component diagrams are types of diagrams used in UML to represent the static system structure.
- Dynamic view: Sequence diagrams, activity diagrams and state machine diagrams are types of diagrams used to represent dynamic behaviour of the system.

### 2.5.4 README

README is a file used by developer to tell about the project and provides information on how to use it and sometime also how to continue the development of the project.

### 2.5.5 Wireframes

Wireframes are also part of the system documentation, and is used to guide development of the front-end. Wireframes are further described in section 2.9.4.

## 2.6 Testing

Tests are a form of quality assurance used in software development. These tests ensure that the application works as expected. There are different kinds of tests to test different components of the software as well as the usability of the application [16].

### 2.6.1 Unit Tests

Unit tests are created to test the low-level components of a system. These tests can be created to test classes, services, or other small functions used throughout the application. Unit tests are usually created with the Arrange-Act-Assert pattern; instances are created and tested to have an expectant value after calling constructors or functions.

### 2.6.2 Integration Tests

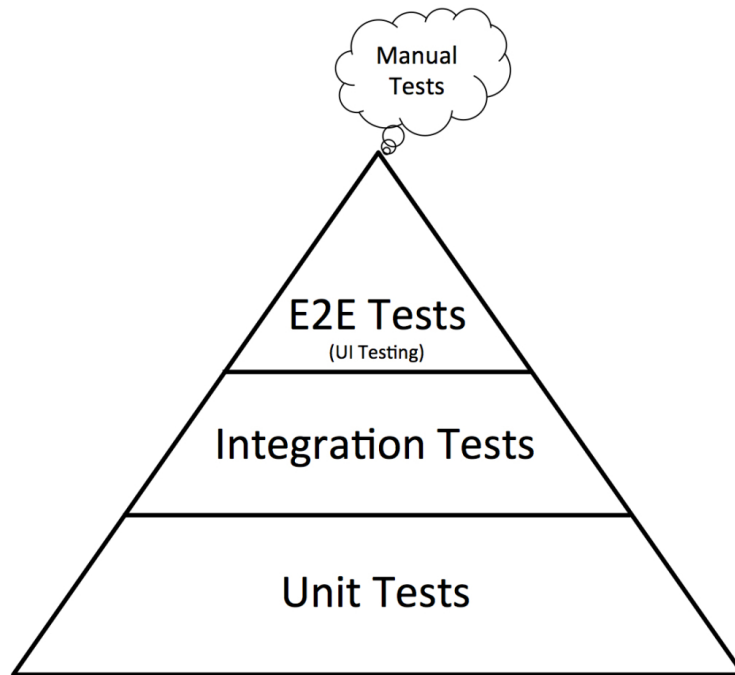
Integration tests are second-level tests that test UI components or the API of an application. The integration tests check that these components behave as expected, and that they can work together [16]. Integration tests also asserts that UI are held to certain criteria like accessibility requirements.

### 2.6.3 End-to-end Tests

End-to-End (E2E) tests are high-level tests that test multiple components of an application, like UI and API together. It will check that a page can be visited, navigated, and handle requests. E2E tests are the most time-intensive test to create and run[17].

### 2.6.4 Test Pyramid

Achieving an appropriate balance between testing and time consumption is important. The testing pyramid, as shown in Figure 2.6.1, provides a helpful framework for understanding the quantities of different tests to make a good balance in a testing plan. Following the test pyramid, unit tests should make up the bulk of the test plan [17]. Unit tests are easy to make and don't take long to run, meaning they are inexpensive in a system. Integration tests are more time-consuming to create and run than unit tests, however, it is still important to ensure that the system functions properly. There should be fewer integration tests than unit tests to keep the balance between tests. E2E tests are the most difficult and time-consuming tests to create [17]. However, they are still valuable for ensuring the system's overall quality. Therefore, a good testing plan should still have a handful of E2E tests.



**Figure 2.6.1:** The test pyramid displays the sought-after quantity of tests for each section [18].

## 2.6.5 Usability Testing

Usability testing involves testing, evaluating, researching a product or service, and analyzing how users interact [19]. Usability tests, also called user tests, are crucial in uncovering potential issues and problems in product design that could negatively impact the product. Observing and questioning a target group as they perform various intended tasks with the product makes it possible to identify less intuitive design decisions and develop better solutions [20].

User tests can be performed on various products as well as on prototypes. Prototype testing allows for early intervention in product development so changes can be made before much effort is put into creating a less-than-ideal design.

### 2.6.5.1 Elements in User Tests

Several methods exist to perform user tests, and various elements are used. Key elements when discussing user tests are as follows:

- **Moderator** - A person who guides the user through the user test, asks the user to perform tasks and answers questions. While not required, user tests are distinguished between moderated and unmoderated and are a key element in user tests [20].
- **Tasks** - A set of realistic actions the user should perform to test the product or prototype. Moderators have a challenging job of learning how to communicate the tasks clearly to the user and avoid misunderstandings and uncertainty about what actions are to be performed, as it can negatively impact the test [20].

- Participant - The test user who should represent a realistic target group for the product or prototype being tested[20].

### 2.6.5.2 Types of User Tests

Some methods used to perform user tests are moderated or unmoderated, in-person or remote, and quantitative or qualitative.

In a moderated user test, a moderator guides participants through predefined actions while answering questions and observing their performance. On the other hand, unmoderated user tests rely on tools such as video recording or eye tracking to observe user behavior. Moderated tests offer the advantage of personalized follow-up questions and the ability to clarify any misunderstandings. However, unmoderated tests allow users to perform in their natural environment and are preferred when collecting data from larger groups [21].

User tests can be conducted in person or remotely. In-person testing involves the moderator and participants in the same room, while remote testing may involve screen-sharing through software. In-person testing allows observing body language and testing on various devices. Still, it can be challenging to set up if participants are in different locations and require travel and equipment. Remote testing is less time-consuming and allows for a larger test group, but it can result in communication issues such as participants being uncomfortable thinking out loud over a call or the test becoming one-sided [22].

Qualitative user tests focus on the user's experience with the product. A moderator would focus on the user's body language and reaction to perform tasks such as eye squinting. Quantitative focuses on how a user performs the given tasks, the success rate, and how long it took to solve the task. When doing quantitative user tests, a larger test group is usually recommended. Qualitative is, therefore, more common as it uses the think-out-loud protocol and does not need a large group of users [23].

### 2.6.5.3 Conducting User Tests

Before conducting user tests, there are many factors to consider. Before the tests, it is important to have a test plan with objectives for the user test. The test plan should also include tasks the users should perform. As previously mentioned, these tasks must be formulated well to minimize misunderstandings that may distort the test results.

When conducting user tests, it is good to have more than one person observing to collect more qualitative data from more than one perspective [24]. The user should also be encouraged to use the think-out-loud method, saying what they think and what they are doing and continuously providing feedback [24]. The moderator should also think about how they speak with the user. There are three primary methods for this; Echo, Boomerang, and Columbo [25]. The Echo method is repeating the last thing the user said in more of an interrogative manner and waits for the user's response[25]. When using the Boomerang method, the moderator asks the user's opinion instead of answering their questions [25]. Lastly, when using the Columbo method, the moderator will intentionally

ask unclear questions with long pauses to let the user fill in [25].

After conducting the user tests, the moderator and the other observers in the user tests should go together and analyze their findings. When an analysis has been made, the team should figure out what changes need to be with the product to address the results of the user tests.

## 2.7 Continuous Integration, Continuous Deployment, and Continuous Delivery

### 2.7.1 Continuous Integration

Continuous Integration (CI) is a practice where developers merge their code changes regularly into a central repository. When the code is merged automated tests and builds are run. Some of the key goals of CI are to improve the quality of the software and code and find bugs and address them quicker [26].

### 2.7.2 Continuous Delivery

Continuous Delivery (CDel) is a practice where the code changes are automatically built, tested, and pushed to a non-production environment, testing, or staging. This allows the developers to manually test and ensure the code is ready for production before it's built and shipped [27].

### 2.7.3 Continuous Deployment

Continuous Deployment (CDep) is a practice where the code changes are automatically tested, built, and released into production [28].

## 2.8 Universal Design

### 2.8.1 Web Content Accessibility Guidelines

Around 1 in 6 people experience serious disability [29], and even more experience disability to a lesser extent. The level of assistance people with disability needs from assistive technologies widely varies. For example, some individuals need screen readers to access web content, while others may need alternative input methods, such as buttons or other devices.

Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C) produced the Web Content Accessibility Guidelines (WCAG) [30] as international standards to ensure that web applications are accessible to everyone, regardless of ability. These guidelines outline minimum requirements for web content to guarantee that it is accessible to all users.

The WCAG is built upon four core principles of accessibility [31]:

- Perceivable - User interface components must be perceivable by all senses so that those with hearing or visual impairments can understand them. This includes providing visual cues for those who are hard of hearing and ensuring that screen readers can read content for those who are visually impaired.
- Operable - User interface components must be operable, meaning that all users can interact with them.
- Understandable - To ensure the content and interface are understandable, clear language and intuitive design must be used.
- Robust - User interface components and the user interface as a whole must be robust, meaning that they are compatible with a wide range of assistive technologies

### 2.8.2 Accessible Rich Internet Applications

The Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) provides a framework and guidelines to make websites more accessible to people who use assistive technologies such as screen readers or alternative input devices [32]. ARIA defines roles and states that can be set as attributes on HTML elements to make it easier for browsers and screen readers to communicate with assistive technology and allow users to navigate the web interface more effectively. Today, ARIA attributes are partially supported in 98.98% of all browsers [33].

### 2.8.3 The Authority for Universal Design of ICT

The Authority for Universal Design of ICT in Norway provides different regulations of Universal Design requirements depending on what sector the system is a part of [34]. A system used in the public sector is required by law to follow WCAG 2.1 [34], while a system used in the private sector is required to follow WCAG 2.0 [34].

## 2.9 Design

### 2.9.1 Design Principles

The layout, appearance, and overall user experience of a product are determined by its User Interface (UI) and User Experience (UX). To ensure a positive user experience, it's important that the UI/UX is user-friendly and easy to navigate. However, creating UI/UX can present several challenges [35]. For instance, rushing through the development process can compromise the quality of research, which is essential in understanding the target audience and catering the design to them. Creating an intuitive UI for complex products or accommodating many features can also be challenging [35].

Don Norman, a highly prolific researcher in user-centered design provides 6 design principles to use for creating an intuitive and user-friendly UI/UX [36].

### **2.9.1.1 Visibility**

According to the first design principle, visibility, all functionality in the user interface should be visible and accessible. This principle can be difficult to achieve in software development when making a product for small screen sizes [36]. To solve this issue, a menu icon is commonly used to indicate that additional functionality is available in the product.

### **2.9.1.2 Feedback**

Don Norman's next design principle is feedback. It is used to inform the user of their location on the product and what is happening [36]. Feedback can be demonstrated through changes in color, size or shadow of elements when they are interacted with, or by displaying a spinner to indicate that a process has begun.

### **2.9.1.3 Affordance**

Affordance is the mapping between how something looks and how it is used [36]. For software development an example is buttons. A button is expected to perform an action and therefore using something that looks like buttons as headings will create confusion as it is not intuitive.

### **2.9.1.4 Mapping**

The design principle mapping dictates that there should be a connection between the appearance of an object and its functionality [36]. For instance, the arrow controls on a keyboard are placed based on their corresponding directional movements and are labeled with arrows pointing in the same direction. In software development, icons used on buttons serve as a more specific example; a plus icon typically signifies addition, while a trash icon deletes.

### **2.9.1.5 Constrains**

The constraints principle refers to visible restrictions in the user interface of a product. This may, for example, be visibly disabling input fields and buttons when they are not allowed to be used [36]. This helps guide the user to how the product should be used.

### **2.9.1.6 Consistency**

The final design principle is consistency. This means that the UI/UX should have a uniform appearance and behavior across the entire product, while also maintaining a consistent layout with similar products. By doing so, the user experience becomes more intuitive and user-friendly [36].

## **2.9.2 Iterative Design Process**

The iterative design process involves continuously improving the product. This process can begin at any stage of development, but typically starts with creating a prototype and conducting user tests, and making design adjustments based on the feedback received [37]. Using the iterative design process allows for improvements to be made throughout the



development cycle. This approach also encourages frequent testing and demos which is a great benefit for clearing up misunderstanding between client and developing teams and ensuring the system requirements meets the user's needs [37]. It may be advantageous to perform the grunt of the design work and more iterations during the prototype stage, as it is easier and less time-consuming to make changes at that point.

### 2.9.3 Design Guidelines

Design guidelines, also known as style guides, are used to ensure a consistent look and feel for products and specifications in a design [38]. They are helpful tools for creating uniformity in components and style sheets that do not already exist. Design guidelines may include various sections such as typography, color schemes, layout, and images to name a few. Larger brands and businesses may have more detailed sections regarding the tone used in the text.

### 2.9.4 Wireframes

Wireframes are a useful tool for planning the layout and concept of an application. They are easy to distribute to all stakeholders and inexpensive to change and redo if there has been a misunderstanding of the product's expectations. Wireframes can be split up into two categories, low fidelity, and high fidelity.

#### 2.9.4.1 Low Fidelity

Low-fidelity wireframes are rough sketches of the layout of the product and do not closely resemble a finished product. These wireframes can be made on the go with just a pen and paper or made with software such as Balsamiq or Figma which also can make the wireframes interactive for prototype testing. Wireframes with low fidelity is better to test the flow of the application [39].

#### 2.9.4.2 High Fidelity

High-fidelity wireframes closely resemble the end product. It shows the layout of the product with its desired look, and feel, possibly using design specifications from the design guidelines. High-fidelity wireframes allow the stakeholders to see a realistic version of the product being planned and therefore can provide more meaningful feedback from usability testing [39]. High-fidelity wireframes can be made through mock-ups using software like Photoshop, Gimp or Figma, or through code using HTML and CSS.

## 2.10 Technologies

### 2.10.1 Programming Languages

#### 2.10.1.1 JavaScript

JavaScript (JS) is a widely-used, lightweight scripting language. JS is used mainly for web pages and in various non-browser environments such as Apache, Node.js, and Adobe Acrobat [40]. JS is also a multiparadim programming language; it supports the programming paradigms such as Object-Oriented Programming and functional programming.

### 2.10.1.2 TypeScript

TypeScript (TS) is a library which allows you to write strongly typed JS and gives you the ability to use type interface without additional code. TS also gives you additional syntax to JS for supporting a tighter integration with your editor and catch errors early. It converts to JS and will therefore be able run everywhere JS runs [41].

### 2.10.1.3 HTML

HyperText Markup Language (HTML) is the fundamental component that builds the web. By defining the underlying structure and meaning of web content, HTML enables web pages to be interpreted and rendered consistently across all devices and browsers [42]. When HTML is loaded on the client side a Document Object Model (DOM) is built. It is the DOM that JS manipulates to change the content and data on web pages.

### 2.10.1.4 CSS

Cascading Style Sheets (CSS) is a stylesheet language that defines the appearance of a web page. By providing a separation between content and presentation [43].

### 2.10.1.5 SQL

Structured Query Language (SQL) is a programming language used to manage and manipulate relational databases. Administrators, developers, and analysts use it to manage and analyze relational databases [44].

### 2.10.1.6 Virtual Machines

A Virtual Machine (VM) is a computer resource that uses software instead of a physical computer to run programs and deploy apps [45].

### 2.10.1.7 Reverse Proxy

A reverse proxy is a server sitting in front of other servers or applications forwarding traffic and requests to those servers or applications from the clients. This is mostly done to increase performance and security. [46]

## 2.10.2 Code Version Control

### 2.10.2.1 Git

Git is a version control system built to have high speed and efficiency. It handles every type and size of project and is free and open source. Developed by Linus Thorvald it quickly took hold as it was used for development of the Linux kernel [47][48].

Git organizes code in repositories, that is split into branches where development can be performed in parallel. A repository can be added to a Git server for collaboration and backup. GitHub, Bitbucket, and GitLab are some examples of popular git server providers.

## 2.10.3 Command Languages

### 2.10.3.1 Bash

Bash is a Command Line Interface (CLI) or shell used in Linux and macOS [49].

### 2.10.3.2 Shell

Shell is a computer program that allows you to control a computer's operating system directly using a CLI. [49]

## 2.11 Web Application

### 2.11.1 Roles of HTML, CSS, and JS

There are various ways to create web applications, but the traditional primary building blocks are HTML, CSS, and JS which are discussed in section 2.10. Each of these technologies has different roles and responsibilities in the web application. HTML is meant to define the document, as any report, pamphlet, or essay would be defined. CSS is responsible for the presentation of the HTML document; it is through CSS that for example colors and font sizes are defined. JS can send and retrieve data from an API and can manipulate the Document Object Model (DOM) to update the webpage. For usability and accessibility purposes it is important to try and keep the use of each technology within their domain.

### 2.11.2 Single-page Application

A single-page application is a web application that only loads the HTML, JS, and CSS for the application once and then re-renders the necessary content using JS, instead of the whole page as traditionally done with multiple-page applications[50]. Single-page applications often perform better than multiple-page application as it only needs to load small contents to re-render parts of the document, and multiple-page applications will load whole documents. However, it can be harder to handle routing and state management on a single-page application.

### 2.11.3 Server-side Rendering

Server-side rendering (SSR) also called universal rendering utilizes both server rendering and client-side rendering [51]. Page documents are rendered on the server and after being loaded on the client-side JavaScript is added to the page document to allow for DOM manipulation. When data changes the client will re-render components through fetching data and using the script. Server-side rendering (SSR) has the advantage that the first contentful page load happens quickly and has better search engine optimization. The downside is that it can take longer to become interactive.

In this chapter, we delve into the methods we used for all the stages of development in our project.

## **3.1 Organization**

### **3.1.1 Team**

The project team consists of three students from NTNU in Aalesund, Espen Otlo, Janita Røyseth, and Sakarias Sæterstøl. We decided to take on specialized roles in the project, and so our team had the following role distribution: Espen Otlo has been the lead systems architect and backend developer, Janita Røyseth has been in charge of the design and frontend development of the application, and Sakarias has been the head of DevOps and DB- and server management. Other responsibilities in the project were shared between all team members.

### **3.1.2 Supervisor**

The supervisor for this project has been Anniken T. Susanne Karlsen. She has acted as a guide and mentor for the team throughout the project. She has also participated in several meetings with the product owner.

### **3.1.3 Client & Product Owner**

The client for this assignment is AxBit. AxBit is a software company with approximately 50 employees divided between their three offices in Møre & Romsdal and Poland. They specialize in the creation of tailored solutions and digital transformation.

Franck Chauvel served as AxBit's representative and the Product Owner for the project and has played an instrumental role in guiding the project to its current state.

## 3.2 Project Planning

### 3.2.1 Preliminary Report

We started working on the preliminary report after the initial meetings with our stakeholders, where we in-depth discussed our assignment. In the preliminary report, we first began by explaining our motivation for choosing this assignment, what our stakeholder's project description and outcome goals of this bachelor assignment are, and what impact goals we wished to achieve for this bachelor thesis. We also explained the organization and roles of this project's team members and stakeholders. Next, we defined the main activities to be done, the different milestones for the project, and how quality assurance would be performed. Lastly, we created a risk assessment for the execution of our project. The preliminary project plan can be found in appendix A.

### 3.2.2 Gantt Diagram

For planning our project's timeline, we created a Gantt diagram. A Gantt diagram is a diagram that shows activities and the timeline for the tasks from start to finish. Each team member added their expected activities and the timeline for them.

### 3.2.3 Work Contract

While planning the project, we also created a work contract where we internally defined the responsibilities of each team member and agreed on different procedures like meeting invitations, document handling, version control, and how to interact with each other.

### 3.2.4 User Requirements

When deciding upon the method of estimation to use for this project, we first saw the need to concretize our constraints, and so, together with the Product Owner, we established the following requirements:

1. The method would need to be easy to adapt to and use by Axbit. Axbit currently uses ideal hours as their basic unit of estimation, and work in scrum teams with differentiating specializations. Their previous estimations varied in detail - some projects were broken down into individual tasks, with estimations made per task, whereas others were only estimated on a feature-level. This meant that in order for our application to be easily adaptable by Axbit, our method of estimation would need to:
  - Be flexible in its level of detail
  - Take the teams' specialization differentiation into account
  - Use ideal hours as the basic unit.
2. Our application is meant to be used as a tool both during project development as well as in the planning phase, so our method would need to be able to estimate a project at any stage of development.

3. The method needed to provide a degree of uncertainty to the estimations. A central tendency alone does not give any hints as to the uncertainty, and therefore cannot be used confidently on its own. A means of illuminating the variance of the estimation was necessary to provide this confidence.

### 3.2.5 User Stories

In addition to the requirements listed above, we (in cooperation with the Product Owner) also established the following user stories to help guide our development. In order for this project to be considered a success we would need to achieve the features marked as (MUST).

As a project manager, I want:

1. (MUST) given an existing project estimated by the team, to get a expected delivery date (calendar time estimation).
2. (MUST) given an existing project estimated by the team, to get a cost estimation.
3. (MUST) to see the most likely release plan, the best and the worst (e.g., burn-up chart), possibly including constraints (i.e., budget, deadline, features).
4. (MUST) to create a project with tasks / add / delete.
5. (MUST) to assign tasks to specific people or group of persons.
6. (MUST) to manage availability for the team (e.g., vacations, illness, part-time allocation, etc.).
7. (MUST) to define cost parameters for the team (e.g., hourly rate, profit margin).
8. (MUST) to set the starting point of the project.
9. (SHOULD) to refine a task into a finer-grain task flow, that can be estimated by developer.
10. (SHOULD) to update the plan with actual project execution data (task started, task ended).
11. (SHOULD) to identify delivery (features) in the plan.
12. (SHOULD) to mark feature as Must-have, could-have, should-have or won't have
13. (COULD) to see the prediction of a specific task, possibly depending on a given starting date.
14. (COULD) to import availability from the Axbit ERP system.
15. (COULD) to decide how t-shirt sizes map to ideal hours.
16. (COULD) to decide how story points map to ideal hours.
17. (COULD) to export existing projects (tasks) from external sources, say JIRA, Asana, etc.

18. (COULD) As the project progresses, to see the current estimation bias and absolute error.
19. (COULD) to see the utilization of staff, who is hiding sleeping, who is overloaded.
20. (COULD) to detect bottleneck tasks.
21. (COULD) to import existing projects (tasks) from external sources, say JIRA, Asana, etc.

As a developer I want:

22. (MUST) to estimate task, in ideal hours.
23. (SHOULD) to be able to say when I start and when I am done with a task.
24. (COULD) to see my own estimation bias and absolute estimation error.
25. (COULD) to estimate task durations in story points.
26. (COULD) to estimate task duration in t-shirt sizes.

## 3.3 Management Tools

### 3.3.1 Teams

We used Teams for meetings and collaboration within the group and with our stakeholders. We chose to use Teams our stakeholders also had access. Teams is a collaboration tool part of the Microsoft Office 365 bundle. Teams have a wide variety of functionality like chat, meetings, voice calls, wiki, and shared team folders for collaboration on documents [52].

### 3.3.2 Confluence

We used confluence to centralize our system documentation and to collaborate with our product owner. Confluence is a collaboration tool for creating a wiki, which has pages with documentation [53].

### 3.3.3 Overleaf

We used Overleaf for writing our thesis. Overleaf is a collaboration tool for LaTeX documents. LaTeX allows more explicitly marking elements making formatting easier. We decided to use Overleaf because we had a better experience using Overleaf in previous courses.

### 3.3.4 GitHub

GitHub is a platform used for hosting code and for collaboration and version control. [54] We used GitHub to store, organize and keep control of issues that we needed to work on.

To make sure that we did not implement any breaking changes to the code, we all made a branch for the issue we were working on. This allowed us to work with one code base without any code-breaking of others' code. When we are done with the issue we make a PR to check for conflict with the code on the development branch.

## 3.4 Developmental Tools and Applications

### 3.4.1 Figma

Figma was used to create wireframes and diagrams. Figma is a collaboration tool for creating prototypes, diagrams, and many other types UI based tools for brainstorming and planning [55]. Figma offers premium functionality for students. We were already well-versed with Figma from our previous group projects together, so Figma was a natural choice.

### 3.4.2 Lighthouse

We used Lighthouse to check our application for issues, particularly looking out for accessibility issues. Lighthouse is an open-source development tool that checks the performance, accessibility, and search engine optimization of web pages [56].

### 3.4.3 VS Code

Visual Studio Code (VS Code) is a free source code editor that is powerful but lightweight and built on open source. Out of the box, it has support for TypeScript, JavaScript, and Node.js. The vast collection of extensions for other runtimes and languages makes it one of the most used editors [57]. We decided to use this as our common editor for easily being able to help debug errors, both in code and other issues that might come up.

### 3.4.4 Docker

Docker is a platform for developing, shipping, and running applications. It allows you to build a container with your application to ship and run on any other platform. This makes it easy to ensure the application runs on the server [58]. Docker makes it easy to spin up and down and ship new builds. It will run on the client's server without any issues. This is the reason we chose to use Docker for our application.

### 3.4.5 pgAdmin

pgAdmin is a database management tool built for PostgreSQL. It can run as a web application making it easy to access from anywhere [59]. We choose to use this to have a standard tool for managing the database for our application.



### 3.4.6 Traefik

For our reverse proxy, we are using Traefik. It's an open-source Edge Router that receives all requests sent to our server and directs the request to the right service. It is natively compliant with Docker, which makes it easy to set up with our services [60].

### 3.4.7 Make

Make is an automation tool to build libraries and executable programs from source code. It utilizes a Makefile file to specify what and how to build [61].

### 3.4.8 Vite

Vite is a platform-agnostic tool for building web applications quickly. It pre-bundles dependencies and delivers source code over native ESM, allowing faster server start times than JavaScript-based bundlers. Because of its speed and out-of-the-box support for TypeScript, we saw it as a good fit for our application.

### 3.4.9 Vitest

Vitest is a framework for unit testing created and maintained by Vue and Vite team members. We chose to use it for unit tests in our application because it utilizes Vite's strengths to provide faster run times and requires little configuration.

### 3.4.10 Storybook

We used Storybook for component documentation and integration tests. Storybook is a tool used for UI documentation and testing. It creates a UI for component documentation and allows for various tests, like interaction tests and accessibility tests [62]. To set up component documentation in the storybook UI, so-called stories must be defined. Stories are files next to your components that document their arguments and what interaction should be done on the component.

### 3.4.11 Playwright

We utilized Playwright for executing E2E tests. Playwright is a tool developed by Microsoft. It uses a Chromium browser that connects to the internet and performs predefined interactions on a live website [63]. We decided to use Playwright for E2E tests because it is a popular and well-established tool that integrates well with VS code since Microsoft developed both.

### 3.4.12 Package Manager

For package management we have used both npm and pnpm. They are both very popular and similar.

#### 3.4.12.1 npm

npm is the world's largest software registry. It's used to store packages to share and borrow to and from other open-source developers [64].

### 3.4.12.2 pnpm

pnpm, short for Performant npm, is a fast and disk space-efficient package manager. It utilizes npm but is up to 2x as fast. The main difference is the way it stores the packages; pnpm stores a package only once. If you use the same package in 100 projects, it will only store it once. If you have a project #101 with an updated version of that package, pnpm will only store the file changed and reuse the same package as for the 100 other projects [65].

### 3.4.13 GitHub Actions

GitHub Actions is an automation tool for CI/CD provided by GitHub. It allows for workflows to be run on predefined events. We have utilized this for our project to run automatically when a PR is created and when it is merged to the development or main branch. For configuring the workflows, we have used YAML (3.6.2) files in a .github folder. GitHub parses this folder automatically and starts the workflow.

## 3.5 Framework, Libraries, Programming- and Scripting Languages

Being an experienced software company, Axbit is well-versed in a variety of languages and constantly has to try out new ones, therefore, there were hardly any limitations to what languages and frameworks we could use.

### 3.5.1 Svelte

Svelte is an open-source front-end JS framework aiming to simplify web development by shifting the work to the compile step when you build your app instead of the browser [66]. Svelte is component-based and is used to create single-page applications.

### 3.5.2 SvelteKit

SvelteKit is a framework for building full-stack server-side rendered web applications with Svelte [67]. It combines Svelte's component-based architecture with server-side rendering to provide a streamlined development experience. We used SvelteKit as we wanted to use a reactive framework to create a Single-page application. Our Product Owner requested the back end to be implemented using Python or JavaScript. Because of our inexperience with Python, we utilized SvelteKit's full-stack capabilities.

### 3.5.3 Typescript

We used Typescript for both front-end and back-end development together with SvelteKit. Typescript is a programming language that is strongly typed and based on Javascript, a multiparadigm scripting language. Because Typescript is typed it is easier to keep the code type-safe and to debug code when issues occur. Typescript and Javascript both support object-oriented programming, which we had planned to use for our system architecture.

### 3.5.4 SCSS

Sassy CSS (SCSS) was used to create the global styling and component styling in the application. SCSS is a CSS preprocessor scripting language that extends the capabilities of CSS [68]. With its support for nested selectors, SCSS provides improved readability and it also allows for more advanced functionality. Additionally, regular CSS can still be used for anyone who may not be familiar with SCSS.

### 3.5.5 Carbon Design Systems

We used Carbon Design Systems' component library and icons. Carbon Design System is a design system developed by IBM. Using a component library allowed us to focus more on developing functionality crucial to the concept of our application.

### 3.5.6 PostgreSQL

PostgreSQL is a powerful open-source object-relational database. It extends the SQL language and adds many features that safely scale and store the most complicated data workloads. We used PostgreSQL as our database. This mostly because we have used it for projects before and because it's open source.

## 3.6 Data and Configuration

### 3.6.1 JSON

JavaScript Object Notation (JSON) is a human-readable lightweight data-interchange format. JSON is easy both for machines to generate and parse and for humans to read and write [69]. We used JSON to pass data to forms.

### 3.6.2 YAML

YAML Ain't Markup Language, is one of the most used languages for configuration files, ending with either `.yaml` or `.yml`. YAML is easy to read as it's a human-readable data serialization language. Serialization means data structures can be converted, translated, and wrapped up in another format. The new format can then be stored or transmitted to a different application or service. As YAML is human-readable and has intuitive syntax it's widely used as the format for writing configuration files for applications, DevOps tools, and programs [70].

### 3.6.3 DTO

A Data Transfer Object (DTO) is a design pattern used in objective-oriented programming, and its main purpose is to encapsulate data and transport it efficiently between different parts of a system. They are typically used for moving data between layers or components of an application, such as between a client, server, and database. DTOs focus solely on data transfer and should not contain any business logic, validation rules, or behavior associated with the data.

## 3.7 System Documentation

### 3.7.1 ER Diagram

We began planning the database structure by creating a simple sketch of a database with what we thought we needed to store. During the development process we discovered that our sketch was not sufficient so we made an ER diagram to have a better understanding of database set up.

### 3.7.2 Class Diagram

In the initial planning stage of the project we created a class diagram to help us plan and design our application. A class diagram is a type of UML diagram, as described in section 2.5.3. The class diagram helped us to communicate, both within the team and with the Product Owner. It was continuously updated as the system evolved, and thereby remained useful throughout the project development.

### 3.7.3 Source Code Documentation

Because we were developing a demonstrator which Axbit intended to continue the development of, thorough documentation of the source code was crucial. As part of development of the application, three forms of source code documentation were used, namely JSDoc, inline comments and Svelte component documentation.

#### 3.7.3.1 JSDoc

As part of our source code documentation we used JSDoc, which is a documentation system for JavaScript, to automatically generate documentation that provides an easy-to-navigate interface for understanding the structure and purpose of the code. The use of JSDoc for our functions and class definitions made it easier for the team to collaborate when developing the application.

#### 3.7.3.2 Inline Comments

Our application contains some functions and methods that are particularly large or complex. For these cases, we wrote inline comments to explain their processes step-by-step, to make them easier to understand and maintain.

### 3.7.4 Svelte Component Documentation

We documented our components using the Svelte Documentation. Svelte Component documentation is denoted by an HTML comment starting the first line with a `@component`. We made sure to document each prop, event, and slot for each component.

## 3.8 Design

### 3.8.1 Wireframing

To ensure that our system’s design aligned with our stakeholder’s expectations, we started the design process by creating wireframes. Wireframes provide an inexpensive way to visualize a system’s layout and functionality. This allowed us to quickly create and iterate on design concepts without investing too much time or resources. While deciding on the first conceptual layouts of the application, we kept in mind Don Norman’s design principles and other applications our client frequently used to ensure a familiar, intuitive layout our client. Using wireframes to show our stakeholders and doing user tests on wireframes for the first test iteration, we could make changes early in the design process when changes were less time-consuming and affordable.

#### 3.8.1.1 Figma

Figma was used to create high-fidelity wireframes that closely resemble the end product. By using high-fidelity wireframes, confusion about the end product would be limited, allowing for better user test feedback. Figma also lets wireframes be interactive, which was a benefit for testing on wireframes. Because Figma is a SaaS, it only requires an internet connection and a browser for prototype testing.

### 3.8.2 Design Guidelines

We created design guidelines before beginning front-end development to ensure our web application’s look and feel stayed consistent. To ensure a cohesive look and feel, design guidelines hold specifications for specific design elements such as spacing, font, color, border-radius, and shadows. They make it easier to develop a coherent design when multiple developers are working together.

### 3.8.3 Accessibility

The application was made to adhere to WCAG 2.0 requirements since it is a web application for a private business. Accessibility was considered while designing the layout, look, and feel. To ensure that our color choices were acceptable regarding the contrast required by WCAG 2.0, we first checked before deciding on what primary color and what background color to use. To make our application usable for disabled people, ARIA attributes have likewise been a part of our development process. Throughout the development process, Lighthouse was regularly used to ensure that the application met the WCAG 2.0 requirements and that we used ARIA attributes correctly.

## 3.9 Testing Methodology

### 3.9.1 Unit Tests

We created unit tests from the beginning of the development process to ensure that the low-level components behaved as expected. The vitest testing framework was used to create unit tests.

### 3.9.2 Integration Tests

We used Storybook to create integration tests by using the Storybook's accessibility test plugin and by writing interaction tests.

### 3.9.3 End-to-end Tests

We used Playwright to create E2E tests. The E2E tests would run on the developer's machine or the development build.

### 3.9.4 Usability Tests

We performed usability tests on wireframes and the Minimum Viable Product. We only performed qualitative, moderated remote user tests where the participants shared their screens while testing. At least two team members were present at each user test to assist in taking notes of the participants' performance. Before each user test, a test plan was created to analyze the results and compare the user tests of that iteration. The test plan would contain predefined tasks the user was to perform, and the plan was used to take notes on how the user performed and note any potential feedback from the user. After each user test, the team analyzed and discussed the results, deciding on the appropriate changes to respond to the feedback received.

#### 3.9.4.1 Prototype Testing

We created a high-fidelity wireframe for prototype testing in Figma, which made it readily available for remote testing. The participant shared their screen while clicking through the Figma prototype and performing actions asked by a moderator.

#### 3.9.4.2 Alpha Testing

We performed a second round of user tests when the MVP was completed. The users shared their screens and entered our applications' development build to test.

## RESULTS

In this chapter, we will describe in detail the results of the project, including findings from our research and the resulting demonstrator application. The demonstrator can be accessed at <https://savvyest.io>.

### 4.1 Method of Estimation

As a result of our research of estimation theory, we have devised a method for project estimation. The method borrows traits from various existing methods currently used by project managers in Axbit. This makes the method easier for Axbit to adopt it as their new standard for project estimation. Following is a detailed explanation of the estimation method we have used in our application.

The estimation method is intended to be used in a collaborative planning session by the project team, and consists of the following steps:

1. **Work breakdown structure** - Start by defining the scope of the project by identifying the key features that will be included. Segment these features into tasks and collaboratively assign them three-point estimates using Planning Poker. If a task has estimates larger than two weeks, break it down into smaller tasks. Repeat this until no tasks have estimates larger than two weeks (as per PERT's 80-hour rule mentioned in section 2.3.1).
2. **Establish relationships** - The completion of some tasks or features may be dependent on the prior completion of other tasks or features. These relationships must be mapped out in order to create a valid workflow.
3. **Assign team members** - Assign the tasks to the member(s) of the team best suited for their completion.
4. **Simulate** - Use the Monte Carlo method to generate multiple iterations of the project and simulate the development of each iteration, creating a joint probability distribution.

The simulation step is part of our application, and further described in section 4.5.2.

## 4.2 Continuous Integration and Delivery

### 4.2.1 CI/CD Process Overview

We have used Continuous Integration and Continuous Delivery (CICD) in our project. This has been done with the use of GitHub Actions (3.4.13). The repository has workflows that are run on PR and merge to the development, or the main branch. Linting and check workflow are run when the PR is made and the deployment workflow runs when the PR is merged.

### 4.2.2 Workflows

**Lint and check workflow** was performing checks for formatting and to check that there were no syntax errors.

**Deploying to development environment** performs an SSH logging to the server and running commands to set up the development environment using the newest code.

**Deploying to production environment** performs a SSH logging to the server and running commands to set up the production environment using the newest code.

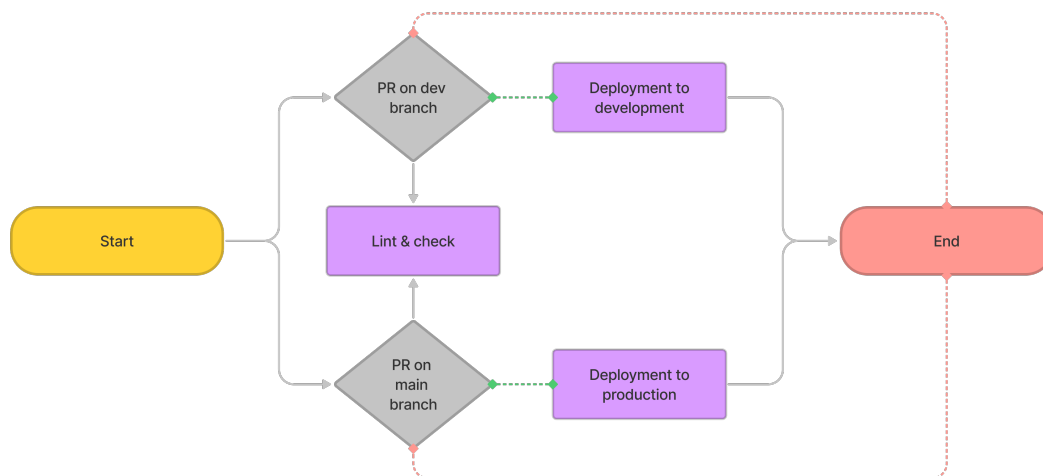


Figure 4.2.1: The Github Action workflow for CI/CD



```
Code Blame 32 lines (29 loc) · 670 Bytes

1  name: CI lint & check
2
3  on:
4    pull_request:
5      branches:
6        - main
7        - dev
8
9  jobs:
10   Lint:
11     runs-on: ubuntu-latest
12     steps:
13       - uses: actions/checkout@v3
14       - uses: pnpm/action-setup@v2.2.4
15       - uses: actions/setup-node@v3
16         with:
17           node-version: '16.x'
18           cache: pnpm
19       - run: pnpm install --frozen-lockfile
20       - run: pnpm lint
21
22   Check:
23     runs-on: ubuntu-latest
24     steps:
25       - uses: actions/checkout@v3
26       - uses: pnpm/action-setup@v2.2.4
27       - uses: actions/setup-node@v3
28         with:
29           node-version: '16.x'
30           cache: pnpm
31       - run: pnpm install --frozen-lockfile
32       - run: pnpm check
```

Figure 4.2.2: CI

```

Code Blame 23 lines (20 loc) · 756 Bytes Raw
1 name: Auto deploy to dev
2
3 on:
4   push:
5     branches:
6       - dev
7   workflow_dispatch:
8
9 jobs:
10  Deploy_to_dev:
11    runs-on: ubuntu-latest
12    steps:
13      - name: Install SSH key
14        uses: shimataro/ssh-key-action@v2
15        with:
16          key: ${ secrets.SSH_KEY }
17          known_hosts: ${ secrets.KNOWN_HOSTS }
18          if_key_exists: fail # replace / ignore / fail; optional (defaults to fail)
19
20      - name: Pull project
21        run: ssh ${ secrets.SSH_USER }@${ secrets.SSH_HOST } "cd CES && git checkout dev && git pull"
22      - name: Deploy project
23        run: ssh ${ secrets.SSH_USER }@${ secrets.SSH_HOST } "cd CES && git checkout dev && sudo docker compose -f docker-compose.yml -f
docker-compose.dev.yml up -d ces-dev --build"

```

Figure 4.2.3: CD to development environment

## 4.3 Environment Files

We have used a `.env` file for environment variables. This file is used by Docker, to set up the containers and by the application it self for connecting to the database.

```

1   # Postgres
2   POSTGRES_HOST=
3   POSTGRES_PORT=
4   POSTGRES_USER=
5   POSTGRES_PASSWORD=
6   POSTGRES_DB=
7   DB_SSL=
8
9   # PGadmin
10  PGADMIN_CONFIG_ENHANCED_COOKIE_PROTECTION=
11  PGADMIN_DEFAULT_EMAIL=
12  PGADMIN_DEFAULT_PASSWORD=

```

Figure 4.3.1: The `.env.example` file used in our project.

## 4.4 Documentation

### 4.4.1 Code Documentation

The back-end of the application is documented inside the source code as JSDoc and by inline comments, and by UML class diagrams.

#### 4.4.1.1 JSDoc

All classes and methods in the application back-end has been documented by JSDoc, explaining their structure and functionality. Displayed below is the JSDoc for the `doWork()` method in the `Assignee` class.

```
(method) Assignee.doWork(project: Project, date: Date, milliseconds: number): Promise<number>  
  
This is the main method in the simulator, in which the assignee does the  
following while they have time to spare and are not on leave:  
  
1. Picks a task to work on,  
2. Registers their work contribution to the task NB. If no task is available,  
   registers time as idle.  
  
@param project — Project the project to work on.  
@param date — When to start working.  
@param milliseconds — The duration of the work.  
@code
```

**Figure 4.4.1:** Code example: JSDoc of the `doWork()` method of the `Assignee` class.

#### 4.4.1.2 Inline Comments

In addition to JSDoc we also added inline comments in the more complex methods and functions to increase readability and maintainability. Following is an example of the `doWork()` method of the `Assignee` class.

```

public async doWork(project: Project, date: Date, milliseconds: number): Promise<number> {
  // If this assignee is on leave - do nothing.
  if (this.isOnLeave(date)) return 0;

  // Work day start
  let myDate = new Date(date.getTime());
  let msRemaining = milliseconds;
  let tasksAvailable = true;
  // While there is time left for work and available tasks to work on
  while (msRemaining > 0 && tasksAvailable) {
    // If not currently working on a task, find a new task to work on.
    if (this.currentTask === undefined || this.currentTask.getActualFinish() !== undefined)
      await this.selectTask(project, myDate);
    // If a task is available to be worked on
    if (this.currentTask !== undefined) {
      // Work on the task until it is finished, or work day is over.
      const timeWorked = Math.min(
        this.currentTask.getRemainingMilliseconds(),
        msRemaining * this.levelOfAssignment * this.employee.getEfficiency()
      );
      myDate = new Date(myDate.getTime() + timeWorked);
      this.currentTask.setRemainingMilliseconds(
        this.currentTask.getRemainingMilliseconds() - timeWorked
      );
      msRemaining -= timeWorked / this.levelOfAssignment / this.employee.getEfficiency();
      this.workedTime += timeWorked;
      // If the current task is finished
      if (this.currentTask.getRemainingMilliseconds() === 0) {
        this.currentTask.setActualFinish(myDate);
      }
    } else {
      // No tasks are available - register remaining time as idle.
      tasksAvailable = false;
      if (project.getUnfinishedTasks().length > 0) {
        this.idleTime += msRemaining;
      }
    }
  }
  return tasksAvailable ? 0 : 1;
}

```

Figure 4.4.2: Code example: doWork() method of the Assignee class has inline comments to clarify the method process.

#### 4.4.1.3 Class diagrams

The class diagram of the entity models can be seen in Figure 4.5.1 and the diagram for the service classes is in Figure 4.7.1.

## 4.4.2 Component Documentation

The Svelte components created for the application are documented in the code as well as through Storybook.

### 4.4.2.1 Component Code Documentation

Figure 4.4.3 displays the code documentation for the `SortableList` component. The documentation covers the name, description, and example of how to implement the component, and if applicable, it will document the props, events, and slots. Figure 4.4.4 showcases how the component code documentation is viewed when implemented.

```

<!-- @component
  @name SortableList
  @description A list of items which can be sorted by dragging and dropping.
  @example
  ```html
  <SortableList
    readonly={false}
    showControls={true}
    ghostImageQuery='article'
    on:reorder={handleReorder}
    bind:items
    let:item>
    <p>{item.name}</p>
  </SortableList>
  ```

  @slots default The default slot, takes an item prop, which is the item to be rendered in the list item.
  @prop {any[]} items - The items to be rendered in the list.
  @prop {string} ghostImageQuery - The query selector for the element to be used as the ghost image when dragging.
  @prop {boolean} showControls - Whether to show the controls for moving items up and down.
  @prop {boolean} readonly - Whether the list is readonly or not.
  @fires reorder - Event fired when the list has been reordered, new list is passed as event detail.
  -->
<ul class={{restProps.class || 'list'}}>

```

Figure 4.4.3: Code example: documentation of the `SortableList` component in code

```

if ($currentProject) searchedFeatures = [...$currentProject.getFeatures()];
(alias) class SortableList
  import SortableList
  @name — SortableList
  @description — A list of items which can be sorted by dragging and dropping.
  @example
  <SortableList
    readonly={false}
    showControls={true}
    ghostImageQuery='article'
    on:reorder={handleReorder}
    bind:items
    let:item>
    <p>{item.name}</p>
  </SortableList>
  @slots — default The default slot, takes an item prop, which is the item to be rendered in the list item.
  <SortableList
    readonly={searchedFeatures.length !== $currentProject.getFeatures().length && ready}
    showControls={searchedFeatures.length === $currentProject.getFeatures().length && ready}

```

Figure 4.4.4: Documentation displayed from `SortableList` component documentation

### 4.4.2.2 Storybook Documentation

Storybook documentation is defined through story files denoted by the extension `*.stories.ts`. The story files reside next to their respective components for which they hold documentation. In the story file, the component is documented, and its argument types are

defined. Using the story file, Storybook mounts the components in the Storybook UI. It is possible to change fields and properties through the UI and see how this changes the component. Figure 4.4.5 displays the Storybook UI's documentation and properties for the button element. Figure 4.4.6 shows the button component with properties edited through the UI. The Storybook component documentation can be found at the link: <https://storybook.savvyest.io>

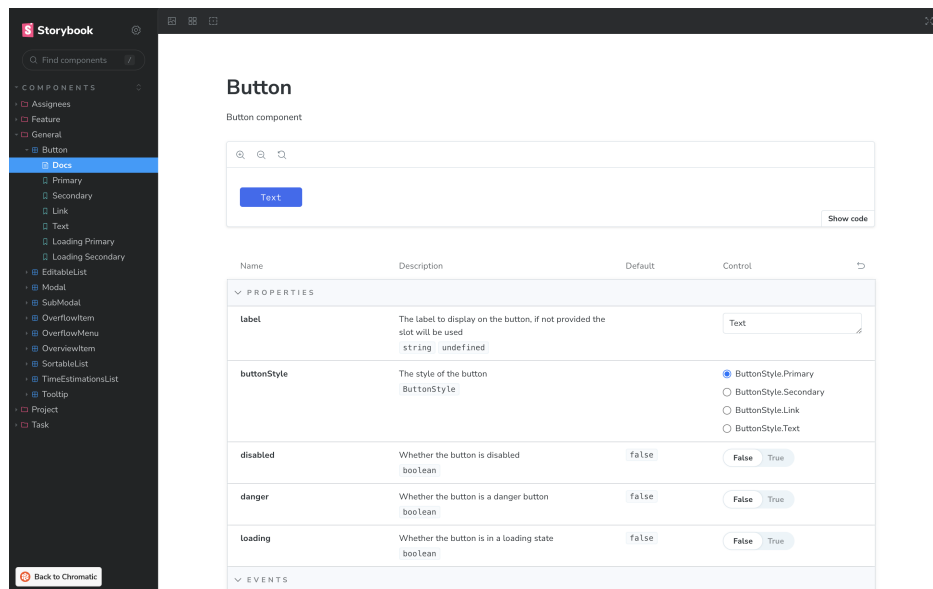


Figure 4.4.5: Button documentation in the Storybook UI

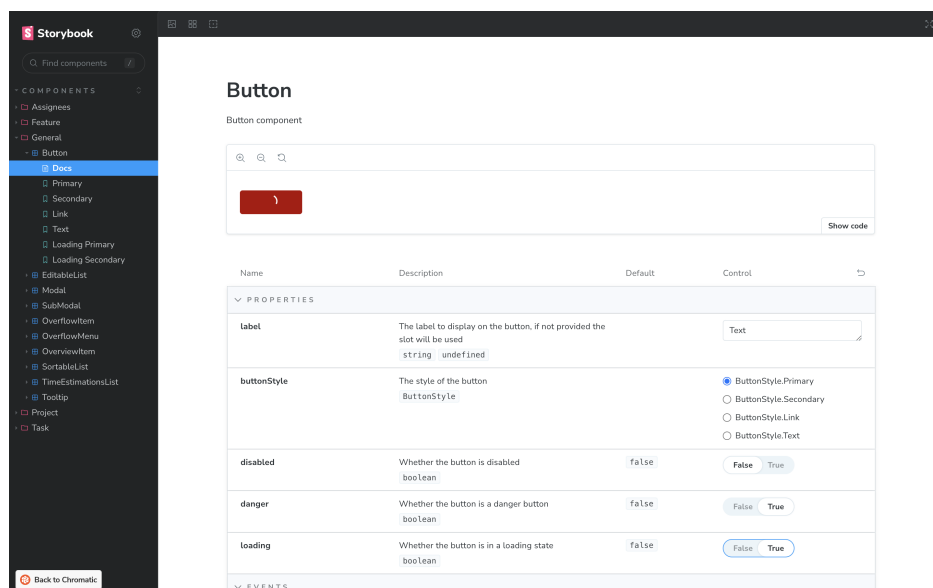


Figure 4.4.6: Storybook Button documentation with edited properties

## 4.5 Backend

### 4.5.1 Entity Classes

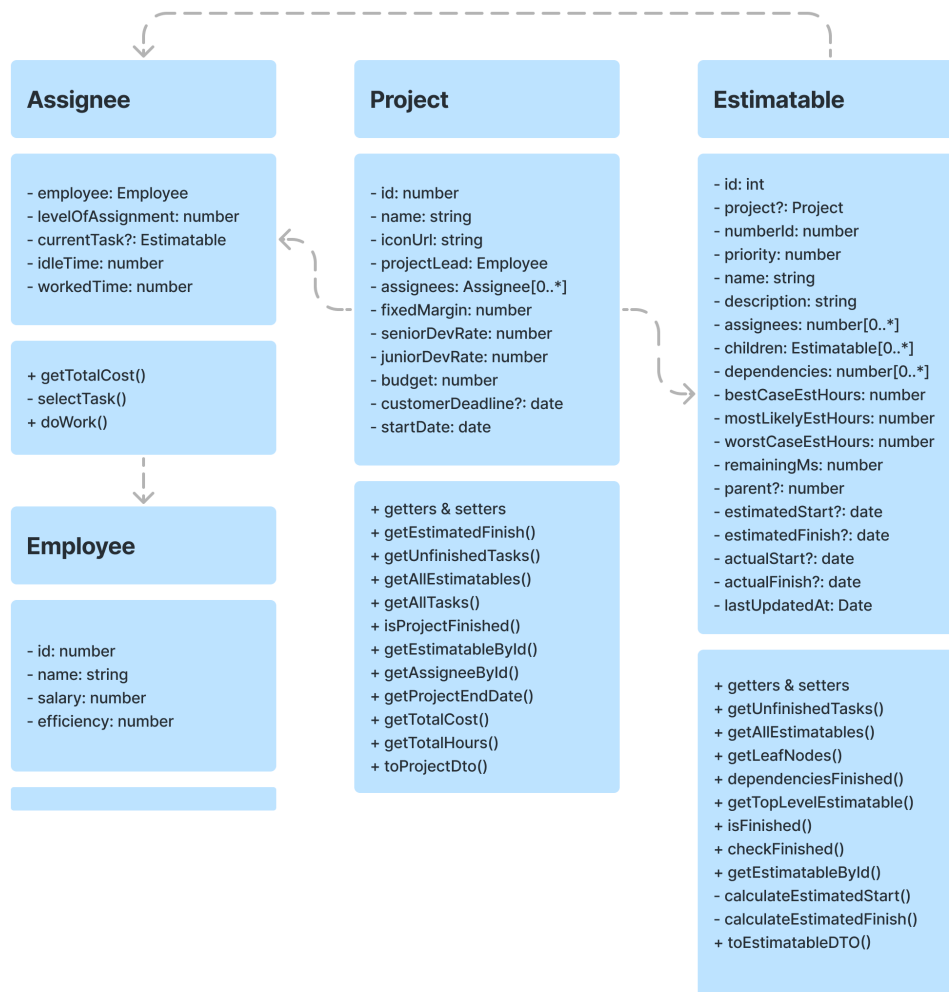


Figure 4.5.1: UML diagram describing the structure of the entity classes.

#### 4.5.1.1 Project

Projects are represented in our application by the `Project` class, which contains information on what the project entails, such as who will be leading the project and who will be working on it, what needs to be done in order to finish the project, as well as hourly rates and cost margin towards the customer.

#### 4.5.1.2 Estimatable

A deliverable, milestone, feature or task all share several traits; they can all be estimated, they all represent work in a project that needs to be completed, and they all may be dependent on other tasks/features, etc. to be completed before work on them can be commenced. Therefore, we have decided to represent these in the application as an `Estimatable`.

An `Estimatable` may be broken down into any number of sub-`Estimatables`, and may have any number of other `Estimatables` as dependencies. As the project is mapped out

with Estimatables, it will take on a tree-like structure, with a root node (the project itself) and one or, most likely, several leaf nodes (Estimatables without children). Each leaf node Estimatable has three-point-estimates in ideal hours (best case, worst case, and the most likely scenario), which form the basis of the project estimation. An Estimatable can be assigned to any number of assignees. If it is not assigned to any specific team members, all employees assigned to the project may contribute towards its completion.

#### 4.5.1.3 Employee

The Employee class represents an employee in the company, without being directly tied to any projects. Employees have an efficiency rating, which is the ratio of hours spent developing per work day. Their level of competence also factors into this value. We found in our research [7][8] that this efficiency factor tends to lie between 40-60%. However, our Product owner requested the default value for this to be set to 85% for their initial testing.

#### 4.5.1.4 Assignee

The Assignee class in our application represents an employee's role in a project. An employee assigned to a project has a level of commitment (the fraction of their working hours they will be spending on the project, and also manages their scheduled leaves. This class also forms the backbone of our simulator.

### 4.5.2 Simulation

This section details the simulation process and explains the different levels of the simulator, followed by code examples.

#### 4.5.2.1 Monte Carlo Method

The project to be simulated is first duplicated multiple times. Every duplication also randomizes the estimated ideal hours of each task using a triangular distribution generated from the task's three-point estimates, thereby creating a large number of iterations of the project. The development process of every iteration is then simulated, as described below.

#### 4.5.2.2 Overview

Our simulator has two levels - The project level and the assignee level. The project level manages date tracking and advancement, while the actual project work and progress is handled on the assignee level.



### 4.5.2.3 Project Level

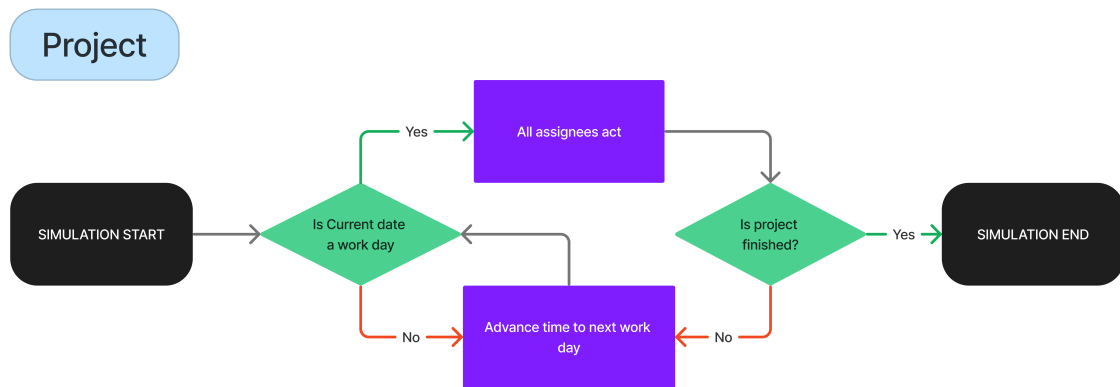


Figure 4.5.2: Describes the simulation process for the project.

The simulation is started on the project level on the date set as the project start date or the latest date a task was registered as either started or finished (in cases where the project is already in development). It will then check if its current date is a work day, and if so it will notify all team members that a new work day has started. Once all team members have acted, it will advance time to the following date and repeat this process, until the project is finished.

```
/**
 * Simulates the development of the given project,
 * and returns the processed project with all relevant
 * values updated.
 *
 * This method is only used by the runNSimulations method in the same class.
 *
 * @param project the {@code Project} to simulate.
 * @returns the updated {@code Project} state after the simulated development process.
 * @throws {@code Error} if the simulation reaches an infinite loop.
 */
private static async runSimulation(project: Project): Promise<Project> {
  if (project.getAssignees().length === 0) throw new Error('No assignees');

  let running = true;
  const date = new Date(project.getStartDate().getTime());
  date.setHours(8);
  date.setMinutes(0);
  date.setSeconds(0);
  while (running) {
    const workDay: boolean = await isWorkDay(date);
    // If the current day is a work day.
    if (workDay) {
      let idleAssignees = 0;
      // Notify all assignees that a work day has started.
      for (let i = 0; i < project.getAssignees().length; i++) {
        idleAssignees += await project
          .getAssignees()
          [i].doWork(project, date, 7.5 * 60 * 60 * 1000);
      }
      // Check if the project is finished.
      if (project.isProjectFinished() === true) {
        running = false;
      } else if (idleAssignees === project.getAssignees().length) {
        // If no assignees are able to find a task to work on:
        throw Error(
          'Error: All assignees are idling! Infinite loop encountered. ' +
          'This is likely due to circular dependencies within the project.'
        );
      }
    }
    // Proceed to next day.
    date.setDate(date.getDate() + 1);
  }
  return project;
}
```

Figure 4.5.3: Code example: Simulation on the project level

#### 4.5.2.4 Assignee Level

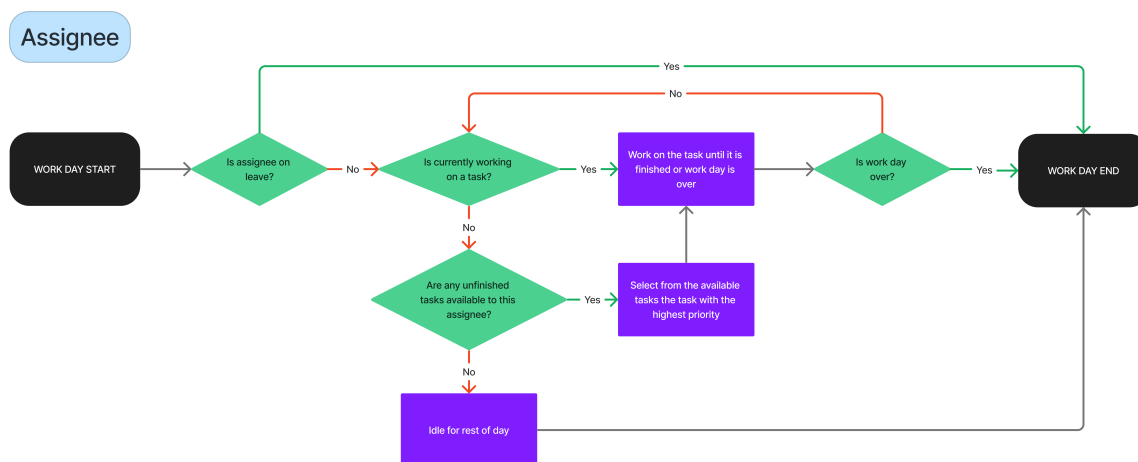


Figure 4.5.4: Describes the simulation process for an assignee.

The Assignee Level manages the actions of each employee working on the project. Every work day, the project will notify all assignees in the project that a new work day has begun, and each assignee will start the following action sequence:

1. The assignee will check if they are on leave, and if so will skip any further actions for the day.
2. Check if they currently have an unfinished task they are working on.
3. If they do not currently have an active task to work on, they will check if there are any unfinished tasks that can be started, that is assigned to them, or open for contributions by all assignees to work on. If multiple such tasks are found, they will select the available task with the highest priority.
4. If they now have a task to work on, they will continue working on this task until either the task is completed or the work day is finished, registering worked hours adjusted by the assignee's level of assignment and efficiency factors. If they finish the task, they will repeat this process from step 3.
5. If not currently working on a task and no unfinished tasks are available to this assignee, they will wait for a new task to become available to them.

```

/**
 * This is the main method in the simulator,
 * in which the assignee does the following while
 * they have time to spare and are not on leave:
 * 1. Picks a task to work on,
 * 2. Registers their work contribution to the task
 * NB. If no task is available, registers time as idle.
 *
 * @param project the project to work on.
 * @param date When to start working.
 * @param milliseconds The duration of the work.
 * @returns {@code number} 1 if the assignee was unable to find work,
 * or 0 if they found work or if they are on leave.
 */
public async doWork(project: Project, date: Date, milliseconds: number): Promise<number> {
  // If this assignee is on leave - do nothing.
  if (this.isOnLeave(date)) return 0;

  // Work day start
  let myDate = new Date(date.getTime());
  let msRemaining = milliseconds;
  let tasksAvailable = true;
  // While there is time left for work and available tasks to work on
  while (msRemaining > 0 && tasksAvailable) {
    // If not currently working on a task, find a new task to work on.
    if (this.currentTask === undefined || this.currentTask.getActualFinish() !== undefined)
      await this.selectTask(project, myDate);
    // If a task is available to be worked on
    if (this.currentTask !== undefined) {
      // Work on the task until it is finished, or work day is over.
      const timeWorked = Math.min(
        this.currentTask.getRemainingMilliseconds(),
        msRemaining * this.levelOfAssignment * this.employee.getEfficiency()
      );
      myDate = new Date(myDate.getTime() + timeWorked);
      this.currentTask.setRemainingMilliseconds(
        this.currentTask.getRemainingMilliseconds() - timeWorked
      );
      msRemaining -= timeWorked / this.levelOfAssignment / this.employee.getEfficiency();
      this.workedTime += timeWorked;
      // If the current task is finished
      if (this.currentTask.getRemainingMilliseconds() === 0) {
        this.currentTask.setActualFinish(myDate);
      }
    } else {
      // No tasks are available - register remaining time as idle.
      tasksAvailable = false;
      if (project.getUnfinishedTasks().length > 0) {
        this.idleTime += msRemaining;
      }
    }
  }
  return tasksAvailable ? 0 : 1;
}

```

Figure 4.5.5: Code example: Simulation on the assignee level

When the simulation completes, it will return the project in it's finished state, with actual start- and actual end dates for all tasks, and the number of hours worked and spent idle for each assignee.

### 4.5.3 Database

The Entity Relation Diagram in figure 4.5.7 describes the relationship between the different entities and linking table. A project have a one-to-many relationship to estimatables and a many-to-many relationship with employees. The relationship with employees can be seen through the linking table `project_assignment_lvl`.

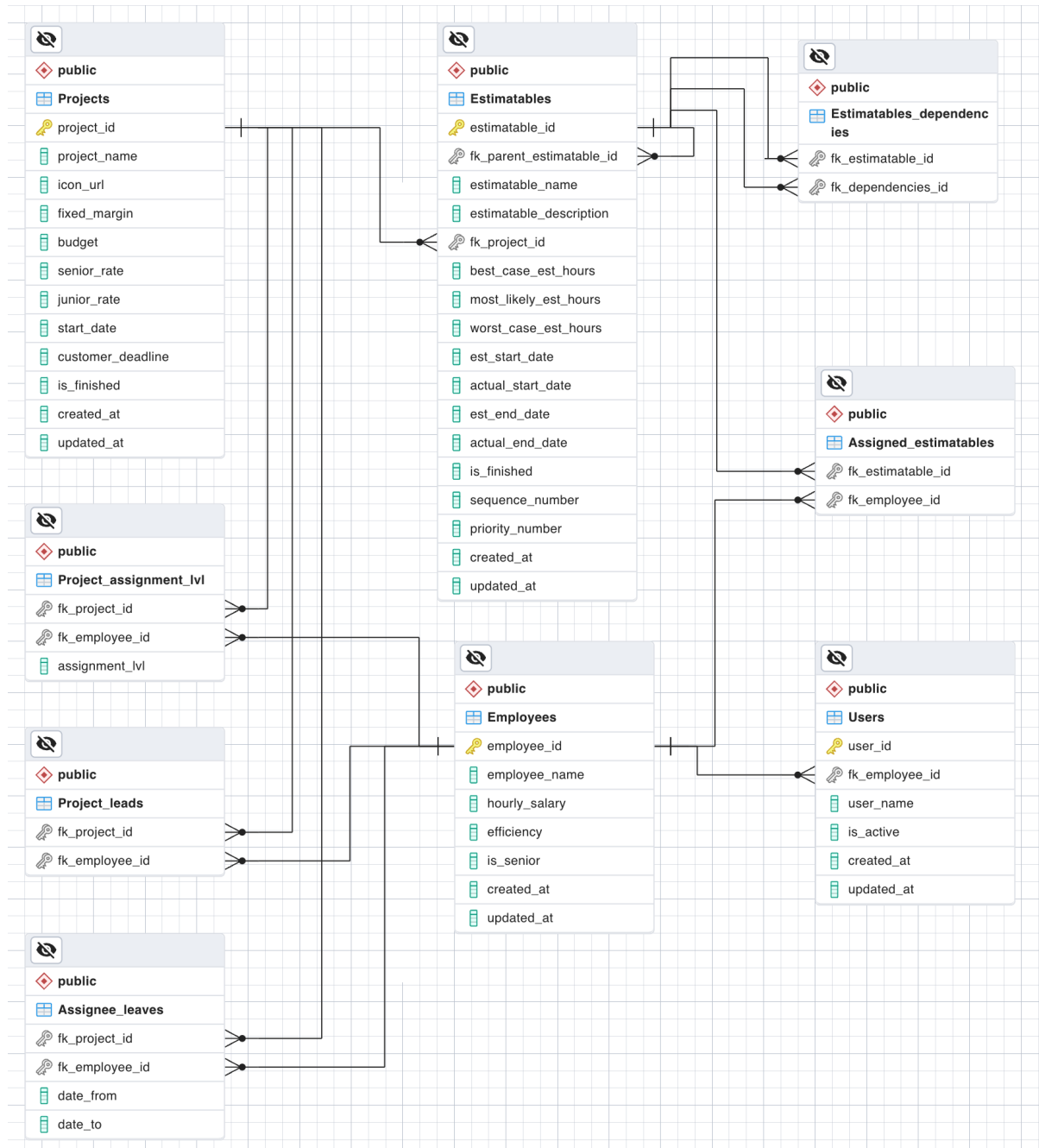


Figure 4.5.6: Entity relation diagram

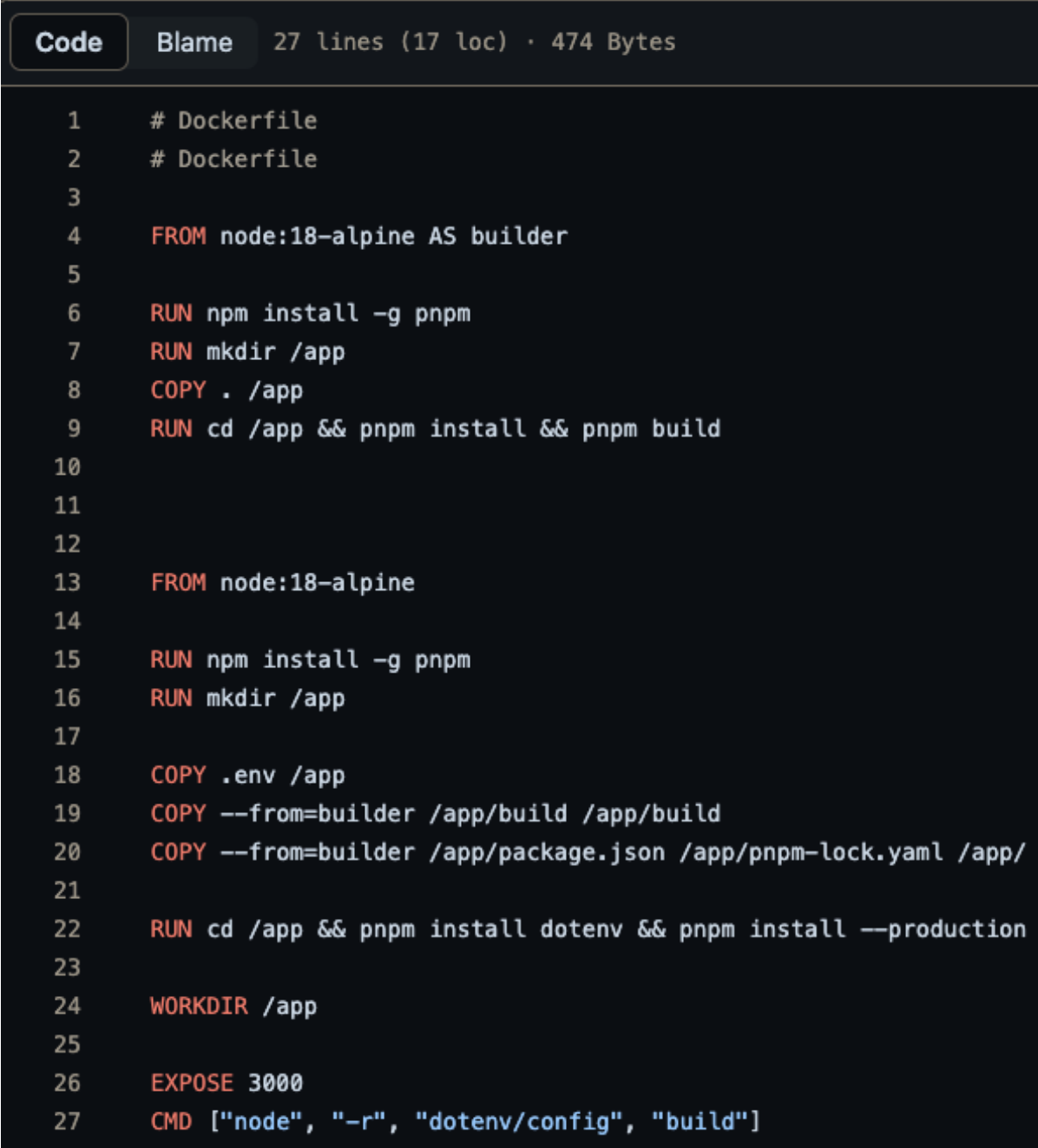
### 4.5.4 Server

For the server NTNU provided us with an Ubuntu server 22.04 from their Openstack [71] solution. The server was setup with a public IPv4 with some restriction on ports. Since

we decided to use Docker for our deployment, we had to set up and configure Docker, Traefik, Node.js and Sveltekit. When this was done the CI/CD workflow took care of the rest of the deployment of the services including the PostgreSQL and pgAdmin container.

### 4.5.5 Docker

Working with Docker we decided to build our own image for the application to be able to have a smaller container deployed. This was done with a Dockerfile.



```
Code Blame 27 lines (17 loc) · 474 Bytes
1 # Dockerfile
2 # Dockerfile
3
4 FROM node:18-alpine AS builder
5
6 RUN npm install -g pnpm
7 RUN mkdir /app
8 COPY . /app
9 RUN cd /app && pnpm install && pnpm build
10
11
12
13 FROM node:18-alpine
14
15 RUN npm install -g pnpm
16 RUN mkdir /app
17
18 COPY .env /app
19 COPY --from=builder /app/build /app/build
20 COPY --from=builder /app/package.json /app/pnpm-lock.yaml /app/
21
22 RUN cd /app && pnpm install dotenv && pnpm install --production
23
24 WORKDIR /app
25
26 EXPOSE 3000
27 CMD ["node", "-r", "dotenv/config", "build"]
```

Figure 4.5.7: Dockerfile

For the actual deployment we set up in total three docker-compose files, docker-compose.yml, docker-compose-dev.yml and docker-compose-prod.yml. This was done for it to be easy

to set up just the PostgreSQL and pgAdmin containers for developing on your own computer, running only the `docker-compose.yml` file. And for the ease use of CD to deploy to our dev environment with the `docker-compose-dev.yml` file and production environment with the `docker-compose-prod.yml` file

Traefik, our reverse proxy, was sat up using Traefik's own guides and example files. And did not need any extra configuration for extra containers.

### 4.5.6 Postgres

PostgreSQL was easy to set up and maintain during the development of this project. We decided to make have a column in our `estimatables` table that would increment for each estimatable added with the same `project_id`. With PostgreSQL this turns out to be easy.

```
122 -- Functions and Triggers
123
124 CREATE OR REPLACE FUNCTION create_project_sequence()
125 RETURNS TRIGGER AS $$
126 BEGIN
127     EXECUTE format('CREATE SEQUENCE project_%s_sequence_number_seq', NEW.project_id);
128     RETURN NEW;
129 END;
130 $$ LANGUAGE plpgsql;
131
132 CREATE TRIGGER create_project_sequence_trigger
133 AFTER INSERT ON "Projects"
134 FOR EACH ROW
135 EXECUTE FUNCTION create_project_sequence();
136
137 CREATE OR REPLACE FUNCTION update_estimatable_sequence_number()
138 RETURNS TRIGGER AS $$
139 BEGIN
140     NEW.sequence_number := nextval('project_' || NEW.fk_project_id || '_sequence_number_seq');
141     RETURN NEW;
142 END;
143 $$ LANGUAGE plpgsql;
144
145 CREATE TRIGGER update_estimatable_sequence_number_trigger
146 BEFORE INSERT ON "Estimatables"
147 FOR EACH ROW
148 EXECUTE PROCEDURE update_estimatable_sequence_number();
149
150 CREATE OR REPLACE FUNCTION drop_project_sequence()
151 RETURNS TRIGGER AS $$
152 BEGIN
153     EXECUTE format('DROP SEQUENCE IF EXISTS project_%s_sequence_number_seq', OLD.project_id);
154     RETURN OLD;
155 END;
156 $$ LANGUAGE plpgsql;
157
158 CREATE TRIGGER drop_project_sequence_trigger
159 BEFORE DELETE ON "Projects"
160 FOR EACH ROW
161 EXECUTE FUNCTION drop_project_sequence();
```

Figure 4.5.8: Code to set up and remove sequence and trigger

With pgAdmin as the management tool we were able to access the database from any computer without the need of downloading any extra software.

## 4.6 Front-end

### 4.6.1 Design

Axbit had no preferences or requirements for the design of the application as we were creating a demonstrator and they would most likely implement their own UI/UX after validating the system. To create the design of the application, Don Norman's design principles and accessibility were considered. In terms of the principle consistency, the design was inspired by two applications used by Axbit, Slack, their main internal communication channel, and Jira, their sprint planner and issue tracker. Jira was also often discussed in the first meetings of how to define features and tasks.

#### 4.6.1.1 Wireframe

The first iteration of the wireframe was started after the initial meetings with the product owner, so we could have time to reiterate the design regarding any misunderstandings. The first draft of the wireframe can be found in appendix B. We created a high-fidelity wireframe to closely resemble the end product and showcase the design more clearly. After three iterations, where each iteration was displayed to the product owner, the wireframe was ready to be tested through usability tests. The second draft of the wireframe can be found in appendix C. The result of the wireframe after the usability test is showcased in Figure 4.6.1 and in appendix D.



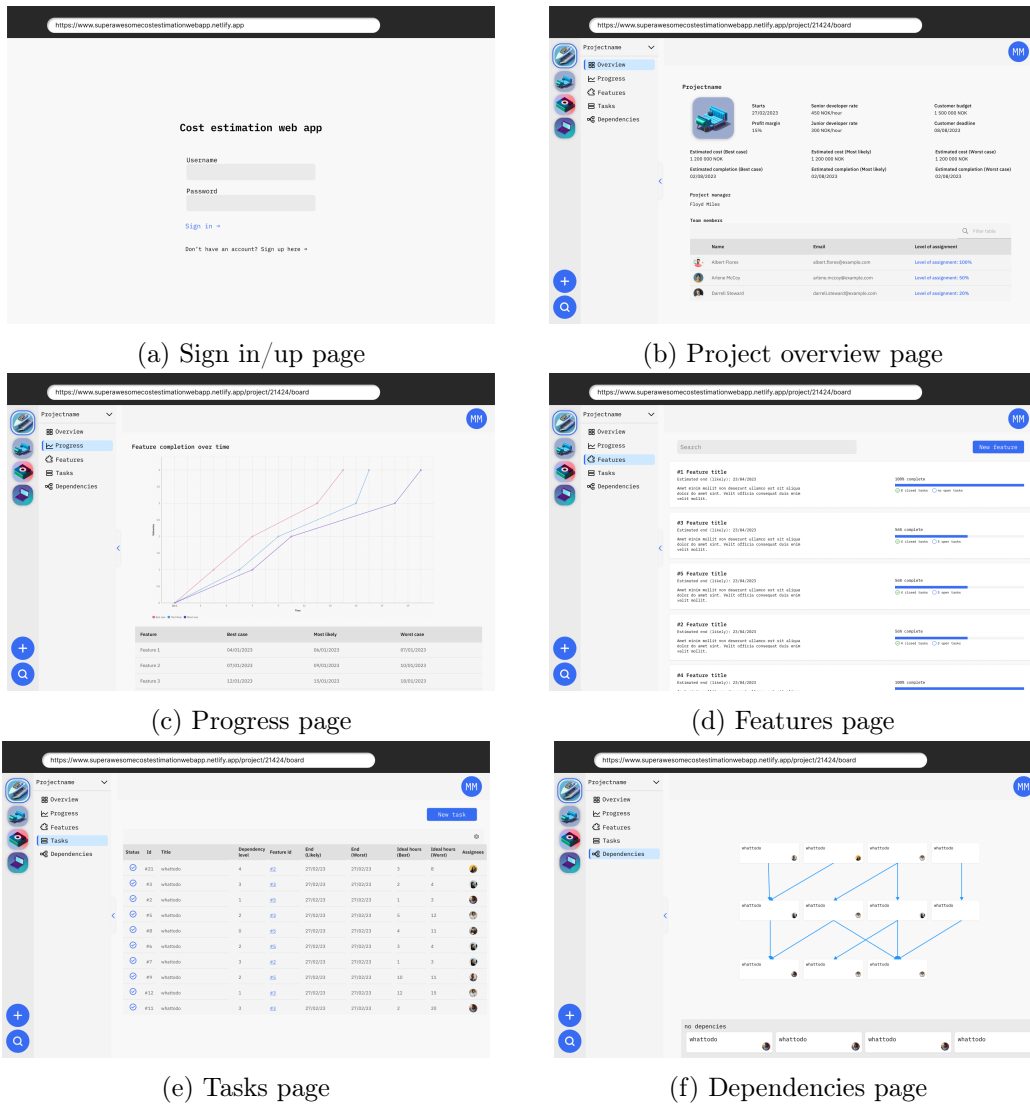


Figure 4.6.1: Final iteration of wireframes

### 4.6.1.2 Design Guidelines

We implemented our own design guidelines as no design guidelines were provided for us. Two distinct shades of blue were used as a primary and secondary colors. Figure 4.6.2 displays the color scheme chosen for the application.

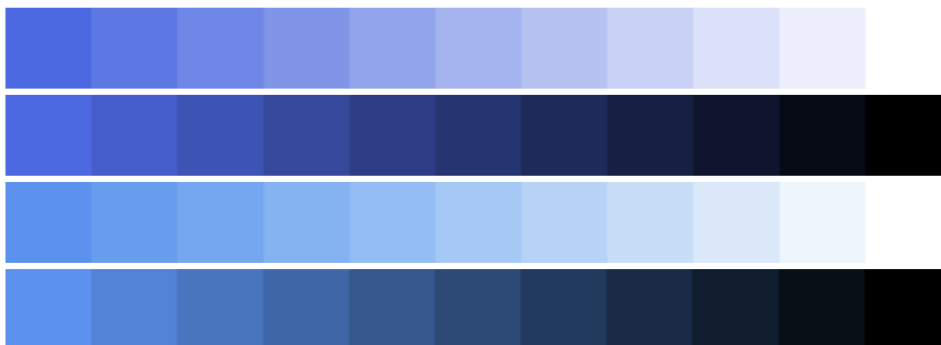


Figure 4.6.2: The color scheme of the application

The monospace font IBM Plex Mono was used as the primary font, keeping on the theme for an application made for a software company. Monospace fonts are serif fonts, known to improve readability, which was beneficial for an information-heavy application. The complete design guidelines developed can be found in appendix E.

## 4.6.2 Components

We used Carbon Design System’s component library for generic components. To tailor the components to our design we applied styling to them. As well as using the Carbon components we also implemented some of our own generic and specialized components which reside in the ‘src/lib/components’ folder. The folder structure for the components is displayed in Figure 4.6.3. The application represents 4 entities, assignees, projects, tasks, and features, where tasks and features are based on the Estimatable data model. The components created for each entity reside in a respective folder of the name and the generic components reside in the general folder.

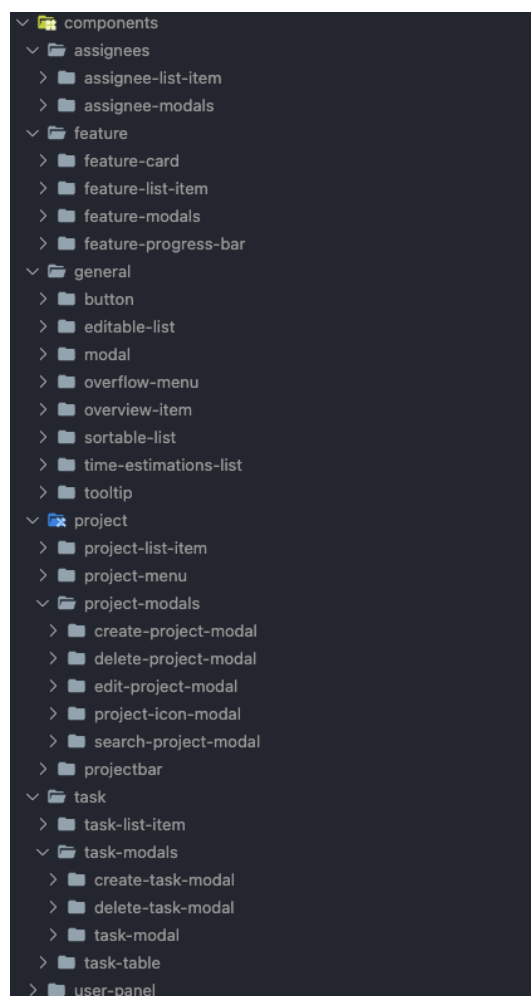


Figure 4.6.3: The folder structure for the components folder

## 4.6.3 Routing

SvelteKit uses file-based routing where directories signify the route as displayed in figure 4.6.4. The routes folder maps to 7 routes in the web application where the slug is a

parameter which in this case is the id of a project.

```
(base) janitaroyseth@Janitas-MBP-2 routes % tree
.
├── +page.svelte
├── +page.ts
├── projects
│   ├── +layout.server.ts
│   ├── +layout.svelte
│   ├── +page.server.ts
│   ├── +page.svelte
│   └── [slug]
│       ├── +layout.svelte
│       ├── +page.svelte
│       └── features
│           ├── +page.server.ts
│           └── +page.svelte
│               ├── overview
│               │   ├── +page.svelte
│               │   ├── project-details
│               │   │   └── project-details.svelte
│               │   └── project-members
│               │       └── project-members.svelte
│               └── progress
│                   ├── +page.svelte
│                   ├── feature-chart
│                   │   └── feature-chart.svelte
│                   ├── feature-table
│                   │   └── feature-table.svelte
│                   └── tasks
│                       ├── +page.server.ts
│                       └── +page.svelte
```

Figure 4.6.4: The file-based routing of the application

The following routes are displayed in the routes directory:

- `/` the route used for signing in to the application.
- `/projects` lists all the projects.
- `/projects/[slug]` This route automatically redirects to the `/projects/[slug]/overview` where `[slug]` is the id of the project.
- `/projects/[slug]/overview` displays information about the project where the slug is the project id.
- `/projects/[slug]/progress` where simulations can be run and estimated progress viewed for the project where the slug is the project id.
- `/projects/[slug]/features` lists all the features in the project where the slug is the project id.
- `/projects/[slug]/tasks` lists all the tasks in the project where the slug is the project id.

#### 4.6.4 State Management

Svelte comes bundled with stores, which are global reactive variables. Global reactive variables update their subscribers upon changes; this allows simpler state management in components, as the state does not explicitly need to be set, and variables don't get passed through many props before their intended use. We implemented 4 different stores, `$projects`, `$employees`, `$currentProject`, and `$app`.

- `$projects` - The `$projects` store contains the complete list of projects accessible to the user in the application. The projects are updated when there are changes to the data through the SvelteKit function `invalidateAll()`, which causes all load functions defined in pages or layouts to reload. For updating data, the `invalidateAll()` function is used after form requests have been responded to.
- `$employees` - The `$employees` store contains the list of employees added to the application. The employee store is also updated when the SvelteKit function `invalidateAll()` is called after form requests have been responded to.
- `$currentProject` - The `$currentProject` store contains the currently displayed project. The store is updated when entering a given valid project. Otherwise, the store is undefined.
- `$app` - The `$app` store is meant to contain general application information; for now, it has only if the application is ready. The `$app` store variable is an object that now only holds one variable `ready`. The store variable `ready` defaults to false and is updated to true once data has loaded and been applied to their respective stores.

Displayed in Figure 4.6.5 is the creation of the `$projects` store. The object returned from the `projectStore()` contains the list of projects, helping functions to update the store, and the store-specific functions.

```
/**
 * Creates a writable store for projects.
 *
 * @returns writable store for projects.
 */
function projectStore() {
  /** List of projects */
  const projects: Project[] = [];

  /**
   * Updates the projects store with the given projects data transfer objects.
   *
   * @param dtos projects data transfer objects to add to the store.
   * @returns the updated projects list.
   */
  function addProjectsFromDTOs(dtos: ProjectDTO[]) {
    update((projects) => {
      projects = dtos.map((dto) => new Project(dto));
      return projects;
    });
    return projects;
  }

  const { subscribe, set, update } = writable(projects);

  return {
    projects,
    addProjectsFromDTOs,
    subscribe,
    set,
    update
  };
}

/** The projects store. */
export const projects = projectStore();
```

Figure 4.6.5: Code example: `$projects` store implementation

## 4.6.5 User Interface

### 4.6.5.1 Login Page



**Figure 4.6.6:** Screenshot of the login page.

Authentication was not implemented because we were making a demonstrator, and if Axbit were to continue using the application, they wished to integrate it with their Enterprise Resource Planner system. Figure 4.6.6 is the login page for the application. The form toggles between sign up and sign in and validates input with regex.

### 4.6.5.2 Projects Page



**Figure 4.6.7:** Screenshot of the projects page.

Figure 4.6.7 is the projects page. The projects page lists all the projects in the database. The buttons have tooltips to make finding the right project and function easier. The projects aren't accessible through tabbing, considering if there are 30 projects, it would make navigating the page for keyboard users inefficient and burdensome. Instead, keyboard users must use the search modal to navigate to a project. The user can also log out, which is only a redirect to the login page for the time being.

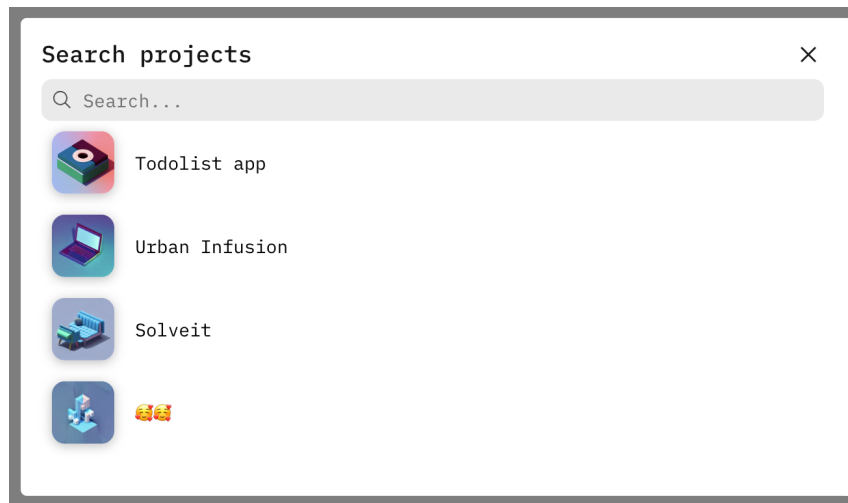
### 4.6.5.3 Projects Layout

The list of projects on the far left in Figure 4.6.7 is a static component for our single-page application. For this route, `/projects`, and any child routes, the project bar remains static. This is defined through the `+layout.svelte` in the projects folder, sample seen in Figure 4.6.8. The layout will always display the project bar and conditionally renders the project menu on whether the slug parameter is defined in the page route. The contents of this page, child page, or child layouts will be visible through the `<Slot/>` component.

```
<div class="projects">
  <Projectbar />
  {#if $page.params.slug}
    <ProjectMenu>
      <ProjectMenuItem
        url={` /projects/${$page.params.slug}/overview`}
        icon={Grid}
        label={'Overview'} />
      <ProjectMenuItem
        url={` /projects/${$page.params.slug}/progress`}
        icon={ChartLineData}
        label={'Progress'} />
      <ProjectMenuItem
        url={` /projects/${$page.params.slug}/features`}
        icon={Sprout}
        label={'Features'} />
      <ProjectMenuItem
        url={` /projects/${$page.params.slug}/tasks`}
        icon={Table}
        label={'Tasks'}
        disabled={tasksLinkDisabled}
        disabledText={'Atleast one feature must exist'} />
    </ProjectMenu>
  {/if}
  <div>
    <UserPanel />
    <slot class="view" />
  </div>
</div>
```

Figure 4.6.8: Code sample for the projects layout.

#### 4.6.5.4 Search Projects



**Figure 4.6.9: Screenshot of a search feature modal.**

Figure 4.6.9 displays the modal used for searching projects. The modal is accessible through the search icon on the left lower corner on any page after `/projects` route, or through the shortcut `ctrl` + `G` or `⌘` + `G` for Mac users, the same shortcut used in Slack which is the main communication app Axbit uses internally.

#### 4.6.5.5 Creating Projects

Figure 4.6.10 displays a screenshot of the create project modal. The modal has four steps:

1. Project - How the project is displayed, its name and icon
2. Details - Project details like profit, start date, deadline, etc.
3. Members - Project lead and assignees are designated in this step.
4. Summary - Summary of the project to be created as seen in figure 4.6.10

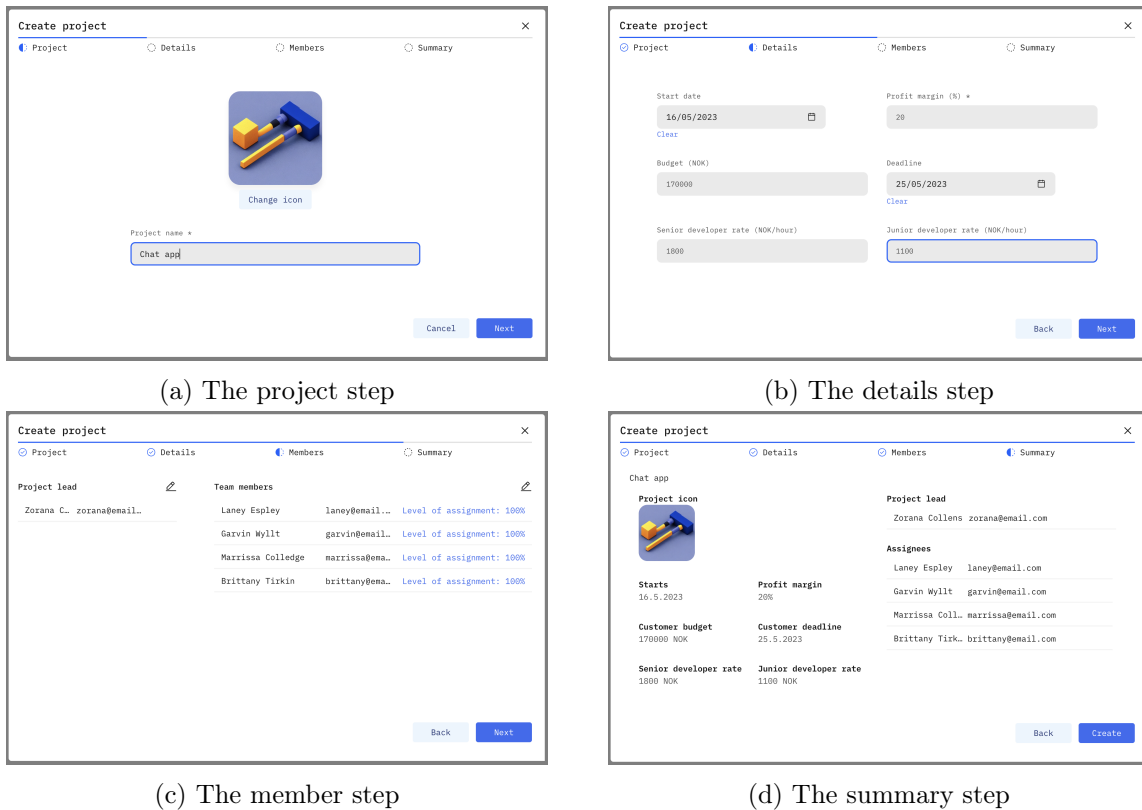


Figure 4.6.10: Screenshot of the create project modal

### 4.6.5.6 Project Overview Page

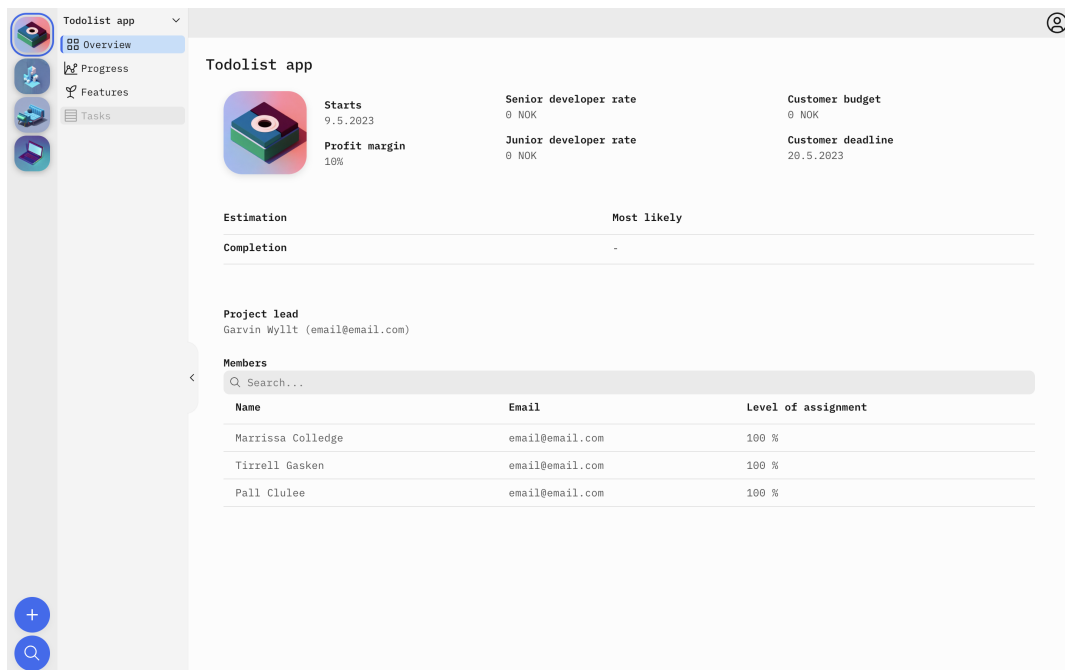
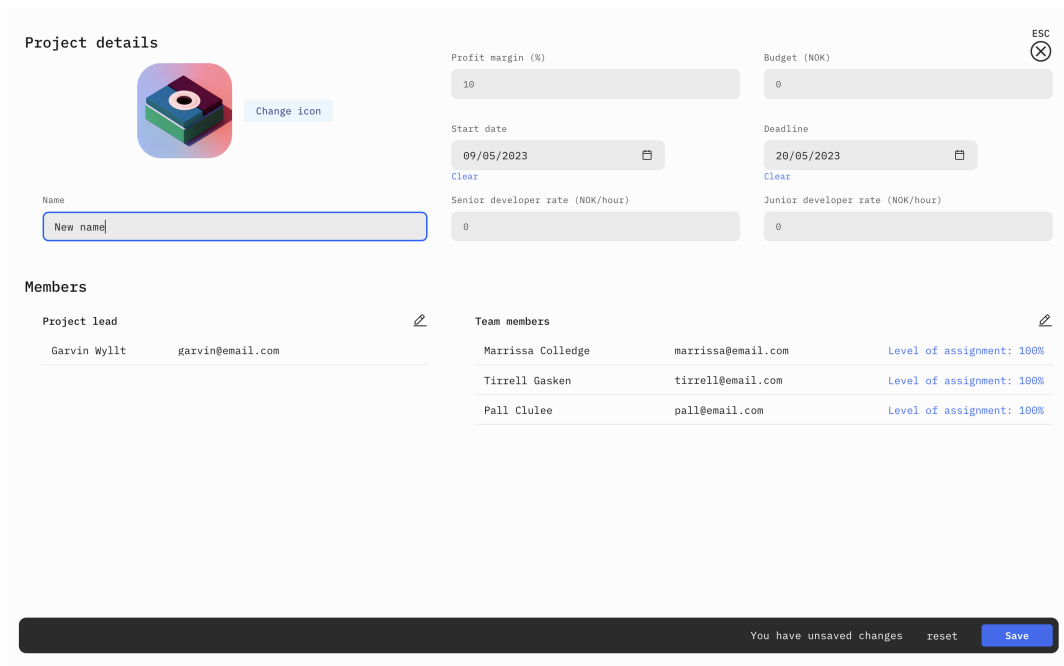


Figure 4.6.11: Screenshot of project overview page

Figure 4.6.11 is the project overview page. It displays details about the projects and their team.



### 4.6.5.7 Editing Projects



The screenshot shows a modal window titled "Project details" with a close button (ESC) in the top right corner. The modal is divided into several sections:

- Project details:** Includes a 3D cube icon with a "Change icon" button. Below it is a "Name" input field containing "New name".
- Profit margin (%):** A text input field with the value "10".
- Budget (NOK):** A text input field with the value "0".
- Start date:** A date picker showing "09/05/2023" with a "Clear" button below it.
- Deadline:** A date picker showing "20/05/2023" with a "Clear" button below it.
- Senior developer rate (NOK/hour):** A text input field with the value "0".
- Junior developer rate (NOK/hour):** A text input field with the value "0".
- Members:** A table with two columns: "Project lead" and "Team members".
  - Project lead:** A single row with "Garvin Wylit" and "garvin@email.com".
  - Team members:** A table with three rows:
    - Row 1: "Marrissa Colledge", "marrissa@email.com", "Level of assignment: 100%"
    - Row 2: "Tirrell Gasken", "tirrell@email.com", "Level of assignment: 100%"
    - Row 3: "Pall Clulee", "pall@email.com", "Level of assignment: 100%"

At the bottom of the modal, there is a dark bar with the text "You have unsaved changes" and two buttons: "reset" and "Save".

Figure 4.6.12: Screenshot of the edit project modal

Figure 4.6.12 displays the edit project modal. The modal takes up the whole screen and will prevent users from leaving if there are unsaved changes. Everything configured while creating the project can also be changed here.

### 4.6.5.8 Deleting Projects

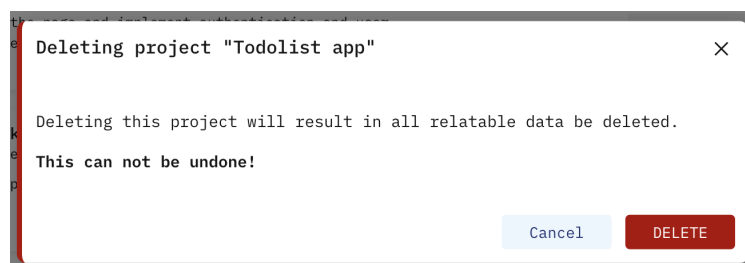


Figure 4.6.13: Screenshot of modal for deleting a project.

Figure 4.6.13 displays the confirmation modal for deleting a project.

### 4.6.5.9 Features Page

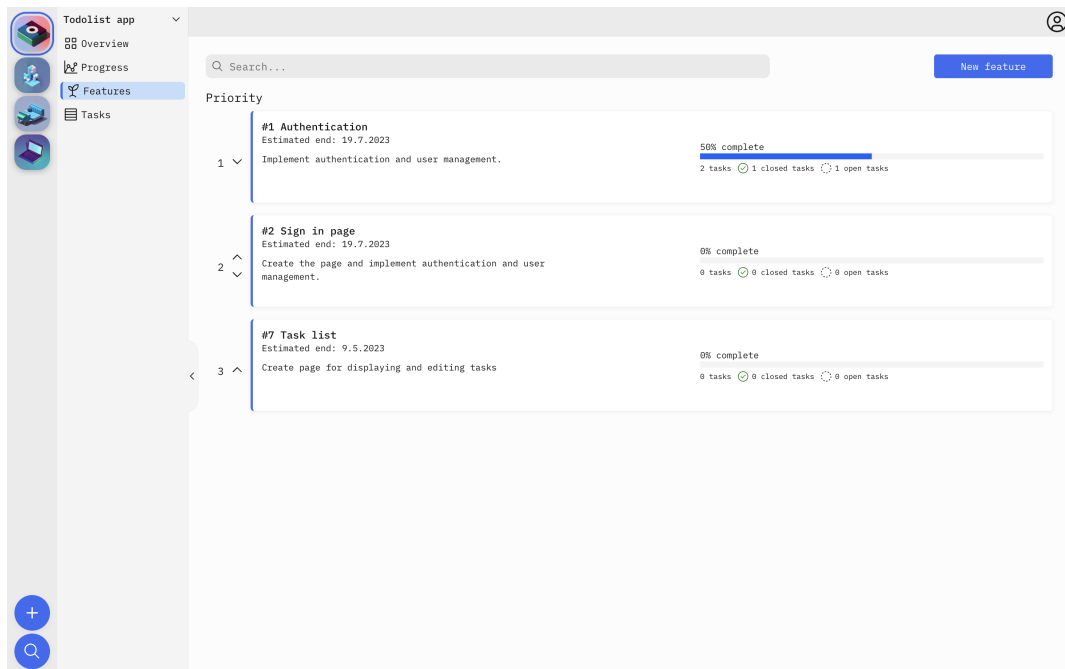


Figure 4.6.14: Screenshot of the features page

Figure 4.6.14 is the features page. All the features for the projects are listed here and can be searched through. Features can also be edited, added, and deleted from this page. The features are ordered by priority; a new feature will always have the lowest priority unless placed higher in the list. The features can be sorted through the buttons or by dragging a feature card and dropping it to another location in the list.

### 4.6.5.10 Feature Details

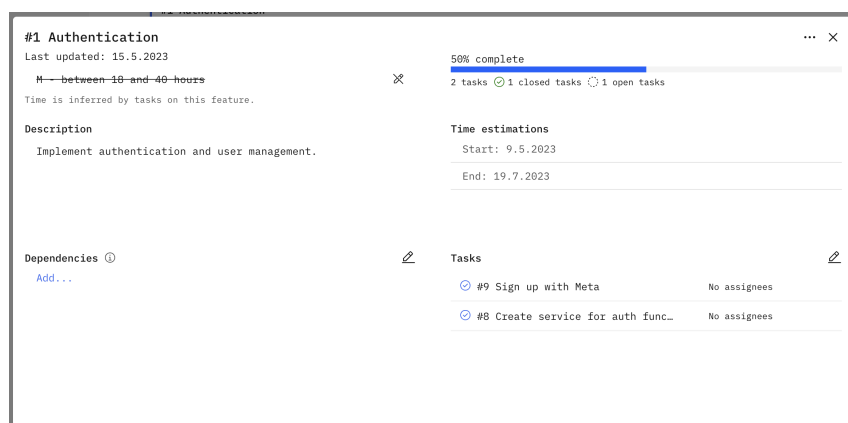
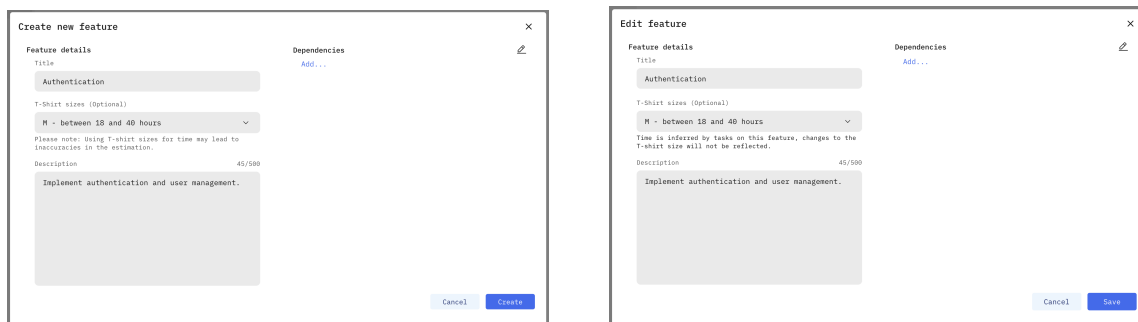


Figure 4.6.15: Screenshot of a feature modal displaying information about a feature.

Figure 4.6.15 displays the feature modal, which shows details about the feature and allows for tasks to be created under the feature and dependencies added. The progress bar shows how many tasks out of the total number of tasks are completed.

### 4.6.5.11 Create and Edit Features



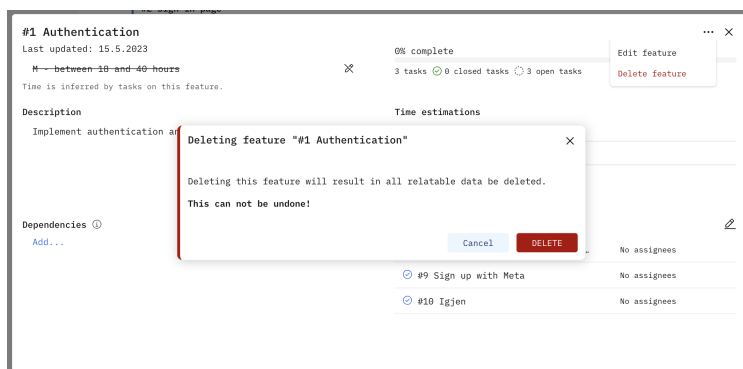
(a) Screenshot of a feature being created

(b) Screenshot of a feature being edited

**Figure 4.6.16: Creating and editing features in the application**

The creation and editing of features happen in the same modal, which has two different modes displayed in figures 4.6.16a and 4.6.16b. Tasks under a feature can not be created under a feature in edit mode. A feature already needs to exist; to save place; this is done under the feature detail modal. A feature can be edited through the options button on the feature modal.

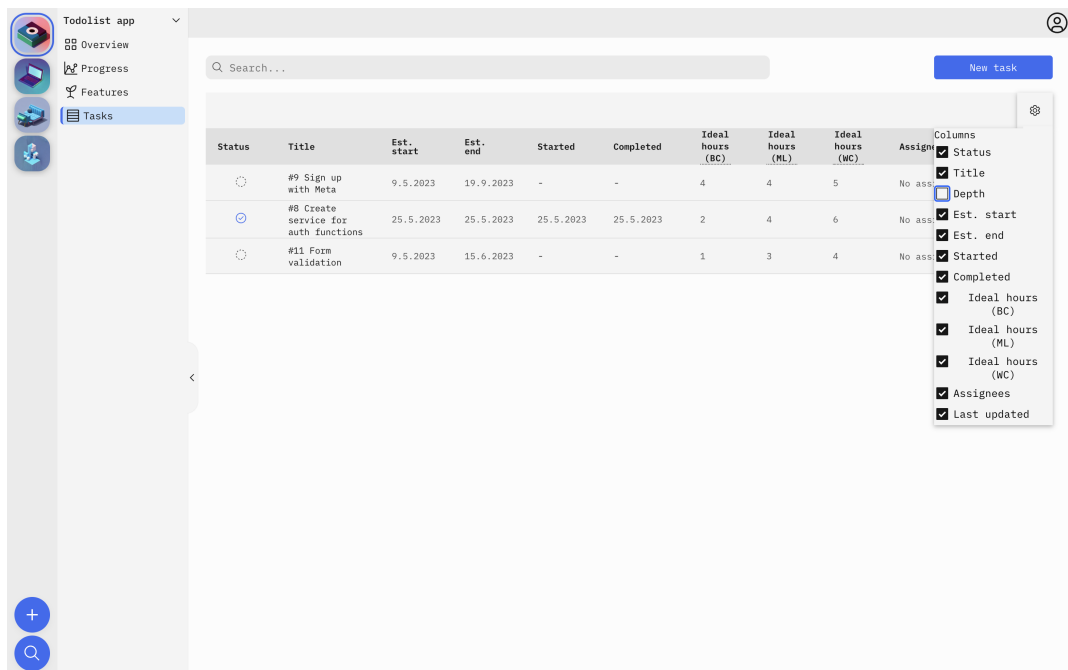
### 4.6.5.12 Delete Feature



**Figure 4.6.17: Screenshot of modal for deleting a feature.**

Figure 4.6.13 displays the deletion confirmation modal on a feature. A feature can be deleted through the options button on the feature modal.

## 4.6.5.13 Tasks Page



Status	Title	Est. start	Est. end	Started	Completed	Ideal hours (BC)	Ideal hours (ML)	Ideal hours (WC)	Assignees
🕒	#9 Sign up with Meta	9.5.2023	19.9.2023	-	-	4	4	5	No assignees
🕒	#8 Create service for auth functions	25.5.2023	25.5.2023	25.5.2023	25.5.2023	2	4	6	No assignees
🕒	#11 Form validation	9.5.2023	15.6.2023	-	-	1	3	4	No assignees

Figure 4.6.18: Screenshot of the tasks page

Figure 4.6.18 displays the tasks page in the application. Viewing, creating, editing, and deleting tasks happen through this page. The page contains a table of tasks in the projects, and the table has customizable headers as displayed.

## 4.6.5.14 Task Details

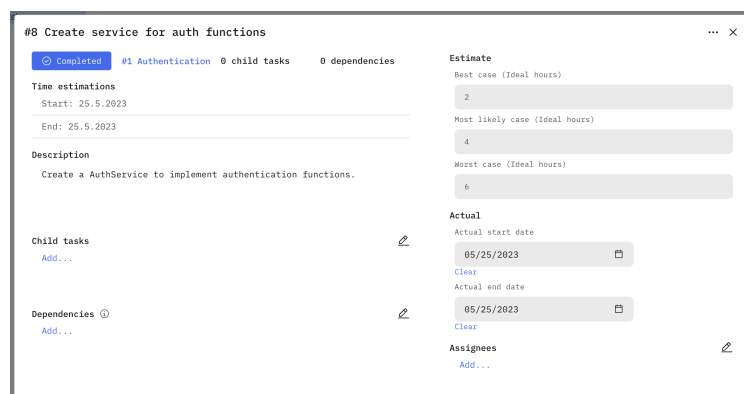


Figure 4.6.19: Screenshot of the task modal, which displays details about a task.

Figure 4.6.19 displays the task modal, which shows details about a task. The modal contains a link back to its parent which can either be a task or a feature, and has a button to update the actual date fields quickly. The modal also allows for editing fields that might frequently change, like estimated time, child tasks, dependencies, assignees, and when the task started and ended.

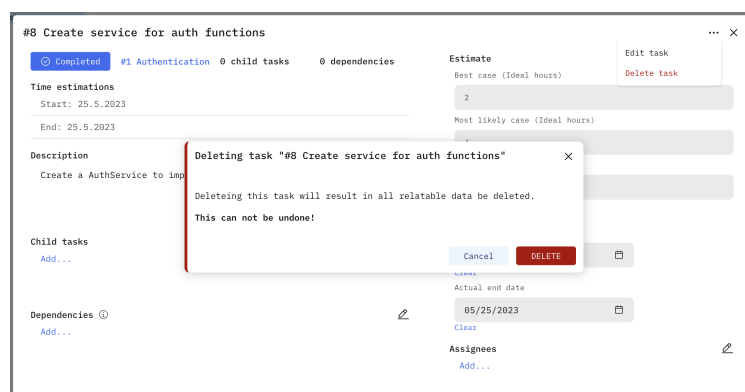
### 4.6.5.15 Create and Edit Tasks



**Figure 4.6.20: Modal in Create and Edit modes**

The creation and editing of tasks are done in the same modal. The modal has a create and edit mode, as displayed in Figure 4.6.20. There are two steps in the modal; the first step is for the details of the tasks, such as name and description and what feature it belongs to. The second step is to estimate how long the task will take and possible dependencies on the task. A task can be edited through the options button on the modal.

### 4.6.5.16 Delete Task



**Figure 4.6.21: Screenshot of modal for deleting a task.**

Displayed in 4.6.21 is task deletion confirmation. A task can be deleted through the options button on the modal.

## 4.6.5.17 Progress Page

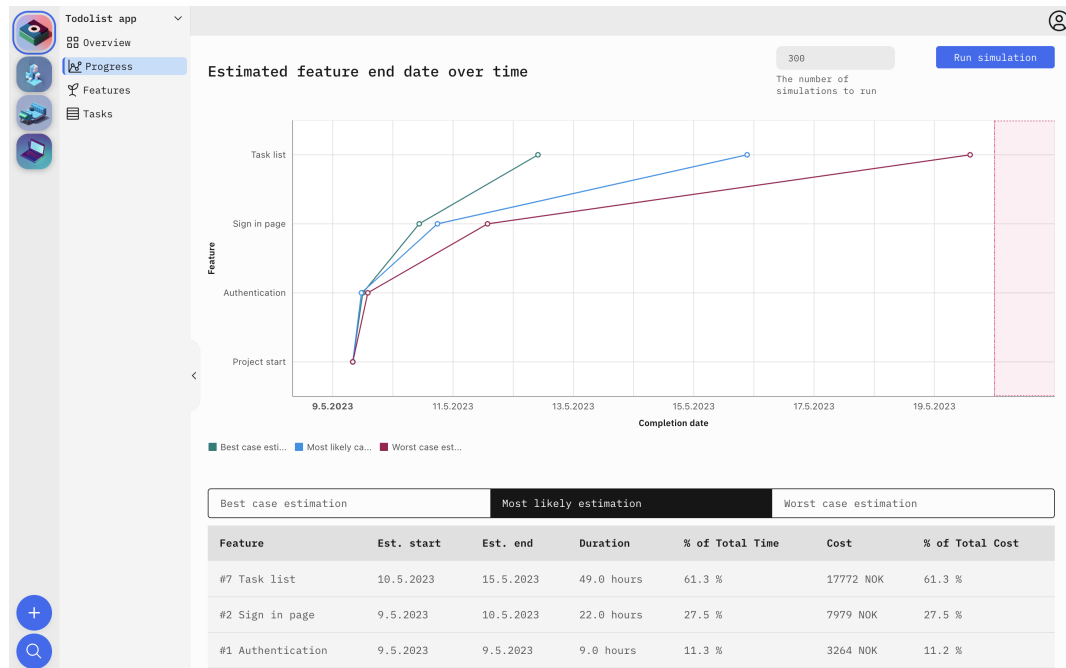


Figure 4.6.22: Screenshot of the progress page

Figure 4.6.22 displays the progress page. From this page, simulations can be run, and the results of the cost estimation simulation are shown in the chart and table. The chart on this page is perhaps the most essential front-end component added to the application, as it provides a clear overview of how the project is estimated to complete. The chart also has a "threshold," a red line, and a highlight, which signifies the project's deadline. The table contains information about how much cost and time each feature is estimated to take, making it easier to cherry-pick important features to finish a project within budget and deadline.

## 4.7 Service Classes

Figure 4.7.1 displays the UML diagram of our implemented service classes. We utilized service classes to define the logic that belonged to neither the models nor the components. The service classes assisted in keeping the models and components object-oriented and prevented code repetition.

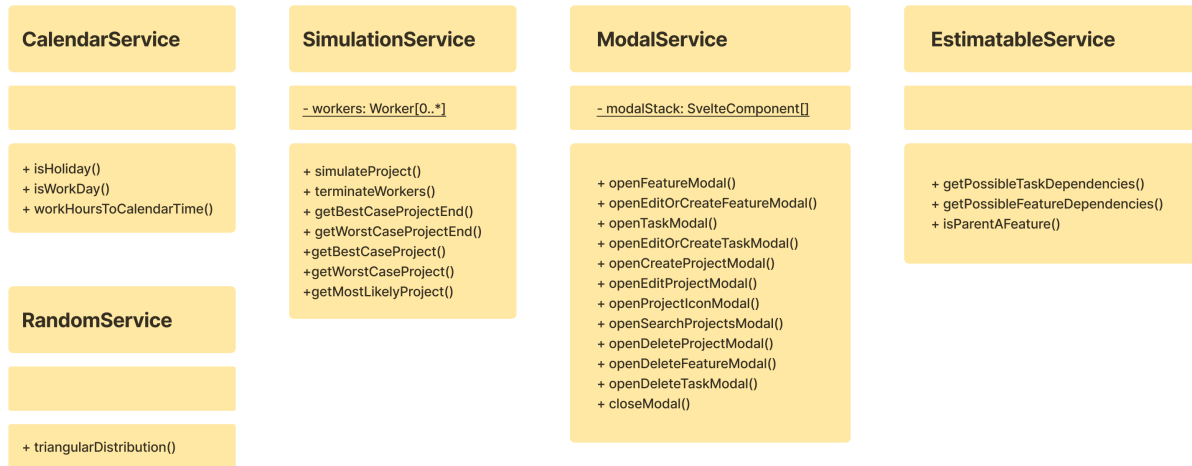


Figure 4.7.1: UML diagram describing the structure of the service classes.

### 4.7.1 CalendarService

The calendar service was created to assist with work-hours to calendar time calculations and provide methods for tracking work days and holidays. The methods `isHoliday`, `isWorkDay` and `workHoursToCalendarTime` are implemented in this service. To retrieve the list of Norwegian bank holidays, the method `isHoliday` uses the API `WebApi.no` [72]. Figure 4.7.2 shows the implementation of the `isHoliday` method in the calendar service.

```

/**
 * Checks whether or not the given date is a Norwegian bank holiday.
 *
 * @param date the {@code Date} to check.
 * @returns {@code true} if the given date is a Norwegian bank holiday, or {@code false} if not.
 */
public static isHoliday = async function (date: Date): Promise<boolean> {
  // Send API request.
  const res = await fetch(`https://webapi.no/api/v1/holidays/${date.getFullYear()}`);
  let json;
  if (res.ok || res.status === 422) {
    const text = await res.text();
    json = text ? JSON.parse(text) : {};
  }

  return json.data
    .map((data: { date: string | number | Date }) => new Date(data.date).toDateStrin())
    .includes(date.toDateStrin());
};
  
```

Figure 4.7.2: Code example: CalendarService

### 4.7.2 EstimatableService

The estimable service was used in front-end components for logic not belonging to the model `Estimatable`, but still required for presenting information about objects of the model. The functions `getPossibleTaskDependencies`, `isParentAFeature`, `getPossibleFeatureDependencies`, `getTaskStatusForChildlessTask`, and `getTaskStatus`

are implemented in the estimatable service. Figure 4.7.3 displays the implementation of the `getTaskStatus` function from the estimatable service.

```
/**
 * Gets the status of the given task. If the task has children, the status is determined by the
 * children. If the task has no children, the status is determined by the actual start and finish.
 *
 * @param task the task to get the status for.
 * @returns the status of the task.
 */
public static getTaskStatus(task: Estimatable): 'open' | 'started' | 'completed' {
  if (task.getChildren().length === 0) return this.getTaskStatusForChildlessTask(task);
  if (task.getUnfinishedChildren().length === 0) return 'completed';

  if (
    task
      .getChildren()
      .filter((child) => child.getActualStart() === undefined || child.getActualStart() === null)
      .length === task.getChildren().length &&
    task
      .getChildren()
      .filter(
        (child) => child.getActualFinish() === undefined && child.getActualFinish() === null
      ).length === task.getChildren().length
  ) {
    return 'open';
  }
  return 'started';
}
```

Figure 4.7.3: Code example: EstimatableService

### 4.7.3 ModalService

Modal service is responsible for the modal windows opened in the user interface. To make the opening of modals through code more straightforward, the modal service creates SvelteComponents through code. The modal service has a stack over the modals currently open in the application. Figure 4.7.4 displays the `openProjectIconModal` function; the SvelteComponent is created and pushed on top of the modal stack. Figure 4.7.5 shows the implementation of `closeModal` function; the last item in the modal stack is removed using the SvelteComponent specific function `component.$destroy()`.

```
/**
 * Opens a modal for selecting a project icon.
 *
 * @param iconUrl the URL of the currently selected icon.
 * @param onIconPick the function to call when the user picks an icon.
 */
public static openProjectIconModal(iconUrl: string, onIconPick: (url: string) => void) {
  this.modalStack.push(
    new ProjectIconModal({
      target: document.body,
      props: { iconUrl: iconUrl, onIconPick: onIconPick }
    })
  );
}
```

Figure 4.7.4: Code example: ModalService open project icon modal



```

/**
 * Closes the topmost modal.
 *
 * @param beforeClose function to call before closing the modal.
 */
public static closeModal(beforeClose?: () => void): void {
  const modal = this.modalStack.pop();

  if (beforeClose) beforeClose();

  if (modal) {
    modal.$destroy();
  }
}

```

Figure 4.7.5: Code example: ModalService close modal

#### 4.7.4 RandomService

The role of the random service is to provide methods for pseudorandom number generation. It contains the triangular distribution number generation method in use by the simulation.

```

/**
 * Returns a pseudorandomly created number weighted by a triangular distribution.
 *
 * @param min lower limit
 * @param max upper limit
 * @param mode central weight point
 * @returns {@code number} pseudorandom number weighted by the triangular distribution.
 */
public static triangularDistribution = function (min: number, max: number, mode: number) {
  const ms = new Date().getMilliseconds().toString();
  const seed = 1 + Number(ms.length > 3 ? ms.substring(ms.length - 3) : ms);

  const rand = triangular.factory(Number(min), Number(max), Number(mode), {
    seed: seed
  });
  return rand();
};

```

Figure 4.7.6: Code example: RandomService

#### 4.7.5 SimulationService

The simulation service manages the simulations of the development of projects through the use of web workers. It also provides helper functions relating to simulated projects, such as `getBestCaseProject()`, which returns the simulated project with the earliest end date, `getMostLikelyProject()`, which returns the project with the latest end date, and `getWorstCaseProject()` which returns a project in which all of the project's tasks' start and end dates are averages of the corresponding tasks in all of the simulations. Below is a code example showing the method `simulateProject()` in the simulation service that creates and instructs the web workers to perform simulations of a project. The internal process of the simulation is described in detail in section 4.5.2.

```

/**
 * Simulates the development of the given project.
 *
 * @param project The project to simulate
 * @param numberOfSimulations The number of simulations to run
 * @returns {Promise<Project[]>} List of simulated projects
 */
public static async simulateProject(
  project: ProjectDTO,
  numberOfSimulations: number
): Promise<ProjectDTO[]> {
  if (!project.assignees || project.assignees.length === 0) throw new Error('no_assignees');
  if (!project.features || project.features.length === 0) throw new Error('no_features');

  /** List of simulated projects */
  const projectSimulations: ProjectDTO[] = [];

  // Ensure there are enough workers in the worker pool.
  if (SimulationService.workers.length < numberOfSimulations) {
    for (let i = SimulationService.workers.length; i < numberOfSimulations; i++) {
      const simulationWorker = new SimulationWorker();
      SimulationService.workers.push(simulationWorker);
    }
  }

  /** List of promises which simulates the project. */
  const promises: Promise<void>[] = [];

  // Tell each worker to start simulation as a promise.
  for (let i = 0; i < numberOfSimulations; i++) {
    promises.push(
      new Promise<void>((resolve) => {
        SimulationService.workers[i].postMessage(project);
        SimulationService.workers[i].onmessage = (message: MessageEvent) => {
          projectSimulations.push(message.data);
          resolve();
        };
      })
    );
  }

  // Resolve all promised simulations and return the results.
  return Promise.all(promises).then(() => projectSimulations);
}

```

Figure 4.7.7: Code example: SimulationService

## 4.8 Test and Quality Assurance

### 4.8.1 Unit Tests

For unit tests, the vitest framework was used, which allowed us to thoroughly test the robustness of the backend models in different states. Automated unit tests, both positive and negative, were made for all model classes and services to ensure code stability, and prevent regression upon the implementation of additional features.

```
it('Employee has correct values with all valid params', () => {
  const employee = new Employee({
    name: 'Lee',
    id: 36,
    salary: 450.0,
    efficiency: 0.9
  });
  expect(employee.getId()).toBe(36);
  expect(employee.getName()).toBe('Lee');
  expect(employee.getEfficiency()).toBe(0.9);
  expect(employee.getSalary()).toBe(450);
});

it('Employee has correct values with invalid params', () => {
  const employee = new Employee({
    name: '',
    id: -36,
    salary: -450.0,
    efficiency: -0.9
  });
  expect(employee.getId()).toBe(0);
  expect(employee.getName()).toBe('NoName');
  expect(employee.getEfficiency()).toBe(0.85);
  expect(employee.getSalary()).toBe(0);
});
```

Figure 4.8.1: Code example: Positive and negative Employee unit tests

### 4.8.2 Integration Tests

Integration tests were created using Storybook. The files for denoting documentation and tests for the components can be found next to the components in the `/src/lib/components` folder. Showcased in Figure 4.8.2 is the configuration for Storybook, where we had imported interaction tests, accessibility tests, and test coverage to see how much our tests covered the code.

```
import type { StorybookConfig } from '@storybook/sveltekit';

const config: StorybookConfig = {
  stories: ['../src/**/*.mdx', '../src/**/*.stories.@(js|jsx|ts|tsx)'],
  addons: [
    '@storybook/addon-interactions',
    '@storybook/addon-essentials',
    '@storybook/addon-coverage',
    '@storybook/addon-links',
    '@storybook/addon-a11y'
  ],

  staticDirs: ['../static'],
  framework: {
    name: '@storybook/sveltekit',
    options: {}
  },
  docs: {
    autodocs: 'tag'
  }
};

export default config;
```

Figure 4.8.2: Storybook configuration

The accessibility is tested automatically by Storybook with no need to write tests, however, interaction tests have to be explicitly written. Showcased in Figure 4.8.3 is the interaction test of `SortableList` component. The resulting test coverage is displayed in Figure 4.8.4

```
SortableListControls.play = async ({ canvasElement }) => {
  const canvas = within(canvasElement);
  const button = canvas.getAllByRole('button', { name: 'Move item down' })[0];
  userEvent.click(button);

  const draggableItems = canvas.getAllByRole('listitem');
  const itemToDrag = draggableItems[0];
  const targetItem = draggableItems[1];

  // Simulate drag and drop
  fireEvent.dragStart(itemToDrag);
  fireEvent.dragOver(targetItem);
  fireEvent.drop(targetItem);
  fireEvent.dragEnd(itemToDrag);
};
```

Figure 4.8.3: Interaction test for `SortableList`

File	% Stmts	% Branch	% Funcs	% Lines
<b>All files</b>	<b>73.52</b>	<b>60.17</b>	<b>33.33</b>	<b>74.25</b>
components/assignees/assignee-list-item	92	93.93	100	95.83
assignee-list-item.svelte	92	93.93	100	95.83
components/assignees/assignee-modal	85	100	100	84.21
assignment-level-modal.svelte	85	100	100	84.21
components/feature/feature-card	91.3	66.66	100	95
feature-card.svelte	91.3	66.66	100	95
components/feature/feature-list-item	87.5	50	100	85.71
feature-list-item.svelte	87.5	50	100	85.71
components/feature/feature-modal/create-feature-modal	60.93	65	0	62.71
create-feature-modal.svelte	60.93	65	0	62.71
components/feature/feature-modal/delete-feature-modal	58.33	0	100	52.38
delete-feature-modal.svelte	58.33	0	100	52.38
components/feature/feature-modal/feature-modal	52.17	41.17	0	52.54
feature-modal.svelte	52.17	41.17	0	52.54
components/feature/feature-progress-bar	82.05	89.65	25	83.33
feature-progress-bar.svelte	82.05	89.65	25	83.33
components/general/button	100	91.42	100	100
button-spinner.svelte	100	100	100	100
button.svelte	100	91.42	100	100
components/general/editable-list	75	69.11	20	80.32
editable-list.svelte	75	69.11	20	80.32
components/general/editable-list/edit-list-modal	78.94	88.88	35.71	83.07
edit-list-modal.svelte	78.94	88.88	35.71	83.07
components/general/modal	97.95	82.14	100	97.82
modal.svelte	100	84.61	100	100
sub-modal.svelte	96.42	80	100	96.29
components/general/modal/modal-actions	100	100	100	100
modal-actions.svelte	100	100	100	100
components/general/overflow-menu	94.73	60	100	94.11
overflow-menu-item.svelte	100	50	100	100
overflow-menu-list.svelte	92.85	62.5	100	91.66
components/general/overflow-menu/stories	100	0	100	100
overflow-menu-story.svelte	100	0	100	100
components/general/overview-item	100	33.33	100	100
overview-item.svelte	100	33.33	100	100
components/general/sortable-list	79.01	75.75	60	82.08
sortable-controls.svelte	92.85	88.88	100	100
sortable-list.svelte	76.11	70.83	60	78.57
components/general/sortable-list/stories	87.5	33.33	100	100
sortable-list-story.svelte	87.5	33.33	100	100
components/general/time-estimations-list	85.71	50	100	100
time-estimations-list.svelte	85.71	50	100	100
components/general/tooltip	83.03	57.14	100	92.94
tooltip.svelte	83.03	57.14	100	92.94
components/general/tooltip/stories	88.88	0	100	100
tooltip-story.svelte	88.88	0	100	100
components/project/project-list-item	88.88	83.33	100	100
project-list-item.svelte	88.88	83.33	100	100
components/project/project-modal/create-project-modal	81.96	65.38	0	76.19
create-project-modal.svelte	81.96	65.38	0	76.19
components/project/project-modal/create-project-modal/assignees-step	100	0	100	100
assignees-step.svelte	100	0	100	100

Figure 4.8.4: Integration test coverage

### 4.8.3 End-to-end Tests

Playwright was used to create E2E tests. We placed the end-to-end tests in the folder `e2e`. Playwright uses the Chromium browser where it loads the denoted web page and performs the interactions specified in the test files on a live site. Displayed in Figure 4.8.5 are the results of our 2 primary tests. We have a smaller test that creates, edits, and deletes a single project, and large tests which take on the whole process of creating, editing, and deleting projects, features, and tasks.

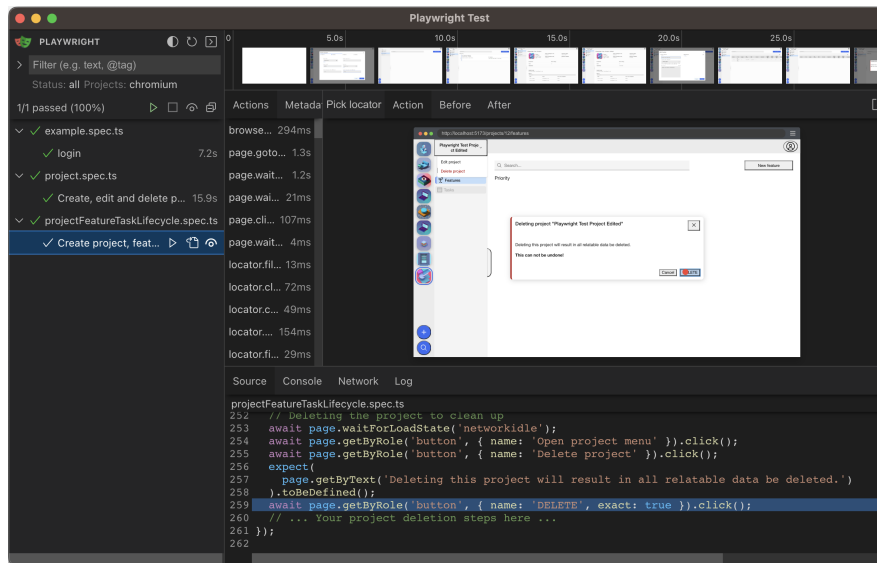


Figure 4.8.5: End-2-end tests results

## 4.8.4 Usability Tests

We executed two usability tests throughout the duration of the project. One prototype test and one test on the MVP. Due to the location of the participants, the usability tests were performed remotely. We did moderated tests with a focus on qualitative finds. The users were encouraged to think out loud and to perform tasks according to a predefined test plan. The test template for the test plan can be found in appendix F.

### 4.8.4.1 Prototype Test

We performed prototype tests on two users before beginning to implement the front end. The prototype tests were performed by the users' sharing their screens and getting a link to the interactive Figma prototype. The main objective of the prototype tests was the navigation of the app, the layout, and the information related to the different concepts of tasks and features. Both users performed well on the prototype tests and gave valuable feedback for changes to improve the layout, navigation, and information displayed. The test plan and test documentation for the prototype test can be found in appendices G, H, and I.

### 4.8.4.2 Alpha Test

When the MVP was ready we performed an alpha test on two users, one of the users was new and had not had a previous introduction to the application. For this iteration of user tests the user shared their screen and was given a link to the applications dev build. The main objective of the alpha test was to test the changes implemented from the prototype tests and to test the simulation and how the readable information in the graph was. The test plan and test documentation for the alpha tests can be found in appendices J, K and L.

### 4.8.5 Code Format and Lint

For Code formatter we decided to go for Prettier, one of the most popular formatters. For linting we have used ESLint. We noticed early on in the development process that we were not used to these tools. We sat up a CI workflow to check if we had formatted the code, and it almost always failed. Luckily this is easy to fix when you see the test fail, you just run the command for formatting and it automatically formats for you. When it comes to ESLint, we got some bigger issues with ESLint not liking all the SvelteKit code. Most of the errors that we came across during development was fixed, but for some we needed to make exceptions in the ESLint configuration, or make a comment in the code for ESLint to ignore it.

## DISCUSSION

In this chapter we will discuss the process, results and decisions we made through this project.

### 5.1 Estimation Method

From our research into project estimation we made several findings that guided us to the method we used in the demonstrator. This section will discuss the choices made regarding the method of estimation.

#### 5.1.1 Basic Unit

Using a time-based basic unit of estimation makes the method heavily reliant on expert judgement, as insight and expertise is required to make reliable estimates. Using ideal hours was, however, a part of the user requirements listed in section 3.2.4. In order to avoid misconceptions, the difference between ideal hours and worked hours should be clearly communicated to the people involved in the project planning.

#### 5.1.2 Three-point Estimates

We used TPE in our method due to their value in providing a degree of uncertainty pertaining to the task's required effort. In cases of minor tasks where the degree of uncertainty is very low, the TPE will have a small spread, and this will be reflected in the simulations, providing increased accuracy. Without TPE the tasks in the simulations would all be subjected to the same degree of variance, and the simulator would therefore produce a less accurate result.

#### 5.1.3 Employee Efficiency

Our simulator uses the efficiency of the employee to increase the accuracy of the simulations. The idea of mapping an employee's ideal hours per work day can pose ethical questions due to the social implications it may lead to, however it also brings large benefits when it comes to project planning. It is also worth noting that a developer's work also includes time spent in meetings or performing other work-related tasks, and not only time spent on development as ideal hours.



### 5.1.4 Method Steps

**1. Work breakdown structure** - The work breakdown structure is important, as smaller tasks are easier to recognize and understand, and thereby easier to estimate accurately. It also makes the task of establishing relationships easier, as each task is more clearly defined.

**2. Establish relationships** - The relationships must be mapped out in order to create a functional workflow in projects where some tasks are dependent on the completion of others. However, in projects that don't have any dependencies between tasks, this step may be skipped. In such cases the simulator is given total freedom in the generation of workflows, which can increase workflow optimization.

## 5.2 Framework

In the project we used SvelteKit as the framework for our full-stack application, which the team members had little prior experience with. We were very pleased with our decision, however it introduced some restrictions on our application. The strict naming convention in the file system of the project made the directories more challenging to browse and navigate through. In addition, this framework uses JavaScript and TypeScript, neither of which have native support for multithreading. This posed some challenge for the implementation of the simulation, however, we were able to solve it using Web Workers. An alternative to Web Workers would have been to perform the simulations on the server side, using a different framework or language. This would entail either dividing our stack or changing the framework for our full-stack application, which would be an alteration too significant to introduce nearing the end of the project.

## 5.3 Security

An application containing sensitive information should always be locked behind a security layer. As our application is a demonstrator intended for integration into Axbit's systems and further developed internally by Axbit, we have not implemented our own authentication system. It still uses the HTTPS protocol and CORS restrictions which are included in modern security standards.

## 5.4 UI/UX

By beginning to plan the user interface through wireframes right after the initial meetings with our product owner, we were able to show our product owner our interpretation of the application from our discussions, and we were able to correct misunderstandings quickly and iteratively. The wireframe was changed a lot during the first weeks until our views and our product owners' views on the application coincided. We proceeded with the layout and performed usability tests on the wireframe.

We used Carbon Components for Svelte to make implementing the front end faster. Using Carbon components allowed us to focus more on implementing functionality key to our demonstrator rather than creating generic components. We had to load the whole

Carbon Design System CSS to use the components, as the components did not come styled and are styled through a global stylesheet. The Carbon stylesheet posed an issue as the stylesheet styled our whole application. The Carbon Design System is similar in style to our application, so only a few elements had to be cascaded through our CSS. This is, of course, an unfavorable implementation of styling as any significant changes to the Carbon Design System CSS may cause substantial changes to the application, and it will require effort to change.

Perhaps the most challenging part of creating the UI/UX was the amount of information needed to be displayed to the user. It took a lot of trials to create something that looked the least amount of messy. In particular, the tasks, both the modal and the table on the tasks page. Due to the amount of information needed to be displayed, which created size restrictions, and that this application is for a private business and only needs to follow WCAG 2.0 requirements, the app was not made mobile-friendly; apart from that, the application was thoroughly and often checked for accessibility violations with lighthouse through Google DevTools, and later with integration tests.

## 5.5 Testing

The unit tests were written simultaneously with their respective models and updated as changes were implemented on the models. They proved valuable throughout the project. As more complexity was added to the models, the unit tests aided us in ensuring the continued integrity of our classes.

In the preliminary report, we had originally planned to use Playwright for integration and E2E testing; however, upon further research, Storybook was discovered. Storybook is mainly used for component documentation however it also offers component testing. Because of Storybook's add-ons for accessibility and its support for interaction tests, we decided to use Storybook for integration tests of the UI components. Throughout the development of the application, integration tests also became down-prioritized as there was a lot of pressure to implement features and functionality so we could perform usability tests on the MVP. After performing alpha tests and correcting the UI according to the feedback of the user tests, integration tests were finally implemented. While the tests were implemented later than hoped, they provided great benefits by warning about accessibility violations ignored by Lighthouse and the IDE.

End-to-end tests were the last tests to be created. We had hoped to create a test database or have separate databases for the dev branch and main branch so as not to pollute the production database with test data. The test written does clean up after itself; however, if a test fails, it will not be able to clean up. Considering that we are creating a demonstrator and not an application ready for production, we concluded it would be more important to implement the end-to-end tests now than to put it off any longer and risk not having end-to-end tests at all.

The usability tests had fewer available users than we had hoped for, with only two users for each iteration. We performed moderated qualitative user tests to make the most out of the user tests. We also made a high-fidelity wireframe to make the end product more

transparent and avoid misunderstandings, as there were few chances to find participants.

## 5.6 Group Dynamic and Methodology

In this section we will discuss the group dynamic and software methodology used during this project. As we will discuss later we have worked together on different projects and subjects earlier and have though those found a good dynamic and method of working that suits us very well.

### 5.6.1 Methodology

As a group we set out to use Scrum, but as defined in The 2020 Scrum Guide

"Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems." "While implementing only parts of Scrum is possible, the result is not Scrum." [73] Since we did not implement the whole Scrum process we did not use Scrum.

We however did work in line with Agile development. For everything needed to be worked on, we made an issue in our GitHub repository. When you started to work on an issue, you assigned yourself and created a new branch. This made it easy for the other members to see what was being worked on. We used our weekly status meeting with the and supervisor, alternating biweekly between them, as start of a new iteration. This made us work towards that meeting on new features and issues. It also gave the and supervisor an insight in how we were doing on our project.

### 5.6.2 Group Dynamic

Our group is an experienced team, having worked together on many projects before embarking on this bachelor thesis. All of our members complement each other's capabilities and interests well; one is highly interested in logic, another in DevOps, and the last in UI/UX. Due to our experience as a group, we were familiar with what expectations we could hold for each other as some members struggled more than others with motivation. We also had another course running alongside the bachelor causing a slow start for the project. It was difficult to pick up the pace when that course ended as we had fallen into a routine with the slow pace. While there has been a significant difference in our hours logged, this does not reflect a difference in efforts. We were aware of this possibility before we started our bachelor assignment and adjusted expectations and workload accordingly, as a team should. All team members performed their roles well, and we are all pleased with each other's efforts and the results we collectively achieved with this bachelor thesis.

## CONCLUSIONS

This chapter will present our conclusions and suggestions for further work based on our results and discussions from Chapters 4 and 5. Our conclusions are based on the requirements for the project compared to the results we have achieved.

### 6.1 Conclusions

The goals of our project were to research project estimation theory, research the existing methods used in AxBit, develop a standardized method for project estimation, and create a demonstrator which implemented the method. In this thesis, we have described our process and the results of completing these goals.

As a result of our research, we have successfully created a method for project estimation that is ready to be adopted by Axbit as their new standard. The method is flexible in its level of detail, considers the team's specializations, and uses ideal hours as the basic unit for estimates, thereby fulfilling all of Axbit's requirements for the method.

We have developed a demonstrator application that meets all expectations set by Axbit in the form of user requirements, and all mandatory user stories have been successfully implemented in the back end. Only user story No.6 in section 3.2.5 was not fully implemented into the front end.

The team is happy with having used the Agile development methodology. This method has made the team adaptable to problems faced and changed during development. We have used GitHub for version control, collaborating, and tracking issues to work on. This can be recommended as it has helped us during development.

In summary, we feel the project has been a success. It has given us a way to test out new frameworks and technologies as well as giving us a better understanding of the complex field of estimation and project management. Axbit and we are very satisfied with how this project turned out, and we are eager to see how the product will evolve as Axbit continues its development.

## 6.2 Future work

This section includes suggestions for future implementation and improvements on the estimation method and the application and its simulator.

### 6.2.1 Validation

The method needs validation through trials with real-life projects. Using the application with historical data and on new projects are ways of achieving this.

### 6.2.2 Integration

Integration with an ERP system would provide further automation to the project creation process. The application could also benefit from being integrated into an existing authentication system, adding the necessary security.

### 6.2.3 Historical data

Our method and application could stand to benefit from adding a feature that uses historical data to adjust its estimates.

### 6.2.4 MTBF & MTTR

Mean time between failure (MTBF) and mean time to repair (MTTR) are metrics typically used with hardware systems.

MTBF refers to the average duration between system failures, but we can adapt them to refer to the average duration between an employee's unscheduled leaves.

MTTR refers to the average time it takes to repair a broken system, but here we adapt it to mean the average duration of an employee's unscheduled leaves.

The simulator in the application currently does not take unforeseen absences of the employees into consideration. Introducing MTBF and MTTR to the assignees in the project could improve the accuracy of estimations. These factors could be calculated from historical data (e.g. from an ERP system), and could be introduced as a chance every time an employee clocks in to call in sick and thereby adding a leave with a duration equalling the MTTR.

## REFERENCES

- [1] Volodymyr Voityshyn Vasyl Teslyuk Anatoliy Batyuk. *Method of Software Development Project Duration Estimation for Scrum Teams with Differentiated Specializations*. July 2022. URL: <https://www.mdpi.com/2079-8954/10/4/123> (visited on 05/12/2023).
- [2] *Principles behind the Agile Manifesto*. 2001. URL: <https://agilemanifesto.org/principles.html> (visited on 05/16/2023).
- [3] *16th State of Agile Report*. 2022. URL: <https://digital.ai/resource-center/analyst-reports/state-of-agile-report/> (visited on 05/16/2023).
- [4] Inc. The Standish Group International. *CHAOS Report 2015*. Aug. 2015. URL: [https://www.standishgroup.com/sample\\_research\\_files/CHAOSReport2015-Final.pdf](https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf) (visited on 03/28/2023).
- [5] Jürgen Laartz Michael Bloch Sven Blumberg. *Delivering large-scale IT projects on time, on budget, and on value*. Oct. 2012. URL: <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value> (visited on 03/28/2023).
- [6] Barry W. Boehm. *Software Cost Estimation with COCOMO II*. Prentice Hall, Aug. 2000.
- [7] Vouchercloud. *How Many Productive Hours in a Work Day? Just 2 Hours, 23 Minutes...* 2019. URL: <https://www.vouchercloud.com/resources/office-worker-productivity> (visited on 04/04/2023).
- [8] Kathy Morris. *HERE'S HOW MANY HOURS WORKERS ARE ACTUALLY PRODUCTIVE (AND WHAT THEY'RE DOING INSTEAD)*. Jan. 2023. URL: <https://www.zippia.com/advice/average-productive-hours-per-day/> (visited on 04/04/2023).
- [9] United States. Bureau of Naval Weapons. Special Projects Office. *Program Evaluation Research Task (PERT): Summary Report. Phase 2*. Special Projects Office, Bureau of Naval Weapons, Department of the Navy, 1958.
- [10] James E Kelley Jr and Morgan R Walker. "Critical-path planning and scheduling". In: *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. 1959, pp. 160–173.
- [11] Mike Cohn. *What Are Story Points and Why Do We Use Them?* Dec. 2022. URL: <https://www.mountangoatsoftware.com/blog/what-are-story-points> (visited on 05/02/2023).

- [12] Samer Zein Ahmed Zarour. *Software development estimation techniques in industrial contexts: An exploratory multiple case-study*. Feb. 2019. URL: <https://www.learntechlib.org/p/207264/> (visited on 05/12/2023).
- [13] Mike Cohn. *Agile Estimation and Planning*. Robert C. Martin Series. Prentice Hall, 2005, p. 330.
- [14] Nikolai Valnakov Metodi Mazhdrakov Dobriyan Benov. *The Monte Carlo Method*. ACMO Academic Press, Aug. 2018.
- [15] Michael H. Goldwasser Michael T. Goodrich Roberto Tamassia. *Data Structures Algorithms in Java, 6th Edition International Student Version*. Wiley, June 2014.
- [16] URL: <https://www.ibm.com/topics/software-testing> (visited on 05/13/2023).
- [17] Grant Ongstad. *The Testing Pyramid: Understanding the Pros and Cons*. 2022. URL: <https://sofy.ai/blog/the-testing-pyramid-understanding-the-pros-and-cons/> (visited on 03/06/2023).
- [18] Daniel Knott. *The Mobile Test Pyramid*. 2015. URL: <https://www.ministryoftesting.com/articles/2102876b> (visited on 03/04/2023).
- [19] U.S Department of Health and Human Services. *Usability testing*. URL: <https://www.usability.gov/how-to-and-tools/methods/usability-testing.html> (visited on 03/30/2023).
- [20] Kate Moran. *Usability testing 101*. 2019. URL: <https://www.nngroup.com/articles/usability-testing-101/> (visited on 04/13/2023).
- [21] Aicha Diallo. *Moderated vs. unmoderated tests*. 2023. URL: <https://help.usertesting.com/hc/en-us/articles/360060344951-Moderated-vs-Unmoderated-Tests-> (visited on 04/29/2023).
- [22] The Story. *Remote usability testing vs. in-person usability testing*. 2022. URL: <https://thestory.is/en/journal/remote-and-in-person-usability-testing/> (visited on 04/28/2023).
- [23] Raluca Budiu. *Quantitative vs. qualitative usability testing*. 2017. URL: <https://www.nngroup.com/articles/quant-vs-qual/> (visited on 04/28/2023).
- [24] Hoa Loranger. *Checklist for Planning Usability Studies*. 2016. URL: <https://www.nngroup.com/articles/usability-test-checklist/> (visited on 05/02/2023).
- [25] Kara Pernice. *Talking with users in a usability test*. 2016. URL: <https://www.nngroup.com/articles/talking-to-users/> (visited on 05/02/2023).
- [26] *What is Continuous Integration? – Amazon Web Services*. [Online; accessed 16. May 2023]. May 2023. URL: <https://aws.amazon.com/devops/continuous-integration>.
- [27] *What is Continuous Delivery? – Amazon Web Services*. [Online; accessed 16. May 2023]. May 2023. URL: <https://aws.amazon.com/devops/continuous-delivery>.
- [28] *Continuous Deployment: An Essential Guide | IBM*. [Online; accessed 16. May 2023]. May 2023. URL: <https://www.ibm.com/topics/continuous-deployment>.
- [29] World Health Organization. *Disability*. 2022. URL: <https://www.who.int/news-room/fact-sheets/detail/disability-and-health> (visited on 03/04/2023).

- [30] Shawn Lawton Henry. *W3C Accessibility Standards Overview*. 2022. URL: <https://www.w3.org/WAI/standards-guidelines/#intro> (visited on 03/04/2023).
- [31] Accessibility Guidelines Working Group (AG WG) Participants. *Understanding the Four Principles of Accessibility*. 2022. URL: <https://www.w3.org/WAI/WCAG21/Understanding/intro#understanding-the-four-principles-of-accessibility> (visited on 03/04/2023).
- [32] Shawn Lawton Henry James Nurthen Michael Cooper. *WAI-ARIA Overview*. 2022. URL: <https://www.w3.org/WAI/standards-guidelines/aria/> (visited on 03/04/2023).
- [33] Can I use. *WAI-ARIA Accessibility features*. 2023. URL: <https://caniuse.com/?search=ARIA> (visited on 03/04/2023).
- [34] The Authority for Universal Design of ICT. *Gjeldende regelverk og krav (Current regulations and requirements)*. 2023. URL: <https://www.uutilsynet.no/regelverk/gjeldende-regelverk-og-krav/746> (visited on 03/08/2023).
- [35] Jésus Husbands. *UI Design Challenges and How to Deal with Them*. 2022. URL: <https://bootcamp.uxdesign.cc/ui-design-challenges-and-how-to-deal%20with-them-2437535cae16> (visited on 04/30/2023).
- [36] Educative Answers Team. *What are Norman's design principles?* 2023. URL: <https://www.educative.io/answers/what-are-normans-design-principles> (visited on 03/04/2023).
- [37] *Design iteration brings powerful results. so, do it again designer!* 2023. URL: <https://www.interaction-design.org/literature/article/design-iteration-brings-powerful-results-so-do-it-again-designer> (visited on 04/26/2023).
- [38] David Martin. *Website style guide - how to create a web design style guide*. 2022. URL: <https://uxhacks.com/website-style-guide/> (visited on 04/28/2023).
- [39] Studio by UXPin. *High-fidelity prototyping vs low-fidelity prototypes: Which to choose when?* 2023. URL: <https://www.uxpin.com/studio/blog/high-fidelity-prototyping-low-fidelity-difference/> (visited on 04/24/2023).
- [40] *JavaScript | MDN*. [Online; accessed 8. Mar. 2023]. Mar. 2023. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (visited on 03/08/2023).
- [41] *JavaScript With Syntax For Types*. [Online; accessed 8. Mar. 2023]. Mar. 2023. URL: <https://www.typescriptlang.org> (visited on 03/08/2023).
- [42] *HTML: HyperText Markup Language | MDN*. [Online; accessed 8. Mar. 2023]. Mar. 2023. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (visited on 03/08/2023).
- [43] *CSS: Cascading Style Sheets | MDN*. [Online; accessed 8. Mar. 2023]. Mar. 2023. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (visited on 03/08/2023).
- [44] Peter Loshin and Jessica Sirkin. *Structured Query Language (SQL)*. Feb. 2022. URL: <https://www.techtarget.com/searchdatamanagement/definition/SQL> (visited on 03/08/2023).
- [45] *What is a Virtual Machine? | VMware Glossary*. [Online; accessed 15. May 2023]. Aug. 2022. URL: <https://www.vmware.com/topics/glossary/content/virtual-machine.html> (visited on 05/15/2023).



- [46] *What is a reverse proxy? | Proxy servers explained.* [Online; accessed 15. May 2023]. May 2023. URL: <https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy> (visited on 05/15/2023).
- [47] *Git.* [Online; accessed 14. May 2023]. May 2023. URL: <https://git-scm.com> (visited on 05/14/2023).
- [48] Contributors to Wikimedia projects. *Git - Wikipedia.* [Online; accessed 21. May 2023]. May 2023. URL: <https://en.wikipedia.org/w/index.php?title=Git&oldid=1155506831>.
- [49] Michael Klein. *What Is Bash Used For?* Oct. 2021. URL: <https://www.codecademy.com/resources/blog/what-is-bash-used-for> (visited on 05/15/2023).
- [50] URL: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (visited on 05/15/2023).
- [51] Addy Osmani and Jason Miller. *Rendering on the web.* Feb. 2019. URL: <https://web.dev/rendering-on-the-web/> (visited on 05/15/2023).
- [52] Luke O'Neill. *What is Microsoft teams? everything you need to know.* 2021. URL: <https://www.techtarget.com/searchunifiedcommunications/definition/Microsoft-Teams> (visited on 05/13/2023).
- [53] URL: <https://www.atlassian.com/software/confluence/use-cases/wiki> (visited on 05/13/2023).
- [54] *Hello World - GitHub Docs.* [Online; accessed 14. May 2023]. May 2023. URL: <https://docs.github.com/en/get-started/quickstart/hello-world> (visited on 05/14/2023).
- [55] URL: <https://www.figma.com/> (visited on 05/13/2023).
- [56] URL: <https://developer.chrome.com/docs/lighthouse/overview/> (visited on 05/13/2023).
- [57] *Documentation for Visual Studio Code.* [Online; accessed 16. May 2023]. May 2023. URL: <https://code.visualstudio.com/docs>.
- [58] *Docker overview.* [Online; accessed 15. May 2023]. May 2023. URL: <https://docs.docker.com/get-started/overview> (visited on 05/15/2023).
- [59] Dave Page. *FAQ.* [Online; accessed 15. May 2023]. May 2023. URL: <https://www.pgadmin.org/faq> (visited on 05/15/2023).
- [60] traefik.io. *Traefik Proxy Documentation - Traefik.* [Online; accessed 15. May 2023]. May 2023. URL: <https://doc.traefik.io/traefik> (visited on 05/15/2023).
- [61] Contributors to Wikimedia projects. *Make (software) - Wikipedia.* [Online; accessed 15. May 2023]. May 2023. URL: [https://en.wikipedia.org/w/index.php?title=Make\\_\(software\)&oldid=1152862650](https://en.wikipedia.org/w/index.php?title=Make_(software)&oldid=1152862650) (visited on 05/15/2023).
- [62] URL: <https://storybook.js.org/> (visited on 05/13/2023).
- [63] URL: <https://playwright.dev/> (visited on 05/13/2023).
- [64] *About npm | npm Docs.* [Online; accessed 15. May 2023]. May 2023. URL: <https://docs.npmjs.com/about-npm> (visited on 05/15/2023).
- [65] pnpm. *pnpm.* [Online; accessed 15. May 2023]. May 2023. URL: <https://github.com/pnpm/pnpm> (visited on 05/15/2023).

- [66] *Svelte • Cybernetically enhanced web apps*. [Online; accessed 8. Mar. 2023]. Mar. 2023. URL: <https://svelte.dev> (visited on 03/08/2023).
- [67] *Introduction • Docs • SvelteKit*. [Online; accessed 8. Mar. 2023]. Mar. 2023. URL: <https://kit.svelte.dev/docs/introduction> (visited on 03/08/2023).
- [68] URL: <https://sass-lang.com/guide> (visited on 05/13/2023).
- [69] *JSON*. [Online; accessed 14. May 2023]. Apr. 2023. URL: <https://www.json.org/json-en.html> (visited on 05/14/2023).
- [70] Dionysia Lemonaki. “What is YAML? The YML File Format”. In: *FreeCodeCamp* (Nov. 2022). URL: <https://www.freecodecamp.org/news/what-is-yaml-the-yml-file-format> (visited on 05/14/2023).
- [71] *Openstack at NTNU - SkyHigh - NTNU Wiki*. [Online; accessed 17. May 2023]. May 2023. URL: <https://www.ntnu.no/wiki/display/skyhigh>.
- [72] Stian Sandberg. *WebApi.no*. 2022. URL: <https://webapi.no> (visited on 05/20/2023).
- [73] *Scrum Guide | Scrum Guides*. [Online; accessed 21. May 2023]. May 2023. URL: <https://scrumguides.org/scrum-guide.html>.



# APPENDICES

## A - PRELIMINARY PROJECT PLAN

---

**Project no. 15**

**Software Project Cost Estimation  
Preliminary project plan**

**Version <1.0>**

---

## Project no. 15

### Revision history

<b>Date</b>	<b>Version</b>	<b>Description</b>	<b>Author</b>
19/Jan/23	0.1	Initial version	Janita, Sakarias, Espen
25/Jan/23	0.2	Finalized first draft	Janita, Sakarias, Espen
01/Feb/23	1.0	Revised according to feedback from supervisor	Janita, Sakarias, Espen

---

# Project no. 15

## Table of contents

Table of contents .....	3
1. Goals and Frameworks .....	4
1.1 Orientation .....	4
1.2 Research question/project description and outcome goals .....	4
1.3 Impact goals .....	4
1.4 Frameworks .....	5
2. Organization .....	6
3. Implementation .....	6
3.1. Main activities .....	6
3.2. Milestones .....	8
4. Follow-up and Quality Assurance .....	8
4.1 Quality assurance .....	8
4.2 Reporting .....	8
5. Risk assessment .....	9
6. Annex .....	9
6.1 Schedule .....	9
6.2 Agreement documents .....	9
6.2.1 Work contract for the bachelor group .....	9
6.2.2 3-party agreement .....	9



---

# Project no. 15

## 1. Goals and Frameworks

### 1.1 Orientation

We chose this task because inaccuracy in cost and time estimation is a widely known problem that all companies working with projects experience. A significant amount of research has been done in this field, but due to the near infinite scope of the issue, a true, guaranteed solution has not yet, and may never be discovered, but conducting more research into finding a solution can bring us more enlightenment and put us in a stronger position to make more confident estimates in the future.

### 1.2 Research question/project description and outcome goals

The task consists of developing a tool that consumes a project description and provides a project plan, including its cost and time. In this thesis, we will focus on building a tool that gives us estimates with low effort: Streamlining (and documenting) the estimation process and clarifying/documenting the inputs it requires. The focus is therefore automation as opposed to accuracy, which the product owner will tackle in a later phase.

The biggest challenge lies in the cost estimation itself; Many factors affect the overall project duration: project complexity, learning curves for the selected technologies, domain expertise, developer experience, developers' availability, uncertainty of the requirements, etc.

Another challenge will be balancing complexity against accuracy. The project owner requires a tool that is minimally complex while still providing estimates of adequate accuracy. Identifying the most significant variables will therefore be a preliminary goal.

There is also an aspect of psychology involved in making estimations that needs to be considered. The human mind has an average time prediction error of 20-30%<sup>1</sup>, so, finding a way to minimize the manual estimation done by the developers could potentially prove beneficial. Eliminating this factor, however, would require defining every possible task in all potential projects for the product owner, which is far too complex and out of scope, so some manual estimation will be required. Because of this we will not be able to remove this factor entirely.

### 1.3 Impact goals

Our goal is to create a demonstrator of an application that provides estimations that are adequately feasible as defined by the product owner, of time and cost for projects that the project owner may use in customer interactions and project planning. Being able to successfully automate these calculations and give reliable and transparent estimates will give the project leads a better understanding of the scope of a project which they may use in customer interactions and result in a better customer satisfaction and a smoother development process. It is however important to note that we are building an example of implementation which our product owner will later validate, and if valid will then continue to develop.

To concretize the goals, the product owner has provided us with the following user stories:

---

<sup>1</sup> Halkjelsvik T, Jørgensen M (2012) From origami to software development: a review of studies on judgment-based predictions of performance time. *Psychol Bull* 138(2):238–271

---

## Project no. 15

As a project manager, I want to...

1. (Must) given an existing project estimated by the team, get an expected delivery date (calendar time estimate).
2. (Must) given an existing project estimated by the team, get a cost estimation.
3. (Must) see the most likely release plan, the best and worst (e.g., burn up chart), possibly including constraints (i.e., budget, deadline, features).
4. (Must) create a project with tasks / add / delete.
5. (Must) assign tasks to a specific employee or group of employees.
6. (Must) define cost parameters for the team (e.g., hourly rate, profit margin).
7. (Must) set the starting date of the project.
8. (Should) manage availability for the team (e.g., vacations, illness, part-time allocation, etc.).
9. (Should) update the plan with actual project execution data (task started, task ended).
10. (Should) identify delivery (features) in the plan.
11. (Should) mark feature as Must-have, could-have, should-have, or will not have.
12. (Should) refine a task into a finer-grain task flow, that can be estimated by the developer.
13. (Could) see the prediction of a specific task, possibly depending on a given starting date.
14. (Could) import availability from AxBit ERP system.
15. (Could) decide how T-shirt sizes map to ideal time.
16. (Could) decide how story points map to ideal time.
17. (Could) export existing projects (tasks) from external sources (e.g., JIRA, Asana, etc.).
18. (Could) import existing projects (tasks) from external sources (e.g., JIRA, Asana, etc.).
19. (Could) as the project progresses, see the current estimation bias and absolute error.
20. (Could) see the utilization of staff, who is hiding sleeping, who is overloaded.
21. (Could) detect bottleneck tasks.

As a developer, I want to...

1. (Must) estimate a task in "ideal time" (effective hours).
2. (Should) be able to say when I start and when I am done with a task.
3. (Could) see my own estimation bias and absolute estimation error.
4. (Could) estimate task durations in story points.
5. (Could) estimate task durations in T-shirt sizes.

These user stories are separated into 'must', 'should' and 'could'. The functionality described as 'must' specify the requirements for the MVP. The 'should' functionalities will have priority over the 'could' when it comes to finalizing the project.

### 1.4 Frameworks

The team, in collaboration with the product owner, has chosen to create a web application, considering the ease of use we wish to achieve with this system. Svelte Kit, a JavaScript framework, will be used for the front-end and backend development. Furthermore, we will use PostgreSQL for data persistency. SASS, Syntactically Awesome Style Sheets, will be used for more comprehensible and readable styling, and Tailwind CSS for faster development of arbitrary styling. Other tools we will consider using are cypress or playwright for integration tests in browser, and Carbon Design System

---

## Project no. 15

to quickly set up a usable application. We will use Docker for containerization of the app for smoother testing and deployment.

GitHub will be used for version control, task management and CI/CD pipeline. Figma will be used for designing and planning the system as well as the development of the system through wireframes and diagrams.

We will need access to a server for the deployment of our application. We are considering an OpenStack server which would be provided to us by NTNU.

### 2. Organization

Franck Chauvel (AxBit) will be involved in the project as the product owner and will provide requirements and necessities, and feedback during the development process.

Anniken Karlsen (NTNU) will be involved as a supervisor, providing feedback and guidance.

Janita Lillevik Røyseth will be the lead UI/UX designer.

Sakarias Sæterstøl will be the lead DevOps architect.

Espen Otlo will be the lead system architect.

In addition to the aforementioned roles, Janita, Sakarias, and Espen will work across roles to ensure continuous development.

### 3. Implementation

#### 3.1. Main activities

**Requirements gathering:** This includes identifying the needs and wants of the app users and product owner and defining the scope and objectives of the project. **Who does it:** All team members. **Why is it done:** To understand the project's need, define the scope, and set the objectives. **How is it done:** By conducting user research and interviews. **When is it done:** At the beginning of the project. **Necessary conditions before it can be done:** Product owner availability, and scope.

**Design and prototyping:** This includes creating wireframes, mockups, and interactive prototypes of the app's user interface and user experience using Figma. **Who does it:** Janita and Sakarias. **Why is it done:** To create a visual representation of the app, to test the usability, and to get feedback. **How is it done:** By using Figma design tool. **When is it done:** After requirements gathering and before development. **Necessary conditions before it can be done:** Requirements and scope defined.

**Development:** This includes writing code to build the app, testing it, and fixing any bugs that are found using Svelte Kit for frontend and backend, and incorporating styling and theme using Tailwind CSS. **Who does it:** All team members. **Why is it done:** To bring the app to life and make it functional. **How is it done:** By using Svelte Kit, Tailwind CSS, and related technologies. **When is it done:** After design and prototyping. **Necessary conditions before it can be done:** Design and requirements defined.

---

## Project no. 15

**Database set up:** This includes setting up the app's database and configuring it to work with the app using PostgreSQL. Who does it: Sakarias and Espen. Why is it done: To add persistence to the app. How is it done: By using PostgreSQL and Docker. When is it done: After design and prototyping. Necessary conditions before it can be done: Design and requirements defined.

**Containerization:** This includes packaging the application and its dependencies in a container using Docker. Who does it: Sakarias. Why is it done: To make the app easy to deploy and run consistently across different environments. How is it done: By using Docker containers. When is it done: During development. Necessary conditions before it can be done: Project skeleton finished.

**Testing:** This includes verifying that the app meets the requirements, is functional and is free of bugs using integration test and unit tests with Cypress and Vitest. Who does it: All team members. Why is it done: To ensure that the app is of high quality, meets the requirements and is usable. How is it done: By using Cypress for automated testing. When is it done: After development and containerization. Necessary conditions before it can be done: Development, Database set up, containerization completed.

**Deployment:** This includes launching the app on the web, on the client's servers or cloud. Who does it: Sakarias. Why is it done: To make the app available to users. How is it done: By deploying the app on the client's servers, or on a server provided by the team/NTNU. When is it done: After testing. Necessary conditions before it can be done: Testing completed and approved.

**Reporting:** This includes drafting and finalizing the detailed report on the project, including design guidelines, what framework we used and why, how we implemented our solutions, how we did our testing and the results we found, diagrams showing our workflow and app designs, how we deployed the app and why we chose to do it this way as well as meeting minutes. Why is it done: To track the progress of the project, identify issues, evaluate our performance, and document our accomplishments. How is it done: By using Microsoft Word. When is it done: Before, during and after development. Necessary conditions before it can be done: Each segment of the report has its own requirements and should be written as soon as these requirements are met.

**Presentation:** This includes preparation of two presentations, one during the project in English and one on the end of the project. Who does it: All team members. Why is it done: To showcase our work to fellow students. How is it done: The presentations are to be made with Google Slides. When is it done: First presentation is to be held in April, the second one in May. Necessary conditions before it can be done: Feedback on preliminary project plan.

---

# Project no. 15

## 3.2. Milestones

Whom/who	Milestone	Deadline
Students, supervisor, product owner	First meeting	13.01.2022
Students	Completing preliminary project plan	27.01.2022
Supervisor	Feedback on preliminary project plan	03.02.2022
Students	English presentation	21.04.2022
Students	Deliver thesis to supervisor	05.05.2022
Supervisor	Feedback on thesis	5 workdays after delivery of thesis
Students	Completing poster for thesis	18.05.2022
Students	Presentation of thesis	~19.05.2022
Students	Delivery of thesis	22.05.2022

Figure 1 Milestones for the project

## 4. Follow-up and Quality Assurance

### 4.1 Quality assurance

Quality assurance is a critical component of software development and can greatly impact the success of a project. "Quality is never an accident; it is always the result of intelligent effort." - John Ruskin. This quote highlights the importance of putting effort into ensuring that the work produced is of high quality. In order to achieve this, all members are responsible for double-checking their own work and conducting peer-reviews of others' work, as well as ensuring that documents are handed off for proof-reading to the supervisor and product owner in a timely manner. Proof-reading by our supervisor and product owner will help to ensure that the deliverables meet the required standards and helps to identify any issues that may have been missed during the internal group meetings. By implementing this quality control process, the team can ensure that all deliverables meet the required standards and that any issues are identified and addressed before they are submitted to the supervisor and product owner. It also ensures that there is enough time for any necessary revisions or feedback to be incorporated before the final deadline.

### 4.2 Reporting

The group will keep their supervisor informed by providing regular progress reports in weekly meetings and will actively involve the product owner in the development of the system throughout the project. This ensures that the product owner is aware of the progress and any issues that may arise and can provide feedback on the direction of the project. The team will also use the weekly meetings as an opportunity to discuss challenges and collaborate to find solutions, align on priorities, and set goals for the next period.

---

# Project no. 15

## 5. Risk assessment

The task at hand is complex and large, with the potential to become even bigger. There is a risk that the project's scope may become too extensive for the group of 3 students to complete within the given timeframe while balancing other coursework and part-time jobs along with their studies. Other risks include limited availability of some group members due to caring for children and other personal obligations, as well as potential challenges in coordinating and cooperating effectively due to differing priorities and working styles among group members. It is important to identify and address these risks early on to mitigate any negative impact on the project's progress and outcome and to develop effective strategies for managing time and resources effectively. The result of these issues could be not meeting deadlines, falling behind on the project's desired progress, and failing the project. However, it is believed that the likelihood of this happening is lower than one would expect, as the group has proven to be strong and capable of overcoming challenges during their 2 years of working together, consistently supporting one another, and going great lengths for their group projects and has achieved favorable results in their previous projects.

Risk	Probability	Impact	Rating
Project scope becoming too extensive	3	5	15
Limited availability of group members	3	4	12
Challenges in coordinating and cooperating	2	3	6

Figure 2 Risk assessment where 1 is considered low and 5 is considered high.

## 6. Annex

The following documents will be delivered as separate files when submitted to Blackboard in January (mandatory work requirement), but not in the final delivery of the main report on May 20th!

### 6.1 Schedule

GANTT.pdf

(In separate file)

### 6.2 Agreement documents

#### 6.2.1 Work contract for the bachelor group

Work Contract.pdf

(In separate file)

#### 6.2.2 3-party agreement

Standard agreement.pdf

(In separate file)

## B - FIRST DRAFT WIREFRAME

### super awesome cost estimation web app

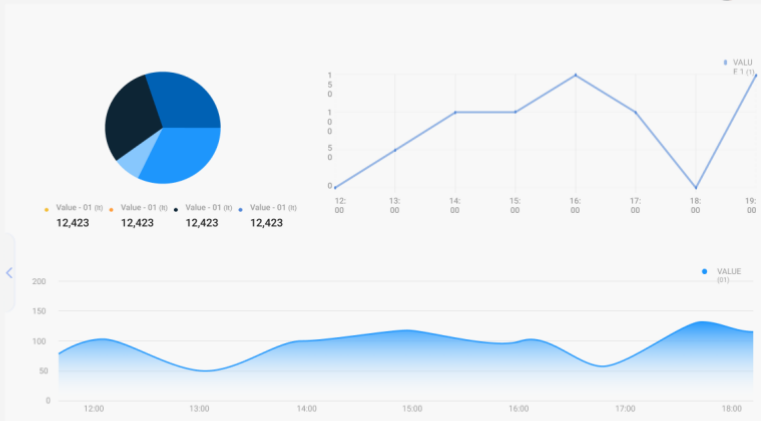
username


password

[sign in →](#)

Marvin McKinney  
Project Manager 




- estimation
- progress**
- board
- roadmap




Marvin McKinney  
Project Manager 

- estimation
- progress
- board**
- roadmap



to do 🕒

- whattodo 
- whattodo 
- whattodo 

in progress 🧑‍💻

- whatdoing 

code review 🧑‍💻

- iswrong 
- iswrong 

done ✓

- whatdone



## C - SECOND DRAFT WIREFRAME

### Cost estimation web app

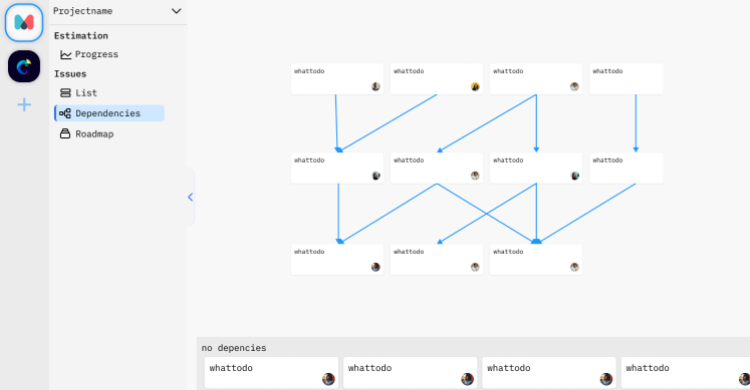
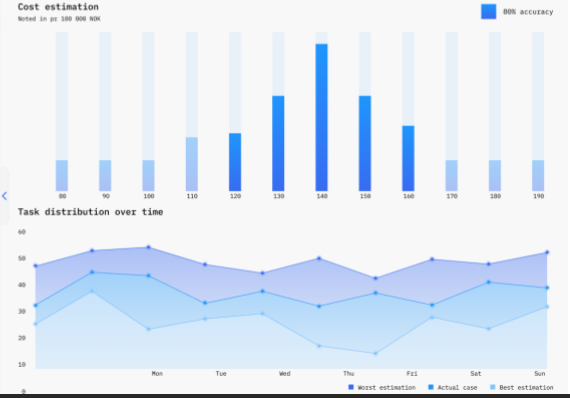
Username

Password

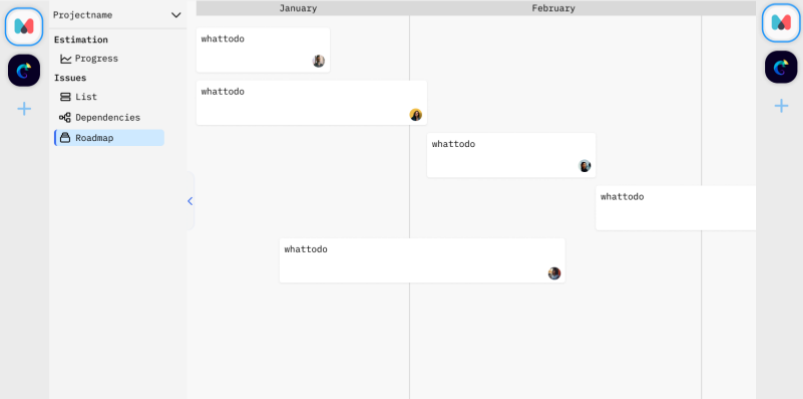
[Sign in](#)

[Don't have an account? Sign up here](#)

- Projectname
- Estimation
- Progress
- Issues
- List
- Dependencies
- Roadmap



Status	Title	Est.start	Est.end	Est.hours	Assignees
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤
🔄	whattodo	31/01/2023	31/01/2023	8 hours	👤



#### Project details



Change project icon

Only .jpg and .png files, 5 MB max file size.

Drag and drop files here or click to upload

Project name

Start date

mm/dd/yyyy

Profit margin

15%

#### Members

Project manager

Floyd Miles

Change project manager...

Team members

Name	Email	Task level	
Albert Flores	albert.flores@example.com	Task level: 100%	Remove member
Arlene McCoy	arlene.mccoy@example.com	Task level: 50%	Remove member
Darrell Steward	darrell.steward@example.com	Task level: 20%	Remove member

## D - FINAL DRAFT WIREFRAME

https://www.superawesomestimationwebapp.netlify.app

## Cost estimation web app

Username

Password

[Sign in →](#)

[Don't have an account? Sign up here →](#)

https://www.superawesomestimationwebapp.netlify.app/project/21424/board

Projectname

- Overview
- Progress
- Features
- Tasks
- Dependencies

Projectname

Starts: 27/02/2023  
Profit margin: 15%

Senior developer rate: 450 NOK/hour  
Junior developer rate: 300 NOK/hour

Customer budget: 1 500 000 NOK  
Customer deadline: 08/06/2023

Estimated cost (Best case): 1 200 000 NOK  
Estimated completion (Best case): 02/06/2023

Estimated cost (Most likely): 1 200 000 NOK  
Estimated completion (Most likely): 02/06/2023

Estimated cost (Worst case): 1 200 000 NOK  
Estimated completion (Worst case): 02/06/2023

Project manager: Floyd Miles

Team members

Name	Email	Level of assignment
Albert Flores	albert.flores@example.com	Level of assignment: 100%
Arlene McCoy	arlene.mccoy@example.com	Level of assignment: 50%
Darrell Steward	darrell.steward@example.com	Level of assignment: 20%

https://www.superawesomestimationwebapp.netlify.app/project/21424/board

Projectname

- Overview
- Progress
- Features
- Tasks
- Dependencies

### Feature completion over time

Feature	Best case	Most likely	Worst case
Feature 1	04/01/2023	06/01/2023	07/01/2023
Feature 2	07/01/2023	09/01/2023	10/01/2023
Feature 3	11/01/2023	15/01/2023	18/01/2023

https://www.superawesomestimationwebapp.netlify.app/project/21424/board

Projectname

- Overview
- Progress
- Features
- Tasks
- Dependencies

Search  [New feature](#)

#1 Feature title  
Estimated end (likely): 29/04/2023  
100% complete  
0 closed tasks 0 open tasks

#3 Feature title  
Estimated end (likely): 29/04/2023  
50% complete  
0 closed tasks 0 open tasks

#5 Feature title  
Estimated end (likely): 29/04/2023  
50% complete  
0 closed tasks 0 open tasks

#2 Feature title  
Estimated end (likely): 29/04/2023  
50% complete  
0 closed tasks 0 open tasks

#4 Feature title  
Estimated end (likely): 29/04/2023  
100% complete

https://www.superawesomestimationwebapp.netlify.app/project/21424/board

Projectname

- Overview
- Progress
- Features
- Tasks
- Dependencies

[New task](#)

Status	ID	Title	Dependency level	Feature id	End (likely)	End (Worst)	Ideal hours (Best)	Ideal hours (Worst)	Assignees
🔄	#21	whattodo	4	#2	27/02/23	27/02/23	3	8	
🔄	#3	whattodo	3	#3	27/02/23	27/02/23	2	4	
🔄	#2	whattodo	1	#5	27/02/23	27/02/23	1	3	
🔄	#5	whattodo	2	#3	27/02/23	27/02/23	5	12	
🔄	#8	whattodo	0	#5	27/02/23	27/02/23	4	11	
🔄	#6	whattodo	2	#5	27/02/23	27/02/23	3	4	
🔄	#7	whattodo	3	#2	27/02/23	27/02/23	1	3	
🔄	#9	whattodo	2	#5	27/02/23	27/02/23	10	11	
🔄	#12	whattodo	1	#3	27/02/23	27/02/23	12	15	
🔄	#11	whattodo	3	#3	27/02/23	27/02/23	2	20	

https://www.superawesomestimationwebapp.netlify.app/project/21424/board

Projectname

- Overview
- Progress
- Features
- Tasks
- Dependencies

no dependencies

whattodo

https://www.superawesomestimationwebapp.netlify.app/project/21424/board

Project details

Change icon or upload icon

Profit margin: 15%  
Budget:

Start date: mm/dd/yyyy  
DeadLine:

Project name:   
Senior developer rate:   
Junior developer rate:

Members

Project manager: Floyd Miles  
[Change project manager...](#)

Team members

Name	Email	Level of assignment	Remove
Albert Flores	albert.flores@example.com	Level of assignment	Remove
Arlene McCoy	arlene.mccoy@example.com	Level of assignment	Remove
Darrell Steward	darrell.steward@example.com	Level of assignment	Remove

## E - DESIGN GUIDELINES

# Design guidelines

## Theme

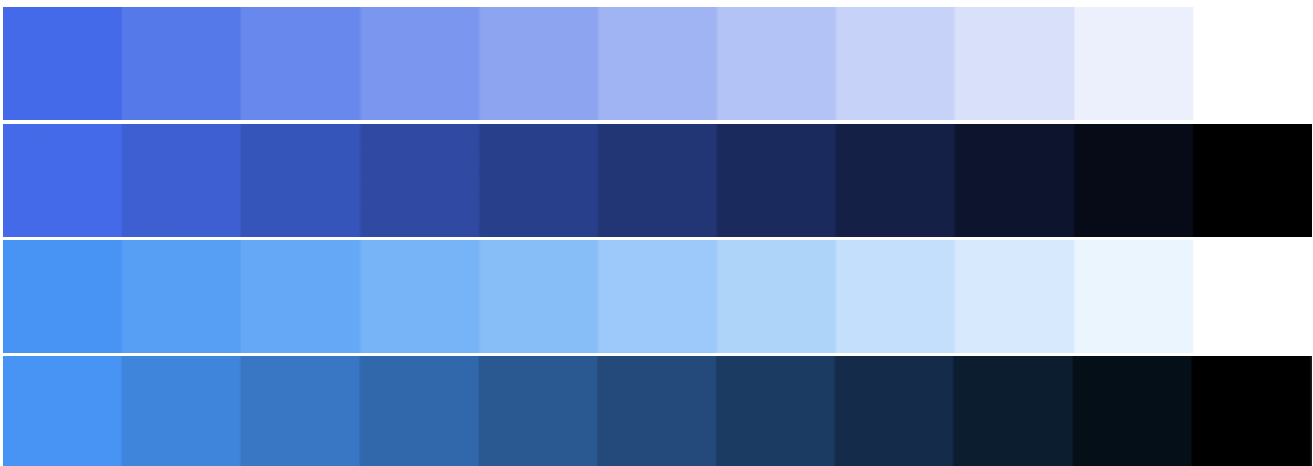
The system we are building is catered to software developing teams. To ensure efficient use, we are placing a strong emphasis on creating a user-friendly and intuitive interface. We aim for a professional and polished appearance, which is why our design concept balances a functional aesthetic with subtle touches of color and rounded edges, to create a visually pleasing and easy-to-use experience that is not overly playful.

## Color Scheme

We have been given the freedom to choose our own colors for this project. Given the professional nature of the cost estimation system we are building, we have chosen to primarily use various shades of grey for the background to create a sleek and sophisticated design. This will also help in differentiating between different sections of the system. As our primary color, we have chosen a shade of blue, as it evokes a sense of trust and reliability, which aligns well with the purpose of the system.

## Colors

Our color palette consists of two distinct shades of blue. Our primary blue color is #376BF1, while the second blue color, #1E96FC, is used for accents. We chose these colors with the goal of creating a professional theme. By using a different shade for accents, we aim to create visual interest without sacrificing the overall serious tone of our design.



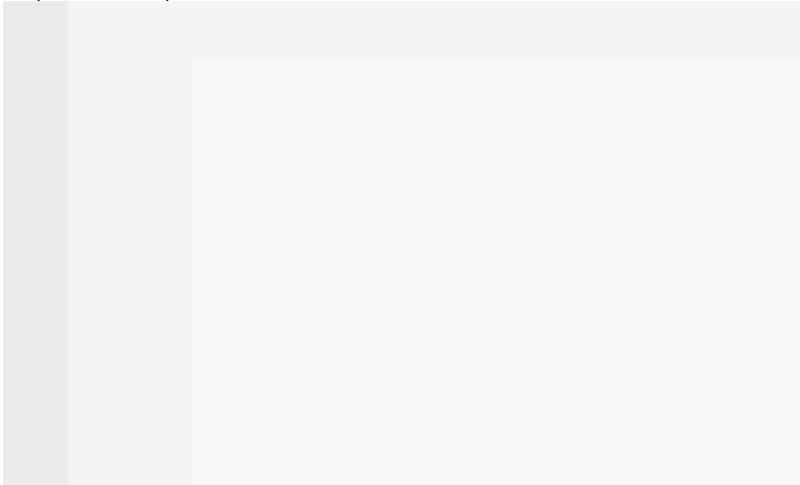
## Hierarchy and Layout

We do not have a specific message or product to promote, but rather the focus is on providing a user-friendly and efficient tool for cost estimation. Our main goal is to make the cost estimation process as smooth and easy as possible for our users. While the first page is important, it is not the primary focus of the system as it is not a standalone, consumer-facing app, but rather a tool for our product owners to use as an add-on to their existing workflow. Therefore, the emphasis should be placed on the functionality and usability of the system rather than the design of the first page.

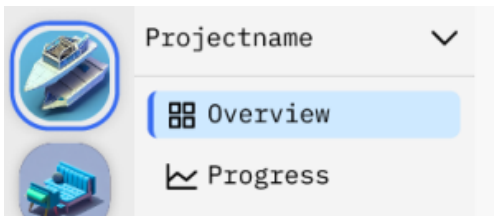
In order to provide a familiar and modern user experience, we have decided to adopt a similar layout as popular team communication tools such as Discord and Slack. Slack, in particular, is widely used by our target audience, our product owner and his team. This familiar layout will make it easy for users to navigate the app. Our main sections will consist of a sidebar for navigating different projects, a second sidebar for switching between different views within a project, and a header/top bar for user-related functionality, such as account settings and notifications, if time allows it. This layout will provide a clear and intuitive structure.

In order to increase the visibility of our main section we will have the project navigation a darker grey than the the project menu. The project menu will be a darker grey than the main section. The main section won't be white as the components shown in the main section should be more visual and not blend in too much with the background.

Simplified visual representation:



We will need to create accents for the chosen the project and the chosen menu item in the project menu to give visual representation to the user were they are.



## Images

We have chosen not to use images for decorative purposes in the design of our cost estimation system. However, projects will have the option to add an icon, and later on, we may allow users to upload their own profile images. Therefore, we have not planned on incorporating text over images, overlay on images, or blurring images in the design. If the need arises to add a decorative element to the design, we will consider these options, but they are not currently part of our plan. Instead, we will focus on creating a clean and professional look that prioritizes functionality and usability.

## Icons

In order to maintain a consistent and intuitive design, we have decided to use icons for different types of buttons throughout the app. Icons provide a clear visual representation of the action being performed, making it easier for users to understand and interact with the system. This approach aligns with our design principles of prioritizing functionality and simplicity. It is important to note that icons will be used only for buttons and not for decorative purposes to keep the system consistent. We will use either black or white icons depending on what's necessary to keep a preferable color contrast in regards to accessibility.

The icon pack chosen is [Carbon Design Systems Icons](#).

## Typography

When selecting the typeface for our system, we considered various options and how the typeface will effect readability and overall design. While sans-serif fonts are popular and considered modern, they may not be the best choice for a system that contains a large amount of information. Serif fonts, on the other hand, are more comfortable for reading large amounts of text. However, to align with our theme, a system for a dev team, we have chosen to use a monospace font. Monospace fonts are commonly associated with coding and technical work, making them a fitting choice for our target audience. This font choice not only enhances the overall theme of the application, but also makes it easier for the users when reading and interpreting the information.

The font chosen for the application is IBM Plex Mono. We will use this font for all purposes in our design.

The font scale is as following:

- 0.625rem / 10px
- 0.750rem / 12px
- 0.875rem / 14px
- 1.000rem / 16px
- 1.250rem / 20px
- 1.500rem / 24px
- 2.000rem / 32px
- 2.500rem / 40px
- 3.000rem / 48px

## Spacing

The space scale is as following:

- 0.125rem / 2px
- 0.250rem / 4px
- 0.500rem / 8px
- 0.750rem / 12px
- 1.000rem / 16px
- 1.500rem / 24px
- 2.000rem / 32px
- 3.000rem / 48px
- 4.000rem / 64px
- 5.000rem / 80px
- 6.000rem / 96px
- 8.000rem / 128px
- ...

## Border radius

With regards to the overall design theme, we aim to create a look that is professional and sophisticated, while still being approachable and user-friendly. To achieve this balance, we will incorporate subtle soft edges in the design, giving it a rounded appearance that is not overly playful. At the same time, we will avoid making the design too round or circular to maintain a serious and professional look that aligns with the purpose of the software.

Buttons: 5px

Project icons: 25px

Project navigation buttons: round

The reason for the difference in rounding is to create a visual distinction between elements that do not have similar functionality. This approach improves the overall usability and user experience by making it easier for users to quickly identify and interact with the different elements on the page.

## Shadows

Due to the same reason specified in the border radius section, we will not have too much shadow, but aim for a small shadow to separate the different elements especially when it comes to modals.



## F - TEST TEMPLATE

# Testing

Template for documenting user tests:

## *User test n'th iteration - Tester name*

*Test performed: dd/MM/yyyy*

### *Introduction*

The introduction we sent to the user and/or how we introduced the test to user.

### *Background*

(Contains information about the user being tested on)

*Name: Testers name*

*Age: Testers age*

*Profession: Testers profession*

*Previous experience: What previous experience does the tester have relating to the system we are building.*

### *Tasks*

*(Predefined tasks the user should test)*

1. *Test feature.*
  - a. *Do this with feature.*
  - b. *Do that with feature.*
2. *Test another feature.*

### *Feedback*

*(How did the user do on the tasks, what more feedback did the user provide beyond the tasks?)*

<b>Task</b>	<b>How did the user perform?</b>	<b>Feedback</b>
1.a	<i>Write how the user performed</i>	<i>What did the user think themselves?</i>
2.b		
2		

*Feedback outside of tasks:*

- *Feedback given outside of a task*
- *should be written in a list like this.*

### *Analysis*

*(At a later stage, we write our conclusion of the user test and write what we have decided to change or why not)*

## G - 1ST ITERATION USER TESTS

# 1st iteration

For our first user test iteration, we have created an interactive wireframe prototype using Figma, [see prototype](#). The wireframe represents a high-fidelity prototype that closely resembles our planned implementation of the software. Our decision to create a high-fidelity prototype was based on the small size and busy schedules of our test group. It was not feasible to create a low-fidelity prototype and do many test rounds as one of our testers is located in another city.

The main objective of the first test iteration is to test the usability of layout and test how this implementation translated to the concept of the application.

The user we are testing on for this iteration is:

- User A, system engineer
- User B, project manager

While having someone a part of our team be a tester is not ideal, Franck is also our product owner and "customer", and so given the limited pool of testers, it will suffice for this project.

The tasks we will ask our users to perform is as following:

1. Sign-in screen
  1. Sign up
2. Progress
  1. Change chart
3. Project
  1. Search project
  2. Create project
    1. Set task level
  3. View project
  4. Edit project
  5. Delete project
4. Feature
  1. View feature list
  2. Create feature
    1. Add connected task
  3. Edit feature
    1. Edit title
    2. Edit description
  4. Delete feature
5. Task

1. View task list
2. Create task
  1. Add dependency
3. Edit task
  1. Edit title
  2. Edit description
  3. Add child issue
  4. Add assignee
6. Dependencies
  1. View dependencies
7. Roadmap
  1. View roadmap

## H - 1ST ITERATION USER TESTS - USER A

# User test 1st iteration – User A

Test performed: 16/02/2023 - 09:00-10:15

## Introduction

User A is a part of our team but is also our "customer" in a sense. Because of the lack of testers, User A was asked to do user tests as well. Because User A is a part of our team, they were given an informal introduction to the user tests and know that we won't implement all the features in the wireframe. It is still really beneficial for our project that User A tests the wireframe so they can give us feedback and make sure that we have interpreted his idea for this cost estimation software correctly.

## Background

Name: User A

Age: 41

Profession: Software engineer

Previous experience: A few years with agile projects, and also other types of estimations.

## Tasks

1. Sign-in screen
  1. Sign up
2. Progress
  1. Change chart
3. Project
  1. Search project
  2. Create project
    1. Set task level
  3. View project
  4. Edit project
  5. Delete project
4. Feature
  1. View feature list
  2. Create feature
    1. Add connected task
  3. Edit feature
    1. Edit title
    2. Edit description
  4. Delete feature

5. Task
  1. View task list
  2. Create task
    1. Add dependency
  3. Edit task
    1. Edit title
    2. Edit description
    3. Add child issue
    4. Add assignee
6. Progress
  1. Change chart
7. Dependencies
  1. View dependencies
8. Roadmap
  1. View roadmap

## Feedback

Task	How did the user perform?	Feedback
1.a	Good	Nothing noteworthy.
2.a	Good	Buttons instead of dropdown menu so it is not hidden.
3.a	Good, found in second attempt.	Nothing noteworthy.
3.b	Good	Optional starting date, customers budget, customer deadline.
3.b.i	Good	Nothing noteworthy.
3.c	Good	Nothing noteworthy.
3.d	Good	Estimated budget, time and cost.
3.e	Good	Nothing noteworthy.



<b>Task</b>	<b>How did the user perform?</b>	<b>Feedback</b>
4.a	Good	Manually order the feature list, grouped releases, time estimation should be bigger, ID on features.
4.b	Good	Nothing noteworthy.
4.b.i	Good	Nothing noteworthy.
4.c	Good	Nothing noteworthy.
4.c.i	Good	Nothing noteworthy.
4.c.ii	Good	Nothing noteworthy.
4.d	Didn't find right away.	Should be easier to delete.
5.a	Good	Ideal duration instead estimated hours. Uncertainty in list and modal, maybe change the uncertainty through buttons and radio buttons to swap between showing start date and end date.
5.b	Good	Nothing noteworthy.
5.b.i	Good	Nothing noteworthy.
5.c	Good	Quicker way to edit ideal duration, a panel instead of a modal. Summary of how many child issues and dependencies.
5.c.i	Good	Nothing noteworthy.
5.c.ii	Good	Nothing noteworthy.
5.c.iii	Good	Nothing noteworthy.
5.c.iv	Good	Nothing noteworthy.

Task	How did the user perform?	Feedback
6.a	Good	Drag and drop dependencies would be nice but not important for now.
7.a	Good	Not useful, gantt is not used in AxBit,

Feedback outside of tasks:

- Show uncertainty is a top priority

## Analysis

### User feedback:

In this [prototype](#) the first page to show when opening a project was the progress page and the charts was toggled through a dropdown menu. It would be better to have buttons to swap between the charts so the options aren't hidden. The landing page when opening a project was not intuitive, it would be better to create a project overview for the project with much of the same information provided in the edit project page and maybe more like the most likely estimated budget and end date. When creating a project the starting date for the project should be optional and possible to add the customers budget and the customers preferred deadline. The feature list should be possible to manually order in regards to the priority of what should be done, the features should also have an id to make it easier to distinguish and the time estimation should be bigger. The features should also be easier to delete. For the task list view there was wrong terminology for estimated hours, a preferred term is ideal duration. The task list was lacking information like uncertainty, which should be a part of the list and modal. Alternatively the uncertainty could be toggled through buttons so you see the estimation for the different levels of estimation, like best case, most likely and worst case. Also it would be good to have some means to change the columns that are shown in the list. When it comes to editing the tasks it would be beneficial to edit some fields quicker in the list itself, instead of opening a modal. It could also be better to have a side panel open instead of a modal, giving a better view of the list and its information while editing a task. The editing view itself could still show more information like number child issues and dependencies, and should have two fields for ideal duration, a best case and worst case. The dependency graph should have more information for what is shown and it would be nice to drag and drop dependencies between task in the graph, but it is however not important for now. The roadmap is redundant as it is not used in AxBit.

**What we will change:**

We will add a project overview page to display the same information as when editing the information and the last estimation for budget and estimated end date. We will also add customers budget and customers deadline as input fields for project, This can be especially helpful in graphs to see how risky a project will be. In terms of creating/editing project the starting date will also be optional. We will also be adding another estimated hours field so we can have best case and worst case as inputs for tasks. In the task details modal we will also add a summary or a number of child tasks and dependent tasks. Also the term will be ideal duration instead of estimated hours. IDs will be added to features and tasks. The drop down menu for the progress page will be removed, and in the case of more graphs being added we will add buttons instead to display the options right away instead of having them hidden. The roadmap will be removed as it is not useful for AxBit.

**What we might change:**

While a highly sought after feature and definitely a priority the manually ordering of features is not easily implemented as of now and will require restructuring of backend. Because of this, the manually ordering of features is a definitely later, not a maybe. A side panel was a wish instead of a modal, it has positive qualities as one can see the list while editing/viewing information, but we wish to get something up right away and might look into it later. There was also a wish for buttons to toggle information of uncertainty in the task list and toggling fields, we believe a way of optionally choosing what features is shown might be a better way than toggling. So we will show the uncertainty when we come to that stage, but the toggling of columns might be later. Dragging and dropping dependencies would be cool in the dependency page however it is far from a priority and is not something to spend time on as of now. Grouping releases was also mentioned however this will wait until later when we implement releases.

**What we won't change:**

We won't be changing the location for deleting tasks and feature, whether it is beneficial to have feature and task easily deleted varied, and keeping it as is keeps the ui clean.

## I - 1ST ITERATION USER TESTS - USER B

# User test 1st iteration – User B

Test performed: 16/02/2023 - 13:00-14:15

## Introduction

User B is not a part of our team and had no knowledge of what this project was about, Franck Chauvel took it upon himself to spend 5 minutes before our user test to show User B the following the powerpoint informing what the project was about:

[user-test.pptx](#)

## Background

Age: 39

Profession: Sales manager / Project manager

Previous experience: 2 years experience with estimation as project manager

## Tasks

1. Sign-in screen
  1. Sign up
2. Progress
  1. Change chart
3. Project
  1. Search project
  2. Create project
    1. Set task level
  3. View project
  4. Edit project
  5. Delete project
4. Feature
  1. View feature list
  2. Create feature
    1. Add connected task
  3. Edit feature
    1. Edit title
    2. Edit description
  4. Delete feature
5. Task
  1. View task list
  2. Create task

- 1. Add dependency
- 3. Edit task
  - 1. Edit title
  - 2. Edit description
  - 3. Add child issue
  - 4. Add assignee
- 6. Dependencies
  - 1. View dependencies
- 7. Roadmap
  - 1. View roadmap

## Feedback

Task	How did the user perform?	Feedback
1.a	Okay	Nothing noteworthy.
2.a	Confusing	One graph view is enough.
3.a	Good	Change position of the search button.
3.b	Good	"Remove" is enough, do not need to add "member", or use a trashcan for removing a member. Back instead of prev.
3.b.i	Good	Coverage?
3.c	Good	Nothing noteworthy.
3.d	Good	Slack integration, an "x" to cancel the editing.
3.e	Good	Confirm dialog.
4.a	Good	Total tasks in feature. Add dependencies between features. Epic instead? Show ID number for feature.

<b>Task</b>	<b>How did the user perform?</b>	<b>Feedback</b>
4.b	Good	Have requirements for feature description, more inputs? technologies?
4.b.i	Good	Nothing noteworthy.
4.c	Good	Nothing noteworthy.
4.c.i	Good	Nothing noteworthy.
4.c.ii	Good	Nothing noteworthy.
4.d	Good	Nothing noteworthy.
5.a	Good	Have IDs for tasks, show dependency degree of task.
5.b	Good	Help with description, remind user of things to think of.
5.b.i	Good	Nothing noteworthy.
5.c	Good	Too simple to edit a task, better to do it through the dot menu? Show which feature the task is connected to. Colors on the estimation of whether its best or worst?
5.c.i	Good	Nothing noteworthy.
5.c.ii	Good	Nothing noteworthy.
5.c.iii	Good	Nothing noteworthy.
5.c.iv	Good	Nothing noteworthy.
6.a	Good	Search bar instead of a horizontal list, simpler to have a list?
7.a	<i>Not tested with User B</i>	

Feedback outside of tasks:

- What role does an assignee have?
- Catered to not use icon for project, dummy icons? three first letters?
- Notification when uncertainty is close to being reached.

## Analysis

### User feedback:

For this [prototype](#) we removed the roadmap as it was not useful in anyway for our software, but this was our only change. We got feedback again with this user test that it would be beneficial to have a project overview as a landing page instead of diving straight into the information. It was also not ideal to have multiple charts, but one chart view. When testing the creation of a project, we got feedback that the wireframe had redundant terminology and could use simpler terms, like "Remove" or a trashcan instead of "Remove member", and use "Back" instead of "Prev" for navigating modals. The project should also be catered to not using an icon as few project does have something visually to be an icon. When editing a project it would be better to have an "x", or a means to escape the editing mode, and deleting projects should have a confirmation modal. With level of assignment in a project there was some feedback regarding temporary leaves. The feature list could show more information like the total number of tasks, and maybe it should be able to add dependencies between features. Epic maybe a better term than feature as it more aligns with Jira. The tasks should have IDs too, and show the dependency degree of tasks. When editing a task it could be useful to have a tooltip for the description, to remind the user of helpful things to add to the description. It was also too easy to edit the task, one would expect one more step with going through the dot menu button and editing. The task modal should also display what feature it belongs to. Another point User B brought up was having colored inputs for best case estimation and worst case estimation when viewing the task information. For the dependency page it might be helpful to have a search bar, and it's simpler to have a list.

### What we will change:

There were similarities between User B's feedback and Franck's feedback, like a project overview as a kind of landing page when opening a project. We will address this and create a project overview. Another common feedback amongst our two testers was ID's of features and tasks, this is a simple change that we will address immediately. Other feedback User B gave us was to simplify the terms used so "Remove member" will be changed to just "Remove", and «Back» instead of «Prev» for navigating the modals. The projects will also be more catered towards not using an icon, as well as still allowing to upload an icon we will maybe provide dummy icons or use three first letter. We will also add a confirmation modal when anyone attempts to delete a project as User B gave us feedback for. It was also sufficient enough with just one graph view, and we will make the progress page simpler and the information more clear. Editing a task and feature was also too simple, it wasn't intuitive so we will add "edit \*\*\*\*\*" in the dot menu for feature and tasks. We will also add more help with the description of feature



and task, perhaps a more descriptive placeholder or a tooltip. When editing projects we will also add an "x" or another means to escape the editing view.

### **What we might change:**

Other feedback User B provided which is thoughts for later is search field on the dependency page, list in dependency tasks and child tasks. We will wait with the search field in the dependency page as the dependency page its not priority. User B also mentioned a sort of warning when the estimation is closing worst case uncertainty as it will give the project manager time to update the customer. This warning is down prioritized till a later stage but it definitely has value and we hope to implement at one point.

### **What we won't change:**

What we will not be implementing however is changing the location of the search button, while it is not in an ideal position having a search bar on top while there are many other search bars will be cluttered with repetitive looks and might be confusing. A search bar over the project menu, which would look akin to how discord is in our opinion not ideal either as it is confusing in terms of what the menu represents, it holds items for a project, and searching projects there is not intuitive. Placing the search button on top won't look good either as the search and add buttons should be grouped, and add buttons are normal to stay on the bottom in this kind of layout. None of our users really struggled with the position, it took at most a second glance and is a learning matter in this case. User B mostly mentioned this as a fun idea but slack integration would be fun when seeing list of members in a project, to easily communicate as they use Slack in AxBit, however given the scope of our project we won't be attempting this for now. The last feedback was given in a discussion about the results of the user test with Franck, where we will add another input field for estimated hours so they can add best case and worst case, in this regard User B mentioned coloring the input fields, while a fun idea a red input field can be confusing as it often mean wrong input, we might play with the idea later as coloring a label but not for now.

## J - 2ND ITERATION USER TESTS

## 2nd iteration

For the second user test iteration, we have an MVP of our demonstrator available on the URL: <https://dev.savvyest.io>. This version of our application is capable of creating and editing projects, features and tasks as well as performing simulation-based estimation through the information provided by the features and tasks in a project.

Our objective with the second iteration of the user test is to test our previous changes with the last user test, but most importantly it is testing the simulation and the data provided and displayed from the simulation. We will emphasize the feature-completion chart and how the users manage to decipher it.

The user we are testing on for this iteration is:

- User A, project manager
- User B, HR and administration

The tasks we will ask our users to perform is as following:

1. Sign-in screen
  1. Sign in
2. Project
  1. View projects
  2. Search project
  3. Create project
    1. Assignees and level of assignment
  4. Edit project
3. Feature
  1. View feature list
  2. Create feature
  3. Edit feature
4. Task
  1. View task list
    1. Information provided in the table (like depth?).
  2. Create task
    1. Dependencies
    2. Child tasks
  3. Edit task
5. Progress
  1. Running a simulation
  2. Reading the chart

### 3. Reading the table

## K - 2ND ITERATION USER TESTS - USER A

# User test 2nd iteration – User A

Test performed: 03/05/2023

## Introduction

User A was a test user for our previous iteration and therefore was familiar with the concept of our application, we sent the following email to her to invite her back for the second iteration of user tests (translated):

*Hi User A!*

*Thank you so much for last time!*

*Our application is nearing readiness for demo. On this occasion, we hope that you would be willing to act as a test user again.*

*We also wonder if you might know more people who could be willing and have the ability to be test users.*

*We hope to have a demo-ready application at the beginning of next week.*

*Hope to hear from you*

## Background

Age: 39

Profession: Sales manager / Project manager

Previous experience: 2 years experience with estimation as project manager

## Tasks

1. Sign-in screen
  1. Sign in
2. Project
  1. View projects
  2. Search project
  3. Create project
    1. Assignees and level of assignment
  4. Edit project
3. Feature
  1. View feature list

- 2. Create feature
- 3. Edit feature
- 4. Task
  - 1. View task list
    - 1. Information provided in the table (like depth?).
  - 2. Create task
    - 1. Dependencies
    - 2. Child tasks
  - 3. Edit task
- 5. Progress
  - 1. Running a simulation
  - 2. Reading the chart
  - 3. Reading the table

## Feedback

Task	How did the user perform?	Feedback
1.a	Excellent	Nothing noteworthy
2.a	Excellent	Nothing noteworthy
2.b	Excellent	Nothing noteworthy
2.c	Good	Plus icon is intuitive for adding a new project. Would be cool to have integration with sales software where information like rates are provided. Don't use gear for adding (project lead or assignees), but also maybe allow to change the title of the project lead?
2.c.i	Good	Hard to hit the exact level of assignment, perhaps a bigger slider or a supporting input field.
2.d	Excellent	Confirmation if you exit editing when there are unsaved changes
3.a	Excellent	Nothing noteworthy
3.b	Excellent	Nothing noteworthy
3.c	Good	Allow for tasks under a feature to be added directly in the feature, instead of going through the tasks page.
4.a	Excellent	Nothing noteworthy
4.a.i	Excellent	Ideal hours should be estimated hours. Negative hours? The icon for completion is nice information, could add another icon for under works.
4.b	Excellent	Nothing noteworthy
4.b.i	Excellent	Nothing noteworthy

4.b.ii	Excellent	Nothing noteworthy
4.c	Excellent	Allow for setting only one date on the start and completion date for tasks. Do not link the status to the date, have a button for "completed".
5.a	Excellent	The number of simulations is not intuitive. If a task has no assignees it should be omitted from the simulation.
5.b	Excellent	It was hard to hit the points to show the info on the graph.
5.c	Excellent	Nothing noteworthy

## Analysis

This was User B's second time participating as a test user for our application. She had no issues logging in or navigating the app. When creating a project User B was puzzled about the gear icon used for editing a list or field (project lead and project assignees), she believed it meant to change something about that field, not to edit the field's content. The level of the assignment slider was also hard to control and she wished for a supporting input field for easier input. When editing a project she wished for a confirmation when exiting if the project had unsaved changes. She also wanted to be able to add tasks under a feature directly, creating them there instead of going back to the tasks page and then creating them. Ideal hours was not a favorable term and User B suggested estimated hours instead. User B also noticed a bug in our app where ideal hours were allowed to be negative. The start date and end date on a task were also confusing, they stand under the "Estimate" section and she thought it was the estimated start and end. Setting the status of a task was also not easy to understand and she wished for a button to do so. She also suggested adding another status for "working", where the actual start date would be set and not finished. She also wished to be able to set only one date, not both which is required at this stage. On the progress page, the number of simulations to run was not clear and should have labels or more text to tell what it means. Hovering over the graph was also not easy to hit the exact point on the graph to see more information. After the test, we discussed the app some more and User B wished that tasks with no assignees would be omitted from the simulation as it meant they had not been approved by the customer, per now we randomize the assignees unless a task has them defined.

Based on the feedback provided we want to change the start date and end date on a task to "actual start date" and "actual end date", and move them out of the estimate section. The date input will be changed from range to two individuals so it does not require both to be set at the same time. We will also make it easier to set the status on a task and add another status for "working". We will also add a confirmation on the edit project page if the user is trying to exit with unsaved changes. The gear icon on the editable list will be changed to a plus icon, and the assignment level slider will get a supporting number input field. We will add that tasks can be created directly under a feature. Lastly, we will change the input field for the number of simulations to run.



We will not change the graph as it is a library and we cannot easily change it. After discussion, we will also not omit task which has no assignees from the simulation and keep that an unassigned task gets a random chosen assignee. The term ideal hours will also be kept.

## L - 2ND ITERATION USER TESTS - USER B

# User test 2nd iteration – User B

Test performed: 04/05/2023

## Introduction

We sent the following email to User B to invite them to the second iteration of user tests (translated):

*Hi \*\*\*!*

*Thank you so much for being willing to be test users. We greatly appreciate your help.*

*Could you give me the times when it would suit each of you?*

*A bit about us:*

*We are a group of three students - Espen Otlo, Sakarias Sæterstøl, and me, Janita Røyseth. We are working on our bachelor's thesis this spring, with AxBit as our client, represented by Franck Chauvel. We are developing a demonstrator for simulation-based cost and time estimation for system development projects.*

*The application we have developed will be ready for a demonstration next week, and with that occasion, we want to perform alpha tests.*

*If you have any questions, please feel free to contact us.*

*Thank you again for taking the time to contribute!*

## Background

Age: 30

Profession: PMO, HR and administration

Previous experience: Project manager for a handful of projects.

## Tasks

1. Sign-in screen
  1. Sign in
2. Project
  1. View projects
  2. Search project
  3. Create project

1. Assignees and level of assignment
4. Edit project
3. Feature
  1. View feature list
  2. Create feature
  3. Edit feature
4. Task
  1. View task list
    1. Information provided in the table (like depth?).
  2. Create task
    1. Dependencies
    2. Child tasks
  3. Edit task
  4. Setting task as finished
5. Progress
  1. Running a simulation
  2. Reading the chart
  3. Reading the table

## Feedback

Task	How did the user perform?	Feedback
1.a	Excellent	Nothing noteworthy
2.a	Okay	Project icons were confusing, they looked like icons for arcade games.
2.b	Okay	The search bar was hard to see, expected on top.
2.c	Excellent	The gear on the project lead and assignees lists was misleading, thought it had another function than adding.
2.c.i	Okay, took two tries before she found it	The slider wasn't pleasant to use, and it was difficult to hit the exact number.
2.d	Excellent	Liked the unsaved changes snack bar.
3.a	Excellent, no issue finding it.	The page is very plain. It doesn't provide any help for what to do.
3.b	Good, didn't find it right away, confused of what it was.	Difficult wording, especially "dependant" (for dependant features).

3.c	Excellent, no issue finding it.	Where to click when adding dependant feature, want to click on the name but the add button is on the far right.
4.a	Excellent	Nothing noteworthy
4.a.i	Good/Okay	Misunderstood the feature column, thought it was hours. Would like an "actual hours" column. Liked that she can hide columns. Didn't understand "depth", explain in simpler terms. Didn't like seeing the assignees when hovered, show the names and if there are many have a hide/show all function.
4.b	Excellent	Nothing noteworthy
4.b.i	Excellent	Still not a fan of the dependent word.
4.b.ii	Excellent	Nothing noteworthy.
4.c	Good	Changes when viewing a task should have a "save" button, as it is used elsewhere.
4.d	Good	Could be easier, thought that the start and end date was for the whole project.
5.a	Good, looked around a bit	The number of simulations is unclear.
5.b	Okay	Didn't understand that the worst case was an estimation, thought it was the actual path of the project. Wanted to scroll horizontally. Liked the color-coded lines.
5.c	Excellent	Nothing noteworthy

Feedback outside of tasks:

- Little visual guidance, don't know where to look.
- Atrocious font, hard to read, to digest what's written.
- Difficult language

## Analysis

User B tested our application for the first time on this iteration and had not seen it before. User B had no issues logging on to our application. Right off the bat User B also did not like our font choice as they found it hard to read and digest the information provided in the application. The project page was also confusing, the project icons looked like something for arcade games. It wasn't easy to understand that it represented a project. For searching projects, User B had to look twice to find the search button but then had no issues navigating to a project. The button for adding a project was easier to find. Problems that occurred when creating a project were that the gear icon was used on top of fields like project lead, or assignees list, but didn't actually have any more functionality than changing the field. Also when changing the level of assignment on

assignees the slider was hard to use. When editing a project, the snack bar which warned the user of unsaved changes was well received. The feature page was very plain and had no visual guide for what to do, or information about what it represents. There were also challenges when creating a project, the language used in the app is kinda hard, especially the word "dependant", also wants to be able to click on the feature name to be able to add a dependent feature, not only the "add" button. User B had no issue finding the tasks page however the columns in the table were somewhat misinterpreted and unfavorable, the feature column wasn't obvious it referred to the features sequence id, the depth should be explained in simpler terms and assignees shouldn't be placed into a tooltip but perhaps instead show the names and then a "hide/show" function for toggling to see all assignees. User B had no problems creating a task. However when quickly (without going into the 'edit mode') editing a task or feature she wished there was a save button as it is used rigorously elsewhere in the application. Setting a task as complete was not intuitive, but it wasn't difficult, however, it could be made easier. Lastly, for the progress page, the number of simulations to run was not clear. The best case and worst case for the graph weren't clear enough that it was estimations, User B thought the worst case was the actual case. In the end, we had a discussion of the app and User B liked the concept and could see how it can be used in discussions with the customer and planning an application.

Based on User B feedback, things we will change are the gear icon on fields, it will be swapped with a plus icon or removed altogether. The assignment level slider also will undergo some improvements, it will receive a supporting number field for easier exact input. We will also try to provide better descriptions for components in the app, as well as rework existing descriptions that were insufficient. We will let clicking on the list item in an editable list/field be sufficient enough for toggling the item. Setting the status for a task will be made simpler. Lastly, the progress page will emphasize that it is showing an estimation, not the actual project

We won't change the project icons this far into the project, it took one moment to learn that the icons represented a project. The font will also not be changed, it is the first and only complaint we have received on it; unless we get more complaints we will not change it.