Linköpings universitet | Institutionen för datavetenskap
Examensarbete på grundnivå, 15hp | Datateknik
2025 | LIU-IDA/LITH-EX-G–25/031–SE

TIDIG - Tidredovisningssystem för Digitaliseringsavdelningen på LiU

TIDIG - Time Reporting System for the Digitalisation Division at LiU

Alice Almgren
Axel Berg
Simon Gunnarsson
Isabel Neubauer
Anton Taber
Jakob Tormalm
Samuel Tuvstedt
Samuel Åkesson
Stina Åström

Handledare: Eric Ekström
Examinator: Lena Buffoni

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
013-28 10 00, www.liu.se

Upphovsrätt

Copyright

Sammanfattning

Den här kandidatrapporten redogör för arbetsprocessen och utvecklingen av ett tidredovisningssystem som beställts av Digitaliseringsavdelningen (DIGIT) vid Linköpings universitet. Rapporten behandlar frågeställningar kring hur man kan skapa värde för kunden, vilka erfarenheter som kan vara värdefulla för framtida projekt, nyttan av att skapa och följa upp en systemanatomi samt konsekvenserna av att dela upp en webbapplikation i frontend- respektive backend-servrar med separata arbetsgrupper. Slutsatser som drogs inkluderar att användningen av ett Scrum-ramverk med tydliga sprintuppgifter bidrog till ökad produktivitet. Samtidigt visade det sig att uppdelningen mellan frontend och backend ställde höga krav på kommunikation mellan grupperna, men möjliggjorde också ett effektivt parallellt arbete. En viktig teknisk lärdom var att tidig planering av datatyper och databasschema underlättade implementationen för både frontend och backend. Den färdiga produktens användbarhet uppmättes till ett genomsnittligt värde på 79 enligt SUS-metoden, vilket indikerar en god användarupplevelse.

Abstract

This bachelor's thesis details the work process and development of a time reporting system commissioned by the Digitalization Division (DIGIT) at Linköping University. The report addresses questions regarding how to provide value for the client, which experiences may be valuable for future projects, the benefits of creating and following a system anatomy, and the consequences of separating a web application into frontend and backend servers with separate development teams. Key findings include that the use of a Scrum framework with clearly defined sprint tasks contributed to increased productivity. Furthermore, the division between frontend and backend imposed high demands on communication between the teams but also enabled efficient parallel work. An important technical lesson was that early planning of data types and database schema facilitated implementation for both frontend and backend. The final product's usability was measured according to the SUS method to an average score of 79, indicating a good user experience.

Tillkännagivanden

Vi riktar ett uppriktigt och stort tack till vår handledare Eric Ekström för det stöd och den vägledning som tillhandahållits under genomförandet av detta kandidatarbete. Vi önskar även tacka vår uppdragsgivare, Digitaliseringsavdelningen vid Linköpings universitet, som har möjliggjort detta projekt. Slutligen vill vi framföra vårt tack till examinator Lena Buffoni och Linköpings universitet.

Innehåll

Tillkännagivanden ..... **iv**
Figurer ..... **viii**
Tabeller ..... **ix**
Ordlista ..... **x**
Projektspecifika termer x
Generella termer x
1 Inledning **1**
1.1 Motivering 1
1.2 Syfte 1
1.3 Frågeställning 1
1.4 Avgränsningar 2
1.5 Kontext 2
1.5.1 Dokument tillhörande kandidatarbetet 2
1.5.2 Roller och ansvar 2
2 Bakgrund **4**
2.1 Kundens bakgrund 4
2.2 Projektgruppens bakgrund 4
3 Teori **6**
3.1 Programspråk, ramverk och bibliotek 6
3.1.1 HTML 6
3.1.2 CSS 6
3.1.3 Tailwind CSS 6
3.1.4 DaisyUI 6
3.1.5 JavaScript 7
3.1.6 TypeScript 7
3.1.7 Svelte 7
3.1.8 C# och .NET 7
3.1.9 ASP.NET 7
3.1.10 Entity Framework Core 7
3.2 Arbetsprocesser 8
3.2.1 Fasplanering 8
3.2.2 Scrum 8
3.2.3 System Usability Scale (SUS) 9
3.2.4 Sustainability Awareness Framework (SusAF) 10
3.3 Verktyg 10
3.3.1 Systemanatomi 10
3.3.2 Kanban-bräde 10
3.3.3 Azure DevOps 10
3.3.4 Figma 11
3.3.5 Visual Studio Code 11
4 Metod **12**
4.1 Projektets faser 12
4.2 Inledande arbetet 12
4.3 Utbildning i form av workshoppar 13

4.4 Arbetsuppdelning i frontend och backend 13
4.5 Utveckling av arbetsprocess 14
4.6 Kvalitetsmätningar 14
4.7 Erfarenhetsinsamling 15
4.7.1 Utvärdering av process 15
4.7.2 Utvärderingsformulär 15
4.7.3 Processdokumentation 15
4.8 Systemanatomi 16
4.8.1 Utvärdering av systemanatomin 17
5 Resultat 18
5.1 Värde för kunden 18
5.2 Kvalitetsmätningar 18
5.3 Processbeskrivning 19
5.3.1 Process i fokus 19
5.3.2 Utvärdering av process 20
5.4 SusAF 22
5.5 Gemensamma erfarenheter 23
5.5.1 Processrelaterade erfarenheter 23
5.5.2 Tekniska erfarenheter 24
5.6 Systemanatomi 25
5.7 Utvärdering av systemanatomins betydelse 26
5.8 Systembeskrivning 27
5.8.1 Frontend 27
5.8.2 Backend 31
5.8.3 Uppdelningen av frontend- och backend-server 31
5.8.4 Uppdelning i separata arbetsgrupper för frontend och backend . . 32
6 Diskussion 33
6.1 Resultat 33
6.1.1 Mätningar på process 33
6.1.2 Kvalitetstester 34
6.1.3 Programmeringsspråk och ramverk 34
6.1.4 Processrelaterade erfarenheter 34
6.1.5 Tekniska erfarenheter 38
6.1.6 Systemanatomins upplevda värde 39
6.1.7 Uppdelningen frontend-backend 39
6.2 Metod 41
6.2.1 Kundkontakt 41
6.2.2 Användarundersökningar 41
6.2.3 Design av applikationen 42
6.2.4 Scrum och samarbete med kunden 42
6.2.5 Erfarenhetsinsamling 42
6.2.6 Skapande av systemanatomi 43
6.2.7 Uppföljning av systemanatomi 43
6.2.8 Källkritik 43
6.3 Arbetet i vidare sammanhang 44

6.3.1 Etiska aspekter 44
6.3.2 Strukturella aspekter 44
6.3.3 Individuella aspekter 44
6.3.4 Sociala aspekter 45
6.3.5 Tekniska aspekter 45
6.3.6 Ekonomiska aspekter 45
6.3.7 Miljömässiga aspekter 45
7 Slutsatser 46
7.1 Hur kan tidredovisningssystemet TIDIG implementeras så att man skapar värde för kunden? 46
7.2 Vilka erfarenheter kan dokumenteras från TIDIG som kan vara intressanta för framtida projekt? 46
7.3 Vilket stöd kan man få genom att skapa och följa upp en systemanatomi? 47
7.4 Vilka för- och nackdelar finns med att dela upp en webbapplikation som TIDIG i en frontend- respektive backend-server med separata arbetsgrupper? 47
8 Referenser 48

Figurer

Figur 1 Sprintuppgifternas placering i Kanban-brädet i slutet av varje sprint. 21
Figur 2 SusAD över produkten 22
Figur 3 Systemanatomi 25
Figur 4 Tidredovisningsvyn i TIDIG 27
Figur 5 Uppgiftsvyn i TIDIG 28
Figur 6 Rapportvyn i TIDIG 29
Figur 7 Statistikvyn i TIDIG 29
Figur 8 Profilvyn i TIDIG 30
Figur 9 Blockschema över systemet TIDIG 31

Tabeller

Tabell 1 Projektspecifika termer som används i rapporten x
Tabell 2 Generella termer som används i rapporten x
Tabell 3 Resultatet av LCP-mätningar i sekunder 18
Tabell 4 Resultat från SUS tester 19
Tabell 5 Medelresultat på reflektionsformulär 20
Tabell 6 Gruppens svar på frågor kring systemanatomins användning. 26

Ordlista

Projektspecifika termer

Tabell 1: Projektspecifika termer som används i rapporten

Administratör	En användare som kan hantera uppgifter och skapa rapporter och sammanställningar
Aktivitet	Ett ID som används för att koppla en uppgift till faktureringställe
DIGIT	Digitaliseringsavdelningen vid LiU
LiU	Linköpings Universitet
Roll	En användares roll i systemet, tidredovisare eller administratör
TIDIG	Tidredovisningssystem för Digitaliseringsavdelningen
Tidredovisare	LiU-medarbetare som redovisar arbetad tid
Tidredovisning	Processen där tidredovisare rapporterar tid arbetad på uppgifter
Uppgift	En arbetsuppgift som tidredovisare kan rapportera tid till

Generella termer

Tabell 2: Generella termer som används i rapporten

Azure	En samling molnbaserade tjänster tillhandahållet av Microsoft
Backend	Den delen av systemet som behövs för att bland annat interagera med databas och autentisera användare
.csv	Ett filformat. Står för *Comma separated values*
Discord	En kommunikationstjänst som erbjuder röstsamtal
EER-diagram	*Extended Entity Relationship*, en datamodell för databasdesign
Feature freeze	Sista fasen i en mjukvaruprodukt, när inga fler funktioner ska byggas, utan endast finjusteringar och buggfix återstår.
Frontend	Den delen av systemet som användare upplever och interagerar med. I projektets fall är det webbsidan.
Kanban	En agil metod för att visualisera arbetsflöden och hantera pågående arbete genom en tavla
LCP	*Largest contentful paint* [1]. En metod för att mäta prestanda. LCP är tiden i sekunder det tar för det största elementet på en webbsida att ladda in
LiU-ID	Ett användarnamn som används för digitala tjänster på Linköpings universitet
Produktbacklogg	Lista över prioriterade funktioner och krav som ska utvecklas i produkten
Produktägare	Ansvarig för produktbackloggen, prioriterar krav och ser till att teamet skapar värde under sprintar

Scrum Master	En coach som säkerställer att scrumprocessen följs och stödjer teamet under sprintar
Sprint	En tidsbegränsad arbetscykel i Scrum som varar en till fyra veckor, där teamet utvecklar och implementerar uppgifter och funktioner från sprintbackloggen
Sprintbacklogg	De uppgifter och funktioner som valts ut från produktbackloggen för den aktuella sprinten
Sprintplanering	Ett möte där teamet väljer vilka uppgifter som ska genomföras under den kommande sprinten
Sprint review	Ett möte i slutet av en sprint där teamet redovisar vilka uppgifter och funktioner som har eller inte har levererats under den senaste sprinten
Sprint retrospektiv	Ett möte i slutet av en sprint där teamet reflekterar över hur det gått, samt identifierar förbättringsområden till kommande sprintar
SQL	*Structured Query Language*. Ett standardspråk för databashantering
SSO	*Single Sign On*, ett verktyg för autentisering av användare
Utvecklingsteam	En grupp som utvecklar och levererar funktionalitet i sprintar

1 Inledning

Den här rapporten redogör det kandidatarbetet som utfördes våren 2025 av gruppen PUM02 i kursen TDDD96 på Linköpings universitet (LiU). Arbetet handlade om att skapa det webbaserade tidredovisningssystemet *TIDIG* (Tidredovisningssystem för Digitaliseringsavdelningen) till *DIGIT* (Digitaliseringsavdelningen på LiU).

1.1 Motivering

DIGIT har under lång tid haft ett undermåligt system för *tidredovisning* för sina anställda. De anställda arbetar ofta på flera projekt parallellt och måste därmed redovisa hur mycket tid de lägger på respektive projekt. Detta är viktigt då de anställdas tidredovisning är underlag för fakturering till diverse kunder. Det nuvarande systemet saknar ett användarvänligt gränssnitt.

1.2 Syfte

Syftet med projektet är att utveckla en användarvänlig och effektiv lösning för tidsrapportering, anpassad efter DIGIT:s behov. Som en del av detta skapas en systemanatomi vars stöd för projektets genomförande utvärderas i rapporten.

Rapporten syftar även till att identifiera erfarenheter gällande att arbeta i en projektgrupp för att utveckla en mjukvaruprodukt. Rapporten behandlar också effekten av att arbeta med Scrum och uppdelningen i två tydliga arbetsgrupper för *frontend* respektive *backend* vid utvecklingen av en webbapplikation.

1.3 Frågeställning

Rapporten besvarar följande frågeställningar:

1. Hur kan tidredovisningssystemet TIDIG implementeras så att man skapar värde för kunden?
2. Vilka erfarenheter kan dokumenteras från TIDIG som kan vara intressanta för framtida projekt?
3. Vilket stöd kan man få genom att skapa och följa upp en systemanatomi?
4. Vilka för- och nackdelar finns med att dela upp en webbapplikation som TIDIG i en frontend- respektive backend-server med separata arbetsgrupper?

1.4 Avgränsningar

Arbetet och projektet har haft följande avgränsningar baserat på kurskrav, kundens krav och projektgruppens val.

Från kurs:

- Projektets omfattning begränsades till totalt 3600 timmar, vilket innebär 400 timmar per gruppmedlem för 9 medlemmar.

Från kund:

- Produkten stödjer inloggning med *LiU-ID* via LiU-*SSO* men andra autentiseringslösningar inkluderas inte.
- Produkten följer LiU:s grafiska profil samt uppfyller tillgänglighetskraven EN 301 549 [2] och WCAG 2.1 [3].
- Produktens backend är baserad på C# och ASP.NET, och systemet driftsätts på DIGIT:s egna servrar.

Från projektgrupp:

- Produktens gränssnitt är på svenska och koden skrivs på engelska. Inga andra språk används.

1.5 Kontext

Kursen *Kandidatprojekt i programvaruutveckling* med kurskod *TDDD96* ges av Linköpings Universitet till studenter som går civilingenjörslinjer inom datateknik och mjukvaruteknik. Kursens upplägg består i huvudsak av att studenter, uppdelade i grupper om åtta till nio personer, åtar sig ett projekt från en extern uppdragsgivare och skriver ett gemensamt kandidatarbete utifrån det projektet. I kursen ingår också en individuell erfarenhetssammanfattning där vardera student beskriver sina erfarenheter under projektet. Vidare ingår också föreläsningar, seminarium och en konferens då projektet presenteras för resterande kursdeltagare. Kandidatgrupperna hade stöd och vägledning av en handledare som de träffade varje vecka.

1.5.1 Dokument tillhörande kandidatarbetet

Utöver denna kandidatrapport har andra dokument upprättats. Dessa dokument är följande:

- Arkitekturdokument
- Gruppkontrakt
- Kravspecifikation
- Kvalitetsplan
- Projektplan
- Testplan

1.5.2 Roller och ansvar

Kandidatgruppen som är ansvarig för kandidatarbetet består av nio tredjeårsstudenter varav fyra medlemmar studerar civilingenjörsutbildningen i datateknik och fem medlemmar studerar civilingenjörsutbildningen i mjukvaruteknik. Varje enskild projektmedlem har en specifik grupproll med tillhörande ansvar. Nedan listas dessa roller och en kort beskrivning följer sedan gällande deras relaterade ansvar.

Teamledare: Teamledaren ansvarar för att leda och fördela arbetet inom gruppen. Hen ser också till att målen uppfylls och att processer följs. Vidare agerar

teamledaren coach och ser till att det råder god arbetsmiljö. Teamledaren representerar kandidatgruppen utåt och ansvarar för att en projektplan skrivs. Teamledaren har rätt att ta det slutgiltiga beslutet om det behövs för att arbetet ska fortskrida.

Analysansvarig: Analysansvarig ansvarar för att hålla kontakten med kunden som beställt produkten. Hen tar reda på kundens verkliga behov och agerar förhandlingspart och ger insyn mot övriga gruppen. Analysansvarig ansvarar för kravspecifikationen, att kraven dokumenteras och ser till att kraven uppfylls.

Arkitekt: Arkitekten ansvarar för arkitekturdokumentet och att arkitekturen över produkten dokumenteras. Hen ser till att en stabil arkitektur tas fram och identifierar komponenter och gränssnitt. Arkitekten har rätt att ta övergripande teknikval och har, om det behövs, det sista ordet i tekniska frågor efter teamledaren. Sist behöver arkitekten kunna kommunicera bärande idéer.

Dokumentansvarig: Dokumentansvarig ansvarar för att ta fram dokumentmallar som ska användas under projektet och underhållning av dessa. Hen ser till att en logotyp tas fram, skapar ändringsrutiner och ansvarar för att leveranser sker till gruppens deadlines.

Kvalitetssamordnare: Kvalitetsarbete utförs av alla roller genomgående under alla delar i projektet. Kvalitetssamordnare ansvarar i synnerhet för initiativtagande och har uppföljningsansvar gällande kvalitetsarbetet. Hen ser också till att erfarenheter dokumenteras och att en kvalitetsplan skrivs och efterföljs. Vidare ansvarar hen för gruppens utbildning och ger inspiration till fortsatt kvalitetsarbete. Kvalitetssamordnaren planerar och budgeterar med övriga i gruppen och ser över hur mycket kvalitet får kosta.

Konfigurationsansvarig: Konfigurationsansvarig bestämmer vad som skall versionshanteras och vilka ändringar som ingår i en utgåva (eng. *release*). Hen ansvarar för att välja och underhålla verktyg för versions- och konfigurationshantering. Sist ser konfigurationsansvarig till att verktygen används på rätt sätt under projektet.

Testledare: Testledaren beslutar om systemets status och sköter den dynamiska verifieringen och valideringen av systemet genom exekvering. Vidare ansvarar hen för att en testplan skrivs och att tester rapporteras. Hen ser också till att hålla rimlig distans till det som testas. Sist testar testledaren kvalitetskraven tillsammans med kvalitetssamordnaren.

Backendutvecklingsledare: Backendutvecklingsledaren ansvarar för den detaljerade designen av produkten. Hen ansvarar för att leda och fördela utvecklingsarbetet för backenddelen av projektet. Backendutvecklingsledaren har rätt att fatta beslut om utvecklingsmiljö och ser till att organisera projektmedlemmarnas tester.

Frontendutvecklingsledare: Frontendutvecklingsledare ansvarar för den detaljerade designen av produkten. Hen ansvarar för att leda och fördela utvecklingsarbetet för frontenddelen av projektet. Frontendutvecklingsledaren har rätt att fatta beslut om utvecklingsmiljö och ser till att organisera projektmedlemmarnas tester.

2 Bakgrund

Kapitlet redogör för kundens bakgrund till projektet samt projektgruppens tidigare erfarenheter av liknande projektarbeten och relevanta kompetenser.

2.1 Kundens bakgrund

DIGIT på LiU startade år 2023 och arbetar med att tillhandahålla, utveckla och förvalta en IT-miljö för medarbetare och studenter på universitetet. De strävar att leverera användarcentrerade lösningar med universitetets behov i fokus som är säkra, stabila och kostnadseffektiva [4].

DIGIT är uppdelad i åtta enheter med olika ansvarsområden som samverkar och träffas regelbundet. Enheterna heter Applikationsenheten, Digitala resursenheten, Projekt- och utvecklingsenheten, IT-arbetsplatsenheten, IT-infrastrukturenheten, IT-serviceenheten och Nära IT-stödenheten [4]. Det nuvarande tidredovisningssystemet skapar främst värde för applikationsenheten och projektutvecklingsenheten.

2.2 Projektgruppens bakgrund

Projektgruppen har teoretisk kunskap inom programutvecklingsmetodik från kursen *TDDC93 - Programutvecklingsmetodik*. Även om de flesta saknar praktisk erfarenhet att tillämpa metoderna i projektarbeten har medlemmarna en grundläggande förståelse vad gäller agila arbetsmetoder, kravhantering och systemdesign. Vidare har samtliga projektmedlemmarna erfarenheter av teamarbete från projektarbeten i sina studier. För studenterna inom datateknik har kurserna *TSEA29 - Konstruktion med mikrodatorer* och *TSEA83 - Datorkonstruktion* givit viktiga erfarenheter av att arbeta utifrån en kravspecifikation och en projektplan.

Utöver detta har projektmedlemmarna deltagit i ett flertal datorlaborationer inom olika kurser med samarbete i par. Vissa medlemmar i projektgruppen har även egna erfarenheter av mjukvaruutveckling, projektarbete och teamarbete genom ideellt arbete och fritidsaktiviteter.

Med dessa erfarenheter är projektgruppen överens om att planering är viktigt för att säkerställa jämn arbetsbelastning. Preliminära veckoplaneringar och regelbundna möten är fördelaktiga eftersom det ger en tydlig överblick av projektets framgång. Kommunikation ses som en av de viktigaste faktorerna för ett framgångsrikt projekt och det är viktigt att alla projektmedlemmar tar eget ansvar för att bidra till ett effektivt samarbete.

Det finns en bred kunskapsresurs i projektgruppen inom webbprogrammering. Vissa i projektgruppen har erfarenheter av frontend-utveckling från personliga projekt i samma skala som kandidatprojektet. Däremot finns potential för utveckling inom det programmeringsspråk som ska användas för frontend, TypeScript och Svelte-ramverket. Det ska också nämnas att studenterna inom mjukvaruteknik har erfarenhet inom systemutveckling som involverar både frontend och backend från kursen *TDDD80 - Projekt: Mobila och sociala applikationer*.

Gällande backend-utveckling har en del i gruppen erfarenhet av databaser, *SQL* och molnplattformen *Azure* från tidigare kurser. En begränsning finns då programmeringsspråket som ska användas för backend är C#, vilket är ett språk som få har erfarenhet av. Däremot har samtliga i gruppen läst kursen *TDDE30/TDDD78 - Objektorienterad programmering och Java* som troligtvis underlättar inlärning och objektorienterad programmering i C#. Vidare finns det bristande kunskap om hur ramverket ASP.NET fungerar.

3 Teori

Detta kapitel beskriver den relevanta teorin och lägger grund för att besvara frågeställningarna i Avsnitt 1.3. Detta innefattar programspråk, ramverk, bibliotek, arbetsmetoder och verktyg.

3.1 Programspråk, ramverk och bibliotek

För att utveckla produkten har gruppen tagit hjälp av ett flertal programspråk, ramverk och bibliotek vilka är beskrivna nedan.

3.1.1 HTML

HTML är en förkortning av *Hyper Text Markup Language* och är den teknologi som definierar strukturen och innehållet för webbsidor [5]. Genom olika element som till exempel text, bilder och sektioner, utgör HTML grunden för att bygga webbsidor.

3.1.2 CSS

CSS står för *Cascading Style Sheets* och är ett språk som används för att ge webbsidor sin layout och utseende [6]. Den styr hur HTML-elementen ska visas på skärmen genom att exempelvis definiera positionering, färger, typsnitt och marginaler. I CSS definieras grupper av stilregler i klasser, vilka kan appliceras på HTML-element för att ändra deras utseende.

3.1.3 Tailwind CSS

Tailwind CSS är ett CSS-ramverk som tillåter utvecklare att skriva CSS direkt i HTML [7]. Ramverket är ”utility-first”, vilket innebär att en uppsättning fördefinierade klasser används istället för att skapa egna CSS-regler. Detta möjliggör snabb och effektiv utveckling av webbapplikationer. Tailwind CSS innehåller klasser för att hantera layout, marginaler och textanpassning.

3.1.4 DaisyUI

DaisyUI är ett insticksprogram till Tailwind CSS som förenklar utvecklingen av användargränssnitt genom att tillhandahålla färdiga komponenter [8]. Den innehåller klassnamn som hjälper utvecklaren att designa och anpassa grundläggande komponenter såsom knappar, textrutor, sökrutor med mera. Detta resulterar i att utvecklaren behöver skriva mindre kod och därmed sparar tid.

3.1.5 JavaScript

JavaScript är ett dynamiskt och objektorienterat programmeringsspråk som används för att skapa interaktivitet och manipulera objekt inom en värdmiljö, vanligtvis webbläsare [9]. Språket erbjuder en flexibel syntax utan typdeklarationer och bygger på objekt med egenskaper och metoder. JavaScript är en grundpelare för modern webbutveckling.

3.1.6 TypeScript

TypeScript är ett programmeringsspråk som bygger på JavaScript och tillför statisk typning [10]. Det kompileras till JavaScript och kan därför köras i alla miljöer där JavaScript fungerar. TypeScript gör det möjligt att upptäcka typrelaterade fel redan vid kompilering, vilket ökar kodens säkerhet och underlättar underhåll. Språket behåller JavaScripts objektorienterade och dynamiska karaktär men adderar verktyg för att hantera större och mer komplexa kodbaser.

3.1.7 Svelte

Svelte är ett modernt ramverk för utveckling av webbapplikationer, som kan användas med JavaScript eller TypeScript [11]. Svelte använder inte en virtuell DOM till skillnad från andra populära ramverk såsom React och Angular. Detta innebär att koden kompileras till optimerad JavaScript som endast uppdaterar specifika delar av DOM:en vid behov, vilket resulterar i bättre prestanda.

En fördel med Svelte är dess enkelhet, då inga ytterligare bibliotek eller beroenden krävs [11]. Dessutom har ramverket en lättförståelig och intuitiv syntax vilket gör det lämpligt för nybörjare inom webbprogrammering.

3.1.8 C# och .NET

C# är ett objektorienterat programmeringsspråk utvecklat av Microsoft [12]. Det används inom bland annat utveckling av hemsidor och datorspel och har fördelarna att det fungerar på många plattformar och har en stor och aktiv användarbas.

.NET är utvecklingsplattformen som C# körs på. Den innehåller runtime-miljön, standardbibliotek, kompilatorer och verktyg som behövs för att bygga, köra och distribuera applikationer.

3.1.9 ASP.NET

ASP.NET är ett ramverk för att utveckla webbservrar i .NET [13]. Alla ändpunkter definieras som metoder i klasser, där dessa klasser kallas för ”controllers”. Varje metod motsvarar ett HTTP-anrop, till exempel GET eller POST, och returnerar vanligtvis ett svar i form av HTML, JSON eller en statuskod.

3.1.10 Entity Framework Core

Entity Framework Core är ett verktyg för att förenkla och abstrahera kommunikationen mellan ett .NET-program och en databas [14]. Verktyget kopplar en klass i C#-koden i .NET-programmet till en tabell i databasen och innehåller även funktionaliteten att uppdatera databasens tabeller efter att en ändring har gjorts i koden.

3.2 Arbetsprocesser

De arbetsprocesser som projektgruppen har arbetat med under projektet och kontinuerligt utvärderat förklaras nedan i detalj.

3.2.1 Fasplanering

Ett projektarbete kan delas in i fyra faser [15], [16]. Den första fasen är *analysfasen*. Syftet med analysfasen är att projektgruppen ska förbereda sig genom att analysera hur projektet ska se ut, bilda sig en uppfattning om hur slutprodukten ska bli och säkerställa att det går att genomföra projektet så att kundens krav och behov blir uppfyllda. Den andra fasen är *planeringsfasen* som är till för att realisera projektbilden och upprätta projektplaner. Efter planeringsfasen kommer *genomförandefasen* vars syfte är att realisera planer och design som tidigare skapats. Den består i sin tur av en etablering-, realisering- och överlämningsfas. Under *etableringsfasen* påbörjas genomförandet av projektet och verifiering av att planeringen följs. Därefter kommer *realiseringsfasen* som är den fas där projektets slutresultat tas fram. I den här fasen implementeras designen som tagits fram i etableringsfasen. Vid start av *överlämningsfasen* är produkten färdigtestad och all dokumentation är upprättad. Vidare blir kund informerad om hur de använder och underhåller produkten. Projektets slutresultat överlämnas och det säkerställs att allt blir formellt accepterat av kunden. Den sista fasen är *avslutningsfasen* och under denna fas dokumenteras erfarenheter från projektets gång.

3.2.2 Scrum

Scrum är en agil arbetsmetod där små tvärfunktionella team samarbetar för att utveckla en produkt i korta cykler, kallade sprintar [17]. I slutet av varje *sprint* presenteras en fungerande version av produkten.

3.2.2.1 Roller

I Scrum tilldelas de olika projektmedlemmarna olika roller med olika ansvarsområden såsom *produktägare*, *Scrum master* och *utvecklingsteam* [18]. Scrum master fungerar som en coach och säkerställer att Scrum följs, och ser till att teamet arbetar effektivt. *Produktägare* representerar kunden och ansvarar för produktbackloggen. Utvecklingsteamet består av ett antal utvecklare som producerar mjukvaran.

3.2.2.2 Möten

En sprint består av flera aktiviteter såsom sprintplaneringsmöte, dagligt Scrummöte, sprint review och sprint retrospektiv [18]. Dessa aktiviteter syftar till att strukturera och utvärdera och kontinuerligt förbättra arbetet.

Sprinten startar med en *sprintplanering* där produktägaren tillsammans med teamet bestämmer vad som ska implementeras under denna sprint [18]. Varje arbetsdag hålls även ett dagligt Scrum-möte på ungefär 15 minuter då samtliga teammedlemmar uppdaterar varandra om sitt arbete och kan be om hjälp vid behov. Under mötet svarade varje projektmedlem på följande frågor:

- Vad har du gjort sedan senaste mötet?
- Vad ska du göra fram till nästa möte?
- Har du fastnat på något och behöver hjälp?

Sprint review är ett möte som hålls i slutet av varje sprint [18]. Produkten inspekteras och produktbackloggen ändras vid behov. Utvecklingsteamet demonstrerar arbetet, besvarar frågor, diskuterar vad som gick bra, problemen som de stötte på och hur dessa löstes. Teamet diskuterar även vad som är nästa steg i implementeringen.

Sprint retrospektiv är ett möte som hålls sista dagen av sprinten, efter sprint review [18]. Teammedlemmarna diskuterar resultatet, lärdomar och tar med sig detta inför planeringen av nästa sprint.

3.2.2.3 Backlogg

Varje sprint har en tillhörande sprint backlogg som är en ordnad lista med de *uppgift*er som ska genomföras under den givna tidsperioden [18]. Denna backlogg baseras på produktbackloggen som innehåller de krav och användarhistorier som är kvar att implementera och hanteras utav produktägaren.

3.2.3 System Usability Scale (SUS)

SUS (*System Usability Scale*) är en skala som mäter användbarhet på system [19]. Den genomförs i samband med användartestning efter att en användare har använt exempelvis en webbsida. Den baseras på 10 påståenden som användare ska rangordna på en skala 1-5. En femma innebär att de håller med helt och hållet, medan en etta innebär att de verkligen inte håller med. Följande påståenden ingår i SUS-formuläret:

1. Jag tror att jag skulle vilja använda denna produkt ofta.
2. Jag tyckte att denna produkt var onödigt komplicerad.
3. Jag tyckte att denna produkt var lätt att använda.
4. Jag tror att jag kommer behöva hjälp av en teknisk person för att kunna använda denna produkt.
5. Jag tycker att de olika funktionerna i denna produkt är väl samordnade.
6. Jag tyckte att det var för mycket inkonsekvens i produkten.
7. Jag kan tänka mig att de flesta skulle lära sig att använda denna produkt mycket snabbt.
8. Jag tyckte att denna produkt var mycket besvärlig att använda.
9. Jag kände mig väldigt trygg när jag använde denna produkt.
10. Jag behövde lära mig mycket innan jag kunde komma igång med denna produkt.

$$S = 2.5 \cdot \sum_{i=1}^{10} s_i \tag{1}$$

$$s_i = \begin{cases} x_i - 1 \text{ , om } i \text{ är udda } (1, 3, 5, 7, 9) \\ 5 - x_i \text{ , om } i \text{ är jämn } (2, 4, 6, 8, 10) \end{cases} \tag{2}$$

Baserat på användarnas svar beräknas en slutpoäng enligt Ekvation 1 och Ekvation 2 [19]. För udda numrerade frågor subtraheras 1 från poängen, medan jämnt numrerade frågor beräknas genom att subtrahera poängen från 5, se Ekvation 2. Detta beror på att udda frågor är positiva påståenden till skillnad från jämna som är negativa påståenden. Justeringen resulterar i att alla frågor får en poäng mellan

1 och 4. Slutligen multipliceras summan av de justerade värdena med 2.5 för att få en slutpoäng som varierar mellan 0 och 100.

Resultatet ger en bra indikation på systemets användbarhet, där ett högre poäng indikerar bättre användbarhet [19]. En poäng på 51 eller lägre tyder på dålig användbarhet och bör åtgärdas omedelbart. En poäng runt 68 anses vara acceptabel, men det finns utrymme för förbättring. En poäng på 80.3 eller högre indikerar en mycket bra användbarhet.

3.2.4 Sustainability Awareness Framework (SusAF)

Sustainability Awareness Framework (SusAF) [20] är en metod för att utvärdera en produkts sociala, individuella, miljömässiga, tekniska samt ekonomiska hållbarhet. Metoden genomförs genom att i grupp diskutera dimensionerna ur olika aspekter enligt en förutbestämd mall.

Resultatet blir ett Sustainability Awareness Diagram (SusAD) som visualiserar den direkta och indirekta effekten som produkten har på samhället. Varje effekt kategoriseras efter dimension och hur direkt- eller indirekt effekten är.

3.3 Verktyg

Under arbetet med projektet har de nedan beskrivna verktygen använts av gruppen.

3.3.1 Systemanatomi

En *systemanatomi* är ett hjälpmedel för att visualisera och konkretisera funktionaliteten av ett system. Järkvik [21, s. 14-16] väljer att definiera begreppet systemanatomi som ett verktyg för att kommunicera om och planera system. Varje byggblock i en systemanatomi kallas för en *anatom*, och motsvarar en användarfunktion [21].

3.3.2 *Kanban*-bräde

Kanban-bräde är ett verktyg som visualiserar arbetsflödet och ger en tydlig överblick av arbetsfördelningen och framstegen [22]. Den består av kort som motsvarar uppgifter som kan röra sig mellan olika kolumner i brädet som motsvarar olika faser:

- **Att göra:** Uppgifter som ingår i sprintbackloggen men som ännu inte påbörjats.
- **Pågående:** Uppgifter som är under arbete men ännu inte färdigställda.
- **Granskning:** Uppgifter som är färdiga men ska granskas av en annan projektmedlem.
- **Godkänd:** Uppgifter som är slutförda och godkända.

Kanban-bräden kan visualiseras i Azure DevOps som beskrivs i Avsnitt 3.3.3.

3.3.3 Azure DevOps

Azure DevOps är en plattform utvecklad av Microsoft med syftet att förenkla processen att utveckla programvara [23]. Den innehåller agila verktyg som stödjer arbete enligt Kanban och Scrum, inklusive olika typer av bräden. Det kan även integreras med GitHub eller tillhandahålla Git-lagringsplatser. Dessutom innehåller Azure DevOps verktyg för testning, kontinuerlig integrering och leverans. Under

arbetet användes även dess funktionalitet med Kanban-bräden och sprinter för att organisera arbetet.

3.3.4 Figma

Figma är ett molnbaserat designverktyg för att skapa, dela och testa designer för webbsidor, mobilapplikationer och andra digitala produkter [24]. Det möjliggör samarbete i realtid, där flera personer kan redigera samtidigt. Verktyget innehåller stöd för att skapa prototyper utan att skriva kod samt möjligheten att skapa återanvändbara UI-komponenter och andra användbara funktioner som effektiviserar designprocessen.

3.3.5 Visual Studio Code

Visual Studio Code är en modern och lätthanterlig programutvecklingsmiljö som utvecklats av Microsoft [25]. Den har stöd för de flesta programmeringsspråk och kan köras på flera olika operativsystem. Den erbjuder flera smarta inbyggda funktioner såsom inbyggd terminal, felsökningsverktyg och ett stort antal tillägg. Dessutom finns det inbyggt stöd för Git, vilket gör det enklare att samarbeta med andra utvecklare.

4 Metod

Detta kapitel beskriver de metoder som användes för att besvara frågeställningarna som beskrivs i Avsnitt 1.3.

4.1 Projektets faser

I början av projektet skapades en projektplanering som omfattade en analysfas, planeringsfas, genomförandefas och avslutningsfas. Fasplanen var som följande:

- I **Analysfasen** ingick att skriva *kravspecifikation*, utvärdera gruppens kapacitet för projektet och genomföra en SusAF-analys. Syftet med SusAF-analysen var att säkerställa en bra och hållbar slutprodukt och resultatet redovisas i Avsnitt 5.4. För vidare information om analysfasen hänvisas läsaren till Avsnitt 4.2.
- **Planeringsfasen** utgick att planera projektet och skapa en design för TIDIG. Relevanta planeringsdokumenten för vårt projekt var *projektplanen* och *kvalitetsplanen*. Vidare skapades en *testplan* och designdokumenten *systemanatomin* och *arkitekturdokumentet* skapades för att strukturera upp hur projektet skulle utvecklas.
- **Genomförandefasen** realiserade planer och design som tidigare skapades.
  - Under **etableringsfasen** utbildades medlemmar i projektets tekniker och en bedömning gjordes utifrån planeringens realiserbarhet.
  - Under **realiseringsfasen** genomfördes det som beskrivits i planeringen och den design som tagit fram i etableringsfasen implementerades.
  - Vid starten av **överlämningsfasen** var produkten färdigtestad och all dokumentation var upprättad. Vidare blev kunden informerad om hur de använder och underhåller produkten. Projektets slutresultat överlämnades och det säkerställdes att allt blev formellt accepterat av kunden.
- Den sista fasen var **avslutningsfasen**. Under denna del av projektet dokumenterades de olika erfarenheter från projektets gång. Varje medlem skrev, enligt bedömningskriterierna för kandidatkursen, en individuell erfarenhetssammanfattning och tillsammans skrev gruppen ett gemensamt kandidatarbete.

4.2 Inledande arbetet

Innan arbetet med produkten påbörjades behövde projektgruppen ta gemensamma beslut gällande arbetssätt och roller. Ett gruppkontrakt togs fram för att fastställa hur samarbetet i gruppen skulle fungera. För att ge en detaljerad bild av hur arbetet skulle genomföras togs ett antal dokument fram, beskrivna i Avsnitt 1.5.1. Dessa

dokument uppdaterades och reviderades kontinuerligt under projektets gång för att säkerställa att arbetsmetoderna följdes och att dokumentationen förblev aktuell.

Därefter beslutades vilka ramverk och programmeringsspråk som skulle användas i projektet. Projektmedlemmar med tidigare erfarenhet presenterade och motiverade sina val och därefter togs ett gemensamt beslut genom omröstning.

Det som avslutade det inledande arbetet av projektet var ett kundmöte där kunden gick igenom sitt nuvarande system för projektgruppen och förklarade sitt behov och sina förväntningar på det nya systemet. Detta lade grunden för de dokument som framställts under projektet, särskilt för kravspecifikationen.

4.3 Utbildning i form av workshoppar

Innan utvecklingen av produkten påbörjades, anordnade gruppen två interna workshoppar. Den ena workshoppen fokuserade på backend-utveckling och konfiguration medan den andra fokuserade på frontend-utveckling. Alla uppmuntrades att delta på båda workshopparna för att bekanta sig med de språk och verktyg som skulle användas under projektets gång. Projektmedlemmar med kunskap inom vardera tekniskt område ansvarade för att förbereda material och uppgifter till övriga medlemmar. Innehållet i dessa workshoppar var en blandning av teoretisk genomgång och praktiska moment, där projektmedlemmarna kunde diskutera, samarbeta och dela kunskap med varandra.

Under backend- och konfigurationsworkshoppen introducerades och installerades konfigurationsverktyg. Uppgiften på backend-workshoppen var att skapa två ändpunkter, en för att ta emot en lista med redovisade tider och en för att lägga till en redovisad tid. För utförandet fanns en enkel förberedd webbsida som visade de redovisade tiderna och gjorde det möjligt att redovisa tid. Ytterligare en förberedelse var given kod med en lista som sparar redovisade tider som en startpunkt.

I workshoppen som fokuserade på frontend följde projektmedlemmarna en handledning i ramverket Svelte där man fick skriva grundläggande kod i TypeScript.

4.4 Arbetsuppdelning i frontend och backend

Projektgruppen delades in i två huvudsakliga arbetsgrupper med fokus på frontend- respektive backend-utveckling med egna utvecklingsledare. Utvecklingsledarna hade ett större ansvar för att leda och samordna arbetet inom sin grupp. Uppdelningen gjordes för att minimera förvirring och överlappande arbete. Genom att dela upp gruppen kunde medlemmarnas varierande kompetens tas tillvara på. Vissa hade sedan tidigare mer erfarenhet av backend och andra av frontend, vilket gjorde det naturligt att arbeta i separata spår.

Valet att arbeta uppdelat grundade sig också i projektmedlemmarnas individuella intressen. Flera av projektmedlemmarna tyckte olika delar av utvecklingsarbetet verkade mer givande. Dessutom speglade valet kring uppdelningen gruppens uppfattning om hur ett projektteam skulle vara strukturerat, med arbetsuppgifter fördelade efter kompetens och intresse.

Grupperna arbetade till stor del självständigt, med egna möten och separata sprintuppgifter. Utöver detta samlades hela projektgruppen två gånger per sprint för gemensamma sprintmöten och avstämning.

Samarbete mellan grupperna skedde löpande under projektets gång, främst när nya gränssnitt implementerades och krävde ny logik från backend-servern. Syftet var att skapa tydliga ansvarsområden inom respektive arbetsgrupp, samtidigt som varje medlem skulle behålla en god helhetsförståelse genom regelbunden kommunikation.

4.5 Utveckling av arbetsprocess

Projektgruppen valde att inledningsvis arbeta med en klassisk version av Scrum, såsom beskrivet i Avsnitt 3.2.2, i kombination med ett Kanban-bräde, som beskrivet i Avsnitt 3.3.2. Till skillnad från en klassisk version av Scrum fokuserade projektgruppen inte på att leverera en fungerande produkt i slutet av varje sprint, och hade därmed inte regelbundna demonstrationer för kunden. Arbetsprocessen anpassades genom kontinuerliga utvärderingar under projektets gång. Syftet med att etablera en tydlig metodik var att skapa ett strukturerat och effektivt arbetssätt, vilket i sin tur förväntades bidra till ökat kundvärde.

I slutet av varje sprint hölls ett avslutningsmöte som motsvarade både sprint retrospektiv och sprint review. Under mötet diskuterade gruppen sprintens resultat, nästa steg i implementeringen, förbättringsförslag till nästa sprint och problem som uppstått. Projektmedlemmarna fick även anonymt besvara ett formulär, enligt beskrivningen i Avsnitt 4.7.2, för att bedöma sprintens genomförande. Gruppen utgick sedan efter förbättringsförslagen och bestämde vad i arbetsprocessen som skulle ändras till nästa sprint.

Denna iterativa förbättringsprocess ledde till att en arbetsmetod utvecklades som var anpassad till projektgruppens behov och arbetssätt. Förbättringarna och resultatet av denna arbetsmetod beskrivs i Avsnitt 5.3.1.

4.6 Kvalitetsmätningar

För att uppfylla de kvalitetskrav som fastställts i projektets kravspecifikation genomfördes kontinuerliga mätningar av prestanda, användbarhet och tillförlitlighet. Syftet med mätningarna var att säkerställa att produkten skapar värde för kunden.

För att mäta prestanda och säkerställa att *LCP* [1] inte överskred 2,5 sekunder i minst 95% av testfallen användes verktyget *lighthouse-ci* [26]. En pipeline sattes upp i Azure DevOps som automatiskt genererade en sammanställning över alla Web Vitals-värden [27], inklusive LCP. Under mätningarna användes en konsekvent bandbredd genom att emulera 4G-nätverk, för att minska varians.

Användbarhetstester, enligt metoden som beskrivits i Avsnitt 3.2.3, genomfördes vid två tillfällen under produktens utvecklingsfas. Det första testtillfället ägde rum under den andra sprinten och då deltog samtliga 9 projektmedlemmar. De utförde totalt 8 definierade uppgifter i systemet, med syftet att identifiera tidiga brister i användarupplevelsen. Det andra testtillfället skedde under den femte sprinten och då deltog 13 externa personer. Varje medlem i projektgruppen hade ansvaret för att hålla i användartester för en till två personer. Vid detta tillfälle fick deltagarna utföra fyra fördefinierade uppgifter, utformade för att täcka största delen av produktens funktionalitet. Syftet var att säkerställa att produkten uppfyllde kvalitetskravet om ett genomsnittligt betyg på minst 68 poäng i en SUS-mätning.

Efteråt fyllde de i ett SUS-formulär med tio påståenden om deras användarupplevelse och betygsatte hur väl varje påstående stämde på en skala från ett till fem.

Utöver SUS-formuläret fick testdeltagarna även möjligheten att ge generell feedback och förbättringsförslag på produkten. Den insamlade feedbacken diskuterades sedan på ett av projektgruppens möten och relevanta åtgärder vidtogs med syftet att förbättra systemets användbarhet.

För att säkerställa systemets tillförlitlighet skapades automatiserade tester med målet att täcka 95% procent av backend-koden genom *statement coverage*. *Statement coverage* innebär att varje rad i koden testas minst en gång. Det var även önskvärt att öka testningens *branch coverage*, som innebär att testerna täcker alla möjliga logiska grenarna i koden.

4.7 Erfarenhetsinsamling

I genomförandet av kandidatprojektet samlades erfarenheter in av gruppen och medlemmarna. Erfarenheterna handlade om projektarbetet, den centrala arbetsprocessen *Scrum* och tekniska lärdomar.

4.7.1 Utvärdering av process

Under tidens gång anpassades den huvudsakliga arbetsmetoden *Scrum*, och även mindre processer, utefter utvärderingar som genomfördes under sprint retrospektiv. Under mötet diskuterades och dokumenterades både positiva och negativa erfarenheter från sprinten. Följande frågor användes som utgångspunkt för diskussionen:

- Vad gick bra respektive dåligt under sprinten?
- Vilka problem uppstod och hur löstes dessa?
- Vad kan förbättras eller göras annorlunda inför nästa sprint?

Insikterna från utvärderingen dokumenterades och sammanställdes av kvalitetssamordnaren och användes sedan som underlag vid planeringen av nästa sprint.

4.7.2 Utvärderingsformulär

I slutet av varje sprint fick samtliga projektmedlemmar anonymt besvara ett formulär för att bedöma sprintens genomförande. Formuläret bestod av skalfrågor där en etta representerar missnöje och en femma nöjdhet.

Insamlad data analyserades för att identifiera förbättringsområden. Formuläret bestod av följande frågor:

1. Var sprintmålen tydliga och realistiska?
2. Hade teamet en god kommunikation och samarbete under sprinten?
3. Kunde de planerade uppgifterna levereras i tid och med god kvalitet?
4. Sprint retrospektiv hjälpte oss att identifiera förbättringsområden och utvecklas som team.
5. Arbetsbelastningen under sprinten var rimlig och hållbar.
6. Jag är överlag nöjd med hur denna sprint genomfördes.

4.7.3 Processdokumentation

För att dokumentera erfarenheter under projektets gång, fördes ett protokoll under varje möte, med undantag för dagligt Scrum-möte. Protokollet var särskilt viktigt under möten som rörde uppstart av sprint och utvärdering av sprint. Det var bety-

delsefullt för att kunna reflektera över det utförda arbetet och följa de förändringar som genomfördes i Scrum-processen. Kvalitetssamordnaren antecknade förändringarna till efterkommande sprint och sammanställde antal avklarade sprintuppgifter.

4.8 Systemanatomi

Under projektets planeringsfas skapades en systemanatomi med syfte att ge en översikt över systemet TIDIG:s funktioner och egenskaper. Projektgruppen delade upp sig i tre arbetsgrupper som designade varsin systemanatomi. Vid skapandet av systemanatomin utgick varje arbetsgrupp från systemets användarfunktioner. För att identifiera användarfunktionerna användes kraven som definierats i kravspecifikationen.

Vid skapandet av varje anatom för systemanatomin identifierades de underliggande funktioner och komponenter som krävdes för anatomen i fråga [21]. Därmed identifierades alla beroenden mellan funktioner och komponenter. Dessa kopplades sedan ihop med pilar baserat på vilka funktioner och komponenter som interagerar med varandra. Slutligen strukturerades anatomerna upp i följande abstraktionslager:

1. Användarfunktioner
2. Användargränssnitt
3. Serverfunktioner
4. Databassystem
5. Hårdvara

Efter att de tre grupperna skapat ett utkast vardera för systemanatomin sammanställdes dessa till en slutlig systemanatomi. Systemanatomin justerades utifrån återkoppling från handledare och examinator i kursen.

Tanken var att under projektets gång kontinuerligt följa upp och uppdatera systemanatomin för att säkerställa en gemensam förståelse av systemets struktur och beroenden. I Avsnitt 5.6 redogörs det faktiska utfallet.

4.8.1 Utvärdering av systemanatomin

Under slutfasen av projektet gjordes en utvärdering för att få en bild över hur gruppen upplevde att systemanatomin hjälpte utvecklingen och om det upplevdes att systemanatomin användes som den skulle. En enkät skickades ut till varje deltagare som värderade olika påståenden på skalan 1-5. 1 motsvarar att påståendet inte stämde och 5 motsvarar att påståendet stämde bra. Enkäten bestod av följande påståenden:

1. Systemanatomin hjälpte mig att få en överblick över hela systemet.
2. Jag förstod bättre hur olika delar av systemet hängde ihop tack vare systemanatomin.
3. Systemanatomin underlättade vårt arbete med att planera utvecklingen.
4. Jag använde systemanatomin regelbundet under projektets gång.
5. Jag tror att systemanatomin hade fungerat bättre om vi la mer tid på att uppdatera den.
6. Jag skulle rekommendera att använda systemanatomi i liknande projekt.
7. Jag upplevde att systemanatomin bidrog till ett bättre slutresultat.

Resultatet för enkäten redovisas i Avsnitt 5.7.

5 Resultat

Detta kapitel beskriver det system och de processer som har utvecklats för att besvara frågeställningarna i Avsnitt 1.3, med stöd av de metoder som beskrivs i Kapitel 4. Här presenteras även projektgruppens gemensamma erfarenheter från arbetet.

5.1 Värde för kunden

Genom att uppfylla alla krav i kravspecifikationen kunde värde för kunden säkerställas. Utöver detta anpassades även applikationen efter användning på mindre skärmstorlekar såsom mobila enheter.

För att minska risken för missförstånd har kontinuerlig kontakt med kunden upprätthållits under projektets gång. Vid oklarheter eller frågor har projektets analysansvarige och konfigurationsansvarige kontaktat kunden via mejl, och vid behov deltagit på kundmöten.

För att få en djupare förståelse för kundens behov genomfördes en undersökning där ett formulär skickades ut till de anställda på DIGIT. Formuläret innehöll frågor om hur det nuvarande tidsrapporteringssystem används, dess för- och nackdelar samt förbättringsförslag. Resultatet gav en djupare förståelse av användarnas behov och önskemål. Detta togs i beaktande vid utformandet av TIDIG för att skapa värde för kunden.

5.2 Kvalitetsmätningar

Resultatet av de mätningar som utfördes på webbapplikationens LCP [1] visas i Tabell 3. Resultatet baseras på innehållet i main-branchen och ändpunkten /tasks, som visar uppgifter som användaren är tilldelad. Alla mätningar, med undantag för den som genomfördes den första maj, resulterade i tider under den övre gränsen på 2.5 sekunder enligt produktens kvalitetskrav. Direkta mätningar i en webbläsare visade istället ett genomsnittligt LCP-värde på 0.3 sekunder, vilket tyder på att den automatiska mätningen ger missvisande resultat.

Tabell 3: Resultatet av LCP-mätningar i sekunder

Datum	**15/4**	**16/4**	**17/4**	**18/4**	**20/4**	**21/4**	**22/4**	**23/4**	**24/4**	**1/5**
LCP	2.2	2.2	2.2	2.2	2.3	2.3	2.3	2.3	2.1	2.9

Resultatet från SUS-mätningarna presenteras i Tabell 4, medan de påståenden som användes i mätningarna beskrivs i Avsnitt 4.6. Användartesterna som utfördes under den andra sprinten utfördes internt inom projektgruppen medan testerna under den femte sprinten utfördes av externa personer. Båda mätningarna resulterade i SUS-poäng som överskred den nedre gränsen på 68 poäng, i enlighet med produktens kvalitetskrav. Återkopplingen från testpersonerna i samband med testerna som genomfördes under den femte sprinten beskrev systemet som visuellt tillfredsställande, lättförståeligt och familjärt då det följde LiU:s gränssnitt. Det fanns även förbättringsrekommendationer som innefattade kommentarer på vissa ord som var svåra att tyda och generella fel i designen. Under testningen upptäcktes även buggar i systemet som därmed kunde åtgärdas av gruppen.

Tabell 4: Resultat från SUS tester

Sprint	**Medelvärde på SUS-fråga**										**Totalt**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
2	4.17	2.0	4.0	1.5	4.3	2.3	4.8	1.7	4.7	1.8	81.7
5	3.92	1.92	3.92	1.46	4.15	2.08	4.46	1.62	4.08	1.77	79.23

För backend-koden skapades automatiserade tester i syfte att uppfylla kvalitetskravet om 95 % testtäckning. I slutet av projektet uppnåddes 77% *statement coverage* och 75% *branch coverage*.

5.3 Processbeskrivning

Avsnittet beskriver hur projektgruppen har arbetat med Scrum kombinerat med Kanban-bräde samt hur arbetssättet har utvecklats under projektets gång.

5.3.1 Process i fokus

Projektgruppen arbetade i sprintar på en till tre veckor. I början av varje sprint tilldelades samtliga medlemmar någon av följande roller: Scrum master, produktägare eller medlem i utvecklingsteamet. Scrum mastern ansvarade för att leda möten, produktägaren hanterade sprintbackloggen och samtliga medlemmar i teamet var ansvariga för att genomföra uppgifterna. Teamledaren hade permanent rollen som Scrum master eftersom båda roller har liknande ansvarsområden. Initialt var kvalitetssamordnaren produktägare men ansvaret fördelades sedan mellan tre projektmedlemmar: frontendutvecklingsledaren, backendutvecklingsledaren och den dokumentansvarige.

En *produktbacklogg* upprättades, baserad på de milstolpar och aktiviteter som definierats i projektplanen. Inför varje sprint skapades även en *sprintbacklogg* som visualiserades med hjälp utav ett Kanban-bräde i Azure DevOps. Mötesprotokoll för sprintplanerings- samt retrospektiv-mötena skapades även för att säkerställa att alla relevanta ämnen behandlades.

Inför varje planeringsmöte fick produktägarna i uppgift att förbereda ett förslag på sprintbacklogg. Inledningsvis gick projektgruppen igenom samtliga sprintuppgifter i helgrupp, tilldelade ansvar och prioriterade dessa på en skala från ett till fyra, där fyra motsvarade högst prioritet. Processen utvecklades därefter baserat på

utvärderingar för att bland annat effektivisera planeringsmötena. Mötet inleddes därefter med att gruppen gemensamt gick igenom de sprintuppgifter som berör samtliga projektmedlemmar, såsom exempelvis dokumentation. Därefter delades gruppen upp i frontend och backend. Varje utvecklingsledare gick då igenom relevanta sprintuppgifter inom sitt arbetsområde. Slutligen samlades hela gruppen igen för gemensam återkoppling, tekniska diskussioner och bedömning av sprintens rimlighet.

De dagliga Scrum-möten hölls initialt på distans via *Discord* klockan 12:30, men detta ändrades sedan till klockan 11:03 för att passa projektmedlemmarnas schema. Till en början hade projektgruppen separata mötesprotokoll för Scrum-relaterade möten respektive vanliga gruppmöten. Dessa mötespunkter integrerades sedan till ett kombinerat mötesprotokoll för att samla all dokumentation på samma ställe.

Uppgifterna organiserades i olika kategorier på Kanban-brädet som baserades på sprintens övergripande mål. Dessa kategorier var generellt dokumentskrivning samt frontend- och backendutveckling. Under den sista sprinten av produktens utvecklingsfas gick utvecklingsledarna igenom kravlistan och milstolparna i projektplanen. Sprintuppgifterna skapades utifrån det som återstod att implementera och organiserades i kategorier beroende på vilket krav de relaterade till.

För att visualisera uppgifternas prioritet färgkodades uppgifterna, där varje färg representerade en viss prioriteringsnivå. De flesta uppgifter tilldelades en ansvarig person under sprintplaneringsmötet, men under de tre första sprintarna lämnades vissa uppgifter utan en utsedd ansvarig. Projektmedlemmar som senare under sprinten hade tid att utföra uppgiften kunde därmed skriva upp sig som ansvariga.

Projektgruppen strävade efter att inte skapa nya sprintuppgifter efter sprintplaneringsmötet avslutats. Om det ändå ansågs nödvändigt att lägga till nya uppgifter, markerades detta tydligt i uppgiftens titel att det var en nyligen tillagd uppgift. Sprintuppgifter som inte hann färdigställas under en sprint flyttades över till nästkommande sprintbacklogg. Ett undantag var den femte sprinten som varade i tre veckor och innehöll ett möte i mitten av sprinten då det var tillåtet att skapa nya sprintuppgifter.

5.3.2 Utvärdering av process

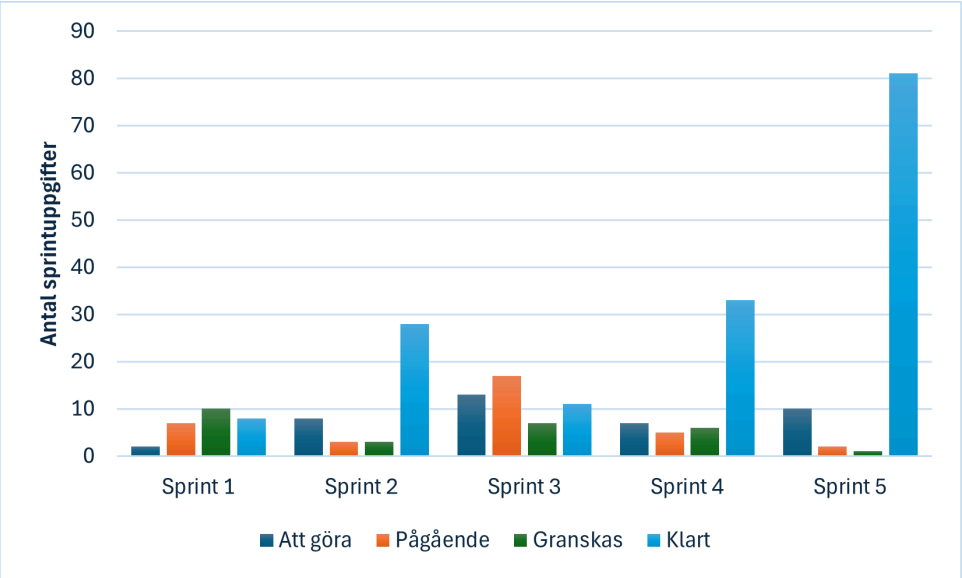
Resultatet från de första fyra sprinterna presenteras i Tabell 5 vars frågor beskrivs i Avsnitt 4.7.2. Fråga 4 besvarades inte efter den första sprinten eftersom sprint retrospektiv inte hade genomförts då.

Tabell 5: Medelresultat på reflektionsformulär

Sprint	Fråga 1	Fråga 2	Fråga 3	Fråga 4	Fråga 5	Fråga 6
1	3.44	2.56	2.56	-	3.67	3.00
2	4.11	3.78	3.67	4.00	3.89	4.22
3	3.22	3.56	3.11	3.56	3.78	3.67

4	4.22	3.00	3.67	3.67	4.33	3.78
5	4.00	3.67	3.44	3.78	3.00	3.89

I slutet av varje sprint, under mötet för sprint retrospektiv, sammanställdes uppgifternas status på Kanban-brädet för att ge en översikt över sprintens resultat. Resultatet presenteras i Figur 1 och visar att antalet uppgifter i sprintarna ökade över tid.

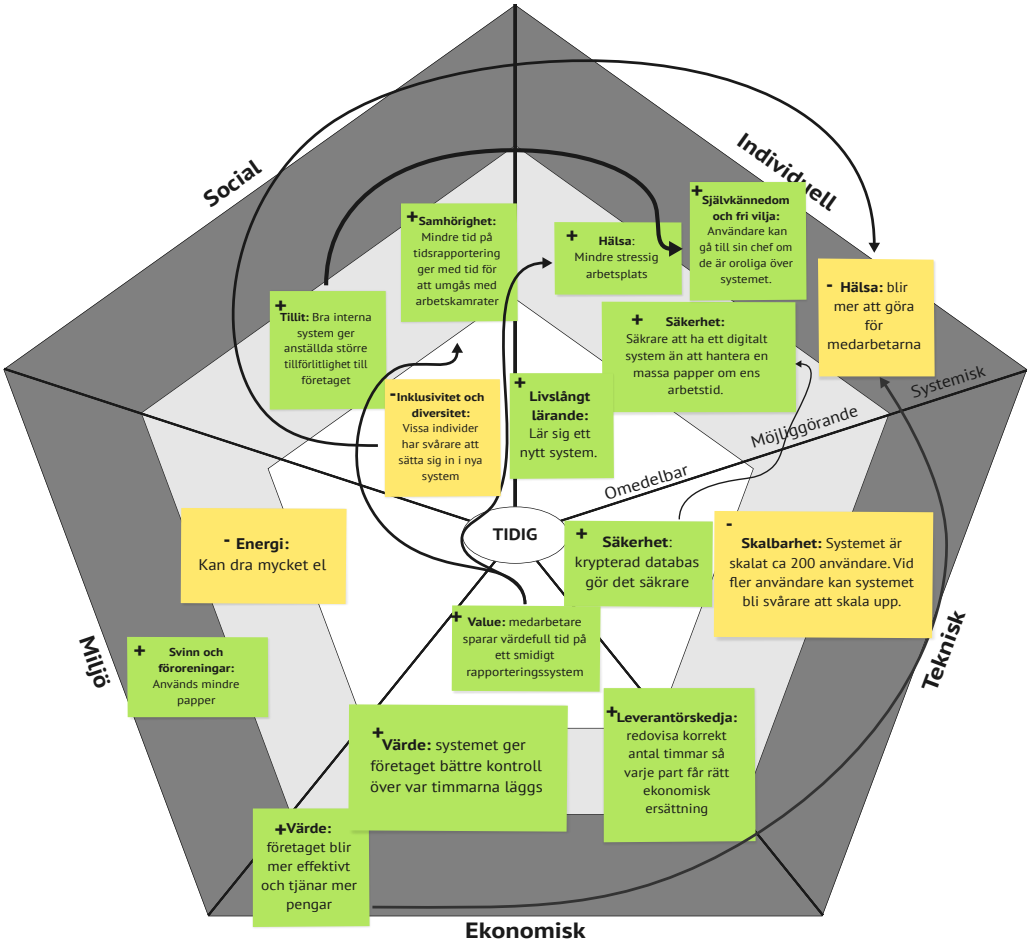


Figur 1: Sprintuppgifternas placering i Kanban-brädet i slutet av varje sprint.

5.4 SusAF

Resultatet av SusAF-analysen är både det Sustainability Awareness Diagram (SusAD) som syns i Figur 2. Gruppen valde att markera positiva effekter med grön färg och ett plus (+) medan negativa effekter markerats gula och ett minus (-).

SusAD synliggör risker och möjligheter som gruppen identifierade med produkten och deras beroenden. Potentiella möjligheter och positiva aspekter med systemet skulle, enligt SusAF-analysen, kunna vara att medarbetare sparar värdefull tid med ett smidigt rapporteringssystem vilket i sin tur hade kunnat bidra till ett mindre stressigt arbetsklimat. En potentiell risk hade däremot kunnat vara att vissa har svårare att sätta sig in i nya system och att systemet därför hade kunnat innebära en stressfaktor för dem.



Figur 2: SusAD över produkten

5.5 Gemensamma erfarenheter

De gemensamma erfarenheterna inkluderar de processrelaterade och tekniska erfarenheter som erhållits av projektgruppen under projektets gång. Scrum i relation till sprintuppgifter behandlas här och för mer ingående information hänvisas läsaren till Avsnitt 5.3.

5.5.1 Processrelaterade erfarenheter

Under projektets genomförande har flera processrelaterade erfarenheter förvärvats. Dessa har uppkommit genom interna utbildningar, genomförandet av Scrum och lärdomar från projektgruppens kommunikation.

5.5.1.1 Projektets faser och planering

Arbetet med olika projektfaser, som beskrivs i Avsnitt 4.1, har resulterat i flera erfarenheter kring planering. Den första fasen, analysfasen, inkluderade arbete med kravspecifikationen, men på grund av kursens tidiga inlämningsdatum för resterande dokument påbörjades även projektplanen och kvalitetsplanen under analysfasen. Detta resulterade i att analysfasen och planeringsfasen kombinerades till en fas där kraven samlades in parallellt med framställandet av projektplan och resterande dokument. Den ursprungliga tidplanen utgick från projektplanens milstolpar som placerades ut i projektets faser och innehöll en veckas tidsbuffert i slutet av projektet för eventuella förseningar.

När halva projekttiden passerat, och projektet befann sig i realiseringsfasen, behövde tidplanen justeras drastiskt eftersom flera milstolpar hade försenats. Det berodde på att det fortfarande saknades en komplett design av webbsida och implementationen med LiU-SSO hade försenats. Denna försening resulterade i att den planerade tidsbufferten tog slut. Detta ledde till en större omorganisering av projektets tidplan då milstolpar tog längre än väntat att uppnå och den gamla tidplanen var för optimistisk. Den nya tidplanen lade ett mindre fokus på projektets faser och etablerade istället nya perioder för produktutveckling och dokumentarbete i enlighet med Scrums iterativa sprintar. De sprintuppgifter som togs fram efter omplaneringen utgick från milstolparna och kraven från kravspecifikationen, till skillnad från tidigare. Vidare bestämdes ett datum för *feature freeze* då sprintuppgifterna skulle vara klara och då fokuset skulle läggas på testning och buggfixar. Slutligen färdigställdes systemet i tid för överlämning till kund.

5.5.1.2 Delning av kompetens

I projektet genomfördes interna utbildningar i form av workshoppar inom konfiguration, backend- och frontendutveckling, enligt Avsnitt 4.3. Utbildningarna inom konfiguration och backendutveckling kombinerades till ett längre tillfälle, där både teoretiska och praktiska moment ingick. Utbildningens omfattning var bred och krävde vissa förberedelser under själva workshoppen, vilket resulterade i att en betydande del av tiden ägnades åt installationer och felsökning snarare än praktiskt lärande. En insikt från workshoppen var att tydliga instruktioner och en avgränsad uppgiftsomfattning underlättade inlärning. Dessa insikter tillämpades i den nästkommande workshoppen inom frontendutveckling, där deltagarna följde en handledning i ramverket Svelte och därefter övade självständigt.

5.5.1.3 Scrum: Formulering av uppgifter

I Avsnitt 5.3.1 står det att under varje sprintplaneringsmöte presenterades förslag på sprintuppgifter i Kanban-brädet. Ett återkommande ämne under sprint retrospektiv var att uppgifterna ofta upplevdes som alltför generellt formulerade och behövde brytas ner i mindre specifika deluppgifter. Exempelvis var uppgifter i första sprinten formulerade: ”Design tidsrapportering dator” eller ”Skriv kravspecifikationen”. Otydliga uppgifter orsakade förvirring över när dessa skulle definieras som klara i Kanban-brädet. Därmed diskuterade gruppen under flera sprint retrospektiv hur uppgifter kunde specificeras bättre till sprinten därpå. I början på den femte sprinten bestämdes att gruppen behövde göra en omplanering. Detta resulterade i ett nytt Kanban-bräde presenterades i mitten av sprinten där varje deluppgift kunde kopplas till milstolpar och krav från kravspecifikationen. Slutdatumet på sprinten fick också större betydelse då slutdatumet på sprinten skulle vara dagen efter projektets *feature freeze*. Denna gång gick inte frontend-gruppen och backend-gruppen separat igenom uppgifterna, utan hela gruppen satt tillsammans när uppgifterna presenterades och delegerades. Detta gjorde att projektmedlemmarna hade möjlighet att ställa frågor på uppgifter de upplevde som otydligt formulerade och därmed kunde formuleringarna korrigeras.

5.5.1.4 Kommunikation

Under projektarbetet uppstod det kommunikationsbrister när utvecklingsarbetet skedde på distans och i separata arbetsgrupper. Det ledde till missförstånd och dubbelarbete. Som åtgärd etablerades gemensamma rutiner vilket inkluderar Kanban-verktyg, dagliga Scrum-möten och en ny Discord-kanal för att informera gruppen med statusuppdateringar. Ytterligare ett problem med bristande kommunikation handlar om kundkontakten. Under vissa perioder av projektarbetet har det varit svårt att få kontakt med kunden, vilket har resulterat i förseningar av funktionalitet. Den bristande kommunikationen med kunden påverkade tillgången till resurser gällande deras inloggningssystem som användes för att logga in i TIDIG. Detta löstes genom att kunden erbjöd oss kontaktuppgifter till en systemutvecklare hos dem, vilket gjorde att kommunikation om de tekniska detaljerna skedde direkt med en systemutvecklare och underlättade därmed utvecklingsprocessen.

5.5.2 Tekniska erfarenheter

Inför projektet hade projektmedlemmarna varierande tekniska erfarenheter, men det fanns alltid någon med grundläggande kunskap om programmeringsspråket eller ramverket som skulle användas. En bra erfarenhet av detta var hur värdefullt det var när personer med erfarenhet tog initiativ och planerade gemensamma workshoppar, skickade ut studiematerial och svarade utförligt på frågor från gruppen. Detta bidrog till att hela projektgruppen fick en gemensam förståelse av projektets tekniska grund.

Inför projektet hade ingen i projektgruppen erfarenhet av Azure DevOps. Genom projektet har gruppen fått praktiska erfarenheter att använda verktyget, både för att hantera processen via Kanban-bräden, men även för testning och arbetsflöden. Medlemmar i gruppen hade tidigare erfarenhet att implementera

databas genom att skriva SQL-kod. Men under detta projekt användes ASP.NET med C#. Detta ledde till att gruppen behövde lära sig ett nytt programmeringsspråk. De kunde dock använda tidigare teori inom objektorienterad programmering och databaser för att underlätta inlärning och implementation.

5.5.2.1 Databasdesign och modellering

Utvecklingen av TIDIG har gett projektgruppen teknisk erfarenhet av databasdesign och modellering av data genom *Entity Framework*. Arbetet gick ut på att identifiera de datatyper som gruppen kallar för *hårda datatyper*. Dessa har direkta motsvarigheter som databastabeller. Ett exempel på detta i systemet är `Task` och `User`. Systemets *hårda datatyper* och deras relation med varandra modellerades med hjälp av *EER-diagram* och relationsdiagram. En viktig sak som gruppen insåg under projektet var att denna modellering inte täckte allt och att data hanterades olika i frontend och backend. Detta resulterade i att *tunna datatyper* utvecklades. Exempel på dessa är `TaskView` och `UserView`. *Tunna datatyper* innehåller den data från databasen som är relevant för frontend. Dessa datatyper tillät alltså systemets backend att skicka mindre data till frontend-servern. Exempelvis var detta bra då användarvyn inte behövde visa all data som backend hade om `Task`-objekten och därför kunde `TaskView` användas istället.

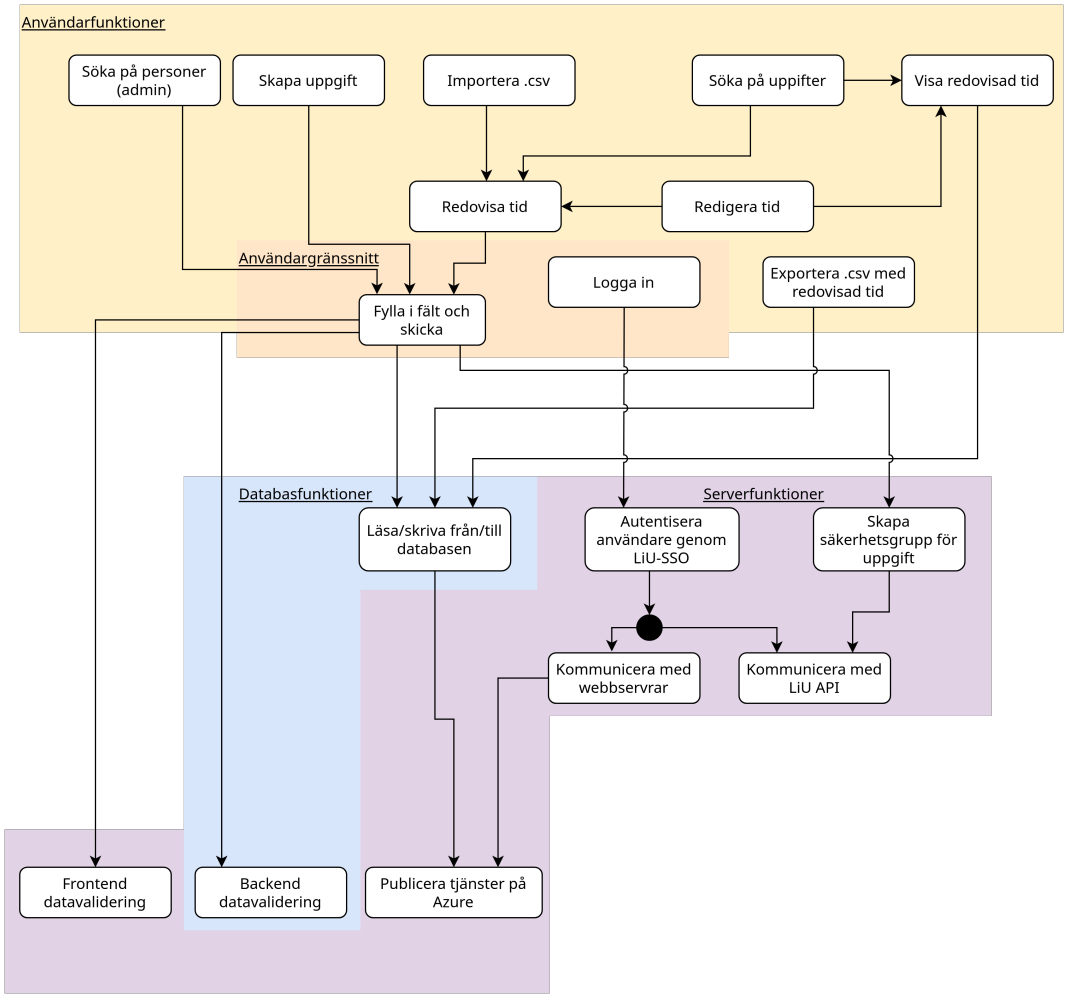
På grund av bristande kommunikation mellan arbetsgrupperna utvecklades sedan vissa stora datatyper som skulle skickas till frontend-servern och detta bröt mot den ursprungliga designen och strukturen. En viktig sak att nämna är även att arbetsgrupperna för backend och frontend undvek att prata ihop sig om ändpunkter och data som frontend-servern förväntade sig från backend-servern. Det skedde ett möte då detta togs upp men bara vissa ändpunkter hann behandlas.

5.6 Systemanatomi

Figur 3 visar den systemanatomi som projektgruppen skapade. Den skiljer sig från planerna beskrivet i Avsnitt 4.8. Det var exempelvis inte relevant att skapa ett lager för hårdvara, eftersom systemet endast är en mjukvarutjänst.

Systemanatomin visade sig vara som mest hjälpsam under projektets planerings- och etableringsfas. Den var till hjälp både för att skapa övergripande blockdiagram över systemet, se Figur 9, och vid utformningen av webbsida, då det fastställdes vilka vyer som skulle implementeras.

Systemanatomin hade även viss användbarhet vid planering av testning. Den har hjälpt till att identifiera vilka delar av systemet och vilka funktioner som ska testas, speciellt vilka integrationer som är viktiga. Under realiseringsfasen har systemanatomin inte aktivt använts då den varken granskats eller uppdaterats.



Figur 3: Systemanatomi

5.7 Utvärdering av systemanatomins betydelse

Tabell 6 innehåller resultatet på enkäten med frågorna samt dess medelvärde och standardavvikelser. Värderingen gjordes på skalan 1-5, där 1 motsvarade att påståendet inte stämde så bra och 5 motsvarade att påståendet stämde bra. Medelvärdet låg mellan 1 och 2.7, vilket kan tolkas som ett lågt resultat. Standardavvikelsen ligger som högst på 1.4, vilket betyder att svaren har medel till låg variation.

Tabell 6: Gruppens svar på frågor kring systemanatomins användning.

Fråga	Medelvärde (1-5)	Standardavvikelse
Systemanatomin hjälpte mig att få en överblick över hela systemet	2.7	1.2
Jag förstod bättre hur olika delar av systemet hängde ihop tack vare systemanatomin.	2.3	1.4

Fråga	Medelvärde (1-5)	Standardavvikelse
Systemanatomin underlättade vårt arbete med att planera utvecklingen.	1.9	1.4
Jag använde systemanatomin regelbundet under projektets gång.	1	0
Jag tror att systemanatomin hade fungerat bättre om vi la mer tid på att uppdatera den.	2.4	1
Jag skulle rekommendera att använda systemanatomi i liknande projekt.	1.7	0.7
Jag upplevde att systemanatomin bidrog till ett bättre slutresultat.	1.8	1

5.8 Systembeskrivning

TIDIG består av en backend-server som hanterar logik och interaktion med databas, samt en frontend-server som hanterar användarinteraktioner. Detta avsnitt utforskar systemets struktur och funktioner.

5.8.1 Frontend

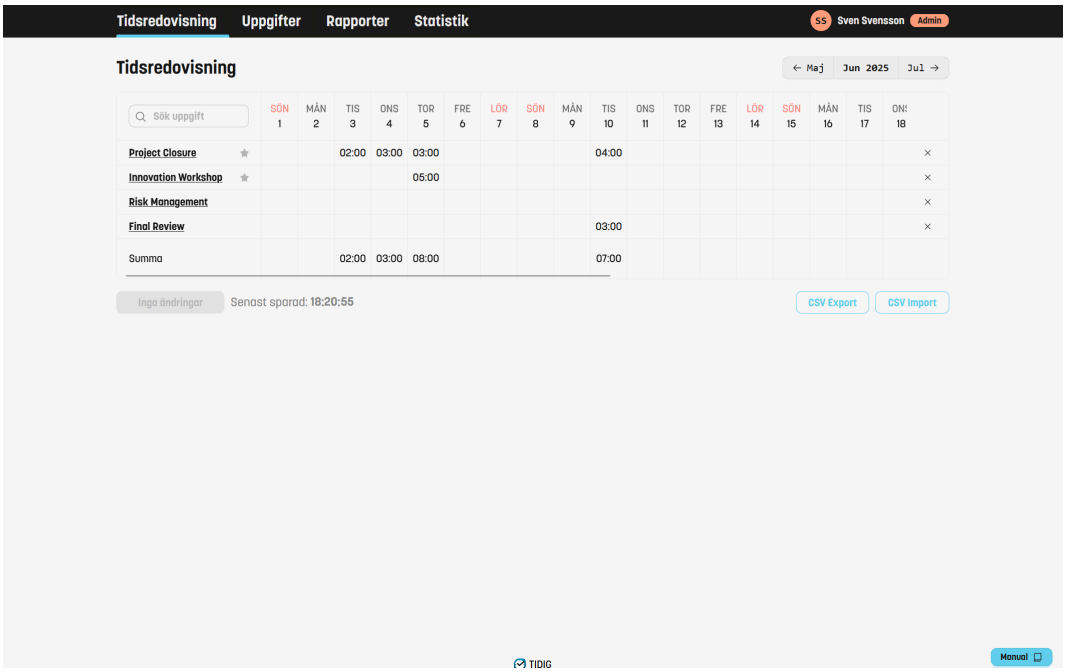
Frontend utvecklades i Svelte med TypeScript och använde Tailwind CSS för att ge komponenter styling, samt externa bibliotek som tillhandahöll färdiga CSS komponenter. All data hämtas från backend-servern via API-anrop.

När en användare öppnar webbapplikationen omdirigeras denne automatiskt till inloggning via LiU-SSO, där även användarens behörighet hämtas. Systemet TIDIG stödjer två roller: *administratörer* och *tidredovisare*. Administratörer är tidredovisare med utökade behörigheter, vilket innebär att deras version av systemet inkluderar extra funktionalitet.

Användare kan navigera mellan vyerna för tidsredovisning, uppgifter, rapporter, statistik och profiler. Vyerna finns tillgängliga via ett navigeringsfält på stora skärmar eller via en rullgardinsmeny på mindre skärmar. Vyerna har även ett anpassat utseende baserat på skärmstorlek och dessa behåller även samtliga funktionaliteter.

5.8.1.1 Tidredovisning

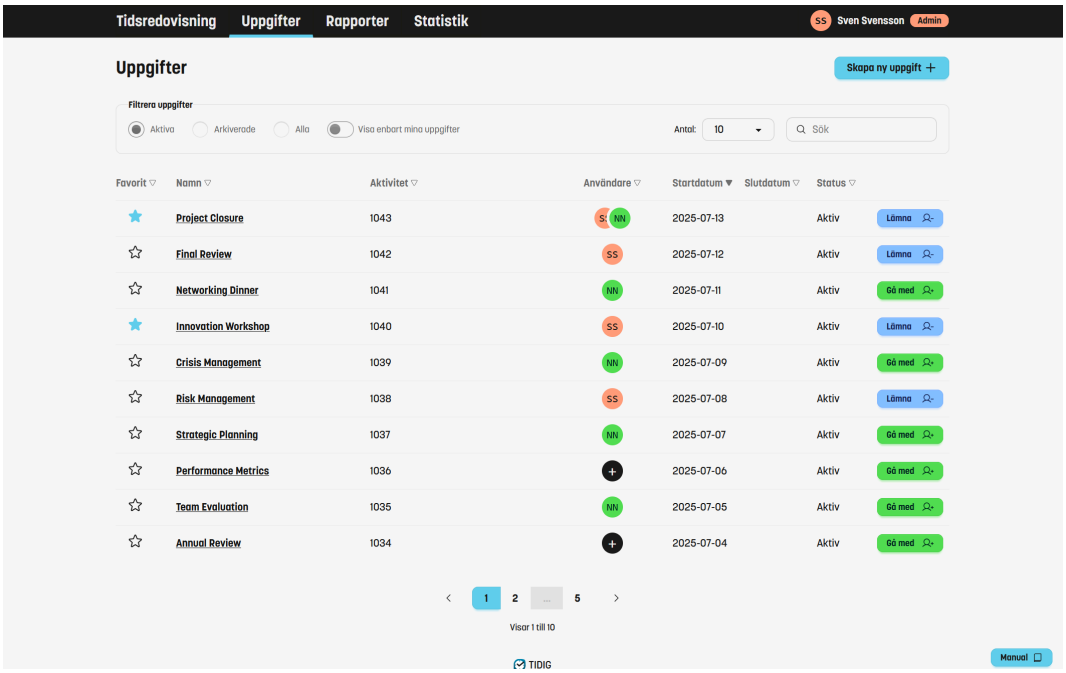
Tidredovisningsvyn är startsidan för TIDIG. Där kan användare redovisa sin arbetade tid på de uppgifter användaren har tillgång till. Figur 4 visar tidredovisningsvyn för stora skärmar.



Figur 4: Tidredovisningsvyn i TIDIG

5.8.1.2 Uppgifter

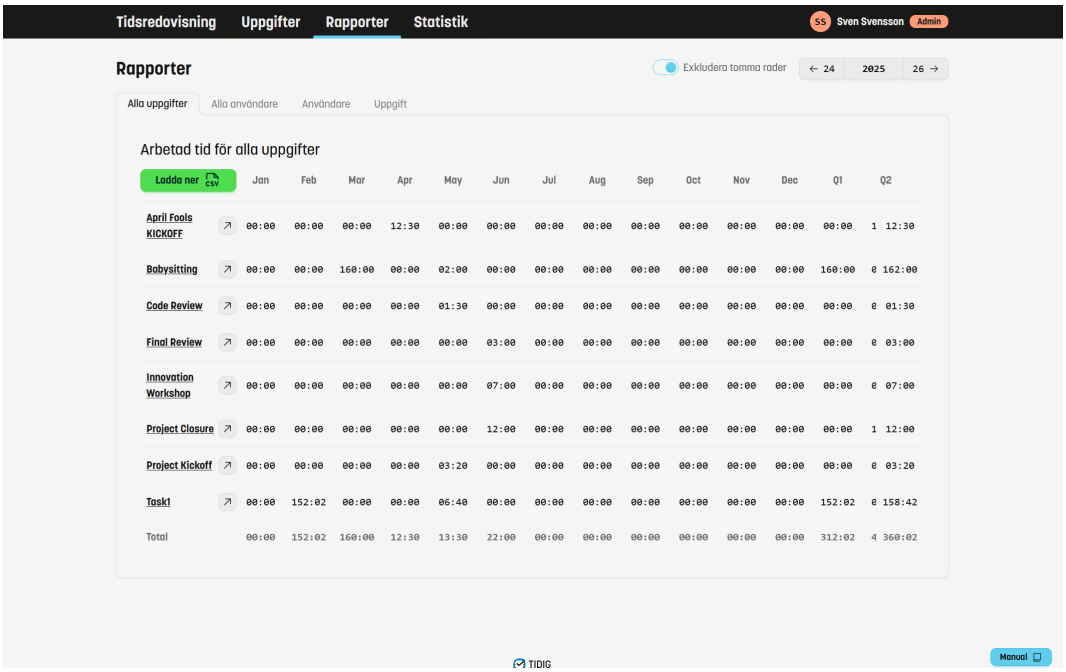
Uppgiftsvyn, se Figur 5, ger användare möjlighet att se, gå med i, samt lämna samtliga uppgifter. I denna vy kan administratörer även skapa, redigera och arkivera uppgifter. Filtreringsalternativen är till för att filtrera vilka uppgifter som syns i uppgiftslistan.



Figur 5: Uppgiftsvyn i TIDIG

5.8.1.3 Rapporter

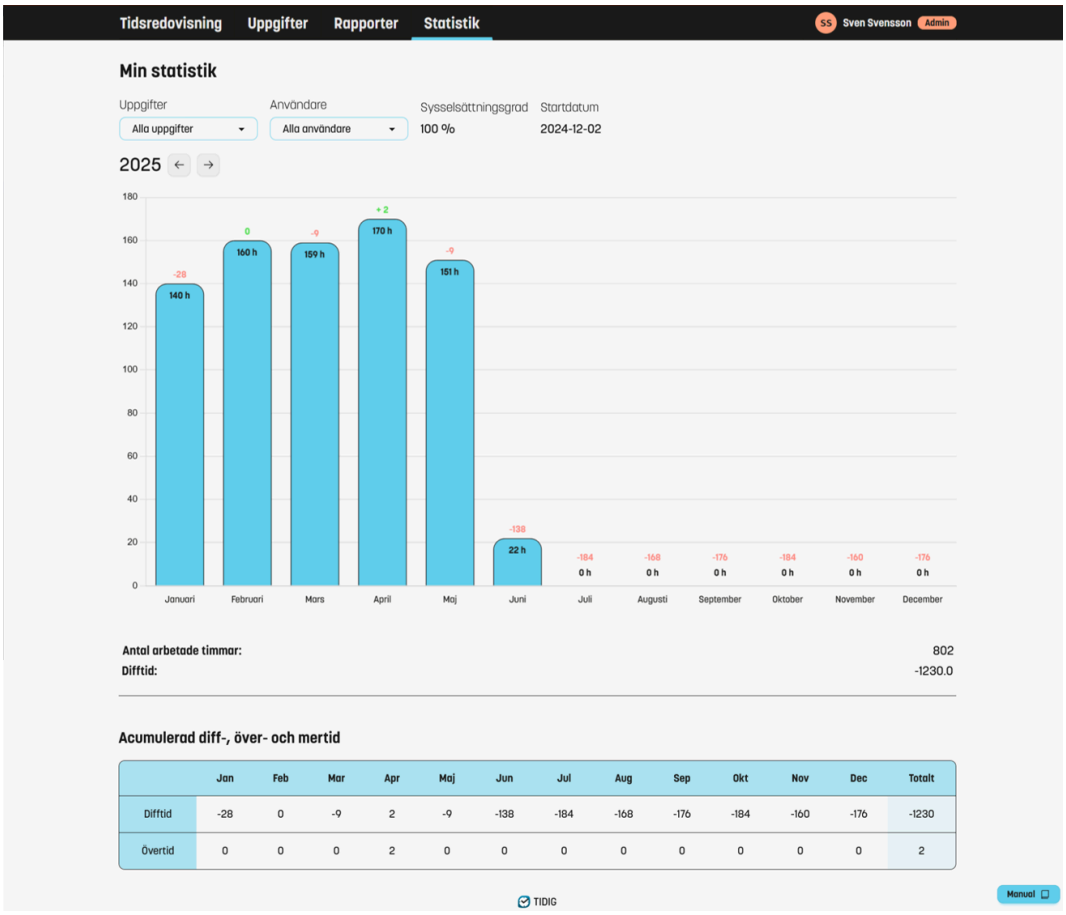
I rapportvyn, se Figur 6, kan användare få en översikt av sin redovisade tid. Dessa rapporter ger en överblick på månadsbas med möjlighet att se över historik från tidigare år. Administratörer kan även ta fram rapporter för enskilda projekt där enskilda tidredovisares tider kan tas fram. Administratörer kan även exportera rapporter i *.csv*-format för att hanteras externt.



Figur 6: Rapportvyn i TIDIG

5.8.1.4 Statistik

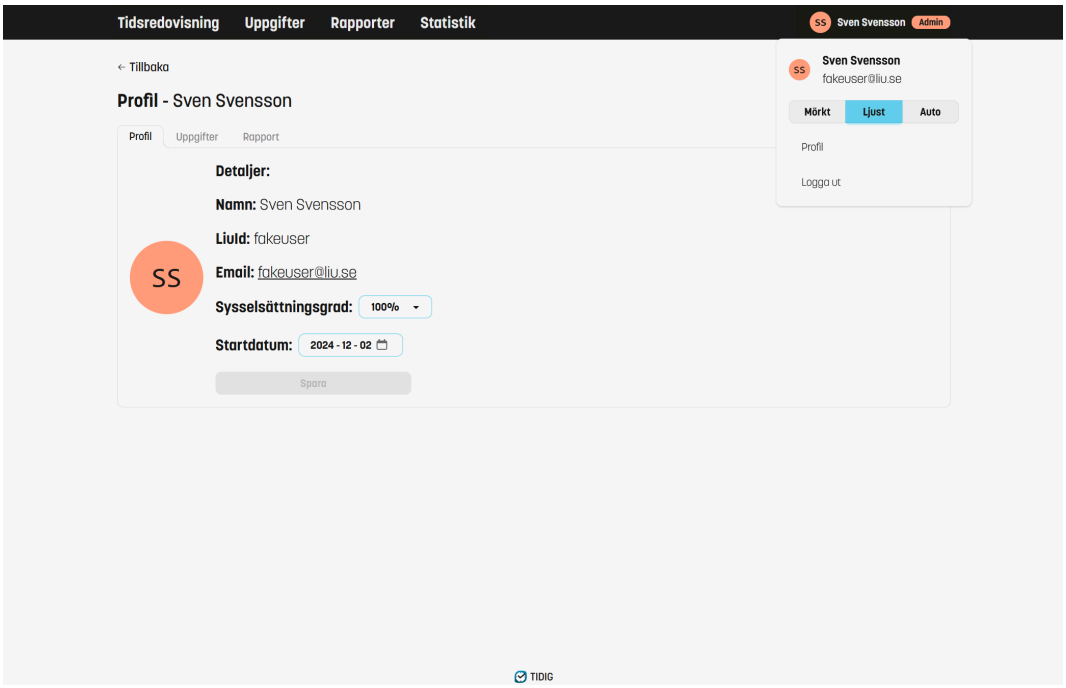
Statistikvyn, se Figur 7, visualiserar användarnas arbetstid för samtliga och enskilda uppgifter i form av ett stapeldiagram. Vyn innehåller även information om förväntad arbetstid, arbetad tid och flextid.



Figur 7: Statistikvyn i TIDIG

5.8.1.5 Profil

I profilvyn, se Figur 8, finns information om de olika användarna samt deras aktiva uppgifter och rapporter. Profilsidan agerar även som inställningsmeny där användare kan ändra sin sysselsättningsgrad och sitt startdatum. Startdatumet används för beräkning av statistiken i statistikvyn.



Figur 8: Profilvyn i TIDIG

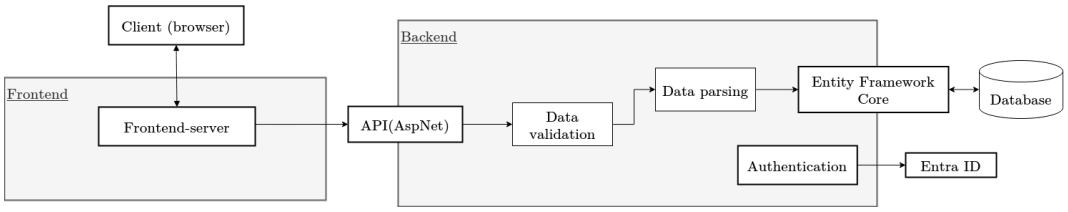
5.8.2 Backend

Backend-servern har som uppgift att utföra all logik och datainsamling. Den ansvarar för att verifiera användarnas åtkomst genom LiU-*SSO*, hålla reda på användares sessioner och hämta samt spara data genom databasen. Allt detta görs tillgängligt genom ett webb-API utvecklad i ASP.NET. Denna kan frontend-servern använda för att få tillgång till den data som backend-servern hanterar. Varje ändpunkt och dess förväntade in- och utdata kan hittas i backend-serverns API-dokumentation som automatiskt genereras med verktyget NSwag. För att hantera databasen används verktyget Entity Framework Core, som gör det möjligt att generera databastabeller utifrån C#-klasser. Verktyget gör det möjligt att skapa kopior av databasen eftersom strukturen definieras på ett ställe i koden. Detta har varit användbart då varje projektmedlem kunde ha en lokal databas under utvecklingen av TIDIG.

5.8.3 Uppdelningen av frontend- och backend-server

På det sättet som systemet byggdes är backend-serverns ändpunkter bara tillgängliga genom frontend-servern. Det går alltså inte att skicka förfrågningar direkt till backend över internet. Denna struktur visas i Figur 9. Fördelen med detta är att kontrollen för vilka förfrågningar som kan skickas stannar hos frontend. En nackdel med uppdelningen är att vissa ändpunkter behöver skapas två gånger. En gång på frontend-servern och en gång på backend-servern. Anledningen till detta är att vissa ändpunkter behöver kunna användas även efter att sidan laddats.

Att systemet är uppdelat i frontend- och backend-server har resulterat i att produkten fullföljer kravet om att backend är i ASP.NET med C# samtidigt som frontend-utveckling är i Svelte.



Figur 9: Blockschema över systemet TIDIG

5.8.4 Uppdelning i separata arbetsgrupper för frontend och backend

Att dela upp webbapplikationen i två distinkta servrar: frontend och backend, gjorde att två arbetsgrupper naturligt kunde jobba parallellt på funktionalitet. Den tydliga uppdelningen förde dock med sig en del problem.

En svårighet med att ha distinkta arbetsgrupper var att arbetet som den ena gruppen utförde ofta berodde på att den andra gruppens arbete fungerade. Från frontends sida krävdes exempelvis backend-ändpunkter för att kunna testa att koden fungerade. Från backends sida var det samtidigt svårt att utveckla ändpunkter när frontend inte på förhand visste exakt vilka de skulle behöva och hur de skulle fungera. Att backend-gruppen inte kunde utveckla ändpunkterna i förväg gjorde vid många tillfällen att ny frontend-kod inte kunde testas förrän flera dagar eller ibland över en vecka senare. Även när en ny ändpunkt förfrågades hände det ofta att dess funktionalitet att den ändrades under projektets gång. Stora delar av koden som skrevs i backend behövde därför skrivas om flera gånger vilket resulterade i arbete som kunde ha undvikits.

6 Diskussion

Detta kapitel problematiserar och analyserar resultat kring värdeskapande för kunden, erfarenhetshantering, systemanatomi och uppdelning i arbetsgrupper. Den belyser även källkritik och de samhälleliga och etiska aspekter med produkten som har skapats.

6.1 Resultat

Gruppens erfarenheter från projektet och Scrum kan tolkas ur flera perspektiv. De tekniska beslutens påverkan på projektet och alternativa implementationssätt diskuteras.

6.1.1 Mätningar på process

Resultatet från projektgruppens utvärdering av sprintarnas genomförande presenterades tidigare i Tabell 5. Sammanställningen av svaren visar inga stora variationer mellan sprintarna, utan samtliga påståenden har generellt fått relativt höga medelvärden. Eftersom samtliga värden ligger över mittpunkten på skalan (2.5 av 5), kan resultatet tolkas som övervägande positivt. Formuläret har främst varit värdefullt som underlag inför sprint retrospektiv, då det hjälpt projektgruppen att identifiera förbättringsområden. Ett konkret exempel är under den fjärde sprinten, där ett lägre resultat på frågan om kommunikation uppmärksammades. Detta låg till grund för en diskussion inom gruppen, vilket i sin tur möjliggjorde att orsaker identifierades och åtgärder kunde vidtas inför kommande sprint. Formuläret har därmed bidragit till att utveckla en effektiv arbetsprocess, vilket i sin tur har möjliggjort för projektgruppen att leverera värde till kunden.

Uppgifters placering på Kanban-brädet visualiseras i Figur 1. Med tanke på att målet är att färdigställa samtliga uppgifter under en sprint har projektgruppen presterat relativt dåligt i den aspekten. Detta resultat stämmer överens med projektgruppens uppfattning att det är mycket svårt att skapa tydliga uppgifter inför sprinten som förblir relevanta och innebär en rimlig arbetsbelastning. Det bästa resultatet uppnåddes under sprint fem, då 86% av uppgifterna slutfördes. Denna förbättring kan sannolikt förklaras av att sprinten var längre, vilket gav mer tid för genomförande av uppgifter, samt att uppgifterna var tydligt definierade då de kopplades till specifika krav. Detta understryker vikten av att kontinuerligt anpassa arbetssättet, vilket ökar projektgruppens effektivitet och möjliggör leverans av en bättre produkt som skapar ett högre värde för kunden.

6.1.2 Kvalitetstester

Vid både de interna och de externa användbarhetstesterna fick produkten en SUS-poäng som tydligt överskred den gräns som angavs i produktens kvalitetskrav, vilket även speglades i återkopplingen från testpersonerna. Ett genomsnittligt SUS-resultat på 68 poäng indikerar att produkten är godkänd ur ett användbarhetsperspektiv [19]. Det positiva resultatet på de interna testerna innebar att implementeringsfasen kunde påbörjas utan att några omfattande ändringar i designen krävdes. Eftersom den nästan färdigutvecklade versionen av produkten TIDIG uppnådde ett SUS-resultat på cirka 79 poäng och återkopplingen från testerna var lätta att åtgärda, kunde projektgruppen i slutfasen fokusera på andra aspekter än användbarheten. Användbarhetstesterna resulterade i att buggar upptäcktes och gränssnittet förbättrades för att öka användarvänligheten. Detta var särskilt värdefullt för kunden eftersom de betonat vikten av ett användarvänligt system.

Även prestandamätningarna visade konsekvent goda resultat, där LCP-värdena låg under den tröskel som sattes i kvalitetskraven. Detta är sannolikt ett resultat av att projektgruppen tidigt tog hänsyn till prestanda såsom att exempelvis utföra tyngre beräkningar i systemets backend. Ett undantag var dock den senaste mätningen, där orsaken till avvikelsen inte har kunnat fastställas. Det kan vara relaterat till den stora skillnaden mellan testar utförda automatiskt eller manuellt i webbläsare. Ingen orsak till skillnaden har identifierats.

6.1.3 Programmeringsspråk och ramverk

Ramverket Svelte användes för frontendutvecklingen. En avgörande faktor för valet var att kunden kommer att ta över systemet efter projektets avslut, vilket ställde krav på att kodbasen ska var lättförståelig och underhållbar. Svelte bygger på webbtekniker som HTML, CSS och JavaScript, vilket minskar inlärningströskeln och gör det enklare för framtida utvecklare att sätta sig in i projektet, även utan tidigare erfarenhet av ramverket. Dessutom kräver Svelte minimalt med mallkod, vilket gjorde det möjligt för projektgruppen att snabbt komma igång med utvecklingen. En ytterligare fördel var att det är enkelt att integrera Svelte med externa JavaScript-bibliotek, vilket möjliggjorde användningen av tredjepartslösningar. Det öppnade upp för utnyttjandet av beprövade verktyg med långsiktigt underhåll och bred användning, vilket accelererade utvecklingen. Ramverket Svelte bidrog därmed till ett ökat kundvärde, då den lättförståeliga kodbasen gav projektgruppen mer tid att fokusera på produktutvecklingen.

6.1.4 Processrelaterade erfarenheter

Erfarenheterna som beskrivs i Avsnitt 5.5 tar upp flera aspekter från projektarbetet. De processrelaterade erfarenheterna behandlar följande områden: Planering, kommunikation, Scrum och delning av kompetens. I detta avsnitt kommer lärdomarna från erfarenheterna jämföras med varandra.

6.1.4.1 Planering

Det finns flera lärdomar om planering att nyttja från de erfarenheter som togs upp i Avsnitt 5.5.1.1. En generell lärdom är att det är svårt att planera relativt långt i förväg. Därför ska man vara förberedd på att planen kan och troligtvis kommer att

ändras. Vidare blir frågan hur man som grupp bör hantera när planeringen fallerar. Detta hanterades genom att planera om tidplanen och prioritera det viktigaste som skulle göras. Därefter hölls flera möten för att gå igenom milstolparnas och kravens status och informera om återstående uppgifter och deras slutdatum.

Den andra gången fungerade den nya planen bättre än den tidigare. Detta kan delvis berott på att man inte längre behövde planera så långt fram i framtiden, men det kan även bero på en ny insikt gällande att omplanera: Det räcker inte bara med att uppdatera interna deadlines, utan man ska också se över vad som är kvar att göra i projektet, relaterat till projektets krav och milstolpar.

En annan lärdom som man kan tänka på i framtiden är vikten av en tidsbuffert, speciellt att ha en större tidsbuffert när man saknar erfarenhet av att planera större projekt. Denna lärdom kommer från erfarenheten då gruppen hade en vecka i tidsbuffert som visade sig inte vara tillräckligt. Konsekvensen var att slutdatumet för att färdigställa TIDIG flyttades fram från att ha legat i början på maj, till slutet av maj. Från detta kan ännu en lärdom dras gällande beroende av tredje part: Om framsteg i ett projekt baseras på en tredje part behöver gruppen vara förberedd på förseningar hos denne och därmed behöver tidplanen innehålla en tidsbuffert som kompenserar detta.

6.1.4.2 Delning av kompetens

De workshoppar som genomfördes i syfte av kompetensdelning var lyckade i vissa aspekter, och misslyckades i andra. Dels gjorde de att projektmedlemmar snabbt kom igång med verktygen och kunde få ställa frågor till de som var mer kunniga, men i andra aspekter var det kunskap om irrelevanta verktyg som lärdes ut. Den kunskap som lärdes ut under backend-workshopen visade sig vara irrelevant för de flesta i utvecklingsteamet.

En möjlig anledning till varför, var att workshoppen hölls tidigt i arbetet då de tekniska verktygen ännu inte hade bestämts. Vidare var en central lärdom från denna workshop vikten av tydliga instruktioner och en avgränsad uppgiftsomfattning för att underlätta inlärning. Som tidigare nämnts i Avsnitt 5.5.1.2 tillämpades dessa insikter i workshoppen inom frontendutveckling och resulterade i en ökad kompetens som sedan underlättade implementationsarbetet av TIDIG. Det kan även nämnas att den andra workshoppen i Svelte hade fördelen av att redan vara ett bestämt verktyg. En lärdom är alltså att det är fördelaktigt att avvakta med utbildning innan tekniska verktyg är etablerade för att på så sätt minska risken för tidsåtgång åt irrelevanta saker.

Ytterligare ett sätt att dela kompetens inom projektgruppen var genom kodgranskning via pull requests i Azure DevOps. Enligt Bacchelli [28] har kodgranskning visat sig vara ett bra verktyg för att öka kunskapsdelning inom ett mjukvaruprojekt. Detta stämmer överens med vad projektgruppen upplevt under utvecklingen av TIDIG.

6.1.4.3 Kommunikation

Under projektarbetet har det framkommit att god kommunikation i teamet har varit viktigt för att lyckas med projektarbetet och leverera en bra produkt. Missförstånd

och dubbelarbete uppkom som ett resultat av bristande rutiner när det gällde att informera om vad man jobbar på. Så som redan tagits upp i Avsnitt 5.5.1.4 resulterade distansarbetet och separata arbetsgrupper till minskad kontakt mellan projektmedlemmarna vilket kan ha bidragit till problemet.

Resultatet på fråga 2 i Tabell 5 visar hur gruppen har upplevt kommunikationen under varje sprint. Kommunikationen har uppfattats som bristfällig, och trots att flera åtgärder har vidtagits för att stärka kommunikationen mellan medlemmarna, visar resultatet att dessa insatser endast haft begränsad effekt. De vidtagna åtgärderna har syftat till att etablera nya kommunikationskanaler och att prioritera möten mellan de separata arbetsgrupperna, i syfte att främja en gemensam förståelse för den pågående produktutvecklingen. Dessa åtgärder har konkretiserats i form av rutiner såsom att informera projektmedlemmar vid påbörjandet av ny funktionalitet samt att konsekvent använda Kanban-brädet för ökad transparens och samordning.

Enligt Ken Schwaber [29], medskaparen av Scrum, är det vanligt att missförstånd och dubbelarbete uppstår i stora team som arbetar med Scrum. Enligt Schwaber innebär stora team 7 personer eller fler. Det innebär att det som teamet har upplevt som ett problem är vanligt och kan vara förväntat i stora projekt, och det finns åtgärder att tillämpa för att förhindra att det sker igen. I de fall som dubbelarbete uppstod kommunicerades problemet mellan medlemmarna och rutiner infördes för att minska risken att det sker igen. En lärdom som följer är alltså att rutiner gällande kommunikation bidrar till projektframgång.

Gällande den bristande kommunikationen med kunden är en orsak till detta teamets ringa erfarenhet av att utveckla en produkt åt extern beställare. Detta projektarbete har givit lärdom angående hur man hanterar en extern kund. Lärdomar från detta har varit att försöka arbeta sig runt problemet med kundfrånvaro genom att fortsätta utveckling på de delar av systemet som inte kräver kundens svar och att söka alternativa sätt att kommunicera, vilket beskrivs i Avsnitt 5.5.1.4. Ytterligare en lärdom har varit att ta initiativ till uppföljande mejl om kunden inte återkopplar inom rimlig tid.

6.1.4.4 Scrum: Kombination av andra arbetsmetoder

Att ha en process som arbetsfokus och sedan kontinuerligt anpassa den till gruppen har varit en lärorik erfarenhet. Scrum är den arbetsprocess som låg i huvudfokus för gruppen och Kanban fanns till som ett extra tillägg, se Avsnitt 5.3.1. Trots detta följde gruppen inte en klassisk agil Scrum-metod med flera iterativa moment. Istället använde gruppen sig av sprintar som pågick under faser och hade en specialanpassad produktbacklogg som bestod av milstolparna. Orsaken till att fasplaneringen integrerades med Scrum var att flera dokument, däribland designdokumentet, skulle lämnas in tidigt i projektprocessen. Därav hade gruppen svårt att till en början etablera ett iterativt arbetssätt då kursen krävde en design- och planeringsfas av studenterna redan i början på kursen. I det fortsatta arbetet hanterades detta genom att faserna innehöll sprintar, där arbetet bedrev iterativt inom respektive fasområde. Exempelvis reviderades och omarbetades koden successivt under imple-

mentationsfasen. En intressant lärdom kan därför vara att det är möjligt att kombinera den agila arbetsmetoden Scrum med linjär fasplanering i ett grupprojekt.

Erfarenheten av att integrera Kanban i Scrum har varit en intressant erfarenhet för gruppen. Det finns många fördelar när Kanban används på rätt sätt, medan den för med sig nackdelar när det används på fel sätt. Exempelvis ger Kanban en god översikt över sprintuppgifterna och deras status. Detta gör det enkelt att se vem man ska höra av sig till vid frågor och funderingar. De negativa aspekterna som projektgruppen har upplevt med Kanban är osäkerhet, ofärdiga uppgifter och dubbelarbete. De gånger som projektmedlemmar inte hade tilldelats ansvar för uppgifter under en sprint, hade dessa uppgifter dessutom en lägre chans att bli avklarade under sprinten. Å andra sidan förekom det situationer där medlemmar åtagit sig uppgifter utan att uppdatera Kanban-brädets status utefter uppgifternas status. Detta kunde resultera i dubbelarbete och missförstånd kring huruvida uppgifter var slutförda eller inte, samt andra liknande problem. Lärdomen som följer är att ett Kanban-bräde endast fungerar effektivt om det underhålls kontinuerligt, vilket förutsätter ett aktivt deltagande från teamet och att varje uppgift tilldelas en ansvarig person.

6.1.4.5 Scrum: Formulering av uppgifter

Sprintuppgifter var något som uppkom varje sprint retrospektiv och utvecklades för varje sprint. Under sprintar som varade under två veckor eller mer upplevdes det som extra utmanande att formulera tydliga, relevanta uppgifter för hela sprinten och samtidigt hålla en rimlig arbetsbelastning. Gruppen fick erfarenhet av att tydliga, realistiska och mätbara uppgifter hade större sannolikhet att bli färdiga under sprinten. Denna lärdom stämmer väl överens med konceptet SMART:a mål [30], vilket handlar om att skapa mål som är specifika (S), mätbara (M), accepterade (A), realistiska (R) och tidssatta (T). Enligt Bahrami *et al* [30] har dessa typer av mål större chans att avklaras än andra. En intressant aspekt är att projektmedlemmarna i början av varje sprint uppfattade uppgifterna som tydliga i den mån att de kunde uppfattas som SMART:a. Men under sprintens genomförande framkom att flera uppgifter i praktiken saknade tillräcklig konkretisering och vissa var för tidskonsumerande i relation till sprintens längd. Denna erfarenhet tydliggör vikten av att gå igenom uppgifter och kritiskt granska och bedöma deras tydlighet, omfattning och genomförbarhet i relation till gruppens resurser och sprintens varaktighet. En annan sak som möjligtvis bidrog till att vissa sprintuppgifter inte blev avklarade var att de under fyra sprintar saknade tydlig koppling till milstolparna och kraven.

Under projektets gång ökade antalet sprintuppgifter markant, se Figur 1, vilket ledde till ett omfattande arbete för produktägaren. Detta kan ha bidragit till att uppgifterna uppfattades som stora och generella av projektgruppen. Detta resulterade i att projektgruppen hade svårt att slutföra uppgifterna och även att fördela dem till specifika personer. Därmed beslutades det att dela på produktägaransvaret mellan frontendutvecklingsledare, backendutvecklingsledare och dokumentansvarig. Detta resulterade i att uppgifterna som presenterades vid sprintplaneringsmötet blev mer specifika och hanterbara. Lärdomen från detta var att uppdelning av produktägare resulterade i detaljerade uppgifter som var fokuserade på respektive

utvecklingsområde. Det innebar också att mindre arbetsbelastning per produktägare. En möjlig nackdel med fördelningen var att detta resulterade i att ingen enskild person i projektgruppen hade fullständig insikt i alla detaljer kring sprintens uppgifter. Detta ställde höga krav på projektgruppens kommunikation för att säkerställa ett samordnat arbete.

6.1.4.6 Scrum: Utveckling av en process

Slutligen är det viktigt att lyfta den iterativa och processförbättrande delen av Scrum och projektgruppens arbetsprocess. Sprint retrospektiv-möten har varit fundamentalt för processförbättringar under projektets gång. De har hjälpt gruppen att inse vad som fungerat bra och vad som fungerat mindre bra under sprintarna. Detta har sedan lett till att gruppen kommit på förbättringsförslag som sedan applicerats på nästkommande sprint. Följaktligen är anpassning av Scrum-arbetsprocessen efter gruppens behov och önskningar viktigt för att stärka gruppens produktivitet och möjliggöra projektframgång.

Vidare kan liknande lärdomar dras gällande det generella arbetssättet som kandidatgruppen använde. Gruppen använde en modifierad stil av Scrum som involverade både Kanban-bräde och fasplanering och alla dessa modifierades under projektets gång. Detta ger då en viktig insikt vilket är att arbetsprocesser behöver anpassas till gruppen och de förutsättningar projektet har.

6.1.5 Tekniska erfarenheter

Genom kunskapsdelningen av de mer erfarna fick resten av gruppen en större förståelse för de tekniska detaljerna. Gällande databasdesign och modellering finns några aspekter att ta upp. En lärdom som gruppen har fått är att en databasdesign och modellering av datatyper underlättar implementationsarbetet. Det underlättar planering och ger projektmedlemmarna en förståelse för vad de ska implementera. Det ska även tilläggas att design och modeller kontinuerligt bör följas upp och revideras under projektets gång. Detta möjliggör anpassning till förändrade behov under utvecklingen och bidrar till att de förblir ändamålsenliga verktyg i arbetet. Ytterligare en lärdom är att ändringar av datastrukturen under implementeringen av ett system bör kommuniceras. Syftet är att uppnå samförstånd mellan projektmedlemmarna samt att säkerställa att de nya ändringarna inte bryter mot designen.

Det har också visat sig vara viktigt att arbetsgrupperna som arbetar med frontend och backend tidigt samordnar kring vilken data som förväntas överföras mellan servrarna. Syftet med detta är att skapa en modellstruktur för backend som sedan kan efterföljas under implementeringen utan att större ändringar behöver göras i ett senare skede.

Om projektet skulle genomföras igen skulle ett närmare samarbete mellan frontend- och backendgruppen, exempelvis genom parprogrammering, kunnat minska antal missförstånd och behovet av omarbetningar i backend-koden. Trots att detta skulle innebära en högre inlärningströskel för att sätta sig in i både backend och frontend, skulle det sannolikt leda till en effektivare utvecklingsprocess och färre tekniska hinder.

Det finns många fördelar fördelar kring frontendramverket Svelte som tas upp under Avsnitt 6.1.3, men branchstandard är React. Av denna anledning är det möjligt att frontend hade varit skriven i React om projektet gjordes på nytt.

6.1.6 Systemanatomins upplevda värde

Enkäten om systemanatomins värde, vars resultat redovisas i Tabell 6, visade att frågorna hade medelvärden mellan 1 och 2.7 med måttlig till låg standardavvikelse. Det indikerar att majoriteten av projektmedlemmarna ansåg att systemanatomin inte var ett stöd under projektets genomförande.

Svaren om faktisk användning särskiljer sig då samtliga medlemmar svarade att de inte använde systemanatomin regelbundet. Samtidigt indikerade svaren på fråga 1, som hade ett medelvärde på 2.7, att systemanatomin underlättade den övergripande förståelse av systemet. Den höga standardavvikelsen, som låg på 1.2, är en indikation på att det fanns delade åsikter om det stöd systemanatomin var för gruppen men utesluter inte möjligheten att den hade potential att vara tillämpbar vilket kan tyda på att den inte nyttjades korrekt.

Den dåliga uppföljningen av systemanatomin kan ha berott på att det initialt inte fanns alternativ till att använda en systemanatomi, då det var ett obligatoriskt moment under ett tidigt skede i kursen. Det kan ha medfört att arbetet med systemanatomin inte vidareutvecklades efter inlämning då det inte fanns krav på fortsatt uppföljning av den. Taxén och Olow menade att systemanatomin fungerar som en lämplig modell främst i början av projektet, vilket överensstämde med resultaten från enkäten [31].

Systemanatomin låg till grund för flera designbeslut i slutet på planeringsfasen. Figma användes för att planera och illustrera användarflödet för klienten, vars design togs fram med hjälp av systemanatomin som definierat webbapplikationens vyer. EER-diagram och API-ändpunktsdiagram användes för att strukturera databasens entiteter, relationer, attribut och hierarkier samt för att specificera vilka typer av indata som resulterade i en viss utdata från API:n. Även dessa verktyg utgick ifrån systemanatomin när den ursprungliga designen skapades.

6.1.7 Uppdelningen frontend-backend

Som nämndes i Avsnitt 5.8.3 gjorde uppdelningen i två arbetsgrupper det möjligt att jobba parallellt och fokuserat på sitt område. Här diskuteras några upplevda för- och nackdelar med uppdelningen, samt möjliga orsaker till dessa.

Inledningsvis var den inlärning av programmeringsspråk och verktyg som krävdes för en enskild gruppmedlem lägre, vilket bidrog till att skynda på implementeringsfasens uppstart. Men även under arbetets gång fanns det fördelar med att ha gjort en tydlig gruppuppdelning. Eftersom inte alla nio medlemmar arbetade med både frontend och backend, var det färre personer som arbetade i samma kodbas, vilket troligen minskade risken för Git-konflikter vid sammanslagning av olika versioner av samma fil.

En nackdel med uppdelningen var dock den låga förståelsen för funktionaliteten och de implementationer som andra utvecklingsgruppen hade skapat. De som inte arbetade med frontend-koden hade svårare att förstå och hantera felmeddelanden

vid testning av fullstack-implementeringar. Samtidigt upplevde de som enbart arbetade med frontend ofta problem med att få backend att fungera, särskilt när databasfilen inte var uppdaterad eller när gamla eller felaktiga migrationsfiler användes. Ett sätt att minska problematiken med att inte förstå varandras kod hade varit att säkerställa nära och kontinuerlig kommunikation mellan frontend- och backend-grupperna.

Långsiktigt innebar den tydliga uppdelningen att vissa projektmedlemmar bara fick erfarenhet inom ena sidan av projektet. Beroende på framtida karriär och intressen kan detta ha påverkat projektmedlemmarna negativt. Exempelvis genom att man inte fick någon erfarenhet av frontendutveckling under sin studietid men hade behövt det senare i karriären.

Ett alternativt arbetssätt hade kunnat vara att till en början dela upp gruppen i frontend och backend men senare dela in på olika funktioner i systemet istället. Det hade gjort att kunskapsdelningen hade kunnat öka och att alla hade fått en bättre förståelse för hela systemet.

Tidredovisningssystemet TIDIG var en relativt liten och okomplicerad webbapplikation. Att dela arkitekturen i en frontend- respektive backend-server kan därför ha fört med sig mer krångel än nytta. Arkitekturens kvalitet påverkas av dess modifierbarhet, alltså hur komplicerat det är att förändra systemet, och dess pålitlighet, som mäts genom hur många procent av en viss tidsperiod som systemet är i drift [32].

En uppdelad arkitektur kan göra det svårare för underhållaren av systemet att utföra en ändring eftersom det kan vara oklart var i systemet ändringen ska ske. Samma princip gäller när ett problem uppstår. Att systemet är uppdelat i frontend- och backend-servrar kan därför öka tidskostnaden för ändringar och förlänga tiden servern ligger nere vid problem. Fördelen med att ha systemet uppdelat är framförallt att det blir mer skalbart. Exempelvis hade en extra frontend-server kunnat sättas i drift och ta emot anrop utan att en extra backend-server nödvändigtvis behövts, men med tanke på systemets förväntade mängd användare kommer ingen skalning förmodligen behöva ske. Därför skulle en enda server kunna vara ett bättre alternativ för ett liknande projekt.

Ett exempel på problematik som hade kunnat uppstå i ett alternativt implementationssätt, med en öppen backend-server är att en illvillig användare skulle kunna komma åt data som det inte är tänkt att de ska ha tillgång till. Detta hade användaren förhållandevis enkelt kunnat göra med *endpoint discovery*. Endpoint discovery är att kartlägga ändpunkterna på ett API genom att exempelvis skicka ett stort antal förfrågningar med gissningar på ändpunkter [33].

Vidare bidrog uppdelningen mellan frontend och backend till en tydligare struktur i projektet, särskilt vad gäller ansvarsfördelning och var i systemet olika delar av logiken skulle implementeras. Om all utveckling hade skett inom Svelte hade det varit svårare att särskilja vad som exempelvis hörde till databasen. Den tydliga avgränsningen mellan gränssnitt och serverlogik gjorde det enklare att samarbeta över gruppgränser och minskade risken för otydlig kod. Detta stöds av slutsatserna i en studie som utvärderade samspelet mellan en uppdelad frontend och backend, där Drupal användes för backend och ReactJS för frontend [34]. Studien visar att dessa

tekniker kompletterade varandra väl, där Drupal stod för datalagring och auktorisering medan ReactJS erbjöd ett användargränssnitt. Även om slutsatserna inte generaliseras till alla projekt, visar studien att en tydlig separation mellan frontend och backend kan vara fördelaktig för både utvecklingsflöde när kommunikationen mellan dem fungerar väl.

6.2 Metod

I avsnittet diskuteras de metoder som definierades i Kapitel 4. Fokus ligger på att granska metodernas styrkor och svagheter samt vilka risker som kan ha påverkat resultatet. Vid skrivandet av rapporten har källor valts ut noggrant, vilket beskrivs även beskrivs i detta avsnitt.

6.2.1 Kundkontakt

Kontinuerlig kontakt med kunden samt användning av en kravspecifikation har varit viktigt för att säkerställa att produkten skapar värde för kunden. Genom regelbundna möten och avstämningar kunde fel och missförstånd upptäckas tidigt och åtgärdas. En risk med denna metod är att kravspecifikationen främst speglar kundens uttryckta behov och inte nödvändigtvis de faktiska behoven som finns hos slutanvändarna. Eftersom kunden önskade ett system med samma funktionalitet som det befintliga, men med förbättrad användarvänlighet, hade det varit värdefullt att få tillgång till det nuvarande systemet för att skapa en tydlig förståelse för dess funktion. Då detta inte var möjligt har det stundtals varit en utmaning för projektgruppen att få en klar bild av hur det nuvarande systemet faktiskt fungerar. Ytterligare en utmaning har varit att kunden har varit mycket upptagen och svår att nå vilket resulterat i att projektgruppen behövt anpassa implementationen av produkten utefter detta.

6.2.2 Användarundersökningar

Användarundersökningar och SUS-mätningar har använts för att mäta användarvänlighet och säkerställa att TIDIG är lätt att använda. Metoderna bidrar till att skapa värde genom att fånga upp användarnas faktiska upplevelser. En utmaning är dock att svar från användare kan vara subjektiva och är inte alltid representativa för alla användare. En begränsning i projektet är att användarundersökningarna inte genomfördes av produktens slutanvändare. Utvecklingen av produkten skedde i stort sett utan kontinuerlig återkoppling från slutanvändarna. Detta var ett medvetet val med tanke på att det är tidskrävande att kontinuerligt demonstrera produkten för kunden och anpassa den baserat på feedback. Samtidigt hade ett närmare arbete med slutanvändarna kunnat ge djupare insikter i deras behov och därmed skapa större värde för kunden.

Ett högre SUS-poäng på användartesterna under den andra sprinten kan sannolikt förklaras av att projektgruppen redan visste hur produkten fungerade. Det lägre resultatet under den femte sprinten kan förklaras av att testpersonerna inte var insatta i projektspecifika termer och förstod exempelvis inte vad det innebar att vara en administratör. De interna användarundersökningar som genomfördes före utvecklingsfasen fyllde syftet att tidigt identifiera problem och oklarheter i designen. Valet att utföra testerna internt i projektgruppen effektiviserade processen. Å andra

sidan hade ett bredare perspektiv kunnat uppnås genom att inkludera externa testpersoner med exempelvis större åldersvariation. Samtidigt innebar de interna testerna fördelar såsom att samtliga projektmedlemmar, både från frontend- och backendteamet, blev delaktiga i applikationens utformning och design.

6.2.3 Design av applikationen

Kunden uttryckte ingen önskan om produkten skulle vara mobilanpassad. Samtidigt omfattar tillgänglighetskrav anpassning för olika skärmstorlekar, och kunden önskade att produkten skulle vara enkel att använda. Det är fortfarande oklart hur mycket värde mobilvyn skapar för kunden eftersom det först kan utvärderas efter en tids användning av produkten. Däremot är det klart att mobilanpassningen inte har någon negativ påverkan på produktens värde och användarvänlighet.

6.2.4 Scrum och samarbete med kunden

I en klassisk version av Scrum fokuserar man generellt på att leverera en fungerande produkt i slutet av varje sprint, och det är vanligt att hålla regelbundna demonstrationer för kunden [18]. Projektgruppen har inte helt utgått från detta i sin arbetsprocess. Projektgruppen har däremot arbetat för att alltid ha fungerande kod på main-branchen, men det har inte alltid genomförts tester för att säkerställa dess funktionalitet. Fördelen med att leverera en fungerande produkt i slutet av varje sprint hade varit möjligheten att genomföra fler användbarhetstester på produkten och ge kunden möjlighet att delta i en demonstration. Att involvera kunden mer i form av regelbundna demonstrationer hade varit värdefullt med tanke på att produktens största fokus låg i dess användbarhet. Dessutom hade kontinuerlig feedback från kunden kunnat ge ökat värde, då de hade blivit mer involverade i riktningen av produkten. Samtidigt innebär detta en risk för förseningar, med tanke på att kunden periodvis har varit svår att få kontakt med. Sådana förseningar skulle i sin tur kunna påverka produktens kvalitet, med tanke på den begränsade tidsbudgeten.

I en studie som genomfördes i syfte att undersöka kundens påverkan på mjukvaruutvecklingsprocessen observerades flera verkliga mjukvaruprojekt [35]. Resultaten visade att ett aktivt kunddeltagande har en positiv inverkan på projektets kvalitet. Samtidigt överskred samtliga observerade projekt sin budget eller tidplan, delvis på grund av att kunden vid flera tillfällen hade otillräckliga resurser, vilket skapade flaskhalsar och ledde till att viktig kommunikation uteblev. Baserat på denna studie är det oklart om den alternativa metoden skulle ha skapat ett ökat kundvärde, eftersom detta förutsätter att kunden har tillräckliga resurser för att delta aktivt. Utan sådana resurser finns en risk att utvecklingsarbetet försenas.

6.2.5 Erfarenhetsinsamling

Användningen av Scrum har upplevts som ett effektivt sätt att ständigt förbättra arbetsprocesser. Efter retrospektiven har projektgruppen kunnat identifiera vad som fungerade bra och vad som behövde förbättras, genom den processdokumentation som utförts.

Utvärderingar i slutet av sprints har använts för att samla erfarenheter av Scrum. Detta har gett en överblick över projektgruppens upplevelser och identifierat

förbättringsområden. Dock finns risken att en utvärdering kan bli för ytlig och inte ge tillräckligt med insikter för att göra meningsfulla förändringar.

Att använda en erfarenhetsrapport i slutet av projektet har varit en bra metod för att sammanfatta och dokumentera lärdomar. Detta har bidragit till att skapa en gemensam kunskapsbas för framtida projekt. En risk med att skriva rapporten i slutet av projektet var att kunskap kunde förloras om den inte dokumenteras kontinuerligt under projektets gång. En potentiell idé för att motverka detta är att varje gruppmedlem medvetet dokumenterar sina erfarenheter under projektets gång.

6.2.6 Skapande av systemanatomi

Processen för att utveckla systemanatomin, där flera versioner togs fram av olika arbetsgrupper, bidrog till att fånga upp olika perspektiv och förbättra kvaliteten. En risk med denna metod kan dock ha varit att designbeslut som hela gruppen kunde ha diskuterat missades, vilket kan ha lett till att vissa delar av designen blev mindre optimal. Möjligen skulle designen blivit mer komplett om varje gruppmedlem var tvungen att uppvisa en förståelse för alla designbeslut, hur små de än kan ha varit.

Feedback från handledare och examinator har spelat en stor *roll* för att säkerställa att systemanatomins struktur och innehåll var tydliga och relevanta. Det har gjort det enklare att upptäcka problem tidigt i processen och åtgärda dem.

6.2.7 Uppföljning av systemanatomi

Systemanatomin låg till grund för flera designbeslut i slutet på planeringsfasen. Dessa beslut introducerade andra designverktyg, som diskuteras i Avsnitt 6.1.6. Vidare byggdes det varken vidare på systemanatomin samt att den inte användes som referens under implementationsfasen. Detta kan ha lett till att arbetet blev mer oorganiserat och sämre kommunikation mellan medlemmar. Erik Schumann [21, s. 107-114] menar att medlemmar i team som använder systemanatomi har större möjlighet att jobba självständigt eftersom varje medlem har en tydligare bild av vad som behövs göras. Projektmedlemmarna kunde jobba mycket självständig men eftersom systemanatomin inte följdes upp, eller var i fokus i arbetsprocessen, är det svårt att påvisa att systemanatomin bidrog till det.

En anledning till att projektmedlemmarna kunde jobba självständigt kan ha varit att de nya designverktygen valdes som ett alternativt arbetssätt, då de fortsatt följdes upp och uppdaterades vid ändringar i planeringen. De upplevdes mer djupgående och gav ett bättre stöd för projektgruppen under utvecklingen. Projektmedlemmarna kunde få en bättre uppfattning av hur projektet var strukturerat och även få en övergripande uppfattning över hur kommunikationen mellan de olika anatomerna skulle ske, inte bara om vilka anatomerna som skulle kommunicera med varandra.

6.2.8 Källkritik

Källorna i rapporten har noggrant valts ut för att säkerställa en hög kvalitet och relevans. Vetenskapliga artiklar har använts i stor utsträckning för att stödja påståenden och argument, men eftersom projektet använder en stor mängd olika verktyg och teknologier som inte alltid har vetenskapliga artiklar kopplade till sig, har även

icke-vetenskapliga källor använts. I sådana fall har källor närmast verktyget eller teknologin valts, som referensdokumentationer eller manualer skrivna av projektmedlemmarna själva. Dessa källor har granskats kritiskt för att säkerställa att de är pålitliga och relevanta för projektet.

6.3 Arbetet i vidare sammanhang

Projektet har flera viktiga samhälleliga och etiska aspekter. Under Avsnitt 5.4 presenterades gruppens bedömning över systemets hållbarhet. I detta avsnitt utreds mer noggrant vad TIDIG har för påverkan på världen i flera aspekter.

6.3.1 Etiska aspekter

Fysiska sätt att mäta arbetstimmar, exempelvis med stämpelklockor, säkerställer att anställda faktiskt är på arbetsplatsen när de uppger ha varit det. Eftersom TIDIG är en webbapplikation, kan tid redovisas varifrån som helst. På detta vis är en potentiell negativ etisk aspekt med systemet att det möjliggör för oärliga anställda att redovisa mer tid än de faktiskt har arbetat. Å andra sidan kan anställda känna att företaget inte litar på dem om ett mer övervakande system används. I denna mening är det en positiv etisk aspekt att TIDIG ger mycket frihet åt tidredovisare.

En annan viktig samhällelig aspekt är tillgänglighetskraven EN 301 549 [36] och WCAG 2.1 [3]. Genom att säkerställa att TIDIG uppfyller dessa krav bidrar projektet till ökad digital inkludering och jämlikhet på arbetsplatsen. Som en del av LiU:s digitala infrastruktur är det viktigt att TIDIG är tillgängligt för alla användare, oavsett deras tekniska kunskaper eller funktionsvariationer.

6.3.2 Strukturella aspekter

I och med att TIDIG driftsätts på en server på Linköpings universitet behålls kontrollen av den lagrade datan. Det kan i allmänhet vara fördelaktigt att göra sig så oberoende av externa faktorer som möjligt. Molnplattformer bygger ofta på virtualisering av flera servrar på samma fysiska maskin, vilket öppnar för dataläckor mellan olika tjänster [37]. Av den anledningen är en lokal driftsättning av systemet troligtvis det säkraste alternativet ur DIGIT:s perspektiv.

6.3.3 Individuella aspekter

Systemet TIDIG har aspekter som skapar möjligheter för individen som rapporterar sin arbetstid. En sådan kan exempelvis vara att arbetsbelastningen minskar med ett enklare rapporteringssystem. Den snabba inloggningen och att det är få knapptryck som krävs för att rapportera tid, gör att redovisning av tid kräver mindre tid. En annan aspekt som framför allt påverkar nya användare är att med det här nya systemet är inlärningskurvan som krävs för att förstå TIDIG och använda det, förhållandevis låg jämfört med föregångaren. Arbetsbelastningen för administratören har potential att bli mycket mindre med det här systemet. Att få alla tidsrapporter i ett modernt system som sammanställer informationen automatiskt, kan göra att administratören slipper göra en stor mängd onödigt arbete.

6.3.4 Sociala aspekter

Ett bra system för tidredovisning kan ge effekten att tilliten mellan både tidredovisare och administratörer ökar. Ett system som är tillräckligt användarvänligt för att man ska vilja använda det kan föra med sig effekten att man som användare blir noggrannare med rapporteringen. Att användare är noggrannare kan förbättra Digitaliseringsavdelningens anseende på lång sikt.

6.3.5 Tekniska aspekter

TIDIG autentiserar användare genom LiU:s Microsoft SSO. Detta är positivt dels för att användare slipper skapa fler konton, och dels för att Microsoft SSO är ett beprövat och säkert system. TIDIG är inte byggt med oändlig skalning i åtanke, och har inte heller testats i större storleksordningar än 100 samtidigt aktiva användare. I denna aspekt är TIDIG dåligt förberedd på plötslig uppskalning.

6.3.6 Ekonomiska aspekter

I och med att TIDIG har hög användarvänlighet, möjliggör det för kunden att spara arbetstimmar för att det krävs mindre administration med ett smidigt och intuitivt system. Ytterligare en positiv ekonomisk effekt som TIDIG användbarhet möjliggör är att det är svårt att råka rapportera fel. Den sista positiva ekonomiska effekt som TIDIG kan ha för kunden är att det möjliggör bättre översikt över fördelning av arbetad tid, vilket möjliggör utvärdering och effektivisering.

6.3.7 Miljömässiga aspekter

TIDIG har konstruerats för att vara plattformsoberoende och kan därmed driftsättas både på lokala och externa servrar. Eftersom kunden väljer att driftsätta TIDIG lokalt undviks användningen av stora utländska datorhallar, som annars kan ha negativa konsekvenser för miljön. Stora datacenter drar mycket elektricitet och kräver stor volym vatten [38]. Om TIDIG driftsätts lokalt, undviks dessa negativa konsekvenser för miljön, och blir därmed ett miljömedvetet alternativ.

7 Slutsatser

7.1 Hur kan tidredovisningssystemet TIDIG implementeras så att man skapar värde för kunden?

För att skapa värde för kunden analyserades behoven genom kravframkallande möten, användarformulär och en kravspecifikation. För att möta kundens behov utvecklades produkten med fokus på användbarhet, anpassades för olika skärmstorlekar och användbarhetstestades. Ramverket Svelte användes för att effektivisera utvecklingen och frigöra tid till att fokusera på användarupplevelsen.

Ett strukturerat arbetssätt med välskriven dokumentation och visualisering av arbetsflödet med Kanban möjliggjorde ett effektivt arbete inom projektgruppen. I kombination med en kontinuerlig anpassning av arbetsmetoden Scrum frigjordes mer tid för produktutveckling och ökade därmed värdet för kunden. Mätningar av LCP, kodtäckning och användbarhet har skapat kundvärde genom att säkerställa att produkten håller hög kvalitet.

Slutligen, genom effektiva arbetsprocesser, bra samarbete, god uppfattning av kundens behov och bra val av ramverk kan TIDIG implementeras i enlighet med kundens vision och därmed skapa en värdefull produkt.

7.2 Vilka erfarenheter kan dokumenteras från TIDIG som kan vara intressanta för framtida projekt?

Erfarenheterna under projektet har resulterat i flera lärdomar som kan vara intressanta för framtida projekt. Att etablera rutiner som främjar kommunikationen under ett projekt är viktigt för att stödja gruppens samarbete. Det är också viktigt att arbeta runt problem och hitta alternativa kommunikationsvägar vid kundfrånvaro. Vidare underlättar tydliga instruktioner inlärning. Projektplaneringen visade behovet av en tidsbuffert och flexibilitet för att hantera oförutsägbara situationer. Inom Scrum var det värdefullt att utforma sprintuppgifter efter SMART:a mål och att kontinuerligt anpassa Scrum efter gruppens behov. Avslutningsvis kan det konstateras att datamodellering bör utformas i samverkan mellan backend- och frontend-arbetsgrupperna för att främja en effektiv och sammanhängande implementation.

7.3 Vilket stöd kan man få genom att skapa och följa upp en systemanatomi?

Att inleda ett projekt med att skapa en systemanatomi gav värdefulla insikter och låg till grund för arbetet att designa TIDIG. Systemanatomin visade sig även vara till hjälp när ett översiktligt blockschema och arkitekturen för TIDIG skulle tas fram.

Gruppens upplevelse av systemanatomin som verktyg är att det inte var ett stöd för utvecklingen, men att det kan bero på att systemanatomin som skapades inte följdes upp. Det kan även bero på att andra designverktyg användes som upplevdes ge mer stöd för utvecklingen.

7.4 Vilka för- och nackdelar finns med att dela upp en webbapplikation som TIDIG i en frontend- respektive backend-server med separata arbetsgrupper?

Uppdelningen i två arbetsgrupper hade både för- och nackdelar. De främsta fördelarna var att det möjliggjorde ökat parallellt arbete och minskade kodkonflikter samt att utbildningstiden minskade eftersom alla inte behövde lära sig samtliga teknologier.

De främsta nackdelarna var att viss funktionalitet tog onödigt lång tid att implementera, framförallt eftersom det ofta uppstod problem där frontend-gruppen behövde en ändpunkt från backend-gruppen för att testa funktionalitet. Samtidigt var det svårt för backend-gruppen att skapa ändpunkterna i förväg eftersom frontend-gruppen inte visste exakt vilka de skulle behöva och hur de skulle fungera.

8 Referenser

[1] Google, ”Largest Contentful Paint (LCP)”, *Google*. Tillgänglig vid: https://web.dev/articles/lcp. [Åtkomstdatum: 21 januari 2025]

[2] ”EN 301 549: Accessibility requirements for ICT products and services”, European Telecommunications Standards Institute (ETSI), 2021. Tillgänglig vid: https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf

[3] ”Web Content Accessibility Guidelines (WCAG) 2.2”, World Wide Web Consortium (W3C), dec. 2024. Tillgänglig vid: https://www.w3.org/TR/2024/REC-WCAG22-20241212/

[4] Linköpings universitet, ”Digitaliseringsavdelningen (DIGIT)”, *Linköpings universitet*. Tillgänglig vid: https://liu.se/organisation/liu/uf/digit. [Åtkomstdatum: 25 februari 2024]

[5] Mozilla, ”HTML: HyperText Markup Language”, *Mozilla*. Tillgänglig vid: https://developer.mozilla.org/en-US/docs/Web/HTML. [Åtkomstdatum: 07 mars 2025]

[6] Mozilla, ”CSS styling basics”, *Mozilla*. Tillgänglig vid: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics. [Åtkomstdatum: 07 mars 2025]

[7] Tailwind CSS, ”Rapidly build modern websites without ever leaving your HTML.”, *Tailwind CSS*. Tillgänglig vid: https://tailwindcss.com/. [Åtkomstdatum: 07 mars 2025]

[8] DaisyUI, ”The most popular component library for Tailwind CSS.”, *DaisyUI*. Tillgänglig vid: https://daisyui.com/. [Åtkomstdatum: 07 mars 2025]

[9] ECMA International, ”ECMAScript® 2023 Language Specification, ECMA-262, 14th edition, June 2023”, *ECMA International*. Tillgänglig vid: https://tc39.es/ecma262/. [Åtkomstdatum: 23 maj 2025]

[10] TypeScript, ”TypeScript is JavaScript with syntax for types.”, *TypeScript*. Tillgänglig vid: https://www.typescriptlang.org/. [Åtkomstdatum: 07 mars 2025]

[11] S. Bhardwaz och R. Godha, ”Svelte.js: The Most Loved Framework Today”, vol. 0, nr , s. 1–7, 2023, doi: 10.1109/INOCON57975.2023.10101104

[12] Microsoft, ”C# language documentation”, *Microsoft*. Tillgänglig vid: https://learn.microsoft.com/en-us/dotnet/csharp/. [Åtkomstdatum: 10 februari 2025]

[13] Microsoft, ”ASP.NET Core”, *Microsoft*. Tillgänglig vid: https://dotnet.microsoft.com/en-us/apps/aspnet. [Åtkomstdatum: 10 februari 2025]

[14] Microsoft, ”Entity Framework Core”, 11 december 2024, *Microsoft*. Tillgänglig vid: https://learn.microsoft.com/en-us/ef/core/. [Åtkomstdatum: 26 februari 2025]

[15] T. Gustavsson, *Agil projektledning*, 5:e uppl. Stockholm, Sverige: Sanoma Utbildning, 2024.

[16] E. S. Andersen, *Systemutveckling: principer, metoder och tekniker*. Lund, Sverige: Studentlitteratur, 1994.

[17] K. S. och Mike Beedle, *Agile software development with scrum*. Upper saddle river, NJ, USA: Prentice Hall, 2002.

[18] B. B. Frank Tsui Orlando Karam, *Essentials of Software Engineering*, Third edition. Jones, Bartlett Learning, 2014, s. 345.

[19] N. Thomas, ”How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website”, Tillgänglig vid: https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/. [Åtkomstdatum: 14 februari 2025]

[20] Suso Academy, ”Sustainability Awareness Framework (SusAF)”, *Suso Academy*. Tillgänglig vid: https://www.suso.academy/en/sustainability-awareness-framework-susaf/. [Åtkomstdatum: 04 maj 2025]

[21] L. Taxén, *The System Anatomy – Enabling Agile Project Management*. 2011, s. 14–16.

[22] Scrum.org, ”The Kanban Guide for Scrum Teams”, 2018, *Scrum.org*. Tillgänglig vid: https://www.qagile.pl/wp-content/uploads/2018/11/2018-Kanban-Guide-for-Scrum-Teams.pdf

[23] Microsoft, ”Vad är Azure DevOps?”, 25 mars 2024, *Microsoft*. Tillgänglig vid: https://learn.microsoft.com/sv-se/azure/devops/user-guide/what-is-azure-devops?view=azure-devops. [Åtkomstdatum: 11 september 2024]

[24] Figma, ”What is Figma”, *Figma*. Tillgänglig vid: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma. [Åtkomstdatum: 07 mars 2025]

[25] Microsoft, ”Your code editor. Redefined with AI.”, *Microsoft*. Tillgänglig vid: https://code.visualstudio.com/. [Åtkomstdatum: 07 mars 2025]

[26] Google, ”lighthouse-ci”, *Google*. Tillgänglig vid: https://github.com/GoogleChrome/lighthouse-ci. [Åtkomstdatum: 17 februari 2025]

[27] Google, ”Web Vitals”, *Google*. Tillgänglig vid: https://web.dev/articles/vitals. [Åtkomstdatum: 20 februari 2025]

[28] A. Bacchelli och C. Bird, ”Expectations, Outcomes, and Challenges of Modern Code Review”, i *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 2013, s. 712–721. doi: 10.1109/ICSE.2013.6606617

[29] K. Schwaber, *Agile Project Management with Scrum*. Redmond, WA, USA: Microsoft Press, 2004, s. 118.

[30] Z. Bahrami, A. Heidari, och J. Cranney, ”Applying SMART Goal Intervention Leads to Greater Goal Attainment, Need Satisfaction and Positive Affect”, *International Journal of Mental Health Promotion*, vol. 24, nr 6, s. 869–882, 2022, doi: 10.32604/ijmhp.2022.018954

[31] L. Taxén och Peter Olow, ”On the Integration of Project Planning, System Anatomy, and System Architecture”, 2013, Tillgänglig vid: https://www.diva-portal.org/smash/get/diva2:297018/FULLTEXT02.pdf

[32] Z. Qin, X. Zheng, och J. Xing, *Software Architecture*. Springer Berlin, Heidelberg, 2008, s. 222–224. doi: 10.1007/978-3-540-74343-9

[33] A. Hoffman, *Web Application Security*. O'Reilly Media, Sebastopol, 2024, s. 79–82.

[34] L. Lundqvist, ”Drupal and ReactJS: An Evaluation of Decoupled Drupal and ReactJS”, Umeå University, 2023. Tillgänglig vid: https://www.diva-portal.org/smash/get/diva2:1816129/FULLTEXT01.pdf

[35] E. Vanhala och J. Kasurinen, ”The Role of the Customer in an Agile Project: A Multi-case Study”, 2019, s. 208–222. doi: 10.1007/978-3-030-33742-1_17

[36] Myndigheten för digital förvaltning, ”Det här är EN 301 549 och WCAG”, 02 juli 2024, *Myndigheten för digital förvaltning*. Tillgänglig vid: https://www.digg.se/webbriktlinjer/lagar-och-krav/det-har-ar-en-301-549-och-wcag. [Åtkomstdatum: 02 mars 2025]

[37] P. Samarati och S. De Capitani di Vimercati, *Cloud Security*. 2016. doi: 10.1002/9781118821930.ch17

[38] M. A. B. Siddik, A. Shehabi, och L. Marston, ”The environmental footprint of data centers in the United States”, *Environmental Research Letters*, vol. 16, nr 6, s. 64017, maj 2021, doi: 10.1088/1748-9326/abfba1