Testplan

Version 2.0 PUM02 2025-04-14

Projektidentitet

Medlemmar

Namn	Ansvarsområde	Mail
Alice Almgren	Teamledare	alial202@student.liu.se
Anton Taber	Testledare	antta671@student.liu.se
Axel Berg	Arkitekt	axebe390@student.liu.se
Isabel Neubauer	Dokumentansvarig	isane541@student.liu.se
Jakob Tormalm	Analysansvarig	jakto054@student.liu.se
Samuel Tuvstedt	Backendutvecklingsledare	samtu593@student.liu.se
Samuel Åkesson	Konfigurationsansvarig	samak519@student.liu.se
Simon Gunnarsson	Frontendutvecklingsledare	simgu061@student.liu.se
Stina Åström	Kvalitetssamordnare	stias606@student.liu.se

Kund:
Digitaliseringsavdelningen, Linköpings universitet

Handledare:
Eric Ekström

Examinator / kursansvarig:
Lena Buffoni

Dokumenthistorik

Version	Datum	Ändringar	Granskad av	Utfärdat av
2.0	2025-04-14	Ändring efter intern feedback	S.Åk, I.N	A.T, A.B
1.1	2025-03-30	Åtgärdat extern feedback	A.A	A.T, S.Åk
1.0	2025-03-11	Åtgärdat intern feedback	J.T	A.T
0.3	2025-03-06	Åtgärdat intern feedback	A.A	A.T
0.2	2025-03-04	Åtgärdat intern feedback	A.A, S.Ås	A.T, S.Åk
0.1	2025-02-21	Skapats	A.A, A.B, I.N, S.T, S.Åk, S.Ås	A.T

Innehåll

1. Referenser 5

2. Introduktion 6
2.1. Bakgrund 6
2.2. Syfte 6
2.3. Mål 6
2.4. Målgrupp 6
2.5. Definitioner, akronymer och förkortningar 6

3. Testobjekt 7

4. Testomfattning 8
4.1. Funktioner som ska testas 8
4.2. Funktioner som inte ska testas 8

5. Teststrategi 9
5.1. Testnivåer 9
5.2. Testverktyg 9

6. Riskanalys 12
6.1. Risker 12
6.2. Riskhantering 12

7. Testkriterier 13
7.1. Godkännandekriterier 13
7.2. Överlämningskriterier 13

8. Testartefakter 14

9. Miljöbehov 15

10. Roller och ansvar 16
10.1. Testroller 16
10.2. Ansvar 16

1. Referenser

[1] A. Almgren *m.fl.*, ”Projektplan”, technical report, jan. 2025.

[2] A. Almgren *m.fl.*, ”Kravspecifikation”, technical report, feb. 2025.

[3] A. Almgren *m.fl.*, ”Begreppslista”, technical report, jan. 2025.

[4] Svelte, ”Svelte Introduction”, 2024, *Svelte*. Tillgänglig vid: https://svelte.dev/docs/svelte/overview. [Åtkomstdatum: 26 februari 2025]

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

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

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

[8] K. Sandahl, ”Project Management”, 24 september 2024. Tillgänglig vid: https://www.ida.liu.se/~TDDC88/theory/10project-management.pdf. [Åtkomstdatum: 16 februari 2025]

[9] ”Azure Pipelines”. Tillgänglig vid: https://azure.microsoft.com/en-us/products/devops/pipelines. [Åtkomstdatum: 14 april 2025]

[10] 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]

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

[12] ”xUnit”. Tillgänglig vid: https://xunit.net/. [Åtkomstdatum: 14 april 2025]

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

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

[15] typescript-eslint, ”typescript-eslint”, *typescript-eslint*. Tillgänglig vid: https://typescript-eslint.io/getting-started. [Åtkomstdatum: 17 februari 2025]

[16] ”svelte-check”. Tillgänglig vid: https://svelte.dev/docs/cli/sv-check. [Åtkomstdatum: 14 april 2025]

[17] ”Dotnet”. Tillgänglig vid: https://dotnet.microsoft.com/. [Åtkomstdatum: 14 april 2025]

[18] ”Node.js”. Tillgänglig vid: https://nodejs.org/. [Åtkomstdatum: 14 april 2025]

2. Introduktion

Testplanen beskriver de strategier och metoder för testning som ska utföras på den beställda produkten och dess tillhörande system. Dokumentet kommer uppdateras kontinuerligt under projektets gång i takt med ändringar av krav, planering och projektspecifika beslut.

2.1. Bakgrund

Målet med projektet är att skapa en produkt åt Digitaliseringsavdelningen (DIGIT) på Linköpings Universitet. De har beställt ett tidsredovisningssystem för sina anställda. Mer information om projektet hittas i *projektplanen* [1].

2.2. Syfte

Syftet med testplanen är att ge projektgruppen ett underlag för hur testerna ska utföras, vad som ska testas och vem som har ansvar för de olika testerna. Syftet med testerna är att säkerställa produktens funktionalitet och användarbehov enligt *kravspecifikationen* [2].

2.3. Mål

Målet med testerna är att kunna lämna över en produkt till beställaren som följer de krav som definierats i *kravspecifikationen* [2].

2.4. Målgrupp

Testplanen riktar sig till utvecklare, testare, testledare och andra intressenter som är involverade i testningsprocessen.

2.5. Definitioner, akronymer och förkortningar

Förklaringar till de förkortningar, definitioner och akronymer som används i det här dokumentet finns i projektets *begreppslista* [3].

3. Testobjekt

Följande är en lista över systemkomponenter och dokument som ska testas i detta projekt:

A) Tidsredovisningssystemets frontend, Version 1.0

Utvecklat i Svelte 5 [4] med TypeScript [5]. Funktioner som ska testas inkluderar:
- Inloggning och autentisering
- Redovisning av arbetstid
- Skapa och redigera uppgifter
- Import och export av .csv-filer med redovisad tid

B) Tidsredovisningssystemets backend, Version 1.0

Utvecklat i ASP.NET [6] med C# [7]. Backend ansvarar för datalagring och logik.
Funktioner som ska testas inkluderar:
- Att autentiseringen fungerar
- Att varje ändpunkt tar emot och skickar den data som förväntas
- Sortering av uppgiftslistor
- Integration med LiU-API

C) Användartester, Version 1.0

Scenariobaserade tester utifrån följande användarhistorier [8]:
- Som en användare vill jag kunna:
  - logga in och identifiera mig för att få tillgång till systemet och relevanta funktioner för min roll.
  - exportera en rapport som .csv-fil för att spara ner och arbeta med lokalt.
  - använda systemet på både en dator och en mobil för att inte begränsa min användning av systemet när jag ska använda det.
- Som en tidsredovisare vill jag kunna:
  - se, lägga till och redigera tider som jag arbetat på en uppgift.
  - exportera en mall, som jag kan fylla i mina arbetstider i för att externt kunna jobba med min tidsredovisning, och sedan importera den till systemet för att redovisa tider.
  - söka och filtrera bland uppgifter samt favorisera de uppgifter som är mest relevanta för mig för tillfället.
  - se statistik över hur många timmar som jag jobbat och hur mycket flextid jag har för att veta hur jag ska planera mitt arbete.
- Som en administratör vill jag kunna:
  - skapa och redigera uppgifter samt tilldela dem till tidsredovisare.
  - skapa rapporter över hur tidsredovisare jobbat och hur mycket tid som redovisats till en uppgift.
- Som en utvecklare vill jag kunna:
  - läsa koden och förstå hur den är strukturerad för att enkelt bygga ut det befintliga systemet.

D) Projektets *kravspecifikation* [2], Version 1.1

Kravspecifikationen specificerar samtliga funktionella, icke-funktionella och kvalitetskrav och står till grund för att verifiera att funktionerna uppfylls.

4. Testomfattning

Testomfattningen beskriver de funktioner i systemet som ska eller inte ska testas.

4.1. Funktioner som ska testas

A) Inloggning och autentisering

B) Redovisning av arbetstider

C) Importering av arbetstider från .csv-fil

D) Redigering av redovisade arbetstider

E) Visning av redovisade arbetstider

F) Skapande av uppgifter

G) Redigering av uppgifter

H) Sökning och filtrering av uppgifter

I) Favorisering av uppgifter

J) Skapande av rapporter

K) Exportering av rapporter till .csv-fil

L) Sökning och filtrering av rapporter

M) Statistikgenerering

N) Responsivitet och tillgänglighet

O) Felhantering

4.2. Funktioner som inte ska testas

A) Användargränssnittets kompatibilitet med äldre versioner av webbläsare innan år 2020

5. Teststrategi

Testerna kommer utföras med två tillvägagångssätt under utvecklingen. Det ena tillvägagångssättet är genom manuella tester, script eller testfiler som exekveras i *Azure Pipeline* [9]. Testfilerna är lokaliserade i samma mapp som den fil som ska testas. Det andra tillvägagångssättet är genom enkäter och användningstester.

5.1. Testnivåer

De olika *testnivåerna* [8] som testningen ska fokusera på är:

- **Enhetstester**

Syftet med enhetstesterna är att testa enskilda funktioner i koden. Målet är att backendkoden ska ha en kodtäckning på 95%, enligt *kravspecifikationen* [2]. Enhetstester skrivs för varje backendfunktion och placeras i tillhörande testfil. Testfallen utgår ifrån en funktion som implementerats i koden och ska träffa alla delar av koden för att testerna ska ha så stor kodtäckning som möjligt. På denna testnivå utförs även kodkvalitetstester för att säkerställa att koden är läsbar och följer de kodkonventioner och formatering som bestämts.

- **Integrationstester**

Syftet med integrationstesterna är att säkerställa att de fåtal komponenter i backend som direkt interagerar med varandra, fungerar tillsammans. Integrationstester ska skrivas där backendfunktioner anropar varandra.

- **Systemtester**

Systemtesterna är manuella tester som utförs av testarna för att säkerställa att alla systemfunktioner är testade. Testarna utför uppgifter baserade på de användarscenarion som tagits fram i *kravspecifikationen* [2]. Efter att testerna är utförda ska testarna svara på en *SUS-enkät* [10] som ger ett underlag för hur systemet var att använda. Målet är att ha ett snitt på minst 68 i SUS-betyg. På denna testnivå utförs även prestandatester för att säkerställa systemets respons och stabilitet under beslastning. Målet är att LCP [11] ska maximalt vara 2,5 sekunder i minst 95 % av fallen.

- **Acceptanstester**

Syftet med acceptanstesterna är att kontrollera att systemet fungerar enligt användarnas behov. De utförs av kunden vid sluttestningen innan överlämningen av produkten för att säkerställa att *kravspecifikationen* [2] är uppfylld.

5.2. Testverktyg

- **xUnit:** För att testa backend används ett externt testbibliotek *xUnit* [12]. Figur 1 visar hur xUnit ska användas i en testfil för att testa en funktion. Genom att köra funktionen och jämföra det med ett förväntat resultat så kan xUnit avgöra om funktionen fungerar som den ska.

```
// Sum.cs
public class Fun
{
    public static int Sum(int a, int b)
    {
        return a + b;
    }
}

// SumTest.cs
using Xunit;

public class SumTests
{
    [Fact]
    public void Add_1_And_2_ShouldEqual_3()
    {
        // Arrange
        int varA = 1;
        int varB = 2;

        // Act
        int res = Sum.Sum(varA, varB);

        // Assert
        Assert.Equal(3, res);
    }
}
```

Figur 1: Mall för enhetstester i C# med xUnit

- **Lighthouse:** För att mäta systemets prestanda används Lighthouse, vilket är ett sätt att automatiskt samla in så kallade *Web Vitals* [13] på en webbapplikation. Mer specifikt används verktyget lighthouse-ci [14] så att det körs vid varje pushad commit i en *Azure Pipeline* [9]. Den genomför då mätningen tre gånger, sammanställer ett medelvärde och laddar till sist upp mätdatan för den aktuella commit:en. På så vis kan alla mätvärden spåras över tid.
- För att testa kodens kvalitet används följande verktyg:
  - **ESLint:** För att hålla koden ren och konsekvent används *ESLint* [15]. Det är ett verktyg framtaget för TypeScript som varnar för buggar och fel i kodstil.
  - **svelte-check:** Likt ESLint så utför *svelte-check* [16] en kontroll så att det inte sker kompileringsfel, att alla variabler används och att sveltekomponenter används korrekt.

  ESLint och svelte-check körs som en *Azure Pipeline* [9] men kan även utföras av utvecklarna i samband med att man slår ihop två utvecklingsgrenar i Azure DevOps. För att genomföra detta manuellt skrivs de kommandon som syns i Figur 2.

```
npm run format
npm run check
npm run lint
```

Figur 2: Manuellt test av kodkvalitet i frontend

- **SUS-enkät:** Efter testerna är klara svarar testpersonerna på en *SUS-enkät* [10] som innehåller följande frågor:
  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 tyckte att det var för mycket inkonsekvens i produkten.
  6. Jag kan tänka mig att de flesta skulle lära sig att använda denna produkt mycket snabbt.
  7. Jag tyckte att denna produkt var mycket besvärlig att använda.
  8. Jag kände mig väldigt trygg när jag använde denna produkt.
  9. Jag behövde lära mig mycket innan jag kunde komma igång med denna produkt.

  Varje test är viktat med Ekvation 1 och resulterar i ett betyg på skalan 1-100.

$$\text{SUS-betyg} = 2.5 \cdot (20 + \text{summan av udda frågor} - \text{summan av jämna frågor}) \quad (1)$$

6. Riskanalys

6.1. Risker

Följande är en lista över de risker som kan påverka resultatet eller omfattningen av de tester som utförs:

A) Försenad integration mellan backend och frontend kan leda till att testningen skjuts upp och enbart görs på en temporär databas kopplad direkt till frontendservern. Dessa tester riskerar att göras i onödan om viss logik som skrivs på frontendservern ska migreras till backendservern.

B) Försenad integration med LiU:s API kan leda till att fiktiv data används för långt in i projektets implementation och resultera i att koden förlitar sig för mycket på dess struktur.

C) Tidsbrist kan leda till att inte tillräckligt många tester utförs och att projektet då inte håller *kravspecifikationens* [2] kvalitetskrav.

D) Testerna har inte en tillräckligt stor kodtäckning.

E) Ändrade krav från kunden kan påverka planeringen och riskerar att tester behöver skrivas om.

6.2. Riskhantering

A) Tydlig kommunikation mellan utvecklarna är viktig för att backend ska ha samma ändpunkter som frontend förväntar sig. På så sätt behöver inte testkoden för klienten skrivas om.

B) Att fokusera på den implementering där utvecklarna inte är begränsade är att prioritera för att kunna börja testa tidigt.

C) Tidsbristen kan undvikas genom att testmoment ingår i varje sprint. I sprintplaneringen ska det förtydligas vad som ska testas, vilka som ska testa och när testningen ska utföras så att det inte blir någon felkommunikation.

D) Genom att köra *Azure Pipeline* [9] genereras en sammanfattning om hur mycket av koden som blev körd.

E) Ändrade krav från kunden kan påverka planeringen och riskerar att tester behöver skrivas om.

7. Testkriterier

Innan projektet är avslutat så måste godkännande- och överlämningskriterier vara uppfyllda för att säkerställa att systemet är färdigt att använda och uppfyller alla ställda krav.

7.1. Godkännandekriterier

För att sluttestningen ska kunna godkännas måste alla testnivåer ha genomförts enligt testplanen. Detta innebär att alla testfall ska ha körts och alla kritiska buggar och fel som identifierats under tester ska ha åtgärdats.

7.2. Överlämningskriterier

För att produkten ska kunna överlämnas måste följande kriterier vara uppfyllda:

- Tester ska ha utförts som bekräftar att produkten uppfyller de krav ifrån *kravspecifikationen* [2] som har prioritet 1.
- Beställaren måste ha godkänt acceptanstesterna och att systemet uppfyller deras behov.
- Systemet ska vara redo för driftsättning.

8. Testartefakter

- Tester rapporteras löpande i en testrapport. Testrapporten innehåller en beskrivning av aktiviteterna, resultatet, de olika avvikelserna i testet och en utvärdering av varje aktivitet. Under projektgruppsmöten används testrapporten för att planera testerna för nästa sprint.
- Resultatet av svaren från *SUS-enkäten* [10] som fylls i efter systemtesterna sparas ner i ett kalkylark och används till grund för utveckling av designen.

9. Miljöbehov

Följande element krävs för att genomföra testerna:

- En dator med en webbläsare som stöds enligt krav NFR7 i *kravspecifikationen* [2].
- *.NET SDK* [17] måste vara installerat för att köra backendtesterna.
- *Node.js* [18] måste vara installerat för att köra frontendtesterna.
- *Lighthouse* [14] måste vara installerat för att köra prestandatesterna.

10. Roller och ansvar

Projektgruppen har olika roller och ansvarsområden. Kapitlet beskriver rollerna och vilket ansvar som medföljer varje roll.

10.1. Testroller

- **Testledare (TL):** Planerar, koordinerar testning och säkerställer testutförande.
- **Utvecklare (U):** Projektmedlem som utvecklar kod i projektet.
- **Testare (T):** Utvecklare inom projektgruppen som testar annan utvecklares kod.
- **Slutanvändare (Kund):** Beställaren av produkten.

10.2. Ansvar

De olika rollerna utför tester inriktade på olika testnivåer där de även är ansvariga för att testerna utförs. Ansvarsområdena för varje roll kan läsas av i Tabell 1.

Tabell 1: Ansvarsområden

Ansvar	TL	U	T	Kund
Enhetstester		X	X	
Integrationstester		X	X	
Systemtester			X	
Acceptanstester			X	X
Rapportering	X		X	
Dokumentation	X		X	
Godkännande av godkännandekriterier	X			X
Godkännande av avslutskriterier	X			X