

Din Digitala Bovärd

Självständigt arbete

Markus Vickman

Examinator: Lars Lundin, Lars.Lundin@miun.se

Handledare: Mikael Hasselmalm, Mikael.Hasselmalm@miun.se

Författare: Markus Vickman, mavi2302@student.miun.se

Utbildningsprogram: TWEUG, Webbutveckling, 120 hp

Huvudområde: Datateknik

Termin, år: VT, 2025

Sammanfattning

Målet med projektet har varit att skapa en webbapplikation för bovärdar. Först planerades projektet efter MOSCOW-metoden för att skapa en minsta produkt som skapar värde. Sedan skapades designskisser och ER-diagram för produkten som fick namnet "Din Digitala Bovärd". Applikationen använder en uppdelad lösning där backend skapades i PHP-ramverket Drupal. Backend innehåller en blandning av Drupals inbyggda funktioner, tilläggsmoduler samt egenskapat PHP-kod. Den grafiska gränssnittet skapades i Vue och programmerades med TypeScript samt stylades med Tailwind. Vue-applikationen använder en REST API-lösning för att ansluta till Drupal. Metoden för säkerhet är OAuth2. Resultatet blev en webbapplikation där bovärdar kan registrera konton, lägga till egna fastigheter samt bostäder, manuell och automatisk fakturering via e-post med PDF-fakturor och ett system för att hantera felanmälningar från boende. Boende kan även logga in för att se aktuell information från bovärden samt skicka in en felanmälan. Integritet är viktigt och därför lägger applikationen stor vikt vid att personuppgifter hanteras varsamt och följer GDPR. Den är även tillgänglighetsanpassad för att fler ska få åtkomst till tjänsten. Applikationen innehåller en tillgänglighetsredogörelse och följer i hög grad DOS-lagens riktlinjer.

Nyckelord: PHP, Drupal, Vue, Tailwind, WCAG, Tillgänglighet, GDPR, REST API.

Abstract

The projects goal was to create a web application for real estate managers. The project followed the MOSCOW method to produce a minimum viable product. Designs and ER diagrams were created for the produkt which was named "Din Digitala Bovärd". The application's frontend was created in Vue and styled with Tailwind. The frontend is decoupled from the backend which was create in the PHP framework Drupal. Vue connects to Drupal with REST API and the security method OAuth2. To create the Drupal backend a combination of Drupal's main functionality, extra modules and programming in PHP was used. The result was a web application where real estate managers can list there properties. They can send both manual and automatic email invoices included with PDF invoices. The application can also handle fault reports from residents and post information for residents. The application carefully manages personal information and respects users integrity. It complies with GDPR and lets users know which data is collected. Accessibility is important and therefor the application also includes an accessibility statement and have made customizations to follow the accessibility guidelines dictated by the DOS-lagen.

Nyckelord: PHP, Drupal, Vue, Tailwind, WCAG, Accessibility, GDPR, REST API.

Innehållsförteckning

Sammanfattning.....	3
Abstract.....	4
Terminologi.....	7
1 Introduktion.....	1
1.1 Bakgrund och problemmotivering.....	2
1.2 Övergripande syfte.....	2
1.3 Avgränsningar.....	3
1.4 Detaljerad problemformulering.....	3
1.5 Översikt.....	4
1.6 Författarens bidrag.....	4
2 Teori.....	5
2.1 Definition av termer och förkortningar.....	5
3 Metod.....	8
3.1 Utvecklingsmiljö.....	8
3.2 Backend.....	8
3.3 Databas.....	9
3.4 Användargränssnitt.....	9
3.5 Metodlista.....	9
4 Konstruktion.....	10
4.1 Grafisk design i Figma.....	10
4.2 Er-diagram i DrawIO.....	10
4.3 Backend i Drupal.....	13
4.3.1 Installera moduler.....	14
4.3.2 Datatyper.....	16
4.3.3 Skapa anpassad modul.....	16
4.3.4 Inställningar.....	23
4.4 Frontend i Vue.....	23
4.4.1 Vyer.....	24
4.4.2 Komponenter.....	26
4.4.3 Typer.....	28
4.4.4 Services.....	28
4.5 Testning av webbplatsen.....	28
4.5.1 HTML-validering.....	29
4.5.2 Tester för tillgänglighet.....	29

4.5.3	Testning med Lighthouse.....	30
4.6	Publicera applikationens.....	32
4.6.1	Publicera Drupal.....	32
4.6.2	Publicera Vue.....	33
5	Resultat.....	34
6	Diskussion.....	36
6.1	Analys och resultatdiskussion.....	36
6.2	Projektmetod diskussion.....	36
6.3	Etisk och social diskussion.....	37
	Källförteckning.....	38
	Bilaga A: Milstolpe-plan.....	41
	Bilaga B: Första design.....	42
	Bilaga C: Slutgiltiga designskisser.....	43
	Bilaga D: ER-diagram.....	44
	Bilaga E: Hämta fastighetsdata.....	45
	Bilaga F: Lägg till fastighet.....	46
	Bilaga G: Ta bort fastighet.....	47
	Bilaga H: Uppdatera fastighet.....	48
	Bilaga I: Hämta relaterad data.....	49
	Bilaga J: E-postbyggare.....	50
	Bilaga K: Inloggningsservice.....	51
	Bilaga L: Ace it.....	52
	Bilaga M: CRON.....	53
	Bilaga N: Din Digitala Bovärd.....	54
	Bilaga O: Fastighetshantering.....	55
	Bilaga P: Fakturahantering.....	56

Terminologi

Förkortningar

MoSCoW	Must have, Should have, Could have and Won't have (this time).
MVP	Minimum Viable Product.
SCB	Statistiska centralbyrån även kallat Statistikmyndigheten SCB.
CMS	Innehållshanteringssystem
DOS-lagen	Lagen om digital offentlig service
WCAG	Web Content Accessibility Guidelines
API	Application programming interface
REST	Representational State Transfer
SQL	Structured Query Language

1 Introduktion

Projektet är en del av ett examensarbete inom datateknik hos Mittuniversitetet och syftar till att utveckla en webbapplikation för bovärdar av hyres- och bostadsrätter. Projektet utfördes hos företaget Websystem. I applikationen ska bovärdar kunna lista alla sina objekt till exempel hus eller lägenhetshus som innehåller lägenheter. Bovärden ska kunna lägga till namn och e-post adresser till boende som är kopplade till lägenheterna. För varje tillsatt bostad ska hyran faktureras ut vid önskat datum via mejl till hyresgästerna. Projektkrav listas nedan enligt *MOSCOW-metoden*. MOSCOW-metoden är ett sätt att tydliggöra för både projektledare och beställare vilka krav en produkt har utifrån ett tydligt prioriteringssystem[1].

Must have

- **Lista fastigheter:** I applikationen ska bovärdar kunna lista alla sina objekt ex. hus eller lägenhetshus med alla innehållandes lägenheter.
- **Fakturering:** Bovärden ska kunna lägga till e-post till boende som är kopplade till lägenheterna. För varje tillsatt bostad ska hyran faktureras ut vid önskat datum via mejl till hyresgästerna.

Should have

- **Bovärdar kan lägga upp information till boende:** Gemensam information som angår flera boende exempelvis ett helt hyreshus kan publiceras offentligt för boende att tillgå.

Could have

- **Felanmälningar:** Felanmälningar behöver kunna hanteras för att underlätta prioritering och effektivisera arbetet för bovärden.

- **Inloggning för boende:** För att de boende ska kunna se status på sina felanmälningar kan konton skapas även för dem.
- **Meddelandesystem mellan bovärd och boende:** Ett meddelande system mellan bovärd och boende kan vara ett tidseffektivt sätt för bovärdar att svara på frågor.

Won't have

- **Underhållsplanering:** Fastigheter har behov av planerade underhåll. En funktion för underhållsplanering ger applikationen mervärde.

Avstämningar kommer regelbundet att ske för att säkerställa att projektplanen följs. Projektet följer även *Minimum viable product*(MVP) och syftar till att producera den minsta produkt som skapar värde[2]. Denna process möjliggör ett flexibelt arbetssätt och säkerställer att projektet kan anpassas efter eventuella förändringar eller nya krav.

1.1 Bakgrund och problemmotivering

Enligt statistik från SCB så bor cirka 50% av Sveriges hushåll i flerbostadshus[3]. Av dessa bor en större del i hyresrätter. Gemensamt för hyresgästföreningar och bostadsrätter är att en avgift betalas in av dem som bor i lägenheterna. Båda boendeformerna behöver hantera underhåll och felanmälningar även om det sker i olika utsträckning. Det finns ett stort antal potentiella kunder som ett digitalt bovärdssystem skulle kunna underlätta för. Därför skapas det här projektet som ett enkelt digitalt bovärdssystem för mindre bovärdar.

1.2 Övergripande syfte

Syftet är att skapa ett digitalt system för att underlätta för mindre bovärdar. Systemet ska spara tid för bovärdar samt ge större överblick över deras fastigheter. Det ska även automatisera fakturering och hantera felanmälningar. Med intuitiv och responsiv design kan applikationen bli ett helhetssystem för bovärdar.

1.3 Avgränsningar

Projektet har en tidsbegränsning på 10 veckor. Dessa veckor inkluderar både det praktiska projektet samt rapportskrivning med projektplanering. Sista veckan i projektet är även avsatt för projektpresentationer. Det kommer att krävas att tid avsätts till att sätta mig in i systemutveckling med *Drupal*. Drupal är ett PHP-baserat ramverk och *innehållshanteringssystem*(CMS)[4]. För att projektet ska leda till en användbar produkt ska agil systemutveckling användas på ett sådant sätt att en minsta värdes produkt publiceras. Denna produkt ska i mån av tid få extra funktionalitet enligt en MOSCOW-prioritering.

1.4 Detaljerad problemformulering

Ett digitalt system ska skapas för att underlätta för mindre bovärdar. Systemet ska spara tid för bovärdar samt ge större överblick över dennes fastigheter. Det ska även automatisera fakturering för bovärdar. En intuitiv, responsiv design som följer *tekniska riktlinjer för webbtillgänglighet 2.1(WCAG)* på AA nivå är ett krav. Följande problem behöver lösas:

DOS-lagen

Applikationen ska vara tillgänglig för bovärdar och boende oavsett behov. Därför ska den följa *lagen om digital offentlig service*(DOS-lagen). DOS-lagen är en lag som styr hur innehåll ska struktureras i applikationer från offentliga aktörer[5].

Kontohantering

Bovärdar ska kunna skapa egna konton. Dessa konton ska ge åtkomst till applikationens funktioner. Det ska även gå att återställa lösenord via mejl.

Lista fastigheter och fakturera boende

Det kan vara ett betungande arbete för hyresvärdar att manuellt sammanställa och skicka ut fakturor. Därför ska applikationen kunna skicka ut e-post-fakturor till alla boende.

Extra funktioner i prioriteringsordning

1. Bovärdar kan lägga upp information till boende.

2. Felanmälningar kan hanteras för att underlätta prioritering och effektivisera arbetet för bovärdar.
3. Ett meddelandesystem mellan bovärd och boende kan vara ett tidseffektivt sätt för bovärdar att svara på frågor.
4. För att de boende ska kunna se status på sina felanmälningar kan konton skapas även för dem.
5. En funktion för underhållsplanering ger applikationen mer värde.

1.5 Översikt

I kapitel 2 förklaras teori om olika tekniker och arbetssätt för projektet. Kapitel 3 innefattar projektets tänkta metoder och planerad arbetsgång. Arbetets utförande beskrivs detaljerat i kapitel 4. I kapitel 5 redogörs för projektets resultat och måluppföljning. Rapporten avslutas i kapitel 6 med en diskussion av projektets genomförande och resultat samt dess etiska aspekter.

1.6 Författarens bidrag

Projektet har till stor del varit självständigt. Val av metoder och tekniker för att lösa projektet gjordes i samråd med Joakim som var min handledare samt frontend och Drupal-utvecklare hos företaget Websystem. Fortlöpande har jag även fått viss rådgivning och genomgång av Drupal.

2 Teori

2.1 Definition av termer och förkortningar

Minimum viable product

Minimum viable product förkortas MVP. Produkten kan i vissa fall vara en prototyp eller en produkt utan substans. Men oftast syftar det till att producera en minsta produkt som skapar värde. Detta snabbar på utvecklingen och gör det tydligare för hur produkten kan utvecklas i framtiden.[2]

Drupal

Drupal är ett öppen källkodsramverk som bygger på PHP-ramverket Symfony. Drupal kan även användas som ett CMS likt WordPress. Det går också att använda Drupal som en komplett backend-lösning. Det finns stöd för att använda färdiga moduler samt programmera egna. Drupal kopplas till en MySQL/MariaDB-databas vid installation.[4]

DOS-lagen

DOS-lagen står för "lagen om digital offentlig service". Det är en lag som säger att innehåll på webbplatser och i applikationer från offentliga aktörer och från offentligt finansierade privata aktörer ska vara tillgänglighetsanpassat. Lagen innefattar att dessa ska ha en tillgänglighetsredogörelse för applikationen samt att de ska följa EU-standarden (EN 301 549 V3.2.1). Men eftersom att det redan finns en vedertagen standard inom webbutveckling med samma innehåll vid namn Web Content Accessibility Guidelines 2.1 så uppfyller den också lagkraven.[5]

Symfony

Symfony är ett öppen källkodsramverk för webbapplikationer. I Symfony kan färdiga moduler användas. Andra ramverk som Laravel, Drupal och Prestashop bygger på Symfony.[6]

Cron

Cron är en tidsbaserad schemaläggare som kan användas för Unix-liknande system som olika Linux varianter eller macOS. Ett cron-

jobb är de uppgifter som automatiseras vid olika tidsintervaller med hjälp av Cron.[7]

RESTful Web Services API

RESTful Web Services API är en modul som ger Drupal ett utökat gränssnitt för att hantera RESTful-APIer. Den gör så genom att använda klassen *RestResource*. *RestResource* är den funktionalitet för RESTful API som är integrerat i Drupal. [8]

Entity API

Entity API är Drupals medföljande funktioner för att hantera objekt av olika datatyper. En vanlig datatyp för att hantera innehåll är noder. Noder fungerar som en SQL-tabell även om de i realiteten består av en tabell per datarad. Specifik data kan hämtas från databasen med *entity query*. [9]

Hooks

Hooks i Drupal är ett sätt att interagera med underliggande funktionalitet eller funktionalitet i moduler. Ett exempel är när en funktion implementerar cron. Då deklarerar cron direkt i funktionsnamnet genom att avsluta med "_cron" t.ex. "function custom_invoice_cron() {}". [10]

Drush

Drush är troligen det vanligaste verktyget för Drupal-utveckling. Drush har funktionalitet som innefattar bland annat generering av kod och export av Drupal installationen. [11]

Composer

Composer är en pakethanterare för PHP som även kan hantera paketens beroenden av andra paket. Composer fungerar väldigt likt *npm*, *yarn* och *bundler* som är pakethanterare för andra programmeringsspråk. [12]

Tailwind CSS

Tailwind CSS är ett CSS-ramverk där CSS skrivs direkt som klasser i HTML-koden. Dessa klasser är ofta en direkt referens till en CSS-egenskap. Det gör att utvecklaren får full frihet i hur stylingen ska se ut utan att använda färdig styling. [13]

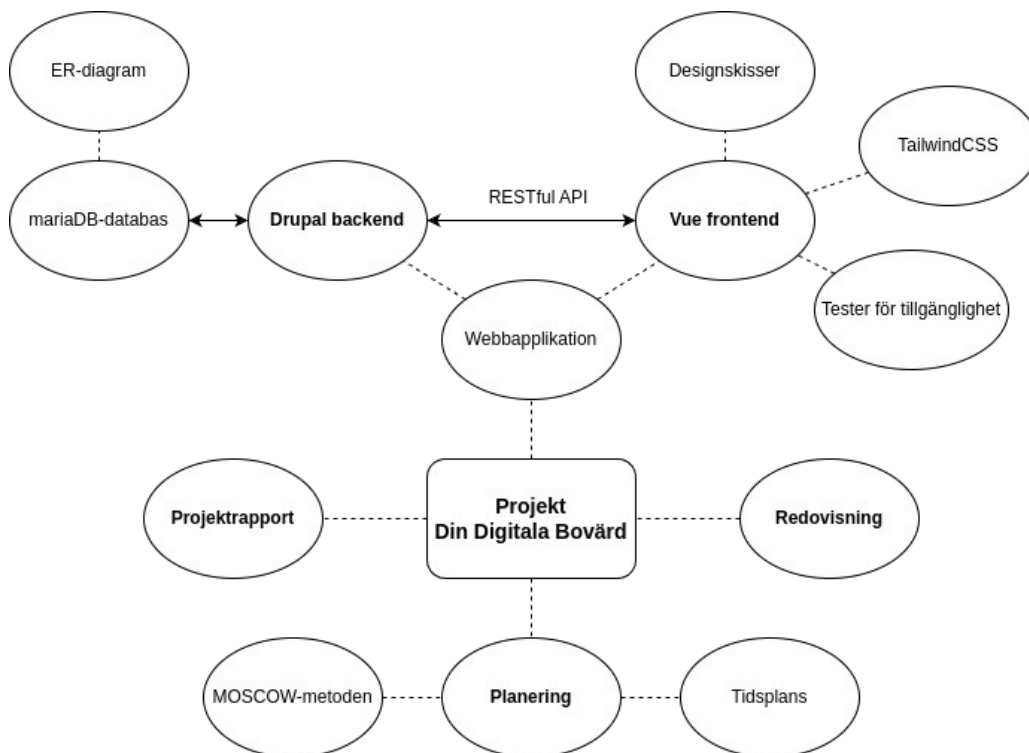
VUE

Vue.js är ett JavaScript-ramverk för att skapa användargränssnitt till webbapplikationer. För att skapa en applikation med Vue används HTML, CSS och JavaScript alternativt TypeScript. Vue kan utvecklas som exempelvis en *Single-Page Applikation(SPA)* där webbsidor inte behöver laddas om för att visa nytt innehåll eller med server rendering där applikationen renderas på servern istället för på klienten. Vue komponenter ligger i vyer eller ibland i andra komponenter. De kallas då att de är en barnkomponent till en förälderkomponent. En barn-komponent kan ta in data från sin förälder med *props* och de kan skicka tillbaka ett svar eller data till en metod hos föräldern med *emits*. [14]

3 Metod

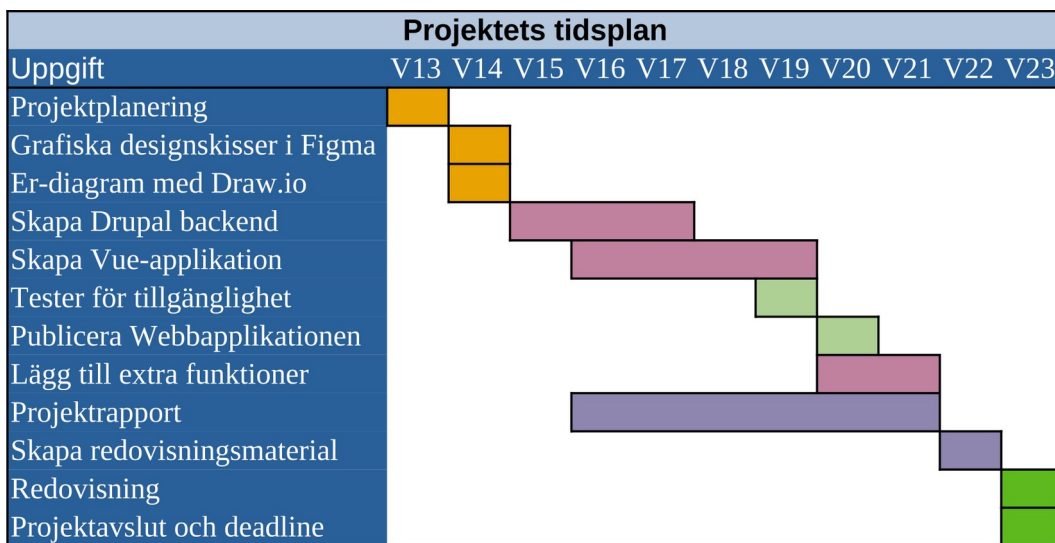
Projektet följer en agil arbetsmetodik. Med principer som MVP och MOSCOW-metoden. I kapitel 1 beskrivs hur applikationens funktionalitet är prioriterad enligt MOSCOW-metoden för att uppnå en MVP-produkt.

I projektplanen skapades även en tankekarta för att visa projektets delar i stort. I figur 1 ser vi att tankekartan även belyser kopplingar mellan olika tekniska val och vilka tekniker som ska användas.



Figur 1: Projektets tankekarta

För att strukturerat arbeta med projektet utifrån MOSCOW-metoden samt tankekartan skapades en tidsplan. I figur 2 visas hur arbetet planerats från vecka 13 till sista dagen vecka 23 som också är projektets slutdatum. Tidsplanen kompletteras av en milstolpeplan för att belysa avslutsdatum för tidsplanens olika delar. I bilaga A visas milstolpe-planen där datumet för "Sammanställa och lämna in rapporten" markerar projektavslutet.



Figur 2: Projektets tidsplan

3.1 Utvecklingsmiljö

Applikationens utvecklingen kommer att ske i operativsystemet Ubuntu. Programmeringen för backend kommer utföras i den integrerade utvecklingsmiljön (IDE) *PHP Storm* eftersom den har bättre stöd för PHP och Drupal. Till frontend programmeringen kommer *Visual Studio Code* att användas som IDE.

3.2 Backend

Drupal ska användas som backend och kopplas som ett RESTful-API till lösningens frontend. Programmeringsspråket är PHP samt Drupals egen syntax och grafiska interface. Drupal valdes eftersom att Websystem är specialiserade på Drupal. Drupal har även många fördelar som ska användas i projektet som exempelvis kontohantering samt dess databas-koppling för lagring och struktur av data.

För att skapa denna backend ska Drupal-moduler installeras genom pakethanteraren Composer. Färdiga moduler ska användas där de fyller ett behov som exempelvis autentisering. En anpassad modul ska också skapas för applikationen. En anpassad modul är en modul som utvecklaren skapat själv. Den kan bestå av lite olika filer. Detta projekt kommer behöva kod för hantering av automatiska e-postutskick schemalagda med *Cron*. Specifik kod kommer även behövas för anpassade e-postmeddelanden samt för skapande av pdf-fakturor. För hantering av data mellan frontend och backend behövs även kod skapas för alla REST-ändpunkter. Dessa

ändpunkter ska kopplas till applikationens olika datatyper för åtkomst och manipulering av data. REST-ändpunkter ska även skapas för manuella utskick av fakturor.

3.3 Databas

Drupal kan använda MySQL eller MariaDB som databas. Därför kommer MariaDB att användas för projektet. Er-tabeller med dess relationer ska skapas för databasen.

3.4 Användargränssnitt

Grafiska designskisser ska skapas för applikationen i designverktyget Figma. Användargränssnittet ska skapas med frontend-ramverket Vue och programmeringsspråket som ska användas är *TypeScript*. För styling av webb-applikationen kommer CSS-ramverket *Tailwind* att användas. Vue och Tailwind ska användas för att de är moderna relevanta tekniker inom webbutveckling. Det är även tekniker som används på företaget Websystem. Eftersom att applikationen vänder sig till en bred grupp av samhället ska applikationen följa DOS-lagen:

- Innehålla en tillgänglighetsredogörelse
- Följ designriktlinjer för WCAG 2.1 på nivå AA

3.5 Metodlista

1. Grafisk design och webbplatskarta i Figma
2. Er-diagram i DrawIO
3. Backend i Drupal
4. Frontend i Vue
5. Testning av webbplatsen
6. Publicera applikationen

4 Konstruktion

4.1 Grafisk design i Figma

Utifrån målgrupp och applikationens tänkta funktioner listade i kapitel 1.4 detaljerad problembeskrivning skapades en grafiska designskisser i Figma. För designskisserna skapades moodboards med färgpalett, typsnitt och textstorlekar. I bilaga B visas hur fastigheter och boenden kunde listas samt dess moodboard. Designskisserna presenterades för företagets frontend-utvecklare samt två externa bedömare. Utvärderingen visade att designen uppfattades som omodern. I samråd med frontend-utvecklaren beslutades därför att en ny, enklare och mer modern design skulle utvecklas. De nya och slutgiltiga designskisserna går att se i bilaga C. Dessa användes som grund till applikationens fortsatta arbete.

I Figma skapades även en webbplatskarta för webbapplikationen. I bilaga D visas hur den listar alla undersidor. För webbplatskartan färgkodades alla undersidorna. Detta för att belysa vilken navigeringsväg och om inloggning krävs för att nå en specifik sida. Lila kräver ett inloggat användarkonto för åtkomst, blå kräver att e-postadressen finns registrerad hos vald bovärd och grön som inte kräver inloggning samt går att nå från alla webbplatsens sidor.

4.2 Er-diagram i DrawIO

I systemutveckling med Drupal används en datatyp istället för en databastabell. De fungerar på liknande sätt för den som utvecklar applikationen. Det är dock viktigt att förstå att för varje rad i en tabell i ER-diagrammet skapas en egen tabell i databasen där alla rader är relaterade till en gemensam rad. Varje datatyp innehåller också många standardrader med data utöver de som skapas manuellt.

För att förstå data och dess relationer behövs ändå ER-diagram för Drupal-utveckling. Ett ER-diagram skapades över databasens datastruktur i draw.io som är en applikation för ritning av grafer och diagram. Tabeller skapades för applikationen tänka funktioner enligt MOSCOW-prioriteringen. Nedan beskrivs alla tabeller men

de går även att se i sin helhet i bilaga E.

Användare(Must have)

Denna tabell skapades för att innehålla uppgifter till applikationens bovärd. Även fast det inte syns i ER-diagrammet så innehåller alla andra tabeller ett fält för vilken användare som har skapat tabellen. I figur 3 visas hur ER-tabellen för användare är uppbyggd.

User(Drupal's egna)	
PK	<u>UniqueID</u>
	UserName
	EmailAdress
	Password
	CreatedDate

Figur 3: Er-tabell användare

Fastighet(Must have)

En tabell skapades även för fastigheter. Varje bovärd kan lägga till flera fastigheter. Förutom fastighetsinformation skapades rader relaterade till fakturering. Dessa rader avgör om bovärderna vill ha automatisk fakturering för fastigheter och vilka datum det ska ske. Rader skapades även för betalningsmetod och betalningsnummer vilket visas i figur 4.

RealEstate	
PK	<u>UniqueID</u>
FK	UserID
	Title
	InvoiceDueDate
	InvoiceSendDate
	PaymentMethod
	PaymentNumber
	AutoInvoice
	StreetAddress

Figur 4: Er-tabell fastighet

Bostad(Must have)

Varje fastighet kan innehålla flera bostäder och därför skapades även en tabell för bostad. Figur 5 visar att bostadstabellen innehåller rader för titel(bostadsnamn), hyra sam namn och e-postadress till boendegästen. Bostäder är relaterade till fastigheter.

Accommodation	
PK	<u>UniqueID</u>
FK	RealEstateID
	Title
	Rent
	EmailAdress
	TenantName

Figur 5: Er-tabell bostad

Faktura(Must have)

För att applikationen skulle bli mer användbar behövdes att fakturor kunde sparas och därför skapades även en tabell för fakturor. Denna tabell strukturerades så att den ska kunna hämta data från andra tabeller när den skapas. Figur 6 visar innehållet i tabellen där även relationen till bostad visas.

Invoice	
PK	<u>UniqueID</u>
FK	AccommodationId
	Title
	InvoiceHTML
	InvoiceNumber
	InvoiceStatus
	TenantName
	EmailAddress

Figur 6: Er-tabell användare

Information(Should have)

Eftersom att bovärdarna bör kunna skicka information till boende i deras fastigheter skapades en ER-tabell för information. I figur 7 visas den relativt enkla tabellen för information till boende. Informationstabellen är relaterad till fastighetstabellen.

Information	
PK	<u>UniqueID</u>
FK	RealEstateID
	Title
	InfoText
	CreatedDate

Figur 7: Er-tabell information

Felanmälan(Could have)

I mån av tid kunde applikationen få ett system för felhantering och därför skapades även en ER-tabell för felrapport. Tabellen innehåller rader för meddelandetyp, meddelande, status och titel. Den innehåller även namn och e-postadress om vem som skrev felrapporten. Relationen till bostad går att se i figur 8.

ErrorReport	
PK	<u>UniqueID</u>
FK	AccomodationID
	Type
	Titel
	Message
	Status
	TenantName
	EmailAdress

Figur 8: Er-tabell felrapport

Serviceplan(Won't have)

Funktionalitet för att planera underhåll kommer inte implementeras i detta projekt utan finns med i projektet som en vision. Underhållsplanering tas med som ett förbättringsförslag. Ett ER-diagram skapades ändå för underhållsplan och går även den att se i bilaga E.

4.3 Backend i Drupal

Websystem hade redan en Drupal-webbplats igång för utveckling så därför användes den under utvecklingen. Men om en ny installation behöver göras så installeras Drupal med *Composer* genom terminal-kommandot "composer create-project drupal/recommended-project:10.4.5". Sedan installeras utvecklingsverktyget *Drush* till projektet med kommandot "composer require drush/drush". Efter installation kan projektet läggas på exempelvis en Apache-servers public-html katalog för tillgång från webbläsares adressrad. För att installera klart Drupal besöks sedan adressen "localhost:port" där port ska bytas ut till aktuell port för servern alternativt om projektet inte publiceras lokalt ska hela adressraden bytas ut. Väl inne på Drupal-webbplatsen kan utvecklaren följa installationsguiden.

Inställningar för CORS gjordes i filen services.php som finns i katalogen default. I figur 9 visas hur *cross-origin resource sharing*(CORS) aktiverades samt beviljade metoder.

```
# Configure Cross-Site HTTP requests (CORS).
# Read https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
# for more information about the topic in general.
# Note: By default the configuration is disabled.
cors.config:
  enabled: true
  # Specify allowed headers, like 'x-allowed-header'.
  allowedHeaders: []
  # Specify allowed request methods, specify ['*'] to allow all possible ones.
  allowedMethods: []
  # Configure requests allowed from specific origins. Do not include trailing
  # slashes with URLs.
  allowedOrigins: ['*']
  # Configure requests allowed from origins, matching against regex patterns.
  allowedOriginsPatterns: []
  # Sets the Access-Control-Expose-Headers header.
  exposedHeaders: false
  # Sets the Access-Control-Max-Age header.
  maxAge: false
  # Sets the Access-Control-Allow-Credentials header.
  supportsCredentials: false

queue.config:
  # The maximum number of seconds to wait if a queue is temporarily suspended.
  # This is not applicable when a queue is suspended but does not specify
  # how long to wait before attempting to resume.
  suspendMaximumWait: 30
```

Figur 9: Er-tabell felrapport

4.3.1 Installera moduler

Moduler valdes ut utifrån de funktioner som webbplatsen behövde samt för om de hade kompatibilitet till Drupal version 10.4.5 som användes i projektet. Dessa moduler valdes från Drupals officiella modul-sida[15]. Drupal-moduler kan ofta användas både från PHP-koden i egna moduler samt från det grafiska gränssnittet. Nedan listas de kommandon för moduler som installerades.

Consumers krävs för att modulen *Simple OAuth* ska fungera och kunna ge tillgång till innehåll. Installationskommando "composer require 'drupal/consumers:^1.19'".

Simple Oauth modulen används för verifiering av användare för bland annat när en separat frontend används. Den har stöd för OAuth2 med *bearer token*. Installationskommando "composer require 'drupal/simple_oauth:^5.2'".

RestUI är en modul som ger ett grafiskt interface för konfigurering av modulen *RESTful Web Services* som också installeras när installationskommandot körs. Här kan inställningar göras för vilka *REST-resurser* som ska vara aktiverade samt med vilka metoder

och vilken säkerhet de ska anslutas till. En REST-resurs är en ändpunkt dit externa användare kan skicka förfrågningar till och kan följa med en modul eller så kan utvecklare skriva egna resurser i kod. Installationskommando `"composer require 'drupal/restui:^1.22'"`.

Symfony mailer modulen är hela e-postsystemet som används i ramverket Symfony. När modulen installeras ersätter den Drupals befintliga e-postfunktion. Den har utökat stöd för att skicka filer och för att formatera e-post med HTML-kod. Installationskommando `"composer require 'drupal/symfony_mailer:^1.5'"`.

Maillog är en modul som används för att journalföra e-postutskick istället för att skicka iväg dem. *Maillog* används främst under utveckling och testning av tjänster som skickar e-post. Installationskommando `"composer require 'drupal/maillog:^1.0'"`.

Serial används på grund av att Drupal som standard använder innehållstyper istället för rena SQL-tabeller i databasen. Därför finns inte stöd för automatisk ökning av värdet på tabellens id. *Serial* ger innehållstyper möjligheten att lägga till ett fält med ett unikt värde som ökar för varje ny tabell som skapas. Installationskommando `"composer require 'drupal/serial:^2.1'"`.

Mpdf modulen gör det möjligt att konvertera HTML-kod till PDF-filer. Den har även stöd för styling, lösenordsskyddade filer samt vattenstämpel. Installationskommando `"composer require 'drupal/pdf_using_mpdf:3.x-dev@dev'"`.

Admin Toolbar används för att ge ett utökat administrationsgränssnitt för Drupal-webbplatser. Detta gränssnitt visar sig som nedfallande menyer med extra funktionalitet. Installationskommando `"composer require 'drupal/admin_toolbar:^3.5'"`.

Efter att alla tilläggen installerats med Composer navigerade jag på Drupal-webbplatsen till menyvalet utöka. Där aktiverades alla de ovan installerade moduler.

4.3.2 Datatyper

Helt enligt er-diagrammen för applikationen skapades datatyper i Drupal. I huvudmenyn valdes "Struktur" och sedan "Lägg till inne-

hållstyp”. I det grafiska interfacet valdes namn för datatyperna samt implementerades alla fält från er-diagrammen.

4.3.3 Skapa anpassad modul

För att ge applikationen dess önskade funktionalitet skapades en egen anpassad modul. Modulen innehåller alla REST-ändpunkter för att hantera applikationens data. I Drupal kallas dessa för *Rest-Resource*. Den innehåller även tre nya typer för e-post som används vid e-postutskick från *RestResource* och för automatisk e-postfakturerings. Dessa e-post typer är typer av *EmailBuilder*. Nedan redogörs för alla filers struktur och hur funktionaliteten skapades för den egenskapade modulen.

För att skapa en anpassad modul på enklaste sätt används utvecklingsverktyget Drush i terminalen. För att se tillgängliga kommandon skrivs ”d” i terminalen, för att se alla tillgängliga mallar som kan genereras skrivs ”d gen”.

Kommandot ”d gen module” användes för att skapa en anpassad modul. Val presenterades för modulen och där valdes modul-namnet till ddb som i det här fallet står för din digitala bovärd. Modulbeskrivningen valdes till RestAPI samt valdes att en modul-fil skulle skapas. Nu skapades två filer, en tom modul-fil och en info.yml-fil. I figur 10 visas hur modulens beskrivning och dess krav deklarerades.

```
name: "Custom API"
type: module
description: "Provides a custom API for 'Din Digitala Bovärd. Includes custom automated invoice mail based on cron.'"
core_version_requirement: ^10 || ^11
package: Custom
dependencies:
  - drupal:node
  - drupal:rest
version: "1.0"
project: "custom_api"
```

Figur 10: ddb.info.yml-fil

För att skapa egna restresurser till RESTful-apiet använde jag Drush kommandot ”d gen rest”. I de val som presenterades valdes samma modul-namn för alla restresurser som för tidigare skapad modul. Där valdes också lämpliga namn för ändamålet. Restresurser programmeras och skapas genom att utöka klassen *ResourceBase*. Programmeringsprocessen av restresurserna beskrivs nedan

i sin helhet för fastigheter och mer konceptuellt för övriga restresurser då struktur och programmeringsgrund är liknande.

RestRealEstate skapades för att hantera datatypen för fastigheter. Drupal använder ofta kommentarer i koden för att styra egenskaper. Därför är vissa kommentarer i Drupal också kod. Ett exempel på det går att se i figur 11 som visar en typisk inledning på en restresurs. Den beskriver id och namn för resursen. Det specificeras även vilka ändpunkter och om de ska ta in html-parametrar.

```
/**
 * Represents positions as resources.
 *
 * @RestResource(
 *   id = "rest_real_estate",
 *   label = @Translation("Real estate Custom REST API"),
 *   uri_paths = {
 *     "canonical" = "api/realestate/{id}",
 *     "create" = "api/realestate"
 *   }
 * )
 */
```

Figur 11: Deklarera restresurs

Följt av det startar själva klassen som innehåller restresursen. Med denna kod utökades klassen ResourceBase: "final class RestResourceConfig extends ResourceBase".

En restresurs kan innehålla en funktion för varje anropsmetod (*get*, *post*, *delete*, *patch*). För att Drupal ska förstå att funktionerna är ändpunkter ska de heta sin metod.

Först skapades funktionen för att hämta data vilket visas i bilaga F. Denna funktion deklarerades på följande sätt: "public function get(): ModifiedResourceResponse {}". Funktionen inleds med att den inloggade användarens id hämtas. Därefter skapades en *entityquery* för att hämta data. Entityquery använder *entity API* för att hämta data från databasen och är Drupal's medföljande funktion för att hantera objekt av olika datatyper[9]. För entityquery bestäms även om en åtkomst kontroll ska utföras. Detta görs för de flesta änd-

punkter i den egna modulen. Nästa steg i koden var att lagra retur-data i en *array*. Detta gjordes med en *foreach-loop* där data hämtades för varje nod av datatypen och lagrades i en array med *JSON*-data.

Efter det skapades funktionen för att lagra fastighetsdata som tar in *JSON*-data med anropet. Funktionen startar med att kontrollera om krävd data är korrekt medskickad annars returneras ett felmeddelande med status-kod 406. Efter det lagras en ny nod för fastigheter. Hela funktionen visas i bilaga G.

I bilaga H visas hur funktionen utformades för att ta bort en fastighet. Funktionen tar emot en *html*-parameter med nodens id. Om ägaren stämmer överens med ägaren på noden så tas den bort.

Klassen innehåller också en funktion för att uppdatera fastigheter. Om användaren stämmer överens med nodens ägare så uppdateras noden med den *JSON*-data som skickades med. Funktionen för att uppdatera fastigheter visas i bilaga I.

I figur 12 visas hur klassen avslutas med en funktion för att hämta en specifik nod med nodens id.

```
// En metod för att hämta noden baserat på ID
2 usages
private function getNodeById($nodeId) {
    // Här hämtas en nod från id parameter
    return \Drupal\node\Entity\Node::load($nodeId);
}
```

Figur 12: Hämta en specifik nod

RestAccommodation klassen skapades för att hantera bostäder fungerar lika dant som ovanstående klass för fastigheter. Anpassningar har gjorts för den data som ska hanteras och dess namn.

RestErrorReport klassen hanterar felrapportering och är även den väldigt lik fastighetsklassen. En skillnad finns som att patch endast läser in om status för felrapporten har ändrats och ingen annan ändring utförs på felanmälan.

RestGetRealEstates är en klass som skapades för att endast hämta alla bovärdensnamn följt av fastigheternas namn. Denna returnerar inte hela fastighetsobjekt utan enbart en array med strängar.

RestInformation skapades för att kunna lägga upp information till bostadsgästerna. Även denna klass är lik övriga men med en skillnad. Anropet för att hämta data hämtar även relaterad data i form av fastighetens namn för att göra det tydligare för användare av applikationen. I bilaga J visas hur koden loopar igenom varje informationsobjekt och hämtar relaterad data med fastighets id.

RestInvoice klassen är till för att hantera fakturor. Fakturor skapas inte manuellt utan är en sammanställning av data. I denna klass kan fakturor hämtas, tas bort och status kan uppdateras men fakturan kan inte skrivas om. Post-funktionen används som ändpunkt för att kontakta en klass för att skicka en e-postfaktura som även innehåller en pdf-faktura. Så denna ändpunkt skickar endast med berörd fastighet, bostads samt bovärdensnamnet till klassen CustomInvoice. I figur 13 visas hur e-post-klassen initieras och hur variabler skickas med som parametrar. Ingående funktionalitet av e-post funktionen går att läsa om för respektive klass.

```
$email_factory = Drupal::service('email_factory');  
$email = $email_factory->sendTypedEmail('custom_invoice', 'invoice', $node, $accommodation_node, $username);
```

Figur 13: Initiera e-postfunktion

RestReminderInvoice använder också post-anrop för att skicka faktura men i detta fall för faktura påminnelser. Denna skickar endast med en redan befintlig faktura nod till klassen CustomInvoiceReminder.

RestTenant skiljer sig på så sätt att det är ändpunkter för ej inloggade. Den har därför ingen åtkomstkontroll. Post-ändpunkten används för att verifiera att den angivna e-postadressen finns registrerad för vald fastighet. Detta räcker som säkerhet då det inte är känslig information utan endast används för att visa relevant information för användaren. Om användaren finns registrerad returneras all information från bovärderna för aktuella fastigheter.

Här finns även en get-ändpunkt som endast hämtar en lista med bostadsnamn där e-posten är registrerad.

Den egna komponenten innehåller också tre anpassade e-postbyggare. Dessa utökar klassen EmailBuilderInterface som använder modulen Symfony mailer. Nedan beskrivs koden och funktionen i sin helhet för CustomInvoice och övriga två beskrivs konceptuellt.

CustomInvoice är en e-postbyggare och det deklarerar i kommentaren direkt i början. Vi ser i figur 14 hur kommentaren gett den utökade klassen ett id samt en undertyp.

```
/**
 * Defines the Email Builder plug-in for test mails.
 *
 * @EmailBuilder(
 *   id = "custom_invoice",
 *   sub_types = { "invoice" = @Translation("Invoice") },
 * )
 */
```

Figur 14: Deklarera e-postbyggare

Sedan deklarerar för att klassen är en utökning av EmailBuilderInterface klassen. Klassen inleds med att i kommentarerna specificera e-postmall och i tillhörande funktion createParams specificera för hur denna undertyp ska ta emot parametrar. Figur 15 visar exakt hur detta skapades.

```
class CustomInvoice extends EmailBuilderInterface {

    /**
     * Saves the parameters for a newly created email.
     *
     * @param \Drupal\symfony_mailer\EmailInterface $email
     *   The email to modify.
     * @param mixed $to
     *   The to addresses, see Address::convert().
     */
    public function createParams(EmailInterface $email, $realestate = NULL, $accommodation = NULL, $username = NULL) {
        // Tilldelar $email medföljande parametrar
        $email->setParam('realestate', $realestate);
        $email->setParam('accommodation', $accommodation);
        $email->setParam('username', $username);
    }
}
```

Figur 15: Deklarera e-postmall samt parametrar

I bilaga K visas huvud funktionen för e-postbyggaren som heter "build". Den läser in parametrar och inställningar från föregående funktion. Variabler tilldelades värden från parametrarna för ett fastighetsobjekt, ett bostadsobjekt samt bovärdens namn. Från fastighet hämtades förfallodag för fakturan samt att en variabel med

dagens datum skapades. Om förfallodatum redan har passerat så förskjuts förfallodatum med en månad.

Sedan skapades en ny nod för innehållstypen faktura med data för bostaden. Med en kombination av data från fastighet, bostad, datum samt faktura id skapades en variabel. Denna är formaterad som html-kod med inbäddad PHP-kod för att skriva ut data. Faktura-noden uppdateras och den nyskapade html-koden läggs till. Vi ser i figur 16 hur en pdf-faktura skapades i en funktion från html-koden.

```
//Funktion för att generera pdf från html.
1 usage
public function generatePdf($html): string {
    // Skapa en ny instans av mPDF.
    $mpdf = new Mpdf();
    // Skriv HTML-innehållet till PDF.
    $mpdf->WriteHTML($html);
    // Få PDF-innehållet som en sträng
    $pdf = $mpdf->Output( name: 'rent_invoice.pdf', dest: 'S');
    return $pdf;
}
```

Figur 16: Skapa pdf-faktura

Till sist skickas fakturan med mejl där html-koden är som grund för både mejlets brödtext och pdf-fakturan. Vi ser i figur 17 hur e-mejlets olika delar tilldelas värden.

```
//Deklarera emailadress, subjekt, bifogar pdf-filen samt skickar med html som body
$email->setTo($accommodation->get('field_emailadress')->value);
$email->setSubject( subject: 'Hyresavi: ' . $realestate->getTitle());
$email->attachNoPath($pdf, 'rent_invoice.pdf');
$email->setBody(['#markup' => $html]);
}
```

Figur 17: Skicka e-post

CustomFaultReport skapades för att ta emot en bostadsbeteckning samt en e-postadress. Utifrån dessa skickar den e-postnotis till bovärderna för att informera att en felanmälan har skapats.

CustomInvoiceReminder är väldigt lik CustomInvoice men här skickas en fakturapåminnelse. Den använder html-kod från en redan befintlig faktura-nod som bas för brödtext samt pdf-fil. Det

som ändras är e-mejlets ämne vilket nu skriver ut "Hyresavi: Påminnelse" samt dagens datum.

Till sist skapades koden i modulens modul-fil för att lägga till automatiska e-postfakturer med CRON. Filen består av en funktion med modulens namn följt av en CRON hook vilket visas i figur 18. Eftersom att funktionen endast ska aktiveras en gång per dygn inleds den med en kontrollera om den redan har aktiverats. Detta görs med att en datumsträng lagras i en nyckel-värdes-databas. Denna kallas i Drupal för state.

```
/**
 * Implements hook_cron().
 */
no usages
function custom_api_cron() {
```

Figur 18: CRON hook

Efter det hämtas alla fastigheter som har automatisk fakturering aktiverad. För alla dessa fastigheter hämtas relaterade bostäder. Om en bostad har en registrerad e-postadress skickas aktuellt fastighetsobjekt, bostadsobjekt och bovärdens namn som parametrar till e-postbyggaren CustomInvoice.

4.3.4 Inställningar

I Drupals gränssnitt började jag med att installera den egenskapade modulen genom utöka-gränssnitten. Efter det valdes REST i menyn och där aktiverades alla skapade ändpunkter samt användarregistrering. För samtliga valdes inställningar att formatet är JSON samt autentiseringsmetod är OAuth2.

Sedan navigerades i menyn till behörigheter. Behörighetsinställningarna är omfattande och går att ändra för nästan allt och där nekas behörighet för det mesta för att öka applikationens säkerhet. För RESTful Web Services gavs resurserna för att lista fastighetsnamn, skapa felrapport, hämta information från bovärd samt användarregistrering rättigheter för både inloggade och ej inloggade användare. För övriga rest-resurser angavs behörighet för inloggade användare.

I menyn valdes ny konfiguration och kontoinställningar. Där anpassades e-postmeddelanden som används vid registrering och återställning av lösenord. Här ändrades även registreringsinställningar så att även besökare själva kan registrera konton samt att det krävs e-postverifiering för att registrera.

Till sist gjordes inställningarna för modulen Simple OAuth som sköter autentisering och verifiering av användare. I menyn valdes konfiguration och Simple OAuth. I inställningarna höjdes utgångstiden för åtkomst-token till en timme. Där valde jag även att generera nycklar för åtkomstkontroll. Till sist skapades en ny konsument för anslutning från tredjeparts konsument. Konsumenten fick nu en klienthemlighet som behövs för att logga in användare med rest-anrop.

4.4 Frontend i Vue

Ett Vue-projekt skapades genom att skriva `create vue@latest` i terminalen. Alternativen som valdes här var TypeScript samt router SPA. Efter det installerades tailwind för projektet genom att skriva `npm install tailwindcss @tailwindcss/vite` i terminalen. Tailwind inkluderades nu i filerna `vite.config.ts` samt i `main.css`.

För applikationen skapades 10 vyer vilket kan likställas med webbsidor. Det skapades även 14 komponenter och 5 filer mer interface för objekts typer. Till sist skapades även 7 service-filer som hanterar REST-anrop.

I filen `index.ts` som återfanns i routerkatalogen specificerades för vilka webbadresser som ska öppna specifika sidor. Sidorna för felrapport, information samt faktura åtkomstkontrolleras här genom kontroll om en åtkomst-token finns lagrad i webbläsarens localstorage.

4.4.1 Vyer

Start-vyn består av enkla text-element samt skärmdumpar för att sälja in applikationen.

Inloggnings-vyn innehåller ett enkelt formulär för inloggning. Under formuläret visas knapp för att visa registreringsformuläret istället. Det finns även en länk för att återställa lösenord. När

knappen aktiveras för att visa registreringsformuläret ändras variabeln för inloggningsväxling till falskt istället. Detta gör att en annan rubrik visas i formuläret samt så att relevant metod anropas när formuläret skickas in.

Fastighets-vyn Implementerar flera komponenter. Dess funktion redogörs för i kapitel 4.4.2. Vyn består av att fastigheter visas i en stor container-komponent i skapas en textcontainer för varje fastighet. Dessa innehåller knappkomponenter för att öppna fastighetsformuläret samt för att visa att relaterade bostäder. Under fastigheterna finns även en knapp för att lägga till en ny fastighet.

Bredvid listas relaterade bostäder för vald fastighet på samma sätt som fastigheter listas. Skillnaden här är att det finns en knapp för att skicka en e-postfaktura som då kallar på den funktionen i bostadsservicen.

Faktura-vyn består av en tabell som listar alla fakturor. Metoder implementerades för sökning, sortering samt paginering. Varje rad i tabellen har en kolumn med en knapp för som skickar med fakturaobjektet till komponenten med modal för faktura.

Felrapports-vyn har funktionalitet som är uppbyggd på samma sätt som i faktura-vyn. Knappar i tabellen öppnar här istället en modal för felrapporter.

Information-vyn implementerar komponenten för informationsformuläret. I vyn finns även en stor containerkomponent där all egen information för fastigheter listas i textcontainer-komponenter.

Boende-vyn används för inloggning av boende hos bovärdens bostäder. Detta görs i ett enkelt formulär som skickar vald e-postadress samt vald bostad med servicen för boende. Om e-postadressen finns registrerad på bostaden returneras ett svar med all aktuell information för fastigheten och användaren visas ett formulär för felhantering samt all information från bovärdens. Felhanteringsformuläret implementeras genom dess komponent samt informationen skrivs ut i stor container-komponent där varje meddelande skrivs ut i textcontainer-element.

Integritetspolicy-vyn innehåller en integritetspolicy och en nedladdningslänk för den samt en mejlkontaktlänk. Integritetspolycyn är enkel men beskriver hur personuppgifter hanteras och vilken data som lagras. Den avslutas med användares rättigheter samt vart de kan vända sig för att ta bort data, få veta vilken data som finns lagrad eller klaga.

Tillgänglighetsredogörelse-vyn innehåller endast en skriven tillgänglighetsredogörelsen, nedladdningslänk för den samt en mejlkontaktlänk. Tillgänglighetsredogörelsen bygger på den officiella mallen från myndigheten för digital förvaltning. Den beskriver vilka delar som är tillgänglighetsanpassade och vilka som är mindre bra anpassade. Det gör det även tydligt vart man ska vända sig för om brister hittas och vilken myndighet de kan kontakta vid bristande efterlevnad av dos-lagen. Det gör också gällande att webbplatsen frivilligt följer riktlinjerna då den egentligen inte omfattas av lagen. Redogörelsen beskriver och några anpassningar som gjorts av webbplatsen för att göra den tillgänglig. Några av anpassningarna:

1. Hela webbplatsen fungerar att styra och komma åt med tangentbord.
2. Knappar, länkar och inmatningsfält har relevanta *aria-labels*. Aria-labels är beskrivning av ett html-elements funktion som används av skärmläsare.
3. Anpassningar för fel som uppkommit under testning har även åtgärdas. Det förloppet beskrivs i kapitel 4.5.2.

Om sidan-vyn innehåller endast text som beskriver webbapplikationen samt en mejlkontaktlänk.

4.4.2 Komponenter

Knappkomponenterna fungerar alla på samma sätt. I figur 19 visas hur de tar in knappnamn som prop och använder det för titel, aria-label och knappnamn. Knappen svara sedan med en emit tillbaka till föräldrakomponenten när den aktiveras.

```
<template>
  <button
    class="bg-blue-500 hover:bg-blue-700 text-white font-bold rounded-full py-2 px-4 hover:cursor-pointer"
    :title="buttonName"
    :aria-label="buttonName"
    @click="$emit('clicked')"
    :id="id"
  >
    {{ buttonName }}
  </button>
</template>

<script lang="ts">
  //Buttonklass med props och emit.
  export default {
    name: 'BlueButton',
    props: ['buttonName', 'id'],
    emits: ['clicked'],
  }
</script>
```

Figur 19: CRON hook

Sidhuvud-komponenten innehåller applikationens sidhuvud inklusive huvudmenyn. Menyn använder *Vue router* som är den inbyggda funktionaliteten för navigering mellan vyer.

Sidfot-komponenten Innehåller endast en enkel sidfot med länkar till vyer om tjänsten, integritetspolicy och tillgänglighetsredogörelse.

Stor container-komponent används som en grå bakgrund och kan även ta emot prop för titel, text och id. Komponentens har även en avsedd plats där andra komponenter kan placeras. Dess platser kallas i Vue för *slot*.

Textcontainer-komponenten fungerar lika dant som föregående komponent men den kan ta emot hela 6 texter med 6 tillhörande underrubriker.

Felanmälningsformulär-komponenten är ett enkelt formulär för att skicka in felanmälningar. V-for används i ett flervälsfält i formuläret för att lista alla bostäder som den boende är registrerad för. Formulärets skicka-knapp skickar med formulärets data till servicen för felrapporter. Ett meddelande med olika text skrivs ut beroende på om post-anropet i servicen lyckades eller ej. Komponentens tar in e-postadress samt registrerade bostäder som *props*.

Informationsformulär-komponenten fungerar på samma sätt men med utökad funktionalitet. Dessa består av att den tar in ett

booleskt värde som prop för om det är ny information eller en information som ska redigeras. Beroende på denna bestäms vilken funktion som ska kallas på i servicen för informations-anrop. För att data ska ändra dynamiskt har lyssnare skapats för att ändra formulärets data när data i props från föräldern ändras.

Fastighetsformulär-komponenten är ett formulär som öppnas när en fastighet väljs. Den fungerar lika dans som ovanstående men med skillnaden att den har en knapp för att ta bort fastighetsobjektet. Om den aktiveras visas för ett val för att säkerställa att den ska tas bort. Om den tas bort blir all data relaterad till den otillgänglig bovärd.

Bostadsformulär-komponenten är funktions och designmässigt identisk med fastighetsformulär-komponenten. Skillnaden ligger i att den hanterar bostadsdata istället.

Modal för fakturor används för att visa utökad information om en faktura. Eftersom att fakturan finns lagrad i sin helhet som en html-sträng i fakturaobjektet skrevs den ut med v-html i ett div-element. Följt av detta finns ett formulär som endast innehåller väl för att ändra fakturans status. Vid statusändring initieras PATCH-funktionen i services för fakturor. Komponentens tar in ett faktura objekt som props samt emits för att stänga modal. Lyssnare läggs till för att dynamisk ändra data när data från props förändras. Till sist finns en knapp för att skapa en faktura. Fakturan konverterar html-kod till en pdf-fil. För att göra det används npm paketet html2pdf.

Modal för felanmälningar har samma funktionalitet som modal för fakturor men med skillnaden att data visas i en tabell istället.

4.4.3 Typer

Typer för alla applikationens objekt deklarerades i interface. Interface skapades för fastigheter, bostäder, fakturor, information samt felanmälan. Dessa skapades alla på liknande sätt efter vilken data objekten kunde innehålla. I figur 20 visas hur typer bestäms för bostadsobjekten.

```
//Interface för boenden
export interface Accommodation {
  id: string
  title: string
  emailaddress: string | null
  rent: string
  tenant: string
  real_estate_id: { target_id: string }[]
}
```

Figur 20: CRON hook

4.4.4 Services

TypeScript-filer används för att skapa services för applikationens REST-anrop. Dessa delades upp mellan filer för vilken data och ändpunkter de skulle nå. Dessa filer är lika varandra och innehåller anrop med metoderna get, post, patch samt delete. Filen Login-Service.ts skiljer sig lite. I den skapades anrop med POST-metoden för både inloggning och för registrering. I bilaga L visas hur utöver användarnamn och lösenord även inloggningsinställningar och klienthemlighet skickas med i anropet. Detta är samma klienthemlighet som hämtas i inställningarna för Drupal Simple OAuth.

4.5 Testning av webbplatsen

Alla tester som utfördes på webbplatsen är på Vue-applikationen. Således gjordes alla ändringar och anpassningar från dessa tester i Vue-applikationen och inte i Drupal.

4.5.1 HTML-validering

Webbplatsens html-kod validerades genom tjänsten *Markup Validation Service*. Tjänsten tillhandahålls av W3C och är till för att säkerställa att html-kod följer html-standard[16].

Varje sida validerades och gav några fel som duplicerade id, typfel på textarea samt att html-språket inte var definierat till svenska. Dessa fel åtgärdades genom att överflödiga id togs bort, typningen av textarea togs också bort samt att språk för html lades till.

4.5.2 Tester för tillgänglighet

Alla applikationens vyer samt modals testades för tillgänglighet med det automatiska testverktyget *Aceit*. Aceit är utvecklat av

företaget *Useit* och är ett testverktyg specifikt för tillgänglighet. Enligt Useit[17] gör Aceit en fullständig tillgänglighetsgranskning enligt kraven i WCAG samt den europeiska standarden EN 301 549.

Testverktyget går att använda på specifika webbsidor eller för en hel webbapplikation. Men eftersom att min webbapplikation kräver inloggning för att visa alla element så testades den genererade html-koden direkt istället. För att göra det kopierade jag html-koden för varje sida och för varje Vue-komponent som renderades på skärmen. Nedan visas resultat och åtgärder från testerna för respektive sida.

Startsidan fick inga fel enligt WCAG 2.1 och fick ett betyg på 118%. I bilaga M går att se hur resultatet för tillgänglighetstestet presenterades.

Om tjänsten lika så fick inga fel och betyget 118%.

Integritetspolicy fick följande fel "Förvarna användaren om länken öppnar ett dokument". Texten "Ladda ner integritetspolicy" lades till som aria-label attribut och länktext. Testet fortsatte ändå att anmärka fel för länken och gav betyget 98%.

Tillgänglighetsredogörelse fick samma fel, åtgärder och resultat som integritetspolicy.

Inloggningssidan gav fel på att nav-element saknades innan användaren loggades in samt att platshållare för inmatningsfält inte är till hjälp för skärmläsare. Nav-elementet för huvudmenyn ändrades så att den alltid syns samt fick attributet aria-label="Huvudmeny". Ett nav-element skapades även för länkarna i sidfoten. Inmatningsfält för registrering och inloggning gavs också attribut för aria-labels. Betyget gick från 91% till 118%.

Inloggningssidan för hyresgäster gav fel samma fel för platshållare på inmatningsfält och därför lades det till attribut för aria-labels. Betygen gick från 93% till 118%.

Hyresgästsidan gav fel på att tomma element renderas på skärmen. Det tomma elementet vad ett p-element som ska innehålla eventuella felmeddelanden. För formuläret adderades v-

`if="errorMessage"` till p-elementet för felmeddelande för att det endast skulle skriva om det faktiskt fanns ett felmeddelande. V-if används för att villkorsmässigt rendera en block om uttrycket är sant[18]. Betyget gick från 95% till 118%.

Fastighetssidan gav samma fel som uppkommit tidigare som tomma element och saknad attribut för aria-label på inmatningsfält. Dessa åtgärdades med v-if för att kontrollera om felmeddelande fanns samt lades aria-label till. Betyget gick från 93% till 116%.

Fakturasidan gav även den fel för tomma element samt saknade aria-label attribut och därför utfördes samma åtgärd som för föregående sida. Betyget gick från 93% till 116%.

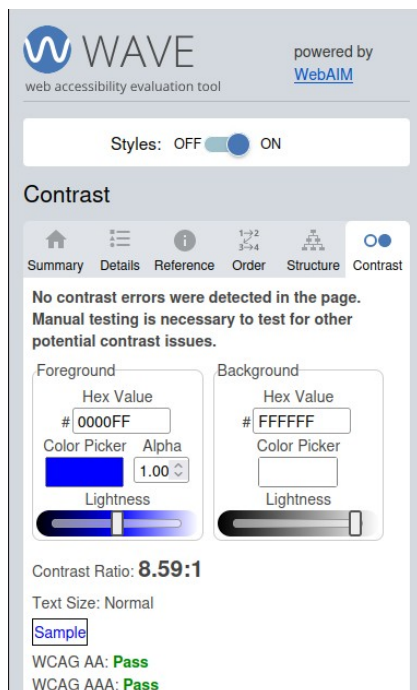
Felhanteringssidan gav även den fel för tomma element och saknade attribut för aria-labels. Förutom det gavs fel på att en tabellkolumn saknade attributet `scope="col"`. Attributet används för att specificera att tabellcellen är ett kolumnhuvud. Dessa fel åtgärdades. Betyget gick från 88% till 118%.

Informationssidan gav endast fel på ett tomt element och detta fel åtgärdades med en v-if. Betyget gick från 95% till 118%.

4.5.3 Testning med Lighthouse

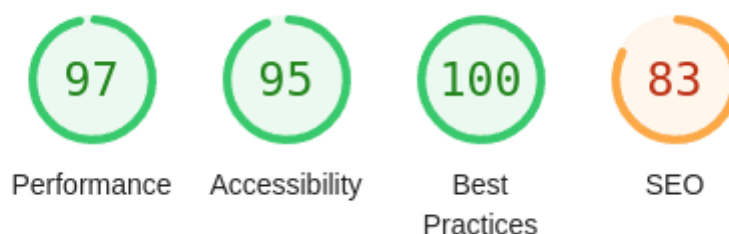
Testverktyget Lighthouse användes för att testa webbapplikationen. Lighthouse är ett inbyggt testverktyg i webbläsaren Google Chrome och är till för att testa webbplatser för prestanda, tillgänglighet, bästa metoder och sökmotoroptimering[19].

Alla webbplatsens sidor testades och resultatet blev ganska lika för alla sidor. För tillgänglighet gav testet fel för kontrasten mellan de blåa knapparna och texten. Eftersom att jag var nöjd med kontrasten kontrollerades även webbplatsens kontraster med webbläsartillägget Wave. Wave är utvecklat av Webaim och är till för att testa webbplatser för tillgänglighet enligt kraven WCAG standarder[20]. I figur 21 kontrasttest visas ett exempel på hur ett kontrasttest ser ut i Wave.

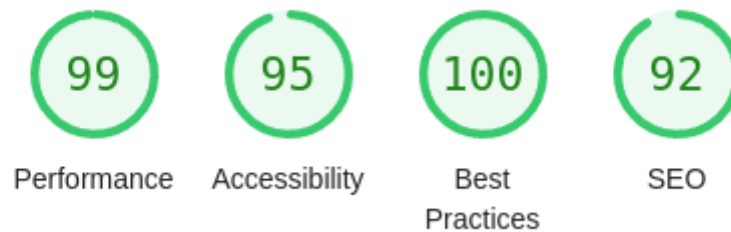


Figur 21. Kontrasttest

Resultatet från Lighthouse var bra med godkända värden på allt utom sökmotoroptimering. Fel för sökmotoroptimering var att en robots.txt-fil saknades. Robots.txt ska innehålla instruktioner för hur webbplatser ska indexeras av bland annat sökmotorer[21]. Webbplatsen saknade även en meta-beskrivning för webbplatsens innehåll. För att lösa dessa problem skapade jag en textfil i Vue-projektets public mapp med namnet robots.txt. I den skrev jag texten "User-agent: * Disallow: /". Denna text är till för att blockera sökmotorindexering och det passade bra eftersom att jag inte behövde att webbplatsen indexerades. I figur 22. "Lighthouse" och figur 23. "Lighthouse åtgärdat" går det att se hur testresultatet för sökmotor optimering ökade från 83 till godkänt 92.



Figur 22. Lighthouse



Figur 23. Lighthouse åtgärdat

4.5.4 Användartester

De sista testen som genomfördes för webbplatsen var användartester. Användartesterna gick ut på att testa hur intuitiv och användarvänlig webbplatsen var. I bilaga N visas testdokumentet som användes. Testet genomfördes på så vis att testpersonerna fick först avgöra webbplatsens syfte och sedan skapa ett konto samt logga in. Efter det skulle de skapa en fastighet med en tillhörande fastighet. För denna skulle automatisk fakturering aktiveras samt skulle en faktura skickas ut. Denna faktura skulle de även ladda ner. Till sist skulle de publicera information om fastigheter för dess boende. Testpersonerna var Christoffer, Isak samt Max med åldrarna 49år, 26år respektive 27år. Alla testpersoner hade stor datorvana.

Testresultatet visade brister på webbplatsens startsida där varken Isak eller Max uppfattade dess syfte eller dess målgrupp. Alla testpersoner upplevde osäkerhet kring hur formulär skulle fyllas i samt vad vissa knappars funktion faktiskt var. Isak upplevde att fler laddningssymboler och notiser för att informera vad som händer hade givit en bättre upplevelse. Webbplatsen hade uppenbara brister framförallt i texter och rubriker.

Utifrån testresultaten ändrades texter i formulären och knappars namn ändrades. Exempelvis i formuläret för fastighet ändrades "Förfallodag" ändrades till "Faktura förfallodag" och i bostadsformuläret ändrades "E-postadress" till "Bostadsgästens e-postadress". Även informationstexterna på startsidan skrevs om för att mer träffsäkert belysa webbplatsen syfte och målgrupp.

4.6 Publicera applikationens

4.6.1 Publicera Drupal

Som webbhotell för Drupal valdes Inleed eftersom att det har stöd för många funktioner. Tjänsten erbjuder egna mariaDB-databaser, e-postkonton, Cron-funktionalitet[22]. Dessutom går det att specificera PHP-version samt komma åt sitt webbhotell via SSH om behov finns.

Inne på Inleed valdes databas i kontrollpanelen. Sedan valdes "skapa databas". Namn för databasen valdes till ddb. Användarnamnet för databasen blev då samma samt så skapades ett slumpmässigt lösenord. Dessa uppgifter sparades då de skulle komma att behövas vid Drupal installationen.

Ett e-postkonto skapades också för applikationen. I Inleeds kontrollpanel valdes e-post samt skapa konto. Där valdes användarnamn och lösenord samt domän för e-postkontot. Valen gav e-postadressen "din.digitala.bovard@markuswebb.se".

Drupal installerades om igen på samma sätt som vid nyinstallation. Först installerades Drupal med Composer och sedan installerades alla publika tillägg även de med Composer på samma sätt som i kapitel 4.3.1. I projektets modul-katalog skapades en ny katalog vid namn custom där den egenskapade modulen kopierades in. Inställningar för CORS gjordes lika dant som i kapitel 4.3.

Nu kopierades hela projektkatalogen över till katalogen public-html hos webbhotellet. Installationsprocessen för Drupal startades genom att index.php-filen besöks genom webbläsaren. Webbadressen blev då "domännamn/web". I installationsguiden valdes rekommenderad installation samt att uppgifter för databas samt e-post fylldes i.

Inställningar för applikationen gjordes nu på samma sätt som i kapitel 4.3.3.

För att aktivera CRON på webbhotellet navigerade jag in på "Cron jobs" under "Advanced Features" i Inleeds kontrollpanel. I bilaga O visas hur jag skapade ett nytt CRON job i Inleed. Väl i inställningarna valde jag att varje morgon klockan fem skulle

kommandot "curl -L -s" följt av Drupal-webbplatsens webbadress för att köra CRON. Webbadressen för detta återfinns i Drupal-webbplatsens meny under konfiguration/system/schemalagda aktiviteter.

4.6.2 Publicera Vue

Vue-applikationen publicerades till Netlify. Netlify är en webbhost för statiska webbsidor och webbapplikationer. För mer tillförlitlig routing vid sidladdning skapades en fil med namnet _redirects innehållandes koden "/* / 200". Denna fil hjälper Netlify att veta hur applikationens routing ska hanteras. Därefter kördes kommandot "npm run build" i terminalen för att transpilera Vue-applikationen. Applikationens dist-katalog kopierades nu in i deploy-gränssnittet hos Netlify. Efter publicering kunde applikationens namn väljas.

5 Resultat

Resultatet blev en användbar applikation för bovärdar som enkelt lever upp till förväntningarna på ett MVP-projekt. Projektet lever upp till alla krav och täcker av de flesta funktioner från prioriteringslistan enligt MOSCOW. I bilaga P visas den publicerade versionen av "Din digitala Bovärd" och den är tillgänglig för bovärdar att använda. Utvärdering av resultatet utifrån problembeskrivningen:

DOS-lagen

Både automatiska tester samt manuella tester visar att applikationen testar bra mot standarden WCAG 2.1. Den innehåller även en egen sida för tillgänglighetsredogörelse vilket är ett krav i DOS-lagen. Standarden är dock omfattande och mer manuella tester hade kunnat förbättra upplevelsen ytterligare. Slutsatsen är att applikationen löser problemet.

Kontohantering

Bovärdar kan skapa egna konton. De kan även återställa lösenord vid behov. Inloggade personer får tillgång till en stor mängd funktioner samt hantering av data. Med en enkel, tydlig och säker kontohantering möts problembeskrivningen för kontohantering.

Lista fastigheter och fakturera boende

Problemet löses genom att bovärdar enkelt kan skapa fastigheter vilket visas i bilaga Q. Till dessa går det att lägga till tillhörande bostäder. I det här läget går det att skicka ut fakturor med pdf-fakturor till e-postadresserna som är registrerade på bostäderna. Här går det även att aktivera automatisk fakturering och vilken faktureringsdag samt förfallodag fakturan ska ha.

Extra funktioner(information till boende, felanmälningar.)

Utöver grundproblemet implementerar applikationen en hel del extra funktionalitet och viss funktionalitet som fakturahantering är mer utökad än vad som var tänkt.

Funktionalitet är implementerad så att bovärdar kan lägga upp information till boende. Boende kan genom att logga in med endast

e-post och bostad för att ta del av denna information. Ingen känslig information ska läggas upp här, men ändå implementerades en enklare kontroll. På denna sida kan även meddelanden och felanmälningar för bostäder skickas in av boende.

En egen sida skapades för fakturor vilken inte var ett krav med det gjorde att fakturahanteringen blev en mer funktionell funktion. Här listas alla skickade fakturor och det går att skicka ut påminnelser på e-post. Status går att ändra för om den är makulerad, betald, ej betald eller om en påminnelse är skickad. I bilaga R visas hur en enskild faktura kan hanteras.

Hantering av felanmälningar fick en egen vy. Där listas alla och det går att ställa in status för felanmälan. När en felanmälan skickas in får även bovärden en e-postnotis om den.

Sammantaget lever applikationen upp till projektets krav och dess problemdefinition. Funktioner som inte implementerades är underhållsplanering samt att status för felmeddelanden inte visas för boenden. Meddelanden går också bara åt ett håll då bovärden inte kan skicka meddelanden till boende.

6 Diskussion

Målet med projektet var att skapa en applikation för bovärdar. Genomförandet var som ett projekt i form av ett examensarbete. Projektet har flutit på väldigt bra. Jag blev väldigt nöjd med hur applikationen löste flera problem för bovärdar enligt problemen som formulerats.

Förbättringsförslag uppskattas då det är svårt för några få att testa en applikation och hitta förbättringar. Förbättringsförslag i nuläget är att utöka felmeddelanden för att hantera svar till den boende också. En större funktion för underhållsplanering där även dokument från besiktningar kan laddas upp hade skapat mervärde. Här skulle då allt planerat underhåll kunna samlas samt att utfört arbete med dess kvitton kan sparas.

6.1 Analys och resultatdiskussion

Applikationen fungerar bra i sitt nuvarande utförande och fungerar bra. Det går dock att finjustera designval. Även tydligare vilka värden som ska fyllas i och vad de faktiskt gör kan förtydligas. Även vilka värden som ska lagras i databasen och vilken tabell de ska tillhöra kan behöva en översyn. Applikation når ändå enkelt sitt mål och löser problembeskrivningen.

6.2 Projektmetod diskussion

Metodvalen för att utföra projektet och skapa en webbapplikation var bra. Det svåraste var att förstå Drupal då inställningar, programmering och modul-funktionalitet blandas. Kvalitet på dokumentationen var också ojämn och ofta bristfällig. Däremot ger Drupal en väldigt säker backend med bra felrapportering. Funktionaliteten blir också väldigt skalbar.

Alla metoder passade projektet bra och Vue passade bra för frontend. Däremot känns det som att en större planering av hur funktionerna faktiskt skulle fungera hade underlättat skapandet av applikationen. Med fördel skulle en prototyp göras i Figma för hela applikationen och dess tänka funktionalitet. Designen hade inte be-

hövs vara slutgiltig för alla sidor men det hade givit en bättre grund för skapandet av ER-diagram, backend samt applikationen grafiska interface.

Användartesterna utfördes i ett sent skede av projektet och alla brister har därför inte hunnit åtgärdats innan publicering av webbapplikationen. Till kommande projekt behöver användartester vara med i projektplaneringen och dess metod redogöras för tydligare i rapportens metodkapitel.

6.3 Etisk och social diskussion

Webbapplikationen "Din Digitala Bovärd" faller under dataskyddsförordningen (GDPR). Information som namn och e-postadresser kopplas ihop med bostäder och bovärdar. Även uppgifter om vilken hyra de betalar samt om de har betalat hyran registreras. Detta kan vara känslig information och den får inte hanteras vårdslöst eller delas utan giltig anledning och godkännande. Applikationen har en integritetspolicy som beskriver vilken information som lagras och hur den hanteras. Det går även att kontakta "Din Digitala Bovärd" för att kontrollera vilken data som finns lagrad. Det är viktigt att bovärdar kontrollerar med boende i deras fastigheter innan de registreras i tjänsten. Det står tydligt i integritetspolicyen att ett godkännande krävs och att det bör ha med i exempelvis kontraktet för bostaden.

Bovärdar kan lägga upp information till boenden och där är det viktigt att ingen privat information skrivs ut. Det står även tydlig information om det innan information skickas ut.

Det är problematiskt när möjligheter begränsas. En tillgänglighetsanpassad webbplats tillåter åtkomst och möjligheter åt fler. Därför är tillgänglighetsanpassning viktigt. När den här typen av applikation är skapad av ett privat företag eller av en person omfattas den inte av DOS-lagen. Jag valde ändå att anpassa den efter DOS-lagen då tillgänglighet för alla är viktigt. En diskussion kan föras kring vart gränsen ska ligga för vilka applikationer som behöver anpassas. Jag har inte svaret på den frågeställningen men allt kommer inte fungera eller vara passande för alla men när det är möjligt så bör anpassningar göras.

Källförteckning

- [1] Projektledning, "MoSCoW metoden"
<https://projektledning.se/moscow-metoden/> Uppdaterad 2021-08-10. Hämtad 2025-04-25.
- [2] Into the commerce, "What is A minimum viable product(MVP)? – A Complete Guide"
<https://intothecommerce.com/business/what-is-a-minimum-viable-product-mvp/> Uppdaterad 2025-03-31. Hämtad 2025-04-25.
- [3] SCB, "Boende i Sverige"
<https://www.scb.se/hitta-statistik/sverige-i-siffror/manniskorna-i-sverige/boende-i-sverige/> Uppdaterad 2023-07-05. Hämtad 2025-03-24.
- [4] Drupal, "Overview of Drupal"
<https://www.drupal.org/docs/getting-started/understanding-drupal/overview-of-drupal> Uppdaterad 2025-02-03. Hämtad 2025-04-25.
- [5] DIGG, "Om lagen om tillgänglighet till digital offentlig service"
<https://www.digg.se/analys-och-uppfoljning/lagen-om-tillganglighet-till-digital-offentlig-service-dos-lagen/om-lagen> Uppdaterad 2024-01-04. Hämtad 2025-04-25.
- [6] Symfony, "What is Symfony"
<https://symfony.com/what-is-symfony> Hämtad 2025-04-25.
- [7] W3Schools, "Bash crontab Command – Schedule Tasks"
https://www.w3schools.com/bash/bash_cron.php Hämtad 2025-04-28.
- [8] Drupal, "RESTful Web Services API overview"
<https://www.drupal.org/docs/drupal-apis/restful-web-services-api/restful-web-services-api-overview> Uppdaterad 2024-03-29. Hämtad 2025-04-25.

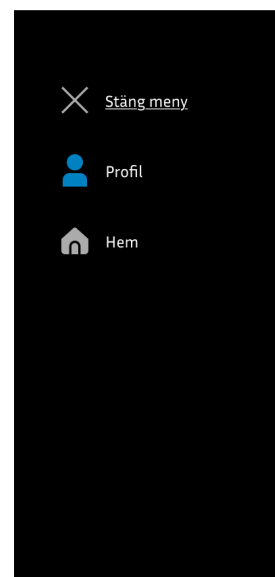
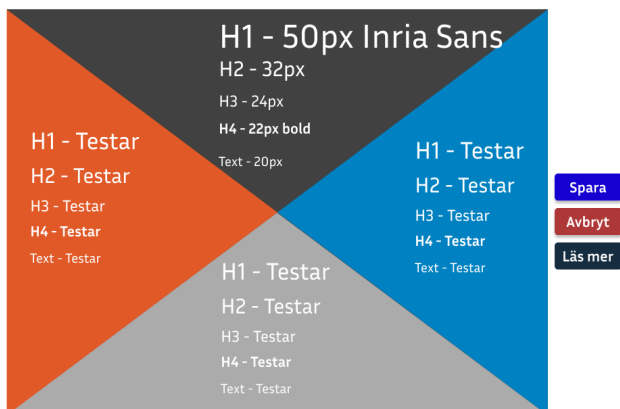
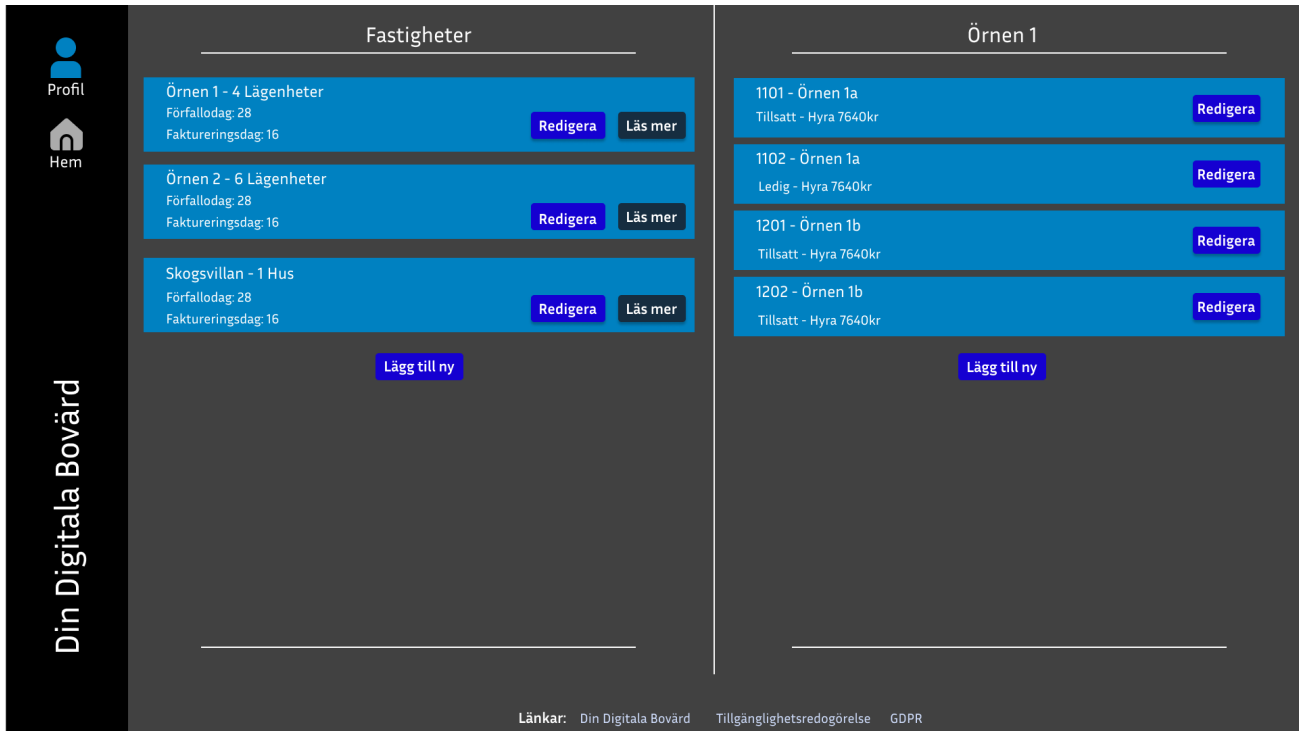
- [9] Drupal, "Working with the Entity API"
<https://www.drupal.org/docs/drupal-apis/entity-api/working-with-the-entity-api> Uppdaterad 2024-03-25. Hämtad 2025-04-25.
- [10] Drupal, "Understanding hooks"
<https://www.drupal.org/docs/develop/creating-modules/understanding-hooks> Uppdaterad 2025-03-14. Hämtad 2025-04-25.
- [11] Drupal, "Drush"
<https://www.drupal.org/docs/develop/development-tools/drush> Uppdaterad 2024-04-03. Hämtad 2025-04-25.
- [12] getcomposer, "Introduction" <https://getcomposer.org/doc/00-intro.md> Hämtad 2025-04-28.
- [13] tailwindcss, "Rapidly build modern websites without ever leaving your HTML" <https://tailwindcss.com/> Hämtad 2025-04-25.
- [14] Vue.js, "Introduction" <https://vuejs.org/guide/introduction.html> Hämtad 2025-05-01.
- [15] Drupal, "Drush"
https://www.drupal.org/project/project_module Hämtad 2025-05-08.
- [16] W3C, "Markup Validation Service" <https://validator.w3.org/> Hämtad 2025-05-08.
- [17] Useit, "Testplattform för digital tillgänglighet"
<https://ace.useit.se/ax/about.php> Hämtad 2025-05-08.
- [18] Vue.js, "Conditional Rendering"
<https://vuejs.org/guide/essentials/conditional.html> Hämtad 2025-05-08.
- [19] Chrome for developers, "Introduction to Lighthouse"
<https://developer.chrome.com/docs/lighthouse/overview> Uppdaterad 2016-09-27. Hämtad 2025-05-08.

- [20] Webaim, "Wave web accessibility evaluation tool"
<https://wave.webaim.org/> Uppdaterad 2025. Hämtad 2025-05-08.
- [21] SEO-guide, "Robots.txt" <https://www.seo-guide.se/robots-txt>
Uppdaterad 2014-07-07. Hämtad 2025-05-08.
- [22] Inleed, "Hjälpcenter" <https://login.inleed.net/helpcenter?>
Hämtad 2025-05-08.
- [23] SEO-guide, "Robots.txt" <https://www.seo-guide.se/robots-txt>
Uppdaterad 2014-07-07. Hämtad 2025-05-08.

Bilaga A: Milstolpe-plan

Milstolpe	Slutdatum
Projektplan publicerad i kursforumet	2025-03-28
Skapa grafisk design och sitemap i Figma	2025-04-02
Skapa Er-diagram med Draw.io	2025-04-07
Skapa Drupal backend för grundfunktioner	2025-04-18
Skapa frontend i Vue för grundfunktioner	2025-05-09
Testa plattformen för WCAG 2.1 på AA nivå	2025-05-10
Publicera backend samt frontend med eget domännamn	2025-05-11
Lägg till extra funktioner i mån av tid	2025-05-23
Skriv rapport och lämna in rapport för opponering	2025-05-23
Förbered för och skapa redovisningsmaterial	2025-05-30
Sammanställa och lämna in rapporten	2025-06-08

Bilaga B: Första design



Bilaga C: Slutgiltiga designskisser

Logga in

Din digitala bovärd

Tjänsten skapar värde till din förening. Innehåller funktioner som faktura hantering och felanmälningar med mera..

Länkar: [Din Digitala Bovärd](#) [Tillgänglighetsredogörelse](#) [GDPR](#) [Om tjänsten](#)

Din digitala bovärd

Fastigheter

Profil

Logga ut

Fastigheter

Örnen 1 - 4 Lägenheter
Förfallodag: 28
Faktureringsdag: 16

Redigera

Läs mer

Örnen 2 - 6 Lägenheter
Förfallodag: 28
Faktureringsdag: 16

Redigera

Läs mer

Skogsvillan - 1 Hus
Förfallodag: 28
Faktureringsdag: 16

Redigera

Läs mer

Lägg till

Örnen 1

1101 - Örnen 1a
Tillsatt - Hyra 7640kr

Redigera

1102 - Örnen 1a
Ledig - Hyra 7640kr

Redigera

1201 - Örnen 1b
Tillsatt - Hyra 7640kr

Redigera

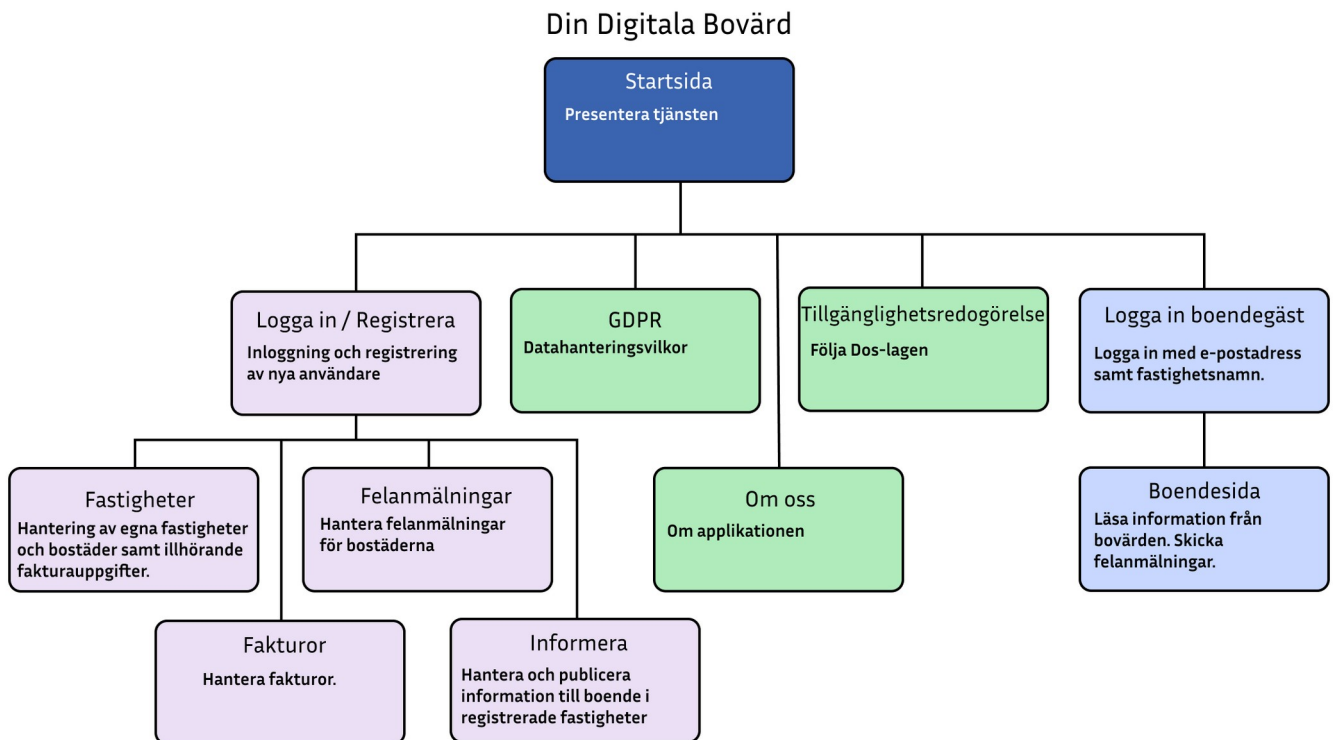
1202 - Örnen 1b
Tillsatt - Hyra 7640kr

Redigera

Lägg till

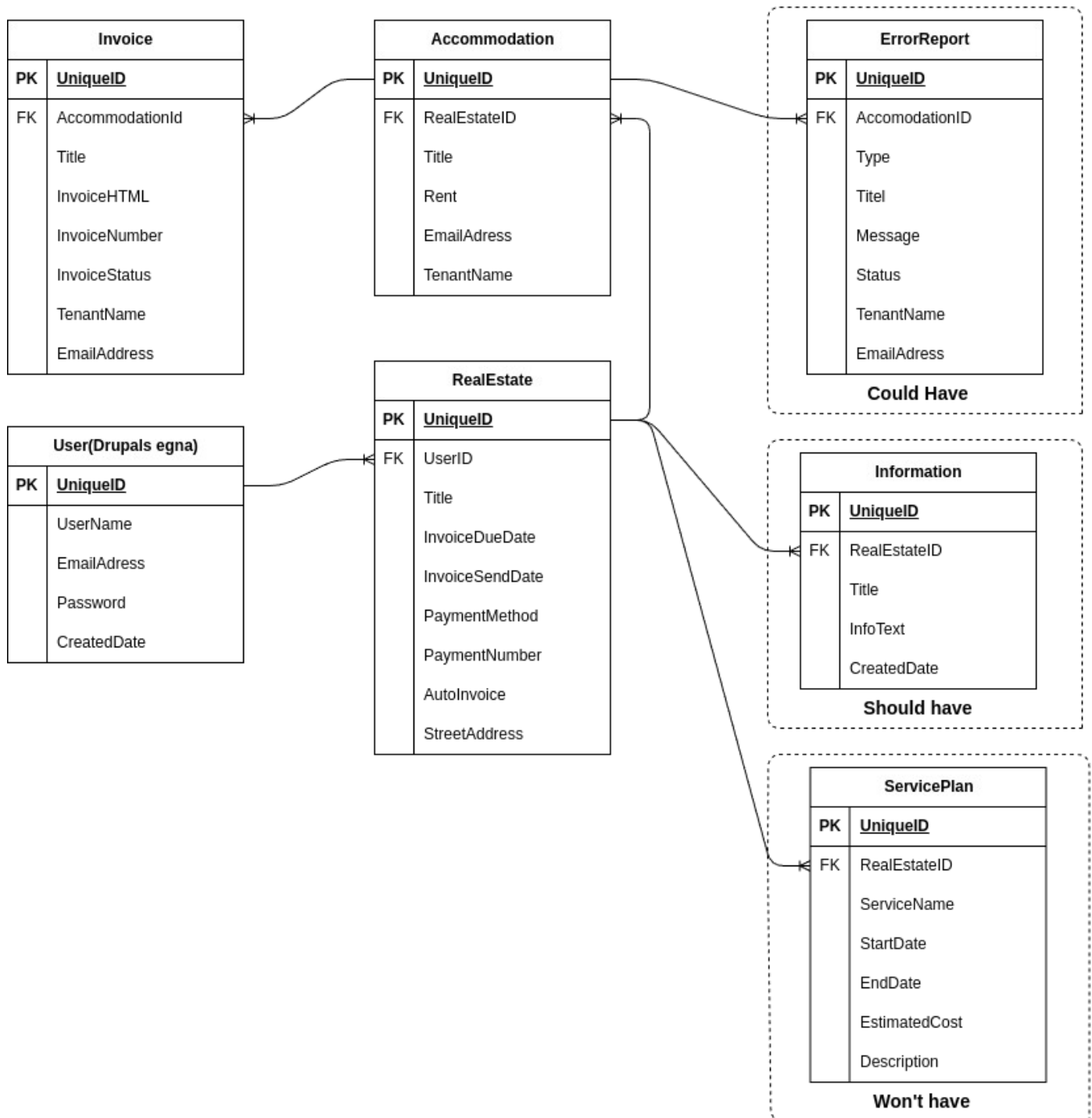
Länkar: [Din Digitala Bovärd](#) [Tillgänglighetsredogörelse](#) [GDPR](#) [Om tjänsten](#)

Bilaga D: Webbplatskarta



Bilaga E: ER-diagram

ER-diagram Din Digitala Bovärd



Bilaga F: Hämta fastighetsdata

```
final class RestResourceConfig extends ResourceBase {  
  
    /**  
     * Responds to GET requests.  
     *  
     * The response containing real estate data.  
     */  
  
    public function get(): ModifiedResourceResponse {  
        //Hämtar id för inloggad användare  
        $current_user_id = \Drupal::currentUser()->id();  
  
        // databas fråga för att hämta fastighet  
        $query = \Drupal::entityQuery('node')  
            ->condition('field_type', 'realestate')  
            ->condition('field_uid', $current_user_id)  
            ->accessCheck('access_check: TRUE')  
            ->execute();  
  
        //kör fråga  
        $nodes = \Drupal\node\Entity\Node::loadMultiple($query);  
  
        //array som ska returneras som svar  
        $data = [];  
  
        // Loopar igenom alla hämtade object  
        foreach ($nodes as $node) {  
            $data[] = [  
                'id' => $node->id(),  
                'title' => $node->getTitle(),  
                'autoinvoice' => $node->get('field_autoinvoice')->value,  
                'invoice_due_date' => $node->get('field_invoice_due_date')->value,  
                'invoice_send_date' => $node->get('field_invoice_send_date')->value,  
                'streetaddress' => $node->get('field_streetaddress')->value,  
                'payment_method' => $node->get('field_payment_method')->value,  
                'payment_number' => $node->get('field_payment_number')->value,  
            ];  
        }  
  
        return new ModifiedResourceResponse($data, status: 200);  
    }  
}
```


Bilaga G: Lägg till fastighet

```
/**
 * Responds to POST requests.
 *
 * lägger till en till node med fastighet.
 */
public function post(array $data): ModifiedResourceResponse {

    //kontrollerar om data finns
    if (empty($data['title'])) {
        return new ModifiedResourceResponse(['error' => 'Title is required.'], status: 406);
    }
    if (empty($data['streetaddress'])) {
        return new ModifiedResourceResponse(['error' => 'Street Address is required.'], status: 406);
    }

    // deklarerar node data och skapar noden
    $node = \Drupal\node\Entity\Node::create([
        'type' => 'realestate',
        'title' => $data['title'],
        'field_streetaddress' => $data['streetaddress'],
        'field_autoinvoice' => $data['autoinvoice'] ?? FALSE,
        'field_invoice_due_date' => $data['invoice_due_date'] ?? NULL,
        'field_invoice_send_date' => $data['invoice_send_date'] ?? NULL,
        'field_payment_method' => $data['payment_method'] ?? NULL,
        'field_payment_number' => $data['payment_number'] ?? NULL,
    ]);

    // Spara noden
    $node->save();

    // Returnera en framgångsrespons
    return new ModifiedResourceResponse($node, status: 201);
}
```

Bilaga H: Ta bort fastighet

```
/**
 * Responds to DELETE requests.
 *
 *
 * Tar bort en nod med fastighet.
 */
public function delete($id): ModifiedResourceResponse {

    // Hämta noden med det angivna ID:t
    $node = $this->getNodeById($id);

    //hämtar id för inloggad användare
    $current_user_id = \Drupal::currentUser()->id();

    // Kontrollera om noden finns
    if ($node) {
        // Kontrollera om noden tillhör den inloggade användaren
        if ($node->getOwnerId() === $current_user_id) {
            // Ta bort noden
            $node->delete();

            // Return att borttaget
            return new ModifiedResourceResponse( data: NULL, status: 204);
        }
        // Om noden inte tillhör användaren
        else {
            throw new \Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException( message: 'You do not have access to this node.');
```

Bilaga I: Uppdatera fastighet

```
/**
 * Responds to PATCH requests.
 *
 *
 *
 * Tar bort en nod med fastighet.
 */
public function patch($id, array $data): ModifiedResourceResponse {
    // Hämta noden med det angivna ID:t
    $node = $this->getNodeById($id);

    // Hämtar id för inloggad användare
    $current_user_id = \Drupal::currentUser()->id();

    // Kontrollera om noden finns
    if ($node) {
        // Kontrollera om noden tillhör den inloggade användaren
        if ($node->getOwnerId() === $current_user_id) {...}
        // Om noden inte tillhör användaren
        else {
            throw new \Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException('message: 'You do not have access to this node.');
```

Bilaga J: Hämta relaterad data

```
public function get(): ModifiedResourceResponse {  
    //läser in inloggad användares id  
    $current_user_id = \Drupal::currentUser()->id();  
  
    // databas-fråga för att hämta information till boende  
    $query = \Drupal::entityQuery('node')  
        ->condition('field: 'type', 'value: 'information')  
        ->condition('field: 'uid', $current_user_id)  
        ->accessCheck('access_check: TRUE')  
        ->execute();  
  
    //kör frågan  
    $nodes = \Drupal\node\Entity\Node::loadMultiple($query);  
  
    //array för att lagra object till retur  
    $data = [];  
  
    // Loopar igenom alla hämtade object  
    foreach ($nodes as $node) {  
        // Hämtar relaterat realestate_id  
        $realestate_ids = $node->get('field_realestate_id')->getValue();  
        $realestate_title = '';  
  
        // OBS!! Dessa if-satser används för att skicka med extra data i svaret. Det gör det t  
        // Om det finns en referens till realestate  
        if (!empty($realestate_ids)) {  
            // Hämta den första realestate noden  
            $realestate_node = \Drupal\node\Entity\Node::load($realestate_ids[0]['target_id']);  
            if ($realestate_node) {  
                $realestate_title = $realestate_node->getTitle();  
            }  
        }  
  
        //Skickar med ett objekt till svaret för varje informationsnode  
        $data[] = [  
            'id' => $node->id(),  
            'title' => $node->getTitle(),  
            'information' => $node->get('field_information')->value,  
            'realestate_id' => $node->get('field_realestate_id')->getValue(),  
            'realestate_title' => $realestate_title, // Lägg till realestate_title  
            'created' => $node->getCreatedTime(),  
        ];  
    }  
  
    return new ModifiedResourceResponse($data, status: 200);  
}
```

Bilaga K: E-postbyggare

```
/**
 * {@inheritdoc}
 */
public function build(EmailInterface $email) {
    //Deklarerar variabler för medföljande parametrar
    $realestate = $email->getParam( key: 'realestate');
    $accommodation = $email->getParam( key: 'accommodation');
    $username = $email->getParam( key: 'username');

    //Skapar ett datumobjekt för dagens datum
    $today = date( format: 'Y-m-d');
    //Skapar ett datumobjekt med nuvarande år samt dag. Det läggs sedan till värdet från fastighetsnoden för betalningsdag.
    $dueDate = date( format: 'Y-m') . "-" . sprintf( format: '%02d', (int) $realestate->get('field_invoice_due_date')->value);

    //Om förfallodag redan har passerat för denna månad ändras förfallodag till månaden efter.
    if ($dueDate < $today) {
        // Om förfallodatumen har passerat, läggs en månad till
        $dueDate = date( format: 'Y-m-d', strtotime( datetime: $dueDate . ' +1 month'));
    }

    //En ny node skapas för denna invoice
    $invoice_node = $this->createInvoiceNode($accommodation);

    //HTML komponeras utifrån fastighetsnode, bostadsnode, användarnamn samt datumen som skapades ovan.
    $html =
        '<h1 style="font-size:24px;text-decoration: underline;">Hyresavi från <b>' . $username . '</b> </h1><br>' .
        '<h2 style="font-size:20px;text-decoration: underline;">Boendeinformation</h2>' .
        '<p><b>Namn:</b> ' . $accommodation->get('field_tenant')->value . '</p>' .
        '<p><b>Adress:</b> ' . $realestate->get('field_streetaddress')->value . '</p>' .
        '<p><b>E-postadress:</b> ' . $accommodation->get('field_emailaddress')->value . '</p><br>' .
        '<h2 style="font-size:20px;text-decoration: underline;">Faktura uppgifter</h2>' .
        '<p><b>Bostadsbeteckning:</b> ' . $accommodation->getTitle() . '</p>' .
        '<p><b>Faktureringsdatum:</b> ' . $today . '</p>' .
        '<p><b>Förfalldatum:</b> ' . $dueDate . '</p>' .
        '<p><b>Betalningsmetod:</b> ' . $realestate->get('field_payment_method')->value . '</p>' .
        '<p><b>Kontonummer:</b> ' . $realestate->get('field_payment_number')->value . '</p>' .
        '<p><b>Betalningsnummer:</b> ' . $invoice_node->get('field_invoice_number')->value . '</p>' .
        '<p><b>Hyra:</b> ' . $accommodation->get('field_rent')->value . 'kr</p><br>' .
        '<p><b>Betalning</b> ' . $accommodation->get('field_rent')->value . 'kr med <b>' . $realestate->get('field_payment_method')->value .
        '</b> till konto <b>' . $realestate->get('field_payment_number')->value . '</b>. Bifoga betalningsnummer: <b>' .
        $invoice_node->get('field_invoice_number')->value . '</b></p><br>';

    //Den nyskapade invoice noden samt html skickas till en funktion för att uppdateras med html
    $this->updateInvoiceNodeHTML($invoice_node, $html);

    //Samma html skickas även till en funktion för att skapa en pdf-fil
    $pdf = $this->generatePdf($html);

    //Deklarera emailadress, subjekt, bifogar pdf-filen samt skickar med html som body
    $email->setTo($accommodation->get('field_emailaddress')->value);
    $email->setSubject( subject: 'Hyresavi: ' . $realestate->getTitle());
    $email->attachNoPath($pdf, 'rent_invoice.pdf');
    $email->setBody(['#markup' => $html]);
}
```


Bilaga L: Inloggningservice

```
// POST-anrop för att logga in en användare
export function login(username: string, password: string): Promise<string> {
  return (
    axios
      .post(
        'https://www.markuswebb.se/theproject/web/oauth/token',
        {
          client_id: import.meta.env.VITE_CLIENT_ID,
          grant_type: import.meta.env.VITE_GRANT_TYPE,
          client_secret: import.meta.env.VITE_CLIENT_SECRET,
          username: username,
          password: password,
        },
        {
          headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
          },
        },
      )
      .then((response) => {
        //JWT till storage
        localStorage.setItem('access_token', response.data.access_token)
        router.push('/home')
      })

      //Vid fel skrivs felet ut i konsollen samt skriver ut ett meddelande till skärmen
      .catch((error) => {
        console.log(error)
        return error.response.data.message
      })
  )
}

//Funktion för att logga ut en användare. Eventuellt bör en logout token användas och anrop till backend
export function logout() {
  localStorage.removeItem('access_token')
  router.push('/login')
}

// POST-anrop för att registrera en ny användare
export function register(username: string, email: string): Promise<string> {
  return (
    axios
      .post('https://www.markuswebb.se/theproject/web/user/register? format=json', {
        name: {
          value: username,
        },
        mail: {
          value: email,
        },
      })
      .then((response) => {
        return 'Lyckad registrering. Logga in på din epost för att verifiera användare och välj lösenord.'
      })

      //Vid fel skrivs felet ut i konsollen samt skriver ut ett meddelande till skärmen
      .catch((error) => {
        console.log(error)
        return error.response.data.message
      })
  )
}
```

Bilaga M: Ace it

Sammanfattning

Analys av:

- Din digitala bovärd
Direktinmatad_html-kod (1)

Analyserar bara inmatade sidor

Analyserade
sidor
1



Antal tester som hittat fel, i snitt per sida

WCAG 2.1 nivå AA			Bortom WCAG 2.1 nivå AA	
Fel	Troliga fel	Klarade	Fel	Klarade
0	0	26	2	10

Tillgänglighetsindex

118%

Över 99: Verktöget har inte hittat några fel eller troliga fel kopplade mot WCAG.

90-99: Det finns avsteg men troligen inte så omfattande.

75-89: Det finns klara problem.

Under 75: Det finns stora problem på sidan.

0

Antal individuella fel mot WCAG 2.1 nivå AA, i snitt per sida

0

Antal individuella troliga fel mot WCAG 2.1 nivå AA, i snitt per sida

1

Antal individuella övriga fel i snitt per sida

▼ Vad säger siffrorna?

Resultattabell

▼ Berätta om resultattabellen

Visa samtliga fel som lista

#	Sida	Struktur	Rubriker	Bilder	Länkar	Tabeller	Formulär	Ramar	Html	Texter
1	Din digitala bovärd	Dator, med JavaScript								
118%	Förekomster på sidan	5	3	5	8	0	3	0	0	3
	WCAG 2.1 nivå AA:	-	0 0	0	0 0 0	0 0 0	0 0 0	0	-	0 0
	Troligt fel:	0 0	0 0	0 0	0 0	-	0	-	0 0	-
	Bortom WCAG:	0 0 1	0	0	0	0	0	0	-	0 0

Bilaga N: Användartester

Din Digitala Bovärd – Användartester
Markus Vickman

2025-05-19

Användartester

De här testerna går ut på att testa webbplatsen Din Digitala Bovärd. Testerna är avsedda för att testa hur intuitiv och användarvänlig den är. Medan testet utförs ska testpersonerna berätta hur de tänker och hur interaktionen uppfattas.

Testuppgifter

1. Framgår webbplatsens syfte?
2. Skapa ett konto och logga in.
3. Skapa en fastighet och en tillhörande bostad.
4. Aktivera automatisk fakturering samt manuellt skicka en faktura.
5. Navigera till och ladda ner den skickade fakturan.
6. Publicera information till boende för den skapade fastigheten.
7. Logga ut.

Upplevda brister

Här listat de upplevda bristerna hos webbplatsen som orsakade problem eller förvirring. Upplevda brister i detta sammanhang är reella brister hos webbplatsen då den behöver vara lättanvänd.

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____

Bilaga O: CRON

»

☰

💬

👤

⚙️

🌐

EN

🔗

Dashboard > Cron Jobs > Edit Cron Job

markus.webb

Edit Cron Job

Current Time

5/8/2025, 9:49 AM

Minute

0

?

Hour

5

?

Day of Month

*

?

Month

*

?

Day of Week

*

?

Cron job will run: At 05:00 AM

Command

?

curl -L -s https://www.markuswebb.se/theproject/web/cron/hbeoqx17nAlpNVqJCJpxCFkzV5

PREVENT E-MAIL

SAVE

Explanation:

- Valid Cron time values are the numbers indicated and *.
- You can specify exact times using commas to separate them. e.g. 1,2,3 (minutes 1,2 and 3)
- You can specify spans using a dash. e.g. 5-7 (minutes 5 to 7)
- You can specify intervals using a star and a forward slash. e.g. */2 (every 2nd minute)
- You can combine them to create a more precise schedule. e.g. 1,5,11-15,30-59/2 (minutes 1, 5, 11 to 15 and every 2nd minute between 30 and 59)

Bilaga P: Din Digitala Bovärd

Din Digitala Bovärd

Tjänsten som skapar värde till din bostadsförening. Innehåller funktioner för listning av fastigheter och bostäder samt fakturahantering och automatisk fakturering.

Fakturor					
Sök bostad					
Sök...					
FAKTURANUMMER	BOSTAD	BOSTADSGÄST	STATUS	SKAPAD	SE HELA
1	1132b	Markus Vickman	Ej betald	2025-05-05	Visa
2	1132b	Markus Vickman	Makulerad	2025-05-05	Visa
3	1132b	Markus Vickman	Ej betald	2025-05-05	Visa
4	1132b	Markus Vickman	Påmind	2025-05-05	Visa
5	1132b	Markus Vickman	Betald	2025-05-05	Visa
6	1132b	Markus Vickman	Betald	2025-05-05	Visa

Mervärde



Ytterligare funktionalitet finns för att hantera felanmälningar och meddelanden från bostadsgäster. Tjänsten har även ett lätt gränssnitt för att skriva ut aktuell information om fastigheterna.

ÄRENDENUMMER	4
BOSTAD	1132b
E-POST	vick_mark@proton.me
BOENDE	Markus Vickman
TITEL	Lågt vattentryck
MEDDELANDE	Det är lågt vattentryck i duschen
ÄRENDETYP	Felanmälan
STATUS	Ej påbörjad
SKAPAD	2025-05-05
TA BORT	Radera

Ändra status
Välj status

Ändra

Bilaga Q: Fastighetshantering



Fastigheter

Fastigheter

Rondellen
Gatuadress: Ångermanlandsgatan 2
Förfallodag: 2
Faktureringsdag: 9
Auto-faktruteri: 1
Betalningsmetod: Bankgiro
Betalningsnummer: 4532-5423

Redigera

Visa

Solhöjden
Gatuadress: Södra sjögatan 23
Förfallodag: 27
Faktureringsdag: 9
Auto-faktruteri: 1
Betalningsmetod: Swish
Betalningsnummer: 0704254234

Redigera

Visa

Lägg till

Rondellen | Bostäder

1132bb
Epost-adress: vick_mark@proton.me
Hyra: 9870.00kr
Hyresgäst: Markus Vickman

Redigera

Skicka faktura

1131b
Epost-adress: oscar.wik@example.net
Hyra: 6789.00kr
Hyresgäst: Oscar Wik

Redigera

Skicka faktura

Lägg till

Om tjänsten

Integritetspolicy

Tillgänglighetsredogörelse

Bilaga R: Fakturahantering

Fakturor

Sök bostad

Sök

FAKTURANUMMER	SE HELA
1	Visa
2	Visa
3	Visa
4	Visa
5	Visa
6	Visa
7	Visa
8	Visa
9	Visa
10	Visa

Hysesavi från macke5g

Boendeinformation

Namn: Markus Vickman
Adress: Ångermanlandsgatan 2
E-postadress: vick_mark@proton.me

Faktura uppgifter

Bostadsbeteckning: 1132b
Faktureringsdatum: 2025-05-05
Förfallodatum: 2025-05-28
Betalningsmetod: Bankgiro
Kontonummer: 4532-5423
Betalningsnummer: 1
Hyra: 9870.00kr

Betala **9870.00kr** med **Bankgiro** till konto **4532-5423**. Bifoga betalningsnummer: **1**

Ladda ner Skicka påminnelse Radera

Ändra status
Ej betald

Ändra

Om tjänsten Integritetspolicy Tillgänglighetsredogörelse