CHALMERS

Lokalbokningswebbapplikation för universitet och högskolor

En webbaserad prototyp för effektivisering av lokalbokning

Examensarbete inom högskoleingenjörsprogrammet Datateknik

Carl Åberg
Kevin Bondesson

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2026
www.chalmers.se

Examensarbete 2026

Lokalbokningswebbapplikation för universitet och högskolor

Utveckling och utvärdering av webbaserad bokningsapplikation

Carl Åberg
Kevin Bondesson

CHALMERS

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
Göteborg 2026

Lokalbokningswebbapplikation för universitet och högskolor
Carl Åberg
Kevin Bondesson

Handledare: Inari Listenmaa, Institutionen för data- och informationsteknik
Examinator: John J. Camilleri, Institutionen för data- och informationsteknik

Examensarbete 2026
Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Illustration av person som bokar eller schemalägger en tid i en kalender.

Skriven i LaTeX
Göteborg 2026

Lokalbokningswebbapplikation för universitet och högskolor
Utveckling och utvärdering av webbaserad bokningsapplikation
Carl Åberg
Kevin Bondesson
Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Detta examensarbete presenterar utvecklingen av en webbaserad lokalbokningsapplikation för universitet och högskolor. Bakgrunden till projektet är att det befintliga bokningssystemet TimeEdit, som används vid Chalmers Tekniska Högskola, upplevs som ointuitivt och erbjuder begränsade möjligheter till filtrering av lokaler, vilket kan leda till sämre resursutnyttjande.

Applikationen utvecklades med T3-stack, som kombinerar TypeScript, Tailwind, Next.js, tRPC, Prisma och NextAuth, och möjliggör ett typsäkert och välstrukturerat system. Implementerade funktioner inkluderar sökning och filtrering av lokaler baserat på kapacitet, utrustning, geografisk närhet, favoritmarkering samt en gruppfunktion för delning av bokningar.

Applikationen utvärderades genom en enkätundersökning med tio deltagare. Resultaten visade att samtliga deltagare upplevde applikationen som användarvänlig och att majoriteten bedömde den som lika bra eller bättre än TimeEdit inom de flesta kategorier. Projektet visar att det är möjligt att utveckla ett användarvänligt alternativ till redan etablerade bokningssystem.

Nyckelord: Webbutveckling, T3-stack, Next.js, TypeScript

Förord

Detta examensarbete genomfördes under vårterminen 2026 som en del av högskoleingenjörsprogrammet Datateknik vid Chalmers Tekniska Högskola.

Vi vill tacka vår handledare Inari Listenmaa för värdefull vägledning och återkoppling under projektets gång, samt de personer som tog sig tid att delta i enkätundersökningen.

Carl Åberg och Kevin Bondesson, Göteborg, Maj 2026

Akronymer

Nedanför listas de akronymer som används i rapporten i alfabetisk ordning:

API	Application Programming Interface
E2E	End-to-end
OAuth	Open Authorization
ORM	Object-Relational Mapping
SSO	Single Sign-On
tRPC	TypeScript Remote Procedure Call

Innehåll

Figurer

1
Inledning

Digitala bokningssystem utgör en central del av den dagliga verksamheten vid universitet och högskolor. För studenter och personal är möjligheten att snabbt och enkelt boka grupprum en viktig förutsättning för effektivt samarbete, planering, och genomförande av studier och arbete. Trots detta upplevs det befintliga bokningssystemet TimeEdit som ointuitivt och begränsat. TimeEdit används vid Chalmers Tekniska Högskola och ligger till grund för de brister som identifierats i detta projekt. Det bör noteras att rapporten är skriven utifrån detta specifika sammanhang, men liknande brister har identifierats i andra webbaserade bokningssystem [1]. TimeEdit saknar bland annat möjligheten till att få en tydlig överblick av lokaler som passar användarens behov, vilket kan påverka både effektiviteten och användarupplevelsen negativt.

Användarupplevelsen i digitala system spelar en central roll för hur effektivt användare kan utföra uppgifter i systemet. Forskning inom människa-datorinteraktion visar att visuell komplexitet och svårnavigerade gränssnitt ökar den kognitiva belastningen, vilket gör det svårare och mer tidskrävande för användare att navigera systemet [2]. I kontexten av lokalbokningar innebär detta att ett svårnavigerat gränssnitt inte enbart försämrar användarupplevelsen utan även påverkar hur effektivt användare genomför bokningar.

1.1 Bakgrund

Att boka grupprum är en återkommande del av vardagen för både studenter och personal. Bokningssystemet används frekvent och möjliggör effektiva grupparbeten, möten och undervisning. Eftersom systemet används regelbundet av de flesta studenter och personal, uppmärksammas även mindre brister i dess utformning. Den nuvarande bokningsprocessen i TimeEdit består av flera steg där användaren behöver navigera genom olika sidor och vyer för att nå bokningsfunktionen och slutligen få en överblick över tillgängliga lokaler. Liknande brister har identifierats i andra webbaserade bokningssystem, där användare upplever systemet som komplicerat när bokningsprocessen är utspridd över flera sidor [1]. Därefter kan användare enbart filtrera lokaler genom att välja campus, byggnad, datum och tid. Information som lokalens utrustning och dess kapacitet finns visserligen tillgänglig före bokning, men systemet saknar möjlighet att filtrera och söka efter lokaler baserat på specifika krav, såsom tillgång till projektor eller en viss minsta kapacitet. Avsaknaden av dessa filtreringsmöjligheter kan leda till ineffektiva bokningsval. Exempelvis kan mindre grupper välja att boka en större lokal med större kapacitet än nödvändigt,

enbart för att det uppfyller gruppens utrustningskrav. Detta kan i sin tur minska tillgängligheten för större grupper som faktiskt är i behov av lokaler med högre kapacitet. Systemets utformning påverkar alltså inte enbart användarupplevelsen negativt, utan även hur effektivt resurser fördelas.

Det är samtidigt viktigt att poängtera att det nuvarande bokningssystemet i TimeEdit är en del av ett större system som hanterar fler funktioner än enbart lokalbokning, exempelvis schemavisning eller administrativa uppgifter. För organisationer är det oftast önskvärt att ha en central plattform som hanterar flera olika funktioner. Samtidigt bidrar detta till att bokningsprocessen blir mer tidskrävande. Studenter behöver navigera genom flera sidor och passera alternativ som sällan är relevanta för dem.

Eftersom TimeEdit är ett slutet system utvecklades istället en fristående applikation med fokus på enbart lokalbokning. Detta möjliggör ett enklare gränssnitt anpassat efter användarnas behov, utan de begränsningar som ett större system medför.

1.2 Syfte

Syftet med projektet är att undersöka hur en webbaserad lokalbokningsapplikation kan utformas för att förbättra användarupplevelsen och effektivisera bokningsprocessen inom universitet och högskolor jämfört med befintliga lösningar.

1.3 Mål

Målet är att utveckla en fungerande och användarvänlig prototyp med följande funktioner.

- Visning av tillgängliga lokaler samt möjligheten att kunna favoritmarkera dessa för snabbare åtkomst vid framtida bokningar.
- Ett användarvänligt och intuitivt gränssnitt för sökning och bokning av lokaler.
- Ett snabbt och effektivt bokningsflöde.
- Filtrering av lokaler exempelvis baserat på kapacitet eller utrustning.
- Föreslå närliggande lokaler till användaren baserat på användarens nuvarande position.

1.4 Avgränsningar

Arbetet avgränsas till utvecklingen av en fungerande webbapplikationsprototyp för lokalbokningar. Projektet omfattar inte integration med befintliga system inom universitet och högskolor, såsom autentiseringstjänster. Webbapplikationen kommer därför inte att hantera registrering eller skapande av nya konton, utan det förutsätts att användaren redan har ett konto som är kopplat till Google. Detta motiveras av att autentiseringen vanligtvis hanteras genom centrala system inom organisationen och ligger utanför projektets omfattning och mål.

Projektet inkluderar en användarstudie med ett begränsat antal deltagare. Användarstudien kommer därför att genomföras i mindre skala och syftar främst till att få återkoppling på användarvänlighet, struktur och funktionalitet, men även för att identifiera eventuella brister och förbättringsområden.

2
Metod

I detta kapitel beskrivs de metoder och arbetsprocesser som låg till grund för projektet. Kapitlet redogör för valt arbetssätt, hur krav identifierades, hur arbetet planerades samt hur utvärderingen av applikationen planerades att genomföras.

2.1 Arbetssätt

Utvecklingen genomfördes med ett iterativt arbetssätt, där funktionalitet utvecklades, testades och utvärderades i flera steg under projektets gång. Ett iterativt arbetssätt bedömdes väl lämpat eftersom projektgruppen bestod av två personer, vilket möjliggjorde snabb återkoppling och flexibilitet att anpassa arbetet efter behov. Detta ligger nära de arbetssätt som används inom agil utveckling, vilket lämpar sig väl för mindre projektgrupper [3].

Iterationerna utgick främst från Gantt-schemat (se figur B.1), som bland annat består av databasdesign, backendutveckling och frontendutveckling, där varje fas utgjorde ett naturligt steg i processen. Varje ny funktion jämfördes mot de krav som identifierats under planeringsfasen innan den integrerades med övriga delar av systemet. Vid identifiering av förbättringsområden åtgärdades dessa löpande.

2.2 Planering

Inför projektet skapades ett Gantt-schema för att planera och strukturera arbetet (se B.1). Schemat låg till grund för projektets genomförande och fungerade som referens under projektets gång. Arbetet delades in i åtta faser: (1) efterforskning och planeringsrapport, (2) UX-design och UI-mockups, (3) systemdesign och databasmodell, (4) frontendutveckling, (5) backendutveckling, (6) tester och användarutvärdering, (7) förbättringar, samt (8) slutrapport och presentation. För varje fas definierades en uppskattad tidsram baserad på fasens omfattning och hur komplex den bedömdes vara, med målet att samtliga faser skulle slutföras inom tidsramen för projektet.

2.3 Kravinsamling

Kraven för webbapplikationen identifierades utifrån systemets syfte att möjliggöra hantering av lokalbokningar. Den befintliga lösningen TimeEdit som används på Chalmers analyserades genom praktisk användning samt granskning av funktioner och användarflöden. Fokus låg på att identifiera centrala moment i bokningsprocessen samt begränsningar i TimeEdit. Utifrån analysen identifierades centrala use cases som låg till grund för applikationens funktionella krav. Kraven prioriterades utifrån hur centrala de bedömdes vara för bokningsprocessen, där grundläggande funktioner såsom sökning, filtrering och bokning prioriterades före mer avancerade funktioner. För fullständig kravspecifikation se appendix A.1.

2.4 Testning

Testning planerades genom manuella funktionstester och även i mån av tid E2E-tester (End-to-End). Funktioner testades kontinuerligt under projektets gång genom att verkliga användarscenarier simulerades, exempelvis bokningsflöden, filtrering och inloggning.

2.5 Användarenkät

För att utvärdera applikationen valdes en enkätundersökning, som genomfördes efter att en fungerande version av applikationen var tillgänglig. En enkät bedömdes lämplig som utvärderingsmetod eftersom det är den vanligaste metoden för att mäta och utvärdera användarvänligheten i mjukvaruprojekt [4]. Enkäten användes för att mäta användarvänlighet och jämföra applikationen med TimeEdit inom specifika kategorier och i sin helhet, samt ge återkoppling på de nya funktionerna. Enkäten gav även deltagarna möjlighet att rapportera buggar och problem, och resultaten låg till grund för eventuella förbättringar av applikationen.

3
Teknisk bakgrund

Teknisk bakgrund redogör för de teknologier och verktyg som ligger till grund för utvecklingen av applikationen. Syftet är att ge en förståelse för de valda lösningarna och varför dessa bedömdes lämpliga för projektet.

3.1 T3-Stack

T3-stack är en webbapplikationsstack som fokuserar på enkelhet, modularitet och typsäkerhet. Den kombinerar flera teknologier, däribland Next.js, TypeScript, tRPC (TypeScript Remote Procedure Call), Tailwind, Prisma och NextAuth [5]. Stacken är utformad så att samtliga verktyg ska fungera väl tillsammans, vilket minskar tiden för initial konfiguration och möjliggör en snabbare start på utvecklingsfasen av projektet.

En central egenskap hos T3-stack är att den möjliggör E2E typsäkerhet mellan front- och backend genom tRPC och TypeScript. Det innebär att typer delas mellan klient och server, vilket minskar risken för fel i kommunikationen mellan applikationens olika lager och minskar behovet av att manuellt definiera typer som i vanligt REST-API (Application Programming Interface). TypeScript valdes framför JavaScript eftersom statisk typning har visats bidra till förbättrad kodkvalitet [6]. T3-stack valdes för projektet då den erbjuder en modern utvecklingsmiljö som tar minimal tid att sätta upp.

3.2 Next.js

Next.js är ett React-baserat ramverk för utveckling av webbapplikationer [7]. Ramverket stödjer både server-side rendering och client-side rendering, vilket möjliggör optimerad prestanda och därmed en förbättrad användarupplevelse. Server-side rendering innebär att innehåll genereras på servern innan det skickas till klienten, och där tar webbläsaren emot färdigt innehåll som kan visas direkt. Detta skiljer sig från client-side rendering där webbläsaren först laddar ner JavaScript och sedan genererar innehållet, vilket kan resultera i längre laddningstider. Server-side rendering kan därmed bidra till kortare laddningstider och förbättrad prestanda [8].

Next.js har även inbyggt stöd för routing och serverfunktioner, vilket möjliggör hantering av både frontend och viss serverlogik inom samma ramverk [7]. I projektet används Next.js som grund för frontendutvecklingen, där React-komponenter

används för att bygga användargränssnittet och Next.js serverfunktioner hanterar routing och serverrelaterad logik.

3.3 tRPC

tRPC är ett bibliotek som möjliggör typsäker kommunikation mellan front- och backend. Genom att dela TypeScript-typer mellan klient och server upptäcks fel i kommunikationen direkt, vilket minskar risken för svårupptäckta buggar [9].

I projektet används tRPC för samtliga API-anrop mellan frontend och backend, vilket garanterar typsäker kommunikation genom hela applikationen.

3.4 Zod

Zod är ett valideringsbibliotek för TypeScript, som använder scheman som definieras för att beskriva den förväntade strukturen på inkommande data [10]. Dessa scheman används för att validera inkommande data och säkerställa att API-anrop uppfyller fördefinierade krav, till exempel en specifik datatyp eller tillåtna värden inom ett intervall.

I projektet används Zod tillsammans med tRPC för att validera inkommande data i backend, vilket säkerställer att felaktiga anrop identifieras och hanteras innan de når applikationens affärslogik.

3.5 Prisma

Prisma är ett ORM (Object-Relational Mapping) för TypeScript som förenklar kommunikationen med databasen [11]. Ett ORM möjliggör interaktion med databasen genom kod, istället för ren SQL, vilket gör modellerna mer läsbara och minskar risken för fel. I Prisma definieras ett schema som beskriver databasens modeller och relationer, vilket används både för att generera databasmigreringar och för att skapa en typad databasmodell.

Prisma valdes i projektet främst på grund av dess möjlighet att automatiskt generera TypeScript-typer utifrån schemat, vilket möjliggör typsäker databashantering och minskar behovet av att manuellt definiera typer [11].

3.6 PostgreSQL

PostgreSQL är ett databashanteringssystem som används för att lagra och hantera data. Databasen stödjer relationer mellan olika datatyper, vilket lämpar sig väl för applikationer där data är sammankopplad.

I projektet finns tydliga relationer mellan den data som lagras i databasen, Exempelvis är en bokning kopplad till både en användare och en lokal under ett specifikt tidsintervall. Genom PostgreSQL kan dessa relationer modelleras med hjälp av tabeller på ett strukturerat sätt [12].

3.7 NextAuth

NextAuth är en autentiseringslösning för Next.js som förenklar hantering av sessionshantering och inloggning [13]. Biblioteket stödjer flera leverantörer av autentiseringstjänster, bland annat Google OAuth (Open Authorization), och hanterar sessioner utan att känslig data som lösenord behöver lagras i applikationens databas.

Efter inloggning skapas en session som används i applikationens backend för att identifiera användaren och skyddar åtkomst i applikationens backend [13].

3.8 Google OAuth

OAuth 2.0 är ett auktoriseringsprotokoll som möjliggör för utvecklare att begränsa mängden känsligt data som hanteras i systemet. Detta genom att ha ett konto hos en tredjepartsleverantör [14].

I projektet används Google OAuth som autentiseringslösning, vilket innebär att användare loggar in med sitt Google-konto istället för att skapa ett nytt konto i applikationen. Valet motiveras med att projektet syftar till att efterlikna ett verkligt bokningssystem i skolmiljö, där inloggning ofta sker via SSO (Single Sign-On). Eftersom tillgång till ett sådant system saknades valdes Google OAuth som en motsvarande tredjepartslösning.

4

Genomförande

Detta kapitel beskriver hur applikationen utformades och implementerades. Beskrivningen följer en logisk ordning utifrån systemets uppbyggnad, arkitektur, databas, backend och slutligen frontend. Kapitlet tar även upp centrala funktioner i applikationen.

4.1 Systemdesign

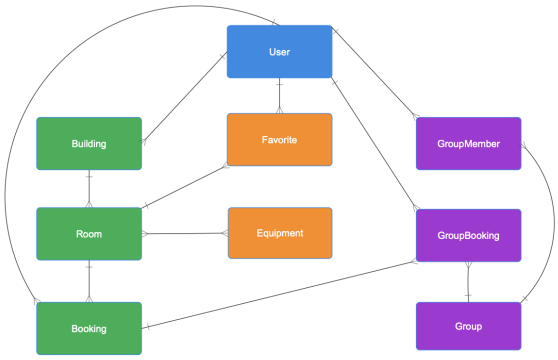
Systemets uppbyggnad baseras på en klient-server-arkitektur, där front- och backend är tydligt separerade med olika ansvarsområden. Designen utgår från de krav som identifierades i planeringsfasen.

4.1.1 Databas

Databasen ansvarar för lagring av applikationens data. En PostgreSQL databas används för att lagra information om lokaler, bokningar och användare tillsammans med Prisma som används för att förenkla kommunikationen med databasen.

4.1.2 Databasmodell

Databasmodellen består av nio modeller: (1) `User`, (2) `Room`, (3) `Building`, (4) `Booking`, (5) `Favorite`, (6) `Equipment`, (7) `Group`, (8) `GroupMember` och (9) `GroupBooking`. Figur 4.1 visar samtliga modeller och hur de relaterar mellan varandra. Dessa relationer är centrala för applikationens funktionalitet.



Figur 4.1: ER-diagram för databasmodellerna.

En `Building` kan innehålla flera `Room` och varje rum är kopplat till en specifik byggnad genom en `foreign key`. Denna relation möjliggör avståndsberäkningar mellan användare och en lokal, eftersom koordinaterna lagras i `Building`-modellen.

`Booking`-modellen kopplar samman en `User` och ett `Room` under ett specifikt tidsintervall. Denna relation är grundläggande för bokningslogiken i backend, där flera kontroller utförs för att säkerställa att affärslogiken följs.

`Equipment` har en many-to-many relation med `Room`, vilket innebär att en lokal kan ha flera olika typer av utrustning och att samma utrustning kan förekomma i flera lokaler. Denna relation möjliggör filtrering av lokaler baserat på utrustning.

Användare tilldelas en roll via `Role` enumet, antingen `USER` eller `ADMIN`, vilket styr åtkomst till administratörssidan tillsammans med tillhörande funktioner.

Nedanstående kodexempel visar ett utdrag ur Prisma schemat med de mest centrala modeller för applikationen.

```
model User {
    id        Int        @id @default(autoincrement())
    email     String     @unique
    role      Role       @default(USER)
    bookings  Booking[]
    favorites Favorite[]
}

model Room {
    id         Int         @id @default(autoincrement())
    name       String      @unique
    capacity   Int
    buildingId Int
    bookings   Booking[]
    equipments Equipment[]
    favorites  Favorite[]

    building Building @relation(fields: [buildingId], references: [id])
}

model Booking {
    id        Int      @id @default(autoincrement())
    userId    Int
    roomId    Int
    startTime DateTime
    endTime   DateTime

    user User @relation(fields: [userId], references: [id])
    room Room @relation(fields: [roomId], references: [id])
}
```

4.1.3 Backend

Backend ansvarar för applikationens affärslogik samt hantering av data. I detta lager hanteras förfrågningar från frontend som kräver åtkomst till eller uppdatering av data. Affärslogiken innefattar bland annat kontroller för att förhindra överlappande bokningar samt begränsningar för total bokningstid per vecka.

Kommunikationen mellan front- och backend sker via tRPC, vilket möjliggör typsäker kommunikation mellan applikationens lager [9]. Backend är strukturerat med tRPC-routrar som hanterar API-anrop från frontend och validerar inkommande data med hjälp av Zod-scheman innan hanteringen av anropet sker [10].

För endpoints som kräver autentisering hämtas användarens unika ID från deras session i backend. Detta innebär att känslig information aldrig skickas direkt från klienten, utan hanteras i backend, vilket bidrar till ökad säkerhet i applikationen.

För att hantera olika behörighetsnivåer används tRPCs stöd för `procedures`. `publicProcedure` används för endpoints som är tillgängliga utan autentisering, `protectedProcedure` kräver att användaren är inloggad och används för endpoints som bland annat hanterar bokning eller avbokning av lokaler, medan `adminProcedure` är begränsade till administratörer och används för endpoints som hanterar att lägga till eller ta bort lokaler.

Nedanstående kodexempel visar en endpoint för att avboka. Endpointen kräver att användaren är inloggad samt att ett boknings-id av korrekt typ skickas med från frontend. Därefter hämtas användarens id från sessionen och används tillsammans med boknings-id som parametrar till servicemetoden som ansvarar för att ta bort bokningen från databasen.

```
cancel: protectedProcedure
    .input(z.object({ id: z.number() }))
    .mutation(async ({ ctx, input }) => {
      const userId = parseInt(ctx.session.user.id);
      return cancelBooking(input.id, userId);
    }),
```

Servicemetoderna kommunicerar direkt med databasen via Prisma och innehåller applikationens affärslogik. Nedanstående kodexempel visar avbokningsservicen som anropas från cancel-endpointen ovan. Metoden tar emot bokningens- och användarens-id och skickar därefter en förfrågan till databasen där båda parametrarna måste matcha, vilket säkerställer att endast användaren som utfört bokningen kan avboka den.

```
export async function cancelBooking(bookingId: number, userId: number) {
  return db.booking.delete({
    where: {
      userId: userId,
      id: bookingId,
    },
  });
}
```

Samma mönster används för samtliga endpoints i applikationen. För bokning, favoritmarkering, och grupphantering. Varje endpoint validerar indata och anropar den tillhörande servicemetoden. Affärslogiken för applikationen återfinns i dessa servicemetoder som till exempel utför kontroller innan bokningen sparas i databasen. Överlappande bokningar förhindras genom att kontrollera om lokalen redan är bokad under det begärda tidsintervallet, samt att användaren inte har en annan aktiv bokning under samma tid. Utöver detta begränsas den totala bokningstiden till åtta timmar per vecka för varje användare, vilket motsvarar den gräns som observerades vid analysen av TimeEdit.

4.1.4 Frontend

Frontend utgör klientdelen av applikationen och ansvarar för användargränssnittet samt dess logik. Det är i detta lager som användaren interagerar med systemet genom funktioner såsom inloggning, sökning, filtrering och bokning av lokaler.

Frontend är uppbyggt av UI-komponenter som enbart ansvarar för rendering av gränssnittet, samt klientkomponenter som fungerar som en brygga mellan användargränssnittet och backend. För att separera logik från gränssnittet används hooks, som hanterar logik och tillstånd samt kommunikationen med backend [15]. När en användare exempelvis klickar på bokningsknappen skickar klientkomponenten anropet till en hook, som i sin tur skickar API-anropet till backend via tRPC. Hooks använder `useState` för att uppdatera komponenter i gränssnittet när deras värden ändras, samt `useEffect` för att hantera sidoeffekter.

Gränssnittet är uppbyggt av återanvändbara komponenter som separerar olika delar av applikationens funktionalitet. Applikationen är organiserad i flera vyer, bland annat en startsida som visar tillgängliga lokaler, favoriter och användarens bokningar, samt en separat vy för att visa samtliga lokaler med funktionalitet för filtrering.

Nedan visas ett förenklat exempel på en `RoomCard` komponent som visar lokalens namn och anropar en funktion när bokningsknappen klickas.

```
export default function RoomCard({ room, onBook }) {
return (
    <div>
        <h2>{room.name}</h2>
        <button onClick ={() => onBook(room.id)}>
        Boka
        </button>
    </div>
  );
}
```

Samma komponentbaserade struktur används genomgående i applikationen där varje vy och funktionalitet är separerad i egna komponenter. För att illustreras hur logik separeras från rendering visas nedan ett förenklat exempel på `useBookingDialog`, en hook som hanterar tillstånd och logik för bokning och avbokning.

```
  const [openBooking, setOpenBooking] = useState(false);
  const [selectedRoomId, setSelectedRoomId] = useState<number | null>(null);

  const handleBook = (roomId: number) => {
    setSelectedRoomId(roomId);
    setOpenBooking(true);
  };

  const handleConfirm = (startTime: Date, endTime: Date) => {
    if (selectedRoomId === null) return;
    bookRoom({ roomId: selectedRoomId, startTime,endTime},
      { onSuccess: () => setOpenBooking(false),
    });
  }

    return {openBooking, handleBook, handleConfirm, error};
}
```

Hooken använder `useState` för att hantera tillstånd som vilken lokal som valts och om bokningsdialogen är öppen. När användaren klickar på bokningsknappen i `RoomCard` komponenten anropas `handleBook`, som uppdaterar tillståndet och då även gränssnittet. Vid bekräftelse anropas `handleConfirm` som skickar en bokningsbekräftelse till backend via `useBookRoom` hooken, vilket visar hur logik är separerad i lager även i frontend.

4.2 Implementation

Detta avsnitt beskriver hur applikationens centrala funktioner implementerats. Implementationen utgår från den arkitektur som presenterats i föregående avsnitt.

4.2.1 Autentisering

Planen var initialt att använda universitetets SSO-lösning för autentisering, men eftersom tillgång till denna tjänst uteblev under projektets genomförande valdes Google OAuth i kombination med NextAuth som autentiseringslösning, vilket beskrivs i 3.7.

NextAuth konfigurerades med Google som leverantör genom att registrera applikationen i Google Cloud Console och hantera OAuth-nycklarna för applikationen. När en användare loggar in omdirigeras den till Googles inloggningssida, samtidigt hanterar NextAuth callback och skapar en session som lagras i backend. Sessionen används sedan i backend för att identifiera användaren och styra dess åtkomst, som beskrivits i 4.1.3.

4.2.2 Bokningsflöde

När en användare bekräftar en bokning i frontend skickas ett API-anrop till backend via tRPC. Anropet tas emot av en endpoint skyddad av `protectedProcedure`, vilket säkerställer att endast autentiserade användare kan genomföra bokningar. Därefter skickas data vidare till bokningsservicen som applicerar affärslogiken beskriven i avsnitt 4.1.3. Om förfrågan godkänns registreras bokningen i databasen och därefter uppdateras användargränssnittet.

4.2.3 Filtrering och sökning

Användaren kan söka efter lokaler baserat på namn samt filtrera resultat utifrån kapacitet, byggnad, utrustning och datum samt tid. När användaren justerar ett filter uppdateras filterparametrarna i frontend och appliceras direkt på den redan hämtade listan av lokaler. Resultatet uppdateras dynamiskt i gränssnittet utan att nya API-anrop behöver skickas till backend, vilket bidrar till en responsiv användarupplevelse.

4.2.4 Favoriter

En lokal kan markeras eller avmarkeras som favorit via en knapp på varje lokalkort i gränssnittet. När användaren klickar skickas ett API-anrop till backend för att lägga till eller ta bort från användarens lista av favoriter.

4.2.5 Administratörssida

Eftersom åtkomst till Chalmers befintliga lokalsystem nekades utvecklades en administratörssida för att manuellt lägga till data. Via gränssnittet kan administratörer lägga till byggnader med namn samt koordinater i form av latitud och longitud, som används för att beräkna avståndet till användaren. Det är även möjligt att lägga till och ta bort lokaler, koppla dem till en befintlig byggnad samt ange kapacitet och utrustning.

Administratörssidan och backendlogiken för att hantera byggnader och lokaler skyddas av `adminProcedure`, vilket säkerställer att endast användare som är registrerade som administratörer i databasen har tillgång till denna sida och dessa operationer.

4.2.6 Geografisk position

Den geografiska funktionaliteten i applikationen används för att beräkna avståndet från användaren till lokaler i närheten. Syftet är att erbjuda användare ett snabbokningsalternativ om de befinner sig på eller i närheten av campus.

Efter användarens godkännande hämtas den geografiska positionen genom webbläsaren. Avståndet mellan användaren och byggnader beräknas genom Haversineformeln, som används för att beräkna det kortaste avståndet mellan två punkter [16]. Förslag på närliggande lokaler visas endast om användaren befinner sig inom en kilometer från campus, eftersom det annars bedöms det irrelevant att visa avstånd.

4.2.7 Gruppfunktionalitet

Efter att användarenkäten genomförts identifierades ett önskemål om möjligheten att utföra en bokning och samtidigt göra den synlig för andra medlemmar i en grupp. Funktionen bedömdes bidra till en mer komplett bokningsapplikation och implementerades som ett tillägg till den ursprungliga planen.

Funktionen möjliggör för användare att skapa en grupp eller gå med i en befintlig via en inbjudningskod. När en användare delar en befintlig bokning med en grupp blir den synlig för samtliga medlemmar, vilket underlättar logistiken vid grupparbeten och projekt. I backend implementerades routrar och services för att hantera skapandet av grupper, medlemskap och delning av bokningar, samtliga skyddade av `protectedProcedure`.

4.2.8 Länkdelning

Utöver delning inom grupper kan en bokning även delas via en länk. Via delningssymbolen i bokningskortets högra hörn kan användaren kopiera en länk med information om bokningen eller dela den direkt med befintliga grupper. När länken delas i applikationer som stödjer Open Graph-protokollet, exempelvis Discord eller WhatsApp genereras en förhandsvisning automatiskt med namn på lokalen, datum och tid, vilket innebär att mottagare inte behöver klicka på länken för att se information. Klickar mottagaren däremot på länken visas en sida med all information om bokningen, alltså lokalens namn, byggnad, datum, tid, kapacitet, utrustning och vem som gjorde bokningen. Sidan är tillgänglig utan inloggning, vilket gör det enkelt att dela en bokning.

4.3 Användarenkät

Enkäten genomfördes digitalt efter att en fungerande version av applikationen var tillgänglig. Studenter, personal vid Chalmers och bekanta tillfrågades att testa applikationen och enkäten besvarades av totalt tio deltagare. De som deltog i undersökningen fick använda applikationen och besvara frågor om användarvänlighet, jämförelse med TimeEdit samt upplevelsen av de nya funktionerna.

4.4 Avvikelser från plan

Under projektets gång visade sig implementationsfasen ta längre tid än planerat. Detta medförde att planerade E2E-tester uteblev, till förmån ytterligare funktionalitet och manuella funktionstester, som genomfördes löpande under projektets gång genom att simulera verkliga användarscenarier, såsom att boka eller avboka en lokal.

5

Resultat

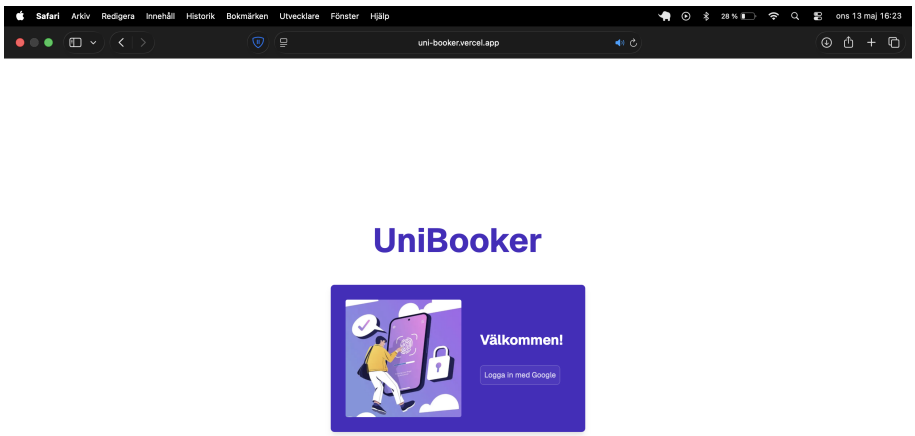
I detta kapitel presenteras resultatet av projektet i förhållande till de mål som identifierades i inledningen, samt applikationens gränssnitt och resultaten av användarenkäten.

Projektets mål var att utveckla en fungerande prototyp med funktioner som förenklar och effektiviserar lokalbokningsprocessen. Samtliga funktionella krav som identifierades i planeringsfasen implementerades, däribland sökning och filtrering baserat på kapacitet med mera. Utöver de ursprungliga kraven implementerades även en gruppfunktion baserat på återkoppling från enkäten.

5.1 Applikation och gränssnitt

Resultatet av projektet är en webbaserad lokalbokningsapplikation som möjliggör för studenter att enkelt söka, filtrera och boka lokaler via ett intuitivt gränssnitt.

Den första sidan användaren möts av är inloggningssidan (se figur 5.1), där autentisering sker via Google OAuth. Detta innebär att ingen manuell registrering krävs, utan endast ett befintligt Google-konto.

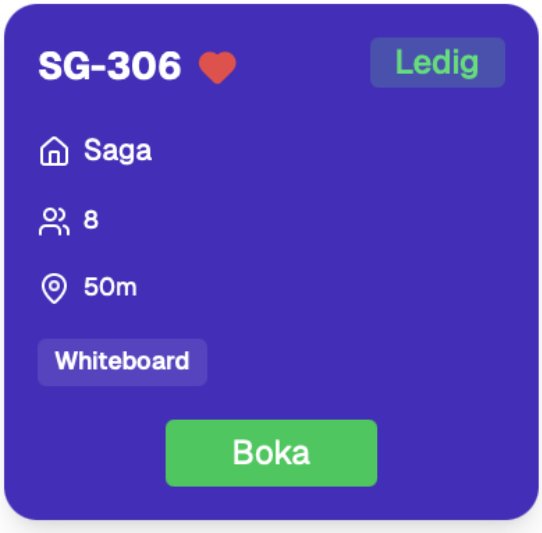


Figur 5.1: Inloggningssida där användaren autentiseras via Google OAuth.

Efter inloggning presenteras en personlig startsida (se figur 5.2) med användarens aktiva bokningar, förslag på tillgängliga lokaler i närheten, förutsatt att användaren godkänt platsåtkomst i webbläsaren, favoritmarkerade lokaler och slutligen längst ner finns möjligheten att se bokade lokaler som delats med grupper. En navbar finns tillgänglig på samtliga sidor med navigering tillbaka till startsidan samt en utloggningsknapp.

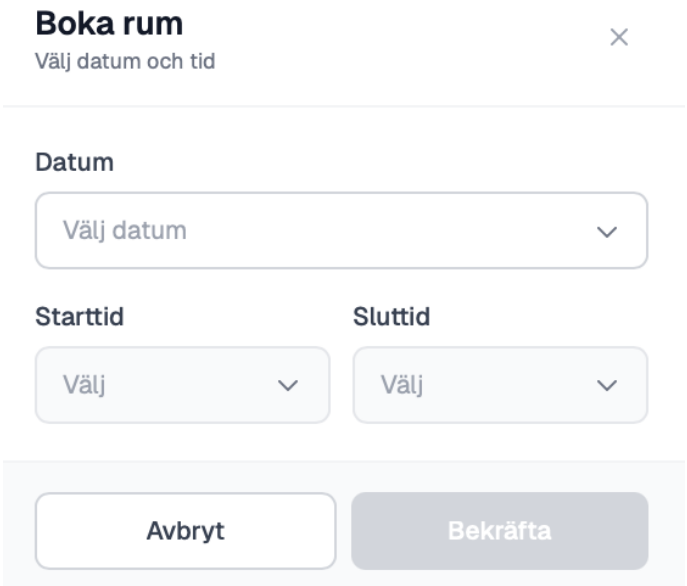
Figur 5.2: Startsidan med aktiva bokningar, lediga lokaler i närheten och favoriter.

Varje lokalkort (se figur 5.3) visar lokalens namn, byggnad, kapacitet och utrustning samt om lokalen är ledig just nu. Användaren kan favoritmarkera en lokal via hjärtsymbolen och om användaren befinner sig i närheten visas även avståndet.



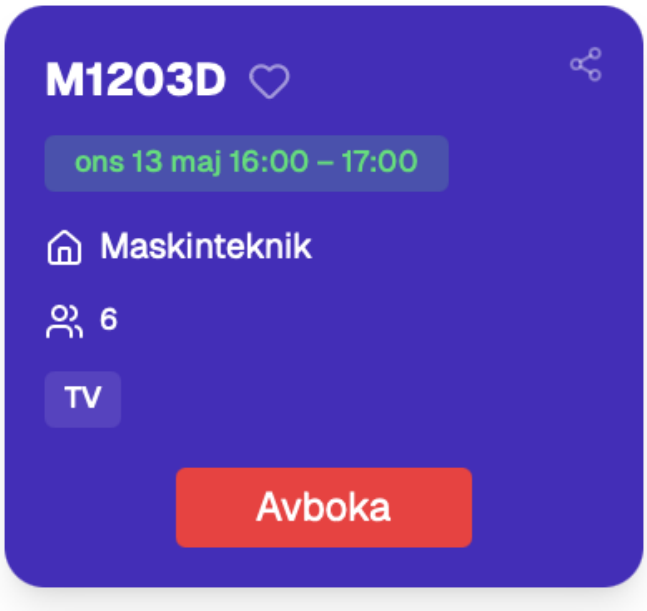
Figur 5.3: Ett lokalkort med relevant information och möjligheten att boka.

För att boka en lokal klickar användaren på "Boka", varpå en dialog öppnas där datum, starttid, och sluttid väljs (se figur 5.4). När bokningen bekräftats visas den under bokningar på startsidan.



Figur 5.4: Dialogrutan för att välja datum, starttid och sluttid.

En aktiv bokning visas som ett bokningskort på startsidan (se figur 5.5) med information om lokal, datum och tid. Användaren kan avboka genom att trycka på "Avboka", varpå en bekräftelsedialog visas (se figur 5.6).

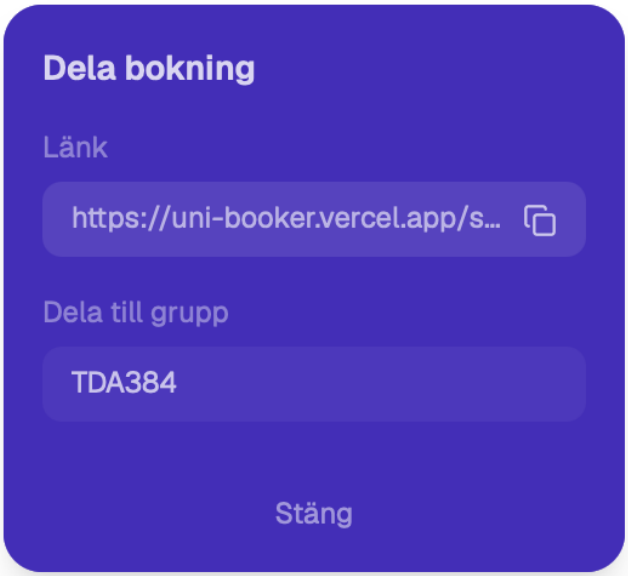


Figur 5.5: Bokningskort med information om den aktiva bokningen samt möjligheten att avboka.

När användaren trycker "Avboka" öppnas en dialog för att bekräfta avbokningen (se figur 5.6).

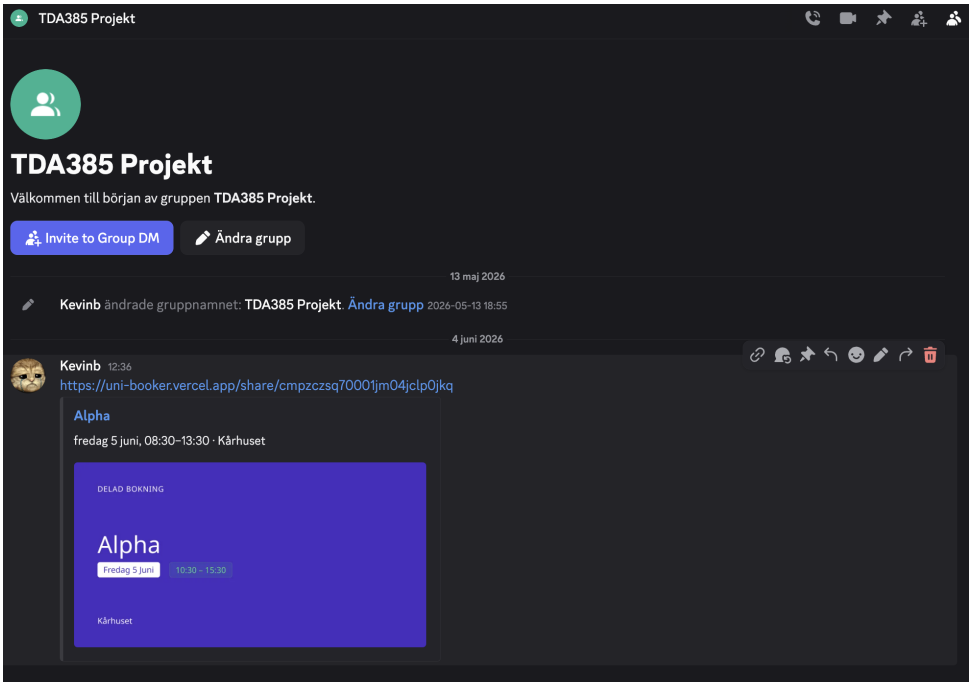
Figur 5.6: Bekräftelsedialog för avbokning

Via en delningssymbol i bokningskortets högra hörn (se figur 5.5) kan användaren kopiera en länk med information om bokningen eller dela den direkt med en befintlig grupp (se figur 5.7).



Figur 5.7: Alternativ för att dela en bokning via länk eller till en grupp

Figur 5.8 visar en förhandsvisning av en delad bokningslänk som visas automatiskt när länken delas i applikationer som stödjer Open Graph, exempelvis Discord eller WhatsApp.



Figur 5.8: Förhandsvisning av bokningsinformation.

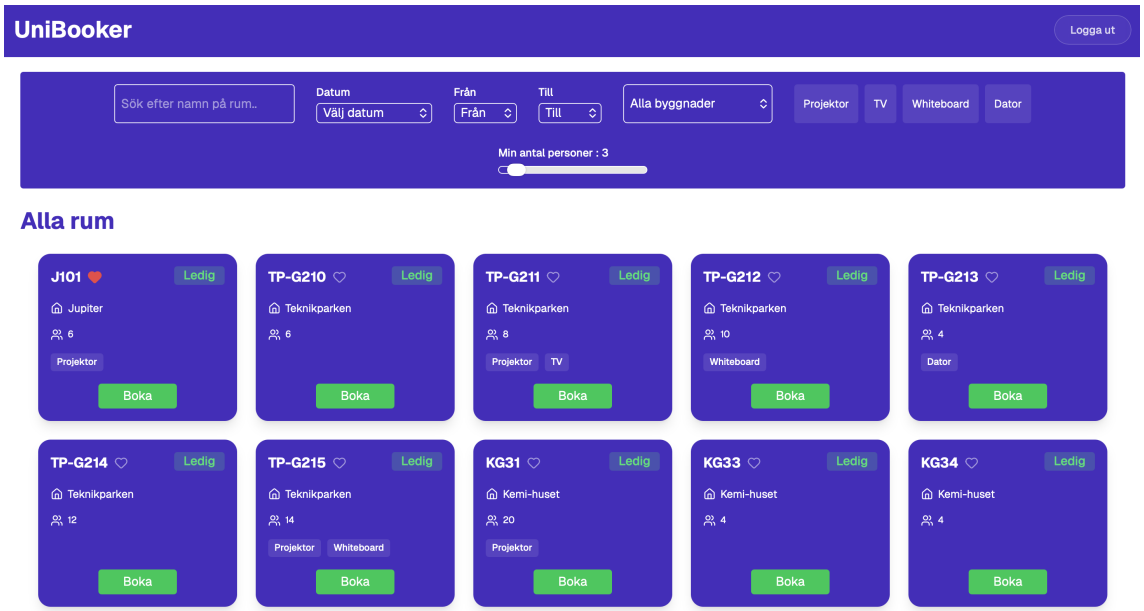
Figur 5.9 visar innehållet i bokningslänken med mer detaljer om den aktuella bokningen.

Figur 5.9: Bokningslänken som innehåller information för bokningen.

Gruppfunktionen finns längst ner på startsidan (se figur 5.10). Användaren kan skapa en ny grupp eller gå med i en befintlig via en inbjudningskod. Inom gruppen kan medlemmar dela bokningar, vilket underlättar att samordna grupparbeten.

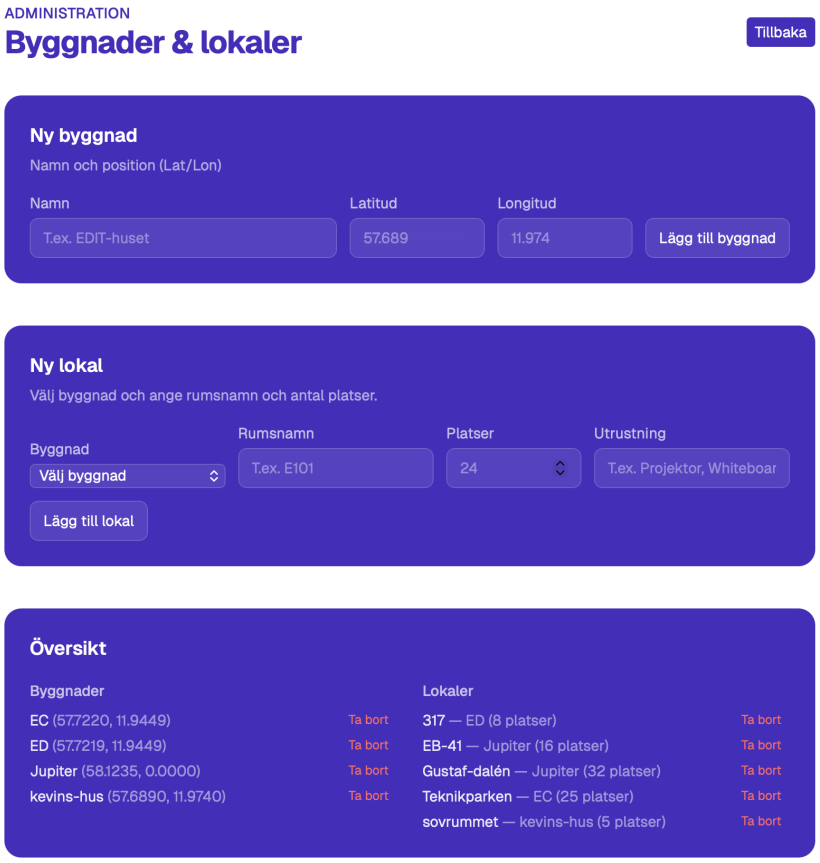
Figur 5.10: Översikt över grupper och dess bokningar.

Via knappen “Visa alla rum” navigerar användaren till en vy som visar samtliga lokaler i systemet (se figur 5.11). Här kan lokaler filtreras efter namn, utrustning, datum, tid och kapacitet.



Figur 5.11: Översikt över alla lokaler, tillsammans med filteralternativ.

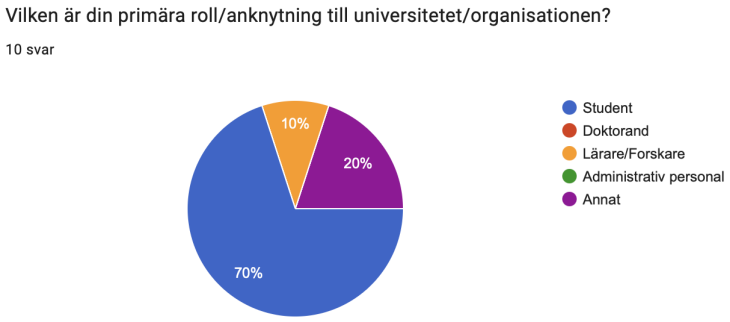
Användare registrerade som administratörer i databasen har tillgång till en administratörssida (se figur 5.12) där byggnader och lokaler kan läggas till, redigeras och tas bort.



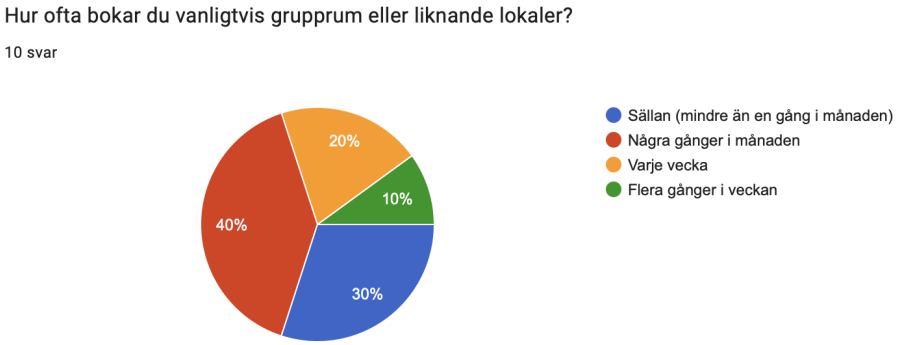
Figur 5.12: Administratörssidan för hantering av byggnader och lokaler.

5.2 Enkätresultat

För att utvärdera applikationen genomfördes en enkät som besvarades av tio deltagare med varierande roller (se figur 5.13), vilket ger en indikation på hur applikationen upplevdes. Av de som svarade var sju studenter, en lärare/forskare och två personer som faller under kategorin annat. Enligt figur 5.14 bokar en majoritet av deltagarna lokaler några gånger i månaden eller oftare, vilket innebär att de som svarade på enkäten har erfarenhet av lokalbokningsprocesser.

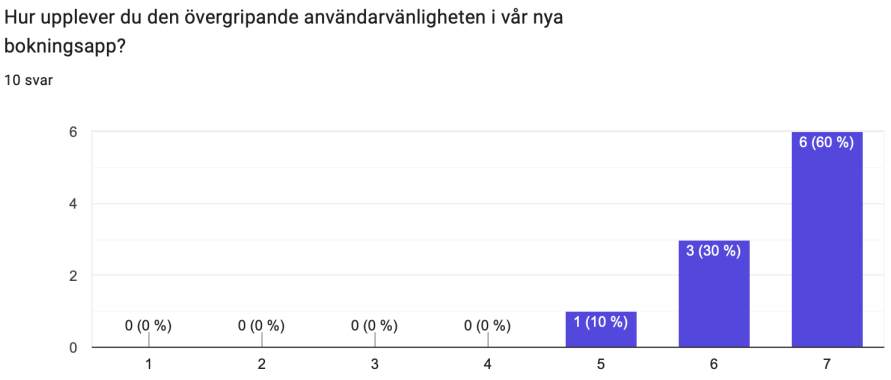


Figur 5.13: Ett cirkeldiagram med roller för deltagarna i användarenkäten.



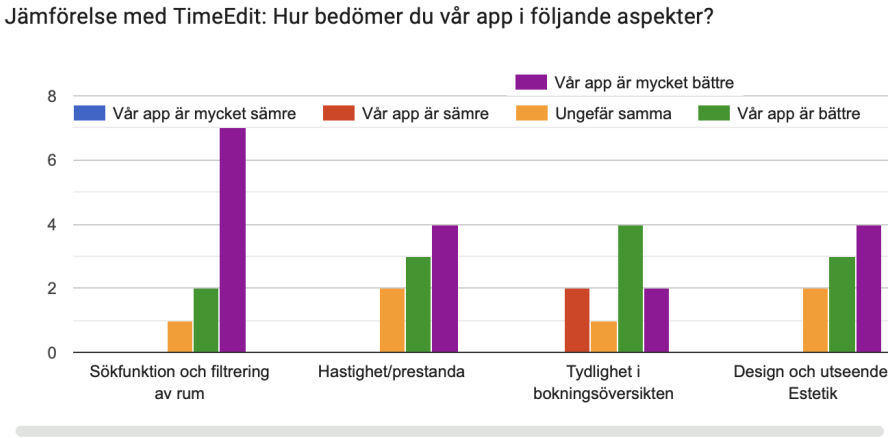
Figur 5.14: Ett cirkeldiagram som visar hur frekvent deltagarna bokar grupprum eller lokaler.

Gällande den övergripande användarvänligheten gav 60% av de deltagande i undersökningen högsta betyg (7 av 7) och 30% gav betyget precis under, vilket tyder på att applikationen upplevdes som enkel att använda (se figur 5.15).



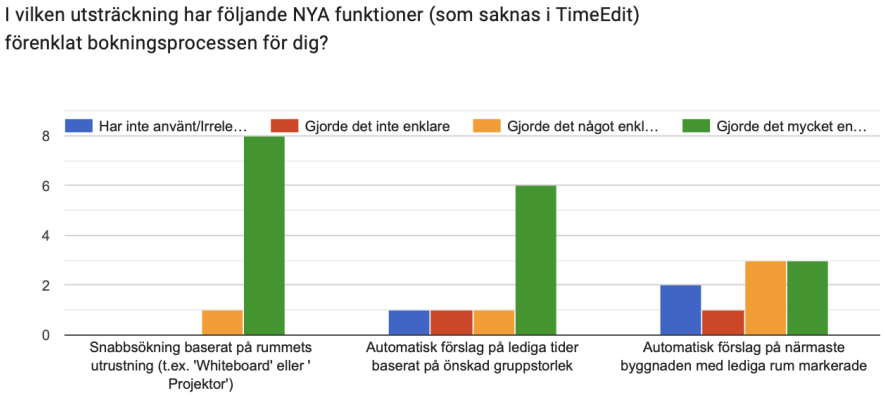
Figur 5.15: Stapeldiagram med deltagarnas betyg på applikationens användarvänlighet.

I jämförelse med TimeEdit bedömdes applikationen som lika bra eller bättre inom samtliga kategorier förutom när det gäller tydlighet i bokningsöversikten. Störst förbättring sågs inom sökfunktion och filtrering, där nio av tio ansåg att applikationen var bättre eller mycket bättre. Tydligheten i bokningsöversikten var den kategori med mest spridda omdömen, där ansåg två respondenter att applikationen presterade sämre (se figur 5.16).



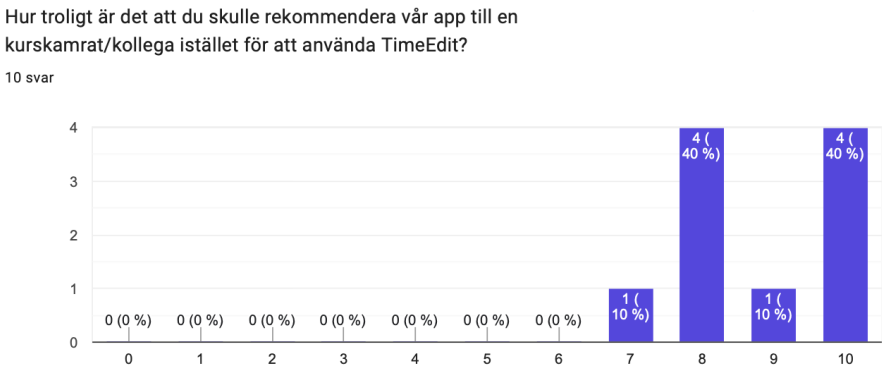
Figur 5.16: Stapeldiagram med deltagarnas betyg på centrala delar av applikationen jämfört med TimeEdit.

De nya funktionerna i applikationen upplevdes överlag som värdefulla. Snabbsökning och filtrering baserat på utrustning och kapacitet ansågs av sex respektive sju respondenter bidra till att göra bokningsprocessen enklare. Funktionen för automatiskt förslag på lediga lokaler i närheten av användare fick mer varierande omdömen jämfört med de andra funktionerna (se figur 5.17). En möjlig förklaring är att funktionen förutsätter att användaren befinner sig inom en kilometers avstånd från campus, vilket inte kommunicerades tydligt till deltagarna i enkäten. Detta kan ha påverkat upplevelsen negativt för deltagarna som testade på distans eftersom de inte fick chansen att testa denna funktionalitet.



Figur 5.17: Stapeldiagram med deltagarnas betyg på nya funktioner i applikationen.

Gällande hur sannolikt det var att respondenterna skulle rekommendera applikationen uppgav samtliga ett värde mellan sju och tio, där 40% gav högsta betyg. Detta indikerar att användarna i högre grad föredrar applikationen före TimeEdit eller liknande bokningssystem (se figur 5.18).



Figur 5.18: Stapeldiagram över hur benägna respondenterna är att rekommendera applikationen.

I övrigt framkom synpunkter om att bokningsbekräftelsen upplevdes som otydlig, problem med läsbarhet på filtreringsfunktionen på vissa webbläsare och även önskemål om funktioner som att kunna ändra befintliga bokningar, en veckovy och funktionalitet för gruppbokningar.

Baserat på den insamlade återkopplingen implementerades en gruppfunktion där användare kan skapa eller gå med i grupper via en kod. Funktionen möjliggör även delning av bokningar inom en grupp, vilket direkt kopplar till ett önskemål från enkäten.

6
Slutsats

6.1 Resumé

Syftet med projektet var att utveckla en webbaserad applikation för lokalbokning inom universitet och högskolor, med fokus på att erbjuda en tydligare överblick över tillgängliga lokaler samt möjligheten att söka och filtrera utifrån behov som kapacitet, utrustning och geografisk position. Samtliga planerade funktioner implementerades och utöver dessa tillkom även stöd för grupper samt möjligheten att dela bokningar genom en länk. Applikationen utvärderades genom en användarenkät med tio deltagare. Deltagarna fick svara på frågor om användarvänlighet, hur väl de nya funktionerna bidrog till att effektivisera bokningsflödet samt hur applikationen upplevdes i jämförelse med TimeEdit. Baserat på resultaten och återkopplingen från enkäten bedöms syftet med projektet ha uppnåtts.

6.2 Kritisk diskussion

Följande avsnitt ger en ärlig reflektion över projektets genomförande och resultat. Diskussionen omfattar de tekniska val som gjordes och deras konsekvenser, begränsningar som identifierades under projektets gång och även diskussion kring säkerhet och etiska aspekter kopplade till hanteringen av användardata. Avslutningsvis lyfts framtida förbättringar fram tillsammans med förslag på vidareutveckling som skulle stärka applikationen ytterligare.

6.2.1 Tekniska val

De tekniska val som gjordes inför projektet visade sig vara väl lämpade. Next.js uppdelning mellan server- och klientkomponenter bidrog till strukturerad kod med tydlig separation. tRPC möjliggjorde typsäker kommunikation mellan front- och backend och fungerade väl ihop med Zod och Prisma, vilket resulterade i ett smidigt flöde mellan applikationens olika lager.

En styrka med T3-stacken är att verktygen är utvalda för att de fungerar väl tillsammans, vilket minskade tiden för den initiala konfigurationen och bidrog till en snabb start på utvecklingsfasen. Detta bedöms vara mycket värdefullt för mindre projekt med begränsad tid och är något som andra projekt av liknande karaktär kan dra nytta av.

Det finns dock nackdelar med de tekniska valen som även bör lyftas. T3-stacken innebär ett starkt beroende av ett specifikt ekosystem, vilken kan bli problematiskt. Exempelvis är tRPC tätt kopplat till TypeScript och Next.js, vilket gör det svårt att byta ut enskilda delar utan att påverka hela stacken.

6.2.2 Begränsningar

En tydlig begränsning är avsaknaden av integration med Chalmers befintliga lokalsystem. Ambitionen var initialt att integrera applikationen med detta system, men på grund av säkerhetsskäl var åtkomsten begränsad. Detta innebär att alla data om byggnader och lokaler måste hanteras manuellt via administratörssidan, vilket påverkar effektiviteten och ökar risken för felaktiga uppgifter. En sådan integration hade förbättrat applikationen avsevärt.

Enkätundersökningen genomfördes med tio deltagare, vilket begränsar möjligheten att dra några större slutsatser om applikationens användbarhet. Urvalet bestod delvis av bekanta, vilket kan ha påverkat objektiviteten i svaren och bidragit till mer positiva omdömen än vad ett mer representativt urval hade gett. En större och mer oberoende urvalsgrupp hade gjort resultaten mer robusta.

Det bör noteras att effektiviseringen av bokningsprocessen inte mättes objektivt, exempelvis genom att jämföra antal klick eller tidsåtgång per bokning med TimeEdit. Slutsatserna om effektivisering bygger istället på deltagarnas upplevelse. En ordentlig jämförelse med antal klick och tidsåtgång hade gett resultaten mer tyngd.

6.2.3 Säkerhet

Autentisering hanterades med NextAuth i kombination med Google OAuth, vilket innebar att databasen inte behövde hantera känsliga uppgifter som lösenord [13]. Efter inloggning skapas en session som används i backend för att identifiera användaren, vilket innebär att känslig information aldrig skickas direkt från klienten. Behörighetskontroll hanterades genom tRPC procedures vilket säkerställde att enbart användare med en session hade tillgång till affärslogiken av applikationen.

Affärslogik som begränsad bokningstid per vecka och logik för att förhindra överlappande bokningar hanterades i backend, vilket säkerställer att användarna inte kan undgå dessa. Sammantaget bedöms säkerheten uppfylla de krav som ställs på en prototyp av denna skala.

En begränsning när det kommer till säkerheten är att bokningslänkarna som genereras för delning är tillgängliga utan inloggning, vilket innebär att vem som helst med tillgång till länken kan se bokningsinformationen, alltså lokalens namn, datum, tid och e-postadressen kopplat till Google-kontot som gjort bokningen. Detta var ett medvetet val för att underlätta delning, men innebär att känslig information potentiellt kan nå obehöriga. Detta är något som hade behövts ses över vid vidareutveckling av applikationen.

6.2.4 Planering och utförande

Det iterativa arbetssättet visade sig fungera väl för ett projekt av denna storlek. Flexibiliteten möjliggjorde att ny funktionalitet, såsom gruppfunktionen kunde tillkomma under projektets gång baserat på återkoppling från enkäten.

En utmaning med tidsplaneringen var att projektet innefattade flera parallella moment, såsom rapportskrivande och implementation, där det var svårt att i förväg uppskatta hur lång tid varje del skulle ta. Implementationsfasen tog längre tid än planerat enligt Gantt-schemat (se B.1), vilket medförde att planerade E2E-tester uteblev till förmån för manuella tester. I efterhand hade en mer realistisk tidsuppskattning för implementationen och tidigare påbörjad testning bidragit till ett mer robust projekt.

6.2.5 Etiska aspekter

Applikationen hanterar personuppgifter i form av användarens e-postadress, som hämtas från Google-kontot vid inloggning. E-postadressen visas för andra gruppmedlemmar i gruppfunktionen samt för de personer man väljer att dela en bokningslänk med. Användare bör därför vara medvetna om att deras e-postadress kan bli synlig för fler än de man initialt delat länken med.

Applikationen använder även geografisk position för att föreslå närliggande lokaler som är tillgängliga. Platsinformation hämtas enbart efter att användaren har gett sitt godkännande i webbläsaren och lagras inte i databasen.

Bokningar lagras i databasen kopplat till användarens id. Vid eventuell vidareutveckling av applikationen bör hantering av användardata ses över och framgå tydligt till användarna, i enlighet med GDPR.

6.2.6 Vidareutveckling

Flera förbättringsområden identifierades under projektets gång.

- Integration med Chalmers lokalsystem för automatisk synkronisering av lokaler och byggnader, vilket skulle ta bort behovet av manuell datahantering.
- Möjligheten att redigera befintliga bokningar, vilket efterfrågades i enkätundersökningen.
- En veckoschemavy för bättre överblick över bokningar över tid, vilket också framkom som önskemål i enkäten.
- Integrerad kartvy som visar lokalens position och vägbeskrivning, vilket skulle underlätta för användare som inte känner till campus. Detta eftersom byggnader på universitet kan vara mycket stora och skyltning kan vara begränsad.

- Tydligare bokningsbekräftelse, något som upplevdes otydligt från svaren på enkäten. En bekräftelsedialog med tid, datum och namnet på lokalen hade löst detta problem.
- Integration med universitetens SSO system för autentisering hade möjliggjort en mer generell lösning där varje universitet kan ha sin egen databas med lokaler och användare, vilket hade gjort applikationen betydligt mer skalbar.
- Möjligheten att använda notifikationer och påminnelser till användaren inför en kommande bokning hade förbättrat användarupplevelsen och minskat risken för missade bokningar, vilket även hade bidragit till bättre nyttjande av resurserna på universitet.
- Stöd för flera språk, framförallt engelska. Detta skulle bidra till att göra applikationen tillgänglig för internationella studenter och personal som inte har svenska som sitt modersmål, vilket är mycket relevant för universitet med en stor andel internationella användare.

Litteraturförteckning

[1] A. F. Hutapea, L. H. Sitohang, S. K. Sihombing, and Sunardi, "User experience evaluation of the booking website using system usability scale and usability testing (study case sports arena)," in *2023 International Conference on Information Management and Technology (ICIMTech)*, 2023, pp. 391–396.

[2] Q. Liao, "Cognitive load in web interface design: A study based on visual complexity and task difficulty," in *2025 6th International Conference on Intelligent Design (ICID)*, 2025, pp. 72–77.

[3] X. Zhang and B. Dorn, "Agile practices in a small-scale, time-intensive web development project," in *2011 Eighth International Conference on Information Technology: New Generations*, 2011, pp. 1060–1061.

[4] F. Paz and J. A. Pow-Sang, "Usability evaluation methods for software development: A systematic mapping review," in *2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*, 2015, pp. 1–4.

[5] T. Browne, "T3 stack," accessed: 2026-03-30. [Online]. Available: https://create.t3.gg

[6] J. Bogner and M. Merkel, "To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022, pp. 658–669.

[7] Vercel, "Next.js documentation," accessed: 2026-03-30. [Online]. Available: https://nextjs.org/docs

[8] S. Kermo and A. Dželihodžić, "Comparative analysis of server-side, client-side, and hybrid rendering approaches in modern web applications," in *2026 25th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2026, pp. 1–6.

[9] tRPC, "trpc documentation," accessed: 2026-05-05. [Online]. Available: https://trpc.io/docs/

[10] Zod, "Zod documentation," accessed: 2026-05-05. [Online]. Available: https://zod.dev

[11] Prisma, "Prisma documentation," accessed: 2026-04-13. [Online]. Available: https://www.prisma.io/docs

[12] P. G. D. Group, "PostgreSQL documentation," accessed: 2026-04-13. [Online]. Available: https://www.postgresql.org/docs/

[13] NextAuth.js, "NextAuth.js documentation," accessed: 2026-04-13. [Online]. Available: https://next-auth.js.org

[14] Google, "Using oauth 2.0 to access google apis," accessed: 2026-05-08. [Online]. Available: https://developers.google.com/identity/protocols/oauth2

[15] M. Platforms, “Reusing logic with custom hooks,” accessed: 2026-05-08. [Online]. Available: https://react.dev/learn/reusing-logic-with-custom-hooks

[16] C. Veness, “Calculate distance, Bearing and More Between Latitude / Longitude points,” 2019, accessed: 2026-05-08. [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html

A

Kravspecifikation

A.1 Funktionella krav

Följande funktionella krav identifierades för webbapplikationen

- Systemet ska kunna visa tillgängliga lokaler för användaren.
- Systemet ska möjliggöra bokning av tillgängliga lokaler.
- Systemet ska möjliggöra avbokning av bokade lokaler.
- Systemet ska visa samtliga bokningar för en användare.
- Systemet ska stödja inloggning via Google OAuth.
- Systemet ska kunna söka efter lokaler baserat på namn.
- Systemet ska kunna filtrera lokaler baserat på datum, tid och kapacitet.
- Systemet ska kunna filtrera lokaler baserat på utrustning (t.ex dator, projektor, whiteboard).
- Användaren ska kunna favoritmarkera lokaler för snabb åtkomst vid framtida bokningar.
- Systemet ska kunna visa unik relevant information för varje lokal (t.ex kapacitet, utrustning, byggnad).
- Systemet ska kunna ge förslag på närliggande tillgängliga lokaler baserat på användarens nuvarande position (förutsatt att användaren godkänner detta).

A.2 Icke-funktionella krav

Dessa icke-funktionella krav har beaktats under arbetet.

- Systemet ska ha ett användarvänligt och intuitivt gränssnitt.
- Systemet ska vara responsivt och fungera väl på olika typer av enheter.
- Systemet ska ge snabb respons vid sökning och filtrering.
- Systemet ska hantera användarens data på ett säkert sätt.
- Platsinformation ska endast användas efter användarens godkännande.

B

Gantt-schema

Figur B.1: Gantt-schema från planeringsrapporten

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS