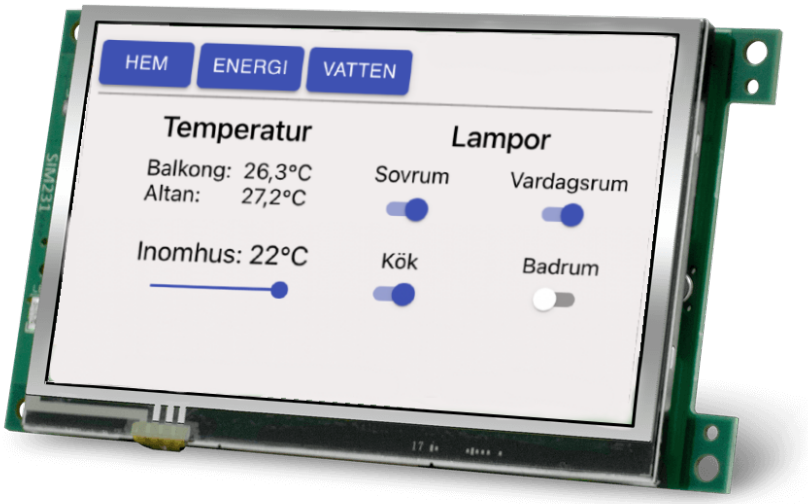
CHALMERS



Generering av grafiska användar-gränssnitt via en webbapplikation

En webbapplikation utvecklad i React

Examensarbete inom data- och informationsteknik

Carl Classon
William Husar

CSE - Institutionen för Data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA, GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022
www.chalmers.se

Examensarbete 2022

Generering av grafiska användargränssnitt via en webbapplikation

En webbapplikation utvecklad i React

CARL CLASSON
WILLIAM HUSAR

CHALMERS

Institutionen för Data- och informationsteknik
Chalmers tekniska högskola
Göteborg, Sverige 2022

Generering av grafiska användargränssnitt via en webbapplikation
En webbapplikation utvecklad i React
Carl Classson
William Husar

Handledare: Joachim von Hacht, CSE
Examinator: Lars Svensson, CSE

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Telefon: 031-772 1000

Omslagsbild: Ett grafiskt användargränssnitt skapat i webbapplikationen.

Institutionen för data- och informationsteknik
Göteborg 2022

Generering av grafiska användargränssnitt via en webbapplikation
En webbapplikation utvecklad i React
Carl Classon
William Husar
Institutionen för Data- och informationsteknik
Chalmers tekniska högskola

Sammanfattning

Fler konsumentprodukter får smarta funktioner för att underlätta i vardagen. Grafiska användargränssnitt behövs för att styra dessa smarta funktioner via en pekskärm. Att programmera användargränssnitt i något programspråk kan vara svårt och tidskrävande. Syftet med detta examensarbete är att förenkla och snabba upp konstruktionen av grafiska användargränssnitt. En webbapplikation har utvecklats i React, där användare kan dra och släppa färdiggjorda grafiska objekt på en rityta för att generera grafiska användargränssnitt. Användarna har möjlighet att spara det genererade användargränssnittet i ett lättviktigt dataformat. Ett enkelt test som genomfördes visade att användandet av webbapplikationen minskade utvecklingstiden av grafiska användargränssnitt avsevärt.

Abstract

An increasing amount of consumer products get smart functionalities to help people with their everyday life. Graphical user interfaces are used for controlling smart functionality using touchscreens. Developing user interfaces in a programming language can be difficult and time-consuming. The purpose of this degree project is to simplify and speed up the development of graphical user interfaces. A web-application has been developed using React, where users can drag and drop ready-made graphical objects onto a canvas in order to generate graphical user interfaces. Users also have the possibility of saving the generated user interfaces in a lightweight data format. A simple test that was performed showed that using the web-application decreased the development time of graphical user interfaces considerably.

Nyckelord: React, JavaScript, webbutveckling, webbapplikation, GUI, generering.

Förord

Vi vill rikta ett stort tack till våra kontaktpersoner på Plejd AB, Iman Habib och Robin Säfström, för deras hjälp och idéer som bidragit till projektets framgång. Vi vill även tacka vår handledare Joachim von Hacht som varit till stor hjälp gällande skrivandet av detta examensarbete.

Göteborg, maj 2022

Terminologi

Nedan listas de förkortningar som har använts i detta examensarbete i bokstavsordning:

DOM	Document Object Model
GUI	Grafiskt användargränssnitt (Graphical User Interface)
JSON	JavaScript Object Notation
MVP	Kravspecifikation (Minimum Viable Product)
SPA	Ensidig applikation (Single-page Application)
UI	Användargränssnitt (User Interface)

Innehåll

1
Introduktion

Fler och fler konsumentprodukter får så kallade smarta funktioner för att underlätta i vardagen, exempelvis att kaffekokaren sätts på automatiskt när alarmet går igång på morgonen. Smarta hushållsapparater som robotdammsugare, kylskåp och kaffemaskiner blir allt mer vanligt. Den tekniska utvecklingen av internetanslutna enheter medför också att allt fler hem blir uppkopplade. Inom en snar framtid kommer exempelvis kaffemaskiner och andra hushållsapparater att kunna styras via en skärm [1]. Idag har över 12% av svenska hem någon form av ansluten mätare, lampa eller fjärrströmbrytare [2]. När apparaterna ansluts till internet skapas nya möjligheter att styra enheten och hämta information.

För att styra anslutna konsumentprodukter behövs någon form av användargränssnitt, numera vanligen grafiska användargränssnitt (graphical user interface, GUI), och en pekskärm. Ett GUI innehåller grafiska komponenter som exempelvis knappar, menyer och textfält. En användare kan interagera med GUI:t genom att trycka på knappar på skärmen, göra val i menyer och så vidare. Ett GUI kan även visa olika former av information om enheten, exempelvis temperatur i en ugn eller när kaffemaskinen behöver rengöras.

För att programmera användargränssnitt till konsumentprodukter används generella programspråk som C. De flesta av dessa programspråk har inga inbyggda språkliga konstruktioner för att skapa grafiska komponenter och därför behöver utvecklare använda speciella tilläggspaket. Detta skapar en brant inlärningskurva då utvecklaren inte bara behöver kunskaper i programspråket utan även om tilläggspaketen.

Ett ytterligare problem med dessa GUI:s är att det kan bli problematiskt att modifiera dem. Även en liten ändring, såsom att flytta på en knapp i GUI:t, kan innebära en långdragen process där flera steg behöver göras om. Dessa steg kan exempelvis vara att göra större ändringar i koden, utföra nya tester på koden, kompilera om alltihop och till sist ladda upp den ändrade koden på enheten. Utvecklare av GUI:s har mycket att vinna på en lösning som förenklar denna utvecklingsprocess.

1.1 Syfte

Syftet är att förenkla och snabba upp konstruktionen av GUI:s för konsumentprodukter.

1.2 Mål

Målet med projektet är att implementera en mjukvara som ger utvecklare möjligheten att visuellt generera GUI:s och därmed sänka kunskapströskeln samt spara tid i utvecklingsprocessen.

1.3 Avgränsningar

Med fullskaligt test avses en situation där flera utvecklare får testa mjukvaran och utvärdera prestandan utifrån ett antal punkter. På grund av tidsbrist kommer det inte utföras något fullskaligt test gällande hur snabbt och enkelt det är att använda programvaran i jämförelse med den traditionella uvecklingsprocessen. Istället kommer ett enklare test att genomföras.

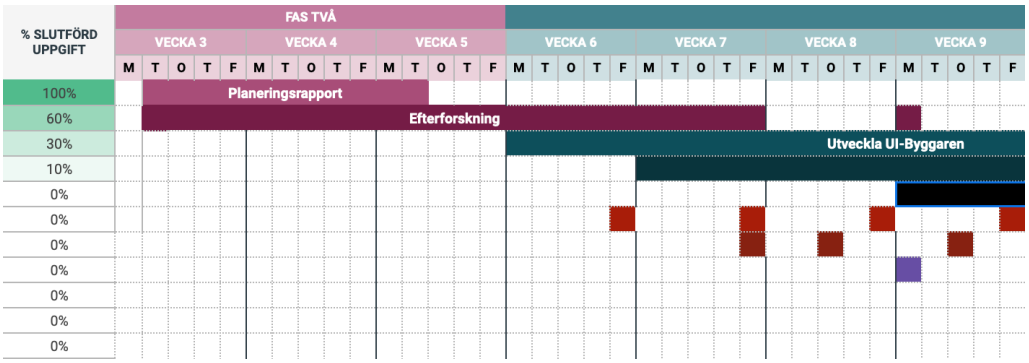
2

Metod

I detta kapitel avhandlas de metoder och tillvägagångssätt som kommer att tillämpas i projektet.

2.1 Projektplanering med Gantt

Vid projektplaneringen kommer ett Gantt-schema användas för att visuellt överblicka projektets alla faser över tid. Ett Gantt-schema innehåller horisontella tidslinjer som representerar en specifik deluppgift av projektet samt dess start- och slutdatum [3]. I schemat framgår det vilka framsteg som görs genom kolumnen: "% slutförd uppgift" (se figur 1). Med hjälp av detta kan man se om den planerade tidsplanen efterföljs. Vidare tydliggör schemat vilka beroenden som finns mellan olika deluppgifter. Se figur 1 nedan för ett exempel på ett Gantt-schema.



Figur 1: Utdrag ur Gantt-schemat.

2.2 Efterforskning

Innan programutvecklingen startas kommer en efterforskning att genomföras. Resultatet av efterforskningen blir ett dokument där relevant information finns samlad för att underlätta vid framtida problem. Informationen kommer från en variation av olika källor och bland dessa ingår bloggar, videor, artiklar och andra projekt på GitHub. Ytterligare information och exempel på liknande problem och lösningar införskaffas genom teknisk handledning från företaget där examensarbetet utförs.

2.3 Testmetod

Följande tillvägagångssätt används för att testa hur väl projektets syfte uppnås:

- Skapa två liknande gränssnitt - ett med den utvecklade mjukvaran och ett med något generellt programspråk.
- Jämför de två grovt tidsmässigt, hur blev resultatet och vilken metod verkar snabbast?

3
Teknisk bakgrund

I detta kapitel introduceras relevant teknisk bakgrund för projektet.

3.1 Inbyggda system

Ett inbyggt system avser ett datorsystem baserat på en mikroprocessor [4]. Ett inbyggt system har utöver en mikroprocessor även alltid någon form av minne och förmågan att skicka ut och ta emot data via ingångar och utgångar. Vidare är ett inbyggt system särskilt designat för att utföra en eller ett fåtal dedikerade funktioner och har i stort sett alltid realtidskrav [5]. Komplexiteten hos ett inbyggt system kan variera kraftig beroende på dess uppgift. De enklaste systemen består enbart av en enskild mikrokontroller men även avancerade system såsom smarta mobiltelefoner kan ses som inbyggda system. De mer avancerade inbyggda systemen har ofta någon form av grafiskt gränssnitt som en användare kan interagera med, till exempel en pekskärm.

3.2 Sakernas internet

Med sakernas internet (Internet of Things, IoT) menas alla enheter som är uppkopplade till internet och inkluderar laptops, smarta TV-apparater, smarttelefoner, bilar, hushållsapparater med flera [6]. En smart enhet kan i princip vara vilken produkt som helst om enheten kan kommunicera med andra enheter över internet.

Ett exempel på en tillämpning av smarta enheter kan vara smart uppvärmning av hemmet. Användaren möts av en digital termostat där det finns möjlighet att avläsa och reglera temperatur. Systemet läser av aktuell temperatur med sensorer och håller reda på om någon är hemma. En molnapplikation hämtar väderprognoser och elpriser. Ett ställdon kan slå av eller på värmepannan och en hubb kopplar samman alltihop [7, s. 12]. Systemet kan alltså aktivt stänga av uppvärmningen när ingen är hemma, och anpassa värmen efter utetemperaturen.

3.3 Öppen källkod

Med öppen källkod menas en datormjukvara som är öppen för användare att använda, ändra, granska och distribuera utan kostnad [8, s. 4] [9]. Det finns heller inga begränsningar gällande vilket syfte datormjukvaran får användas till. Vidare är öppen källkod vanligtvis en form av öppet samarbete [10]; detta innebär att vem som helst som har möjligheten, kompetensen och viljan att bidra till utvecklingen av mjukvaran kan göra det. Det finns även varianter av öppen källkod som inte bygger på öppet samarbete, exempelvis där koden distribueras för att användas fritt men inte tillåter några modifikationer. Vilka regler som gäller för källkoden tydliggörs genom att använda en licens för öppen källkod [9].

3.4 Node.js och Npm

Node.js är en exekveringsmiljö som gör det möjligt att köra JavaScript-kod utanför en webbläsare [11]. Miljön är skapat för att kunna bygga storskaliga server- och nätverksprogram. Node.js är plattformsoberoende vilken innebär att projekt skapade med Node.js fungerar på flertalet olika plattformar som Windows, Linux och macOS.

Vidare har Node.js en mycket stor samling av bibliotek, kallade paket. Dessa hanteras av Node package manager (Npm). Npm ger tillgång till över en miljon olika bibliotek skrivna med öppen källkod. Alla går att installera med enkla kommandon i Npm.

Nedan visas ett exempel på hur bibliotek kan installeras med hjälp av Npm (i kommandotolken). I figur 2 ska “package-name” ersättas av ett unikt namn för biblioteket; detta kan hittas via Npms databas (https://www.npmjs.com)

```
npm install <package-name>
```

Figur 2: Kommando för att installera ett bibliotek med npm.

3.5 Dataformatet JSON

JSON (JavaScript Object Notation) är ett lättviktigt textbaserat dataformat som används för att utbyta och spara data [12]. Det utvecklades för att enkelt kunna representera JavaScript-objekt men idag har i princip alla större programmeringsspråk stöd för JSON. Dataformatet består av två olika datastrukturer, dels objekt i form av attribut-värde-par och dels ordnade listor. Listor kan innehålla värden, objekt, andra listor eller en valfri kombination av dessa. Figur 3 nedan är ett exempel på hur data kan representeras i JSON-format.

```
{
  "namn": "Pelle",
  "ålder": 42,
  "kön": "man",
  "barn": [
    {
      "namn": "Eva",
      "ålder": 17
    },
    {
      "namn": "Johan",
      "ålder": 12
    }
  ]
}
```

Figur 3: Exempel på dataformatet JSON.

3.6 Biblioteket React

React är ett JavaScript-bibliotek för att skapa användargränssnitt (user interfaces, UI) i webbapplikationer [13, s. 9]. Ursprungligen skapades React av Facebook men övergick snart därefter till ett projekt med öppen källkod. Till skillnad från ett GUI så kan UI:s även innehålla icke-grafiska gränssnitt som exempelvis skärmläsare. Detta gör det möjligt för en person med synskador att interagera med en applikation genom ett program som läser upp innehållet [14].

I React används begreppet komponenter för att beteckna funktioner (eller klasser) som returnerar HTML-element [15]. Dessa funktioner kan utöver HTML innehålla logik och tillstånd vilket innebär att en React-komponent både representerar utseende och logik för ett grafiskt objekt. Logiken kan till exempel vara vad som ska ske vid ett knapptryck.

En komponent-funktion returnerar alltid ett HTML-element som kan renderas i en webbläsare. I React sker rendering av HTML-element via React-DOM API:et [13, s. 11]. DOM (Document Object Model) i sig är en standard för hur man renderar, uppdaterar och interagerar med HTML-element [13, s. 13] i en webbläsare. Det som skiljer sig mellan DOM och React-DOM är att React-DOM är en virtuell representation av sidans tillstånd. Detta medför att React inte behöver ladda om hela sidan när en ändring sker, istället laddas enbart den delen av sidan där ändringen har skett. Ett exempel på en sådan ändring kan vara att ett textfält uppdateras med ny text. Detta sparas resurser som minne och processorkraft på serversidan, vilket leder till ökad prestanda. Denna typ av applikation, där hela sidan inte laddas om

vid ändringar, kallas för en ensidig applikation (single page application, SPA) [16].

Till React finns även ett stort utbud av ramverk som bygger ut funktionaliteten hos biblioteket. Många av dessa har skapats av React-användare som publicerat sina ramverk som Npm-paket. Några exempel på detta är Craft.js och MUI.

3.7 Ramverket Craft.js

Craft.js är ett kollaborativt ramverk som skapats av användare på GitHub. Ramverket implementerar drag- och släpp-funktionalitet för React-komponenter. Med hjälp av ramverket kan React-komponenter placeras på en rityta [17]. Vidare så kan tillståndet av hela ritytan serialiseras till JSON-format. Detta medför att en användare kan spara ner det som skapats i ritytan för att senare kunna ladda upp det sparade arbetet och fortsätta där man slutade. Craft.js tillhandahåller även ett antal React-komponenter, bland annat Editor. Editor kan importeras och användas i en React-applikation.

Koden i figur 4 visar hur man i en React-applikation med hjälp av Craft.js kan skapa en rityta som innehåller en text-komponent. TextComponent i figur 4 är ett exempel på en React-komponent.

```
import { Editor, Frame, Canvas } from '@craftjs/core'

const App = () => {
    return (
        <Editor resolver={{ TextComponent }}>
            // Editable area starts here
            <Frame>
                <Canvas>
                    <TextComponent text="I'm rendered here"
                </Canvas>
            </Frame>
        </Editor>
    )
}
```

Figur 4: Craft.js rityta med React-komponent.

3.8 Komponentbiblioteket MUI

MUI är ett komponentbibliotek byggt med öppen källkod. Komponentbiblioteket utvecklas av företaget Material UI. MUI tillhandahåller grafiska komponenter till React-applikationer och är således React-komponenter [18]. Komponenterna för bland annat knappar, ikoner, menyer och grafer med flera finns tillgängliga. MUI används av flera av de ledande företagen på nätet såsom Spotify, Amazon och Netflix [19].

Koden nedan i figur 5 visar hur man med hjälp av MUI kan skapa tre knappar av olika varianter och placera dessa på rad efter varandra.

```
import { Stack, Button } from '@mui/material'

function BasicButtons() {
    return (
        <Stack spacing={2} direction="row">
            <Button variant="text"<Text</Button>
            <Button variant="contained">Contained</Button>
            <Button variant="outlined">Outlined</Button>
        </Stack>
    )
}
```

Figur 5: Tre MUI-knappar i React.

Koden i figur 5 resulterar i att tre knappar visas i webbläsaren enligt figur 6.

Figur 6: Tre renderade knappar i webbläsaren.

4
Genomförande

I detta kapitel presenteras vad som har gjorts i projektet och hur applikationen är uppbyggd.

4.1 Utveckling med Scrum

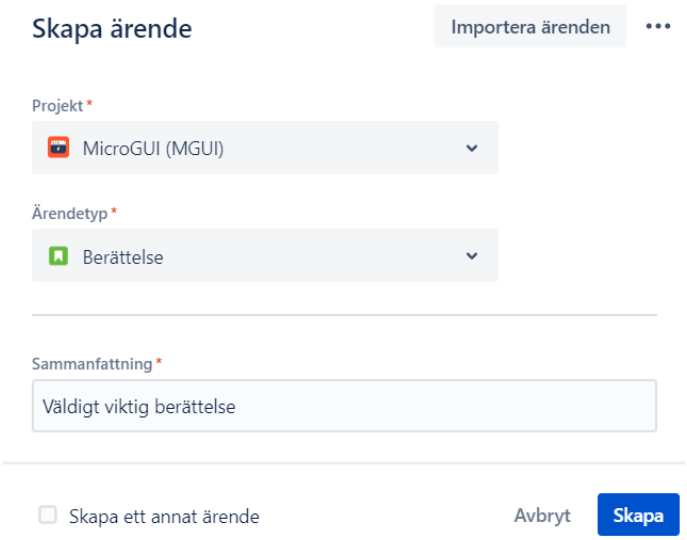
Det agila ramverket scrum tillämpades under utvecklingsprocessen. Scrum är en etablerad metod för produktutveckling där utvecklingen struktureras i iterationer för att uppnå målen [20].

Projektarbetet delades upp i iterationer, så kallade sprintar, fördelade över två veckor. Vid början av varje sprint upprättades unika mål med angiven prioritet och uppskattad tidsåtgång.

4.1.1 Scrum-verktyget Jira

Jira användes för att planera och strukturera programmeringsarbetet under projektets gång. Jira tillhandahåller verktyg för att skapa användarberättelser (user stories), en produktbacklogg och scrum-tavlor (scrum boards) för varje sprint [21]. Det finns även möjlighet att skapa rapporter av föregående sprintar för att kunna mäta och jämföra produktivitet.

Inför varje sprint planerades uppgifter utefter prioriterad funktionalitet i applikationen. För varje uppgift skapades ett ärende i Jira (figur 7) där uppgiften fick en sammanfattning, beskrivning och estimering av mängden ansträngning som krävdes för att slutföra ärendet. Ansträngningen estimerades utefter svårighetsgrad samt uppskattad tidsåtgång.

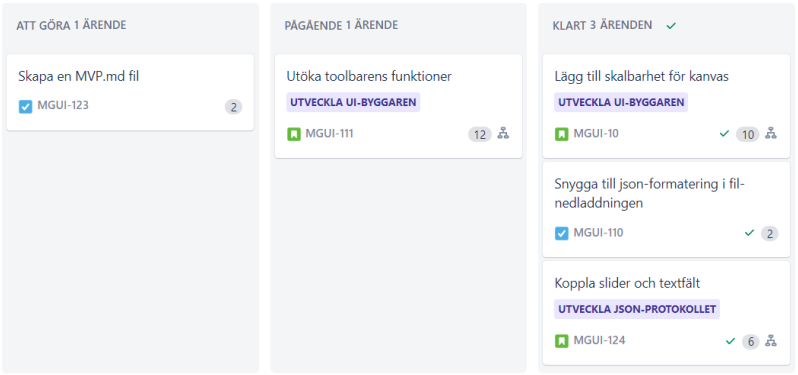


Figur 7: Skapande av användarberättelse.

I produktbackloggen (figur 8) samlades ärenden av olika typer (berättelse, uppgift, bugg) och rangordnades högt till lågt utefter dess prioritet.

Figur 8: Produktbacklogg.

För varje sprint tilldelades de ärenden som hade högst prioritet upp till en viss tröskel av estimerade poäng. Tröskeln ändrades från sprint till sprint utefter en analys av tidigare sprintar, men var i genomsnitt cirka 40 poäng. Ärenden hade olika status beroende på om de var påbörjade, kvar att göra eller helt klara. I figur 9 syns ett exempel på hur scrum-tavlan såg ut under en pågående sprint.



Figur 9: Scrumtavla.

Vid analys av tidigare sprintar användes rapporter med story-poäng över tid. I figur 10 visas ett exempel på hur en sådan rapport såg ut över perioden 22 februari till 6 mars, det vill säga en sprint. I början av sprinten tilldelades ärenden med en estimering runt 25 poäng, och efter en vecka återstod endast ärenden med 5 poäng att göra. I början av den andra veckan lades fler ärenden till och då estimerades arbetet till 20 poäng. Successivt avklarades dessa ärenden och vid avslutandet av sprinten återstod inget arbete.

Figur 10: Sprintrapport.

4.2 Kravspecifikation

Inför implementeringen skapades en kravspecifikation (Minimum viable product, MVP). MVP:n innehöll en lista med specifika krav. Dessa krav behövde alla vara uppfyllda för att projektet skulle klassas som tillräckligt färdigt för att vara godtagbart [22].

Projektet går ut på att skapa en prototyp som består av:

- Ett lättviktigt dataformat som beskriver tillståndet hos GUI:s.

- En webbapplikation som innehåller:
  - En editor där användaren kan bygga ett gränssnitt som sedan automatiskt kompileras till dataformatet.
  - En webbaserad simulator som kan tolka dataformatet och simulera ett GUI.

Applikationen bygger på ett "drag- och släppsystem" där användaren väljer en grafisk komponent och drar den till rätt plats i GUI:t. Applikationen är webbaserad för att göra den så lättillgänglig som möjligt för användare. Detta då en webbapplikation kan köras på alla enheter som har någon form av webbläsare. Detta innebär vidare att kompatibilitet för olika enheter inte behöver tas i beaktning under utvecklingsarbetet.

Kraven för editorn var följande:

- Rityta
  - Det ska finnas en rityta där grafiska komponenter kan ritas ut.
  - Ritytan ska ha ändringsbara attribut för storlek och bakgrundsfärg.
- Komponenter
  - Det ska finnas fyra grafiska komponenter att välja emellan: En knapp (Button), en brytare (Switch), ett reglage (Slider) och ett textfält (Textfield).
  - Komponenterna ska kunna flyttas runt fritt inom ritytans yta.
  - Komponenterna ska ha ändringsbara attribut som exempelvis färg, storlek och händelser (events).
- Spara och ladda
  - Ritytans tillstånd ska gå att spara i JSON-format.
  - Det ska vara möjligt att ladda upp ett sparat tillstånd till ritytan.
- Verktyg
  - En hjälptext ska visas när muspekaren förs över en komponent.
  - Det ska vara möjligt att rensa alla grafiska komponenter från ritytan.
  - Det ska vara möjligt att ångra och göra om ändringar på ritytan.

Kraven för simulatorn var:

- Det ska finnas en simulator där det går att rendera ett GUI lagrat som JSON.
- I simulatorn ska det inte vara möjligt att flytta på komponenter, se hjälptexter, ta bort grafiska komponenter eller ändra attribut hos komponenter.

4.3 Utvecklingsmiljö

Nedan beskrivs de verktyg som användes för att utveckla applikationen. Här följer även instruktioner för att köra applikationen lokalt.

4.3.1 Visual Studio Code

Utvecklingsmiljön som användes vid implementeringen av applikationen var Visual Studio Code (VS Code). VS Code är väl anpassat för utveckling av React-applikationer då det har bra stöd för JavaScript-syntax [23].

4.3.2 Git och GitHub

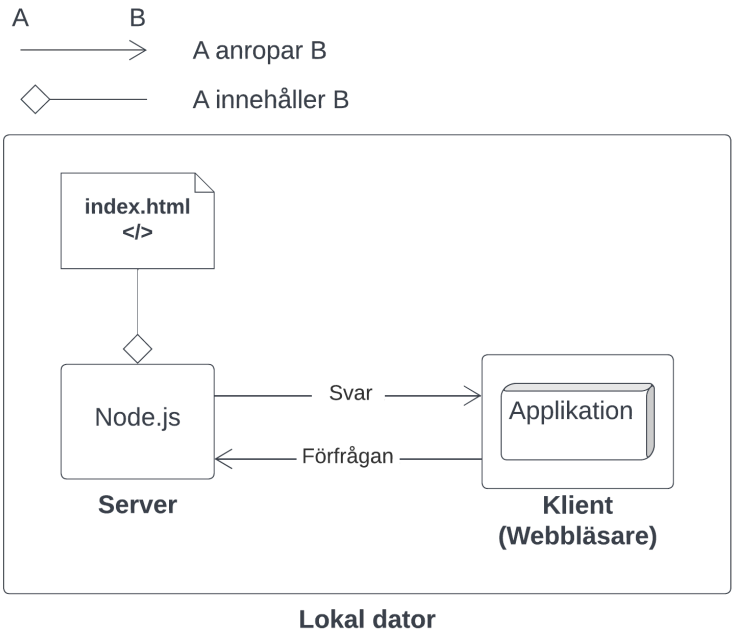
Git användes som versionshanteringssystem. Git lagrar en kollektion av filer i fortlöpande versioner som representerar historiska förändringar av projektets innehåll [24]. Varje sådan version kallas en commit. Systemet möjliggör även simultan utveckling genom parallella grenar, så kallade grenar, som kan delas upp och slås ihop.

GitHub användes som tjänst för att lagra kodbasen. Koden finns tillgänglig att hämta på https://github.com/microgui.

4.4 Systemdesign

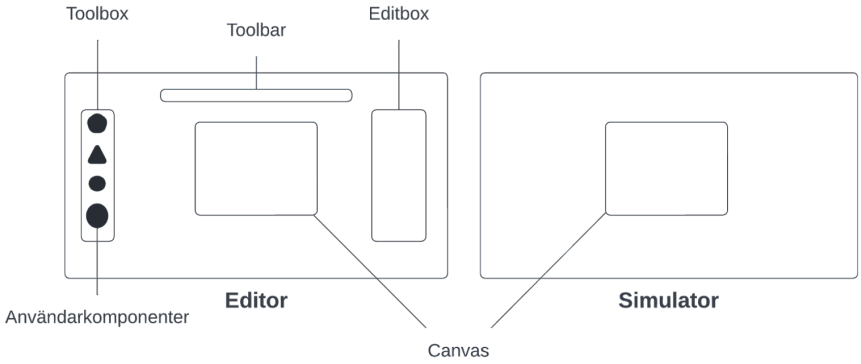
Systemet är en klient-server-applikation, där klienten utgörs av webbläsaren som exekverar en React-applikation. Server-delen består av en Node.js-miljö. Kommunikationen mellan klient och server sker över HTTP.

När React-applikationen ska köras i webbläsaren skickas en förfrågan till servern (Node.js) enligt figur 11. Servern svarar med att skicka en statisk HTML-fil (index.html). Denna fil beskriver var React ska rendera applikationens användargränssnitt.



Figur 11: Systemskiss klient-server.

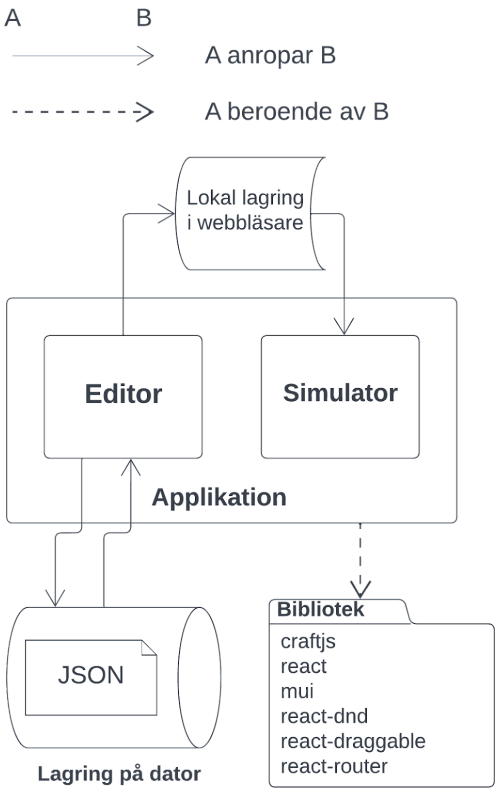
I figur 12 nedan visas en initial mockup av applikationens grafiska utseende. En detaljbeskrivning av de olika React-komponenterna (Toolbox, Toolbar, Editbox, Canvas) ges i avsnitt 4.5. Användarkomponenter beskrivs i avsnitt 4.5.6.



Figur 12: Mockup av applikationens användargränssnitt.

4.5 Applikationen

Applikationen består av två webbsidor med React-komponenter, Editor och Simulator. I figur 13 visas en skiss över applikationens sidor och vilka bibliotek som applikationen är beroende av.



Figur 13: Systemskiss över applikationen.

Editor-sidan är den delen av applikationen där användaren kan skapa sina gränssnitt, medans simulator-sidan är där användaren kan simulera sina gränssnitt. Data kan delas mellan editor-sidan och simulator-sidan via det lokala lagrings-API:et (local storage) i webbläsaren [25]. På editor-sidan är det möjligt att spara och ladda upp JSON-filer som representerar GUI:t. Detta sker via datorns lagringsutrymme (hårddisk). På så vis kan en användare återvända vid senare tillfälle och göra ändringar i sitt GUI efter att ha lämnat applikationen.

4.5.1 Installation av Node.js och Npm

För att kunna skapa en React-applikation och köra den så krävdes två specifika mjukvaror, JavaScript-motorn Node.js och pakethanteraren Npm. Dessa hämtades från https://nodejs.org/en/download och installerades genom att följa den medföljande installationsguiden.

4.5.2 Generering av React-applikationen

Efter installationen av den nödvändiga mjukvaran så kunde en React-applikation genereras genom följande kommando i kommandotolken:

npx create-react-app ui

I kommandot ovanför står npx för en exekvering av Node-paketet create-react-app som skapar grunden för React-applikationen. Applikationen döptes till ui eftersom den representerar ett användargränssnitt.

I figur 14 visas den genererade filstrukturen efter exekveringen av kommandot ovanför. Mappen src (source) innehåller all JavaScript som behöver kompileras till statiska filer när React-projektet byggs. I mappen public finns de statiska filer som inte behöver kompileras.

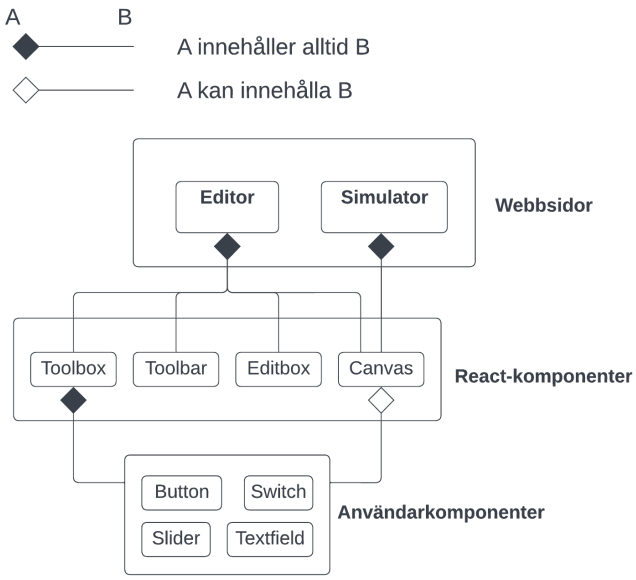
Figur 14: Automatisk genererad filstruktur.

4.5.3 Editor och Simulator

Editor-sidan är uppbyggd av React-komponenter såsom Toolbox, Toolbar med flera (se figur 15). Sidan är baserad på React-komponenten från Craft.js med samma namn, Editor. För att undvika förvirring omnämns denna komponent hädanefter som "React-komponenten Editor".

Simulator-sidan innehåller endast React-komponenten Canvas. I simulator-sidan går det inte att göra några ändringar på GUI:t, det är dock möjligt att interagera med användarkomponenterna. Webbsidan är uppbyggd så att tillståndet av Editor-sidan hämtas från den lokala lagringen i webbläsaren enligt figur 15.

Användarkomponenterna är React-komponenter men de skiljs åt här för att förtydliga vilka grafiska objekt som faktiskt används när man skapar ett GUI i editorn.



Figur 15: Skiss över webbsidor, React-komponenter och användarkomponenter.

I figur 16 syns koden för Editor-sidan. Detta är ett exempel på hur React-komponenten Editor från Craft.js kan användas tillsammans med de övriga React-komponenterna (Toolbox, Toolbar, etc.).

```
<Editor
    enabled={true}
    // A map of every user component in the editor.
    resolver={{
        CanvasArea, Button, Slider,
        Switch, Textfield,
    }}
    // removes the 'drop-indicator' built into craft.js
    indicator={false}
>
    <Stack
        className='row'
        direction='row'
        spacing={0}
    >
        <div className='left'>
            <Toolbox />
        </div>
        <div className='middle'>
            <Toolbar />
            <Frame >
                {/*The canvas element where the user can drop components*/}
                <Element
                    is={CanvasArea}
                    canvas
                />
            </Frame>
        </div>
        <div className='right'>
            <Editbox />
        </div>
    </Stack >
</Editor>
```

Figur 16: Kod för Editor-sidan.

4.5.4 Canvas

React-komponenten Canvas (ritytan) definierar området där användare kan skapa GUI:s (figur 17). Ritytan är definierat som ett drop-område. Detta innebär att en händelse registreras när en komponent släpps inom ritytan. Data som kan avläsas från händelsen är exempelvis koordinater som beskriver var händelsen har skett. Detta i sin tur används för att generera den valda komponenten på de avlästa koordinaterna. Ritytan har flera attribut som kan ändras inom applikationen. Dessa är bakgrundsfärg, bakgrundsbild, bredd och längd.

```
export const CanvasArea = ({ background, image, width, height, children }) => {
    const [, dropTarget] = useDrop({
        accept: "component",
        drop(item, monitor) {
            const offset = monitor.getClientOffset();
            if (offset) {
                const dropTargetXy = document.getElementById('canvasElement').getBoundingClientRect();
                return ({
                    x: parseInt(offset.x - dropTargetXy.left),
                    y: parseInt(offset.y - dropTargetXy.top)
                })
            }
        }
    })

    const { connectors: { connect } } = useNode()

    return (
        <div ref={dropTarget}>
            <Box
                ref={connect}
                id='canvasElement'
                sx={{
                    width: parseInt(width),
                    height: parseInt(height),
                    backgroundColor: background ? `rgba(${Object.values(background)})` : null,
                    backgroundImage: image ? `url(${image})` :
                                     (background ? null :
                                     'radial-gradient(#e2e2e2 20%, #fff 20%)'),
                    backgroundSize: image ? `${width}px ${height}px` :
                                    (background ? null: '10px 10px'),
                    position: 'relative',
                    transition: 'width 0.8s, height 0.8s'
                }}
            >
                {children}
            </Box>
        </div>
    )
}
```

Figur 17: Kod för Canvas.

4.5.5 Toolbox

Toolbox visar alla grafiska komponenter och gör det möjligt för användaren att dra dessa till ritytan. Toolbox är baserad på en Grid från MUI. En Grid är uppbyggt som ett rutnät där föremål placeras i celler. Inom varje cell placeras en unik komponent. I figur 18 visas funktionen CreateNode i Toolbox som innehåller logiken för att rendera en användarkomponent i ritytan. Funktionen useDrag exekveras när en komponent blir dragen från Toolbox och släppt på ritytan. Det som sker sedan är att en freshNode (ny användarkomponent) genereras med ett id och data. Den nya användarkomponenten läggs till i ritytan genom actions.add(node, 'ROOT').

```
const CreateNode = () => {
    const [, dragRef] = useDrag(() => ({
        type: 'component',
        options: {
            dropEffect: 'copy'
        },
        end(_, monitor) {
            // Get data from the drop-event that occurs when
            // a user drops a component on the canvas
            const dropResult = monitor.getDropResult()
            if (dropResult) {
                // Creates a node with attributes of a specific component
                // this is done to create a node with custom id and coords.
                const freshNode = {
                    id: `${component.name}_${count[component.name]}`,
                    data: {
                        type: component,
                        props: {
                            pageX: dropResult.x - 20,
                            pageY: dropResult.y - 10
                        }
                    }
                }
                const node = query.parseFreshNode(freshNode).toNode()
                // add the created node to the canvas.
                actions.add(node, 'ROOT')
                count[component.name] += 1
            }
        }

    }))
    // add our components to our list
    componentList.push(
        <Grid item xs={6} key={index}>
            <Tooltip title={component.name}>
                <IconButton
                    style={{ transform: 'translate(0, 0)' }}
                    ref={dragRef}
                    aria-label={component.name}
                >
                    {icons[component.name]}
                </IconButton>
            </Tooltip>
        </Grid>
    )
}
```

Figur 18: Kod för Toolbox.

4.5.6 Toolbar

Toolbar innehåller verktyg som är relevanta för ritytan. I figur 19 nedan syns implementationen av Toolbar. Alla verktyg är placerade i en rad. Det finns verktyg för att ångra (undo), göra om (redo), rensa ritytan (clear), förhandsvisa tillståndet samt spara och ladda tillståndet av ritytan.

```
export const Toolbar = () => {
    // check if undo/redo is possible using craft.js functionality.
    const { canUndo, canRedo, actions, query } = useEditor((_, query) => ({
        canUndo: query.history.canUndo(),
        canRedo: query.history.canRedo()
    }))

    return (
        <div className='toolbar'>
            <Stack direction='row'>
                <Tooltip title='Undo'>
                    <span style={{cursor: canUndo ? 'pointer' : 'not-allowed'}}>
                        <IconButton
                            // The button is disabled if there is nothing to undo
                            disabled={!canUndo}
                            onClick={() => actions.history.undo()}
                        >
                            <UndoIcon sx={{color: 'grey'}}/>
                        </IconButton>
                    </span>
                </Tooltip>
                <Tooltip title='Redo'>
                    <span style={{cursor: canRedo ? 'pointer' : 'not-allowed'}}>
                        <IconButton
                            // The button is disabled if there is nothing to redo
                            disabled={!canRedo}
                            onClick={() => actions.history.redo()}
                        >
                            <RedoIcon sx={{color: 'grey'}} />
                        </IconButton>
                    </span>
                </Tooltip>
                {/* Renders the clear button of the toolbar */}
                <ToolbarClear />
            </Stack>
            <Stack direction='row' spacing={0.7} sx={{ paddingRight: '10px' }}>
                <Link to='/simulator' target='_blank' rel='noreferrer'
                    style={{ textDecoration: 'none' }}
                >
                    <MaterialButton
                        size='small'
                        variant='contained'
                        color='info'
                        disableElevation
                        onClick={() => localStorage.setItem('data', query.serialize())}
                    >
                        <PlayCircleOutlineIcon style={{ padding: '2px' }} />
                        Simulate
                    </MaterialButton>
                </Link>
                {/* Renders the save/load buttons of the toolbar */}
                <ToolbarSave />
                <ToolbarLoad />
            </Stack>
        </div>
    )
}
```

Figur 19: Kod för Toolbar.

4.5.7 Editbox

Editbox innehåller funktionalitet för att ändra attribut hos komponenter. Variabeln currentlySelectedNodeId (figur 20) håller reda på vilken komponent som användaren har valt i ritytan. När en komponent är vald så visas dess attribut i en lista. Användare kan välja från listan vilken specifik attribut som ska ändras. Varje komponent har unika attribut som påverkar dess utseende.

```
export const Editbox = () => {
    const { active, related } = useEditor((state, query) => {
        const currentlySelectedNodeId = query.getEvent('selected').first();
        return {
            active: currentlySelectedNodeId,
            related:
                currentlySelectedNodeId && state.nodes[currentlySelectedNodeId].related,
        }
    })

    return (
        <div className="editbox">
            <div className="editHeader">
                <Stack
                    direction='row'
                    alignItems='center'
                    spacing={1}
                >
                    <EditIcon style={{ color: 'grey' }} />
                    <h2 style={{ color: 'grey' }}>Edit</h2>
                </Stack>
            </div>
            {active && related.toolbar && React.createElement(related.toolbar)}
            {!active && (
                <h3 style={{ marginTop: '20px', fontWeight: 'normal' }}>
                    Drag a component to the canvas area and
                    click on it to start editing.
                </h3>
            )}
        </div>
    )
}
```

Figur 20: Kod för Editbox.

4.5.8 Användarkomponenter

Fyra olika användarkomponenter går att placera i ritytan och skapa GUI:s med. Dessa är Button, Slider, Switch och Textfield. Alla användarkomponenter som placeras i ritytan lagras i en kontext där tillståndet hos dessa kan hämtas. Nedan följer en kort beskrivning av varje användarkomponent samt dess attribut.

Button - En knapp med text. Ändringsbara attribut för Button är text, storlek, variant (ifylld, enbart text, konturerad) samt färg.

Slider - Ett reglage. Ändringsbara attribut för Slider är storlek, bredd och färg. Det går även att koppla ihop en Slider med en Textfield för att visa det nuvarande värdet av reglaget.

Switch - En brytare. Ändringsbara attribut för Switch är storlek och färg.

Textfield - Ett textfält. Ändringsbara attribut för Textfield är textstorlek, textstil, hur texten ska justeras samt färg.

I figur 21 visas ett exempel på hur koden ser ut för en Button. Kodstrukturen är liknande för de andra användarkomponenterna. Gemensamt för alla användarkomponenter är att Draggable används för att kunna flytta på användarkomponenten.

```
export const Button = ({ custom, onClick, size, variant, background, color,
    text, pageX, pageY, ...props }) => {

    const { enabled } = useEditor((state) => ({
        enabled: state.options.enabled
    }))

    const { id, name, connectors: { connect }, actions} = useNode((node) => (
      {name: node.data.displayName}))

    const nodeRef = useRef()

    return (
        <Draggable
            disabled={!enabled}
            onStop={() => handleStop(actions, nodeRef)}
            nodeRef={nodeRef}
            bounds='parent'
            position={{
                x: getX(pageX, nodeRef),
                y: getY(pageY, nodeRef)
            }}
        >
            <div style={{ position: 'absolute' }} ref={nodeRef}>
                <Tooltip name={name} id={id}>
                    <MaterialButton
                        ref={connect}
                        size={size}
                        variant={variant}
                        onClick={onClick}
                        sx={{ ... }}
                        {...props}
                    >
                        {text}
                    </MaterialButton>
                </Tooltip>
            </div>
        </Draggable>
    )
}
```

Figur 21: Kod för användarkomponenten Button.

4.5.9 Tooltip

När en användare placerar datormusen över en användarkomponent så visas en Tooltip. Denna visar namnet på användarkomponenten och ger användaren möjlighet att radera den. I grund och botten är React-komponenten Tooltip baserad på Tooltip från MUI (se figur 22).

```
export const Tooltip = styled(({ className, name, id, ...props }) => {
    const { actions, enabled } = useEditor((state) => ({
        enabled: state.options.enabled
    }))

    const { isHovered, isSelected } = useNode((node) => ({
        isHovered: node.events.hovered,
        isSelected: node.events.selected
    }))

    const areaRef = useRef(null)

    return (
        <MaterialTooltip
            {...props}
            ref={areaRef}
            open={(isHovered || isSelected) && enabled}
            title={<>
                {name}
                <button
                    style={{ cursor: 'pointer', background: 'none',
                        border: 'none'
                    }}
                    onMouseDown={(e) => {
                        e.stopPropagation()
                        actions.delete(id)
                    }}
                >
                    <DeleteIcon
                        style={{
                            color: 'white',
                        }}
                    />
                </button>
            </>
            }
            classes={{ popper: className }}
            placement="top-start"
        />
    )
})
```

Figur 22: Kod för Tooltip.

4.5.10 Lagring av data

Serialiserad data genereras genom ett anrop till funktionen serialize från Craft.js (figur 23), vilket omvandlar tillståndet hos ritytan till JSON. För att spara dataformatet till datorns hårddisk skapades funktionen saveFile (se figur 24) som skapar en fil utifrån två parametrar. Den ena parametern är vilken data som ska sparas, och den andra parametern är vilket namn filen ska sparas som.

```
<MaterialButton
    onClick={() => saveFile(query.serialize(), formText)}
> Save
</MaterialButton>
```

Figur 23: Anrop för att serialisera ritytan.

```
const saveFile = (data, name) => {
    if (formText !== '') {
        const file = new File(
            [JSON.stringify(JSON.parse(data), null, 4)],
            `${name}.json`,
            { type: 'text/json;charset=utf-8' }
        )
        saveAs(file)
        handleClose()
    }
    if (formText === '') setError(true)
}
```

Figur 24: Funktion för att spara dataformatet.

4.6 Driftsättning

För att köra applikationen lokalt utförs följande steg i kommandotolken:

1. Hämta senaste versionen av Npm (se avsnitt 4.5.1):
   npm install npm@latest -g
2. Hämta en lokal kopia av kodbasen från GitHub:
   git clone https://github.com/microgui/MicroGUI.git
3. Navigera till *ui* i filstrukturen där den lokala kopian har sparats:
   cd <lokal_kopia_plats>/ui
4. Installera alla Npm-paket:
   npm install
5. Kör applikationen:
   npm start

4.7 Test av applikationen

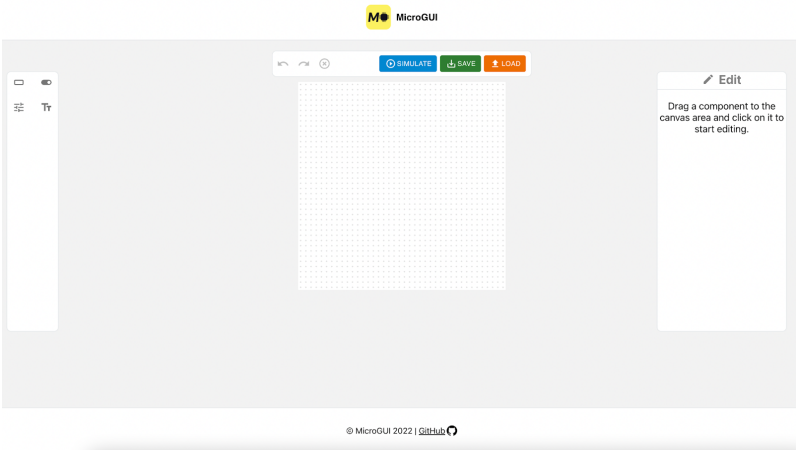
Testet gick ut på att göra två GUI:s - ett i webbapplikationen och ett med hjälp av ett separat programspråk med tilläggspaket. Programspråket som valdes var Java med tilläggspaketet javax.swing. Java valdes då viss erfarenhet av programspråket fanns sedan tidigare. Java Swing är ett API som innehåller grafiska komponenter för att skapa GUI:s, exempelvis JButton, JLabel och JSlider [26]. Resultatet från testet presenteras i avsnitt 5.5.

5
Resultat

I detta kapitel redovisas vad projektet har resulterat i. Målet var att utveckla en mjukvara där användare kan generera GUI:s och därmed sänka kunskapströskeln samt spara tid i utvecklingen av GUI:s.

5.1 Webbapplikationen

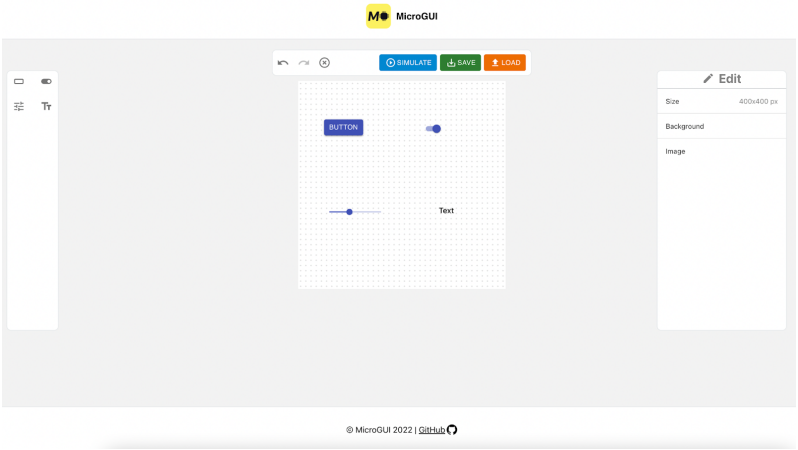
Editor-sidan är det första användaren möts av när webbapplikationen startas upp. I figur 25 visas en skärmdump från webbapplikationen. Utseendet på sidan liknar den mockup som presenterades i avsnitt 4.4, figur 12. Till vänster i bild syns Toolbox tillsammans med ikoner för användarkomponenterna. I mitten av bilden syns Toolbar och Canvas. Till höger finns Editbox som visar en kort instruktion där användaren uppmanas att dra en användarkomponent till ritytan för att börja redigera GUI:t.



Figur 25. Editor-sidan.

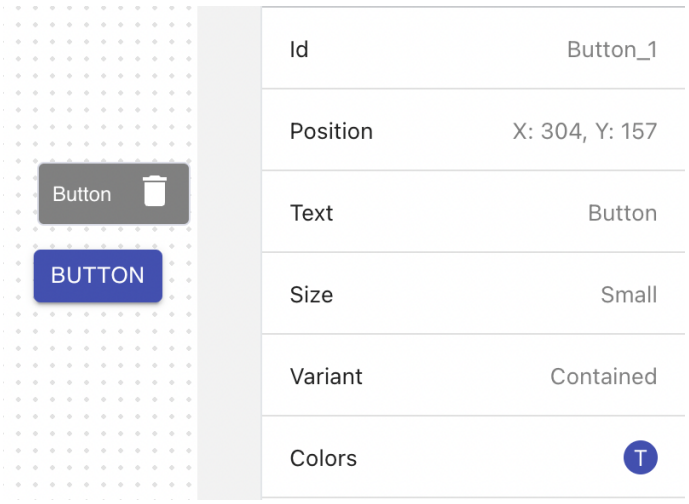
5.2 Att skapa ett GUI

I figur 26 visas en av varje användarkomponent som blivit indragen och släppt i ritytan. En användarkomponent placerar sig där muspekaren befinner sig när användaren släpper datormusens vänsterknapp.



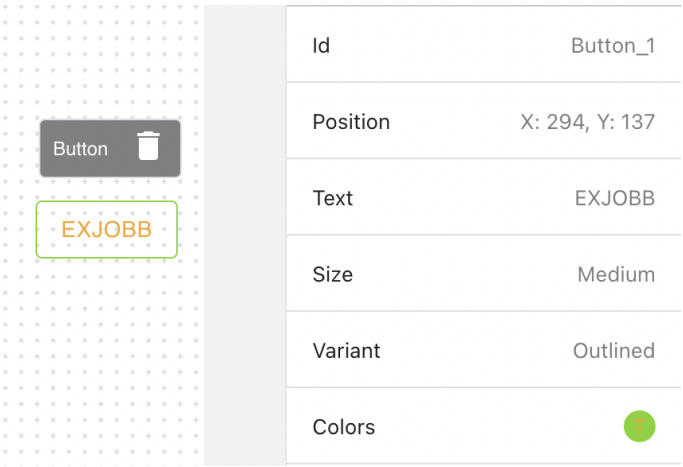
Figur 26. Användarkomponenter i ritytan.

Genom att trycka på en användarkomponent i ritytan blir den markerad. Detta innebär att en Tooltip visas ovanför användarkomponenten (till vänster i figur 27). Det är möjligt att ta bort komponenten genom att trycka på soptunnan i Tooltip. Vidare blir även ändringsbara attribut synliga i Editbox (till höger i figur 27).



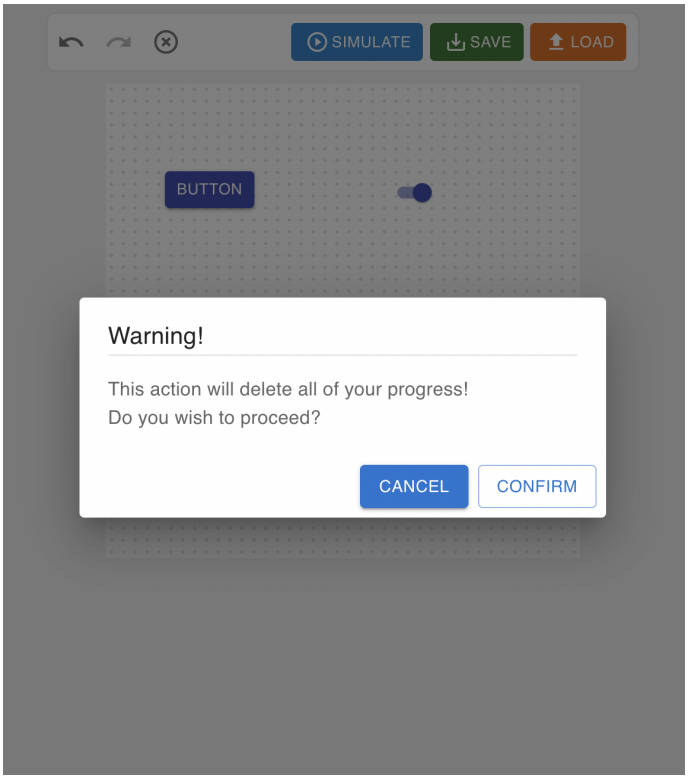
Figur 27. Vald knapp och dess ändringsbara attribut.

Nedan visas hur användarkomponenten ovan blivit redigerad. I figur 28 syns att texten istället är “Exjobb”, storleken är “Medium” och varianten är “Outlined”. Utöver dessa har även färgerna ändrats för bakgrund och text i knappen.



Figur 28. Redigerad knapp.

I Toolbar används en knapp med symbolen "x" för att rensa hela ritytan. När användaren trycker på denna knapp öppnas en dialogruta för att bekräfta handlingen (figur 29). Till vänster om knappen finns även två knappar som möjliggör att ångra och göra om handlingar.

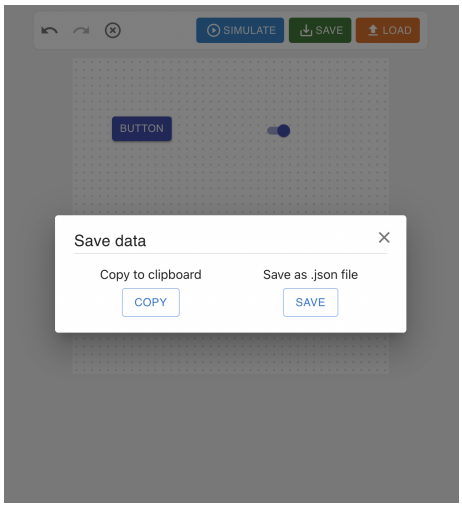


Figur 29. Dialogruta vid knapptryck.

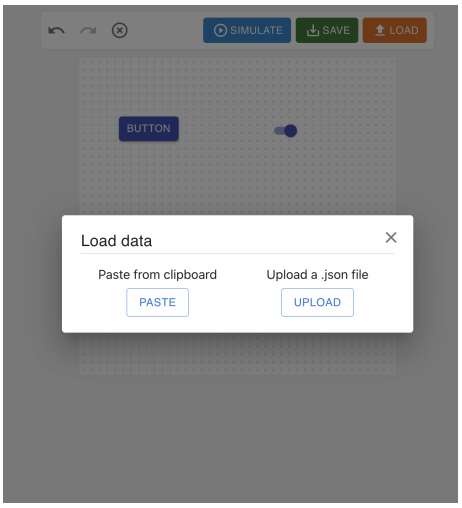
5.3 Spara och ladda upp GUI

För att spara tillståndet i ritytan kan knappen “Save” användas. Genom att trycka på denna knapp öppnas en dialogruta där användaren har möjlighet att spara data på två sätt (se figur 30). Det ena sättet är att kopiera det genererade dataformatet till urklippet på datorn. Det andra sättet är att spara dataformatet som en fil på hårddisken. Om användaren väljer att spara en fil på datorn öppnas en ny dialogruta där användaren får ange ett namn som filen sparas som.

På liknande vis kan användare välja att ladda upp data för att fortsätta använda ett sparat tillstånd av ritytan. Genom att trycka på knappen “Load” visas en dialogruta där data kan laddas upp via datorns urklipp eller via en fil på hårddisken (se figur 31).



Figur 30. Dialogruta vid spara data.



Figur 31. Dialogruta vid ladda upp data.

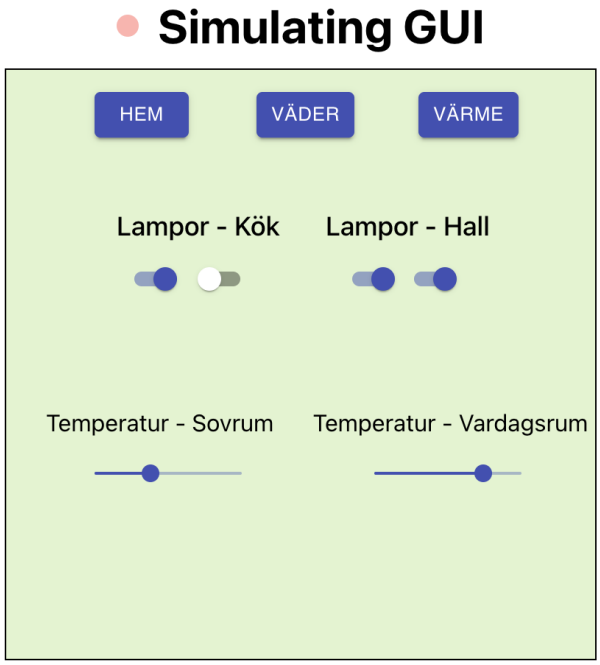
Tillståndet av ritytan och indragna användarkomponenter sparas i ett dataformat enligt figur 32. Själva Ritytan är definierad som ett JSON-objekt “ROOT”. ROOT innehåller attribut som exempelvis “isCanvas”, “props” och “nodes”. Dessa attribut beskriver ritytan. “props” innehåller information som exempelvis ritytans längd och bredd. “nodes” är en lista över alla användarkomponenter i ritytan och refererar i sin tur till ett JSON-objekt för varje sådan användarkomponent. I detta fall finns det en användarkomponent “Button_1” som exempelvis har “props” för text, storlek och variant.

```
{
    "ROOT": {
        "type": {
            "resolvedName": "CanvasArea"
        },
        "isCanvas": true,
        "props": {
            "id": "canvasElement",
            "width": "400",
            "height": "400",
            "background": {
                "r": 245,
                "g": 166,
                "b": 35,
                "a": 1
            }
        },
        "displayName": "Canvas",
        "custom": {},
        "hidden": false,
        "nodes": [
            "Button_1"
        ],
        "linkedNodes": {}
    },
    "Button_1": {
        "type": {
            "resolvedName": "Button"
        },
        "isCanvas": false,
        "props": {
            "text": "Button",
            "size": "small",
            "variant": "contained",
            "background": {
                "r": 63,
                "g": 81,
                "b": 181,
                "a": 1
            },
            "color": {
                "r": 255,
                "g": 255,
                "b": 255,
                "a": 1
            },
            "pageX": 50,
            "pageY": 74
        },
        "displayName": "Button",
        "custom": {},
        "parent": "ROOT",
        "hidden": false,
        "nodes": [],
        "linkedNodes": {}
    }
}
```


Figur 32. Exempel på dataformatet.

5.4 Simulering av GUI

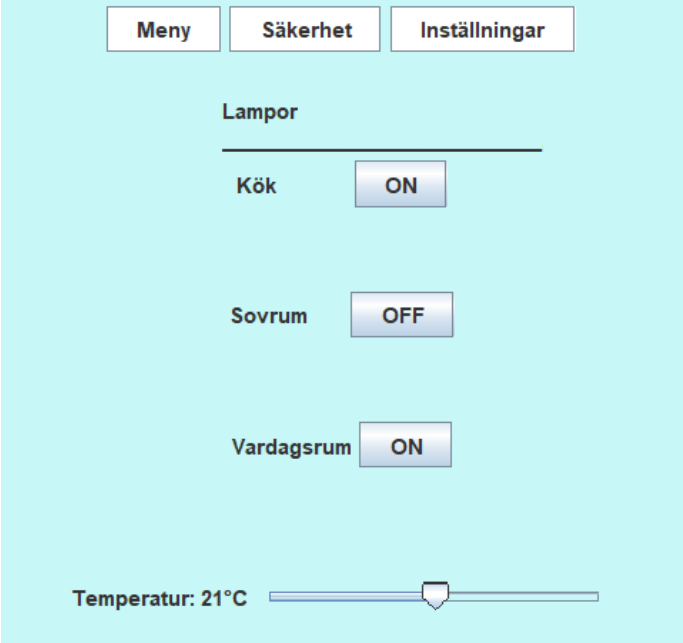
För att simulera ett skapat GUI kan användaren trycka på knappen “Simulate” som finns i Toolbar. Genom att trycka på knappen öppnas en ny flik i webbläsaren där Simulator-sidan visar GUI:t (se figur 33). Som tidigare nämnt så går det inte att flytta på, eller ta bort, användarkomponenter i simulatorn.



Figur 33. Ett GUI i simulator-sidan.

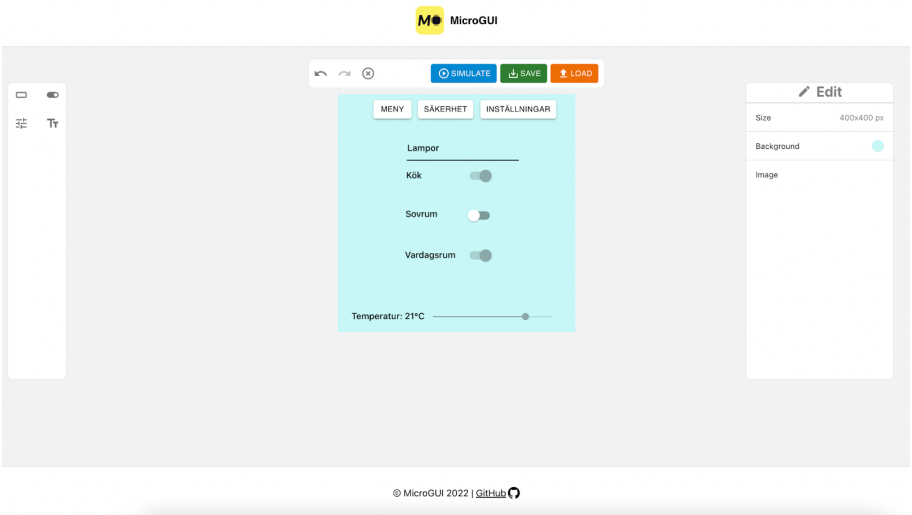
5.5 Testresultat

Som tidigare nämnt gick testet av webbapplikationen ut på att skapa två liknande GUI:s, ett i applikationen, och ett i programspråket Java. I figur 34 visas det GUI som framställdes i Java. Det tog ungefär 60 minuter att skapa detta GUI.

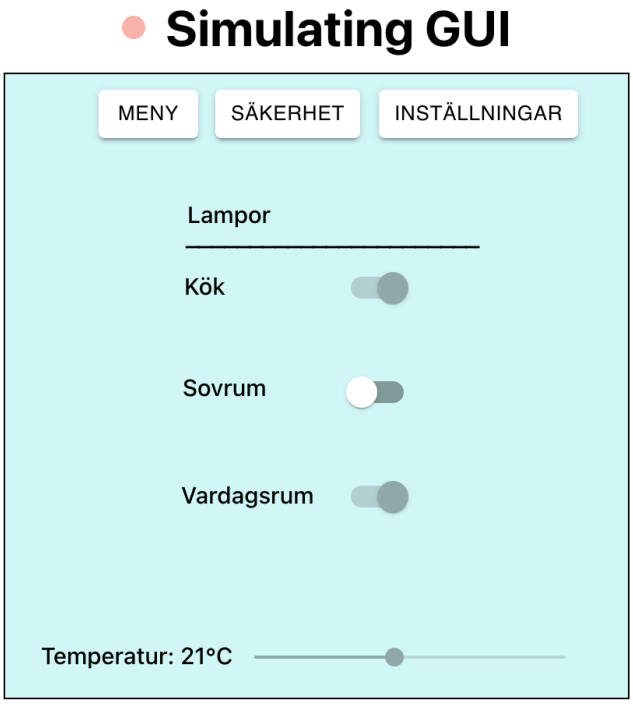


Figur 34. Skapat GUI för test i Java.

Efter att ha skapat ett GUI i Java så skapades ett liknande GUI med hjälp av webbapplikationen. Resultatet av detta syns i figur 35. Detta GUI tog ungefär 5 minuter, eller cirka 8% av tiden, att framställa. I figur 36 simuleras GUI:t via webbapplikationen.



Figur 35. Skapat GUI för test i webbapplikationen.



Figur 36. Skapat GUI för test i simulatorn.

6
Slutsats

Syftet med examensarbetet har varit att förenkla och snabba upp konstruktionen av GUI:s för konsumentprodukter. Utifrån detta syfte har en webbapplikation utvecklats. Webbapplikationen gör det möjligt för användare att generera GUI:s som exempelvis skulle kunna användas i skärmar för smarta enheter. Vi anser därmed att examensarbetets syfte är uppfyllt.

6.1 Diskussion

En av styrkorna under projektets gång har varit användandet av Scrum-metoden för planering. Detta har varit mycket lyckat för utvecklingsarbetet. Varje vecka har det funnits tydliga uppgifter och mål att klara av tack vare planeringen. Planeringen av specifika uppgifter har skett i princip veckovis, men inför varje sprint sattes övergripande, och mer långsiktiga, mål upp.

Efterforskning är något som kunde ha skötts bättre från vår sida. Tidigt i projektet hamnade vi på lite fel bana och tappade en vecka på en lösning som inte alls hade fungerat i det långa loppet. Detta berodde dels på feltolkning av uppgiften, men även på att inte tillräcklig efterforskning hade gjorts för att kunna hitta möjliga lösningar. Efter ett samtal med handledare på företaget blev uppgiften tydligare och mer efterforskning gjordes. Detta ledde till lösningen som senare implementerades och användes.

Ett område som kunde fått större fokus är testning av kod. Vi blev uppmanade av handledare på företaget att läsa om kod-testning och att försöka implementera detta i projektet. Tyvärr hamnade fokus på andra saker och det resulterade i att väldigt begränsad testning av koden genomfördes. Detta är något som vi absolut skulle göra annorlunda och bättre vid ett nytt projekt.

Som testresultatet visade så var det en markant skillnad tidsmässigt att använda applikationen jämfört med Java för att generera ett GUI. För att ytterligare testa huruvida kunskapströskeln sänktes med vår lösning så hade fler tester kunnat göras. Exempelvis skulle en eller flera utomstående utvecklare kunnat testa både applikationen och Java (eller något annat programspråk) och redovisa sin upplevelse. Som redan nämnt i avgränsningar (avsnitt 1.3) var detta inte möjligt på grund av tidsbrist.

Vidare kan vi konstatera att struktur och dokumentation av projektet har fungerat mycket väl. Bland annat finns det dokumenterat hur nya användarkomponenter kan läggas till och hur man på andra sätt kan bidra till projektet. Detta var något som prioriterades högt under hela arbetsprocessen för att säkerställa att projektet skulle kunna leva vidare även efter detta examensarbete.

Detta projekt har behandlat hur GUI:s kan genereras för konsumentprodukter. För att faktiskt kunna använda ett genererat GUI i verkligheten måste mer utveckling ske gällande hur hårdvaran ska kunna tolka dataformatet och visualisera GUI:t. Tanken från handledare på företaget är att detta ska lösas i ett annat examensarbete. Detta är en av anledningarna att dokumentation har varit viktigt - för att nya studenter ska kunna ta vid där vårt examensarbete avslutats.

I övrigt så har ärenden skapats på GitHub innehållande önskad funktionalitet som utvecklare kan bidra med. Exempelvis önskas nya användarkomponenter, en utökning av Simulator-sidan samt möjlighet att kopiera och klistra in användarkomponenter. En utökning av Simulator-sidan hade bland annat kunnat innebära möjlighet att koppla upp mot en enhet och tillståndet av dess användargränssnitt i realtid.

6.2 Hållbarhet och etik

Programutveckling kan, likt många andra yrken med teknik som produkt, påverka samhället på olika vis. Det är därför viktigt att belysa samhälleliga, etiska och ekologiska aspekter som kan komma att påverkas av detta projekt.

Ur en ekologisk synvinkel skulle en vidareutveckling av projektet kunna leda till att minska onödigt resande till och från arbetet. Uppgifter som tidigare krävde att någon var fysiskt på plats skulle nu vara möjligt att utföra på distans. Ett exempel på en sådan uppgift kan vara underhåll av användargränssnitt till en enhet som har upphört att fungera eller är i behov av ny funktionalitet. Ytterligare en aspekt som är värd att belysa är återanvändning. Om det blir lättare att underhålla och återanvända pekskärmar till nya syften skulle det kunna leda till minskat elektroniskt avfall.

Projektet resulterar i öppen källkod som vem som helst kan använda sig av och modifiera helt utan kostnad. Ur ett samhällsperspektiv innebär det att alla har möjlighet att nyttja och förbättra produkten, vilket i sin tur kan leda till bättre lösningar i framtiden. Det finns dock även etiska frågeställningar som kan uppstå beträffande öppen källkod. Med tanke på att vem som helst har tillgång till produkten så finns det inget sätt att begränsa dess användning. Detta innebär att om någon med oetiska intentioner vill använda eller vidareutveckla produkten för ett icke-etiskt ändamål så går inte det att hindra.

Litteraturförteckning

[1] ”Smart Home - Sweden | Statista Market Forecast”, Statista. [Online]. Tillgänglig vid: https://www.statista.com/outlook/dmo/smart-home/sweden. [Åtkomstdatum: 04 mars 2022]

[2] ”Användning av internetanslutna enheter eller system för privat bruk efter kön och redovisningsgrupp. År 2020”, 18 november 2020. [Online]. Tillgänglig vid: https://www.statistikdatabasen.scb.se/pxweb/sv/ssd/START___LE___LE0108___LE0108M/LE0108T52/. [Åtkomstdatum: 04 mars 2022]

[3] ”What Is A Gantt Chart? | Definition & Examples | APM”, 20 september 2017. [Online]. Tillgänglig vid: https://www.apm.org.uk/resources/find-a-resource/gantt-chart/. [Åtkomstdatum: 21 mars 2022]

[4] K. Iniewski, Red., Embedded systems: hardware, design, and implementation. Hoboken, New Jersey: John Wiley & Sons, Inc, 2012.

[5] ”What is an Embedded System? Definition and FAQs | OmniSci”. [Online]. Tillgänglig vid: https://www.omnisci.com/technical-glossary/embedded-systems. [Åtkomstdatum: 25 februari 2022]

[6] ”Definition av sakernas internet”. [Online]. Tillgänglig vid: https://techlib.se/definition/internet_of_things.html. [Åtkomstdatum: 06 april 2022]

[7] T. Sundström, Internet of Things - En guide till sakernas internet, 1:a uppl. 2016 [Online]. Tillgänglig vid: https://internetstiftelsen.se/app/uploads/2021/01/internet-of-things.pdf. [Åtkomstdatum: 21 mars 2022]

[8] A. M. St. Laurent, Understanding Open Source and Free Software Licensing. Sebastopol: O’Reilly Media, Inc., s.4, 2008 [Online]. Tillgänglig vid: http://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=443485. [Åtkomstdatum: 21 mars 2022]

[9] ”The Open Source Definition | Open Source Initiative”. [Online]. Tillgänglig vid: https://opensource.org/docs/osd. [Åtkomstdatum: 21 mars 2022]

[10] S. S. Levine och M. J. Prietula, ”Open Collaboration for Innovation: Principles and Performance”, Organization Science, vol. 25, nr 5, s. 1414–1433, okt. 2014, doi:10.1287/orsc.2013.0872. [Online]. [Åtkomstdatum: 21 mars 2022]

[11] ”Vad Node.js innebär och varför du ska använda detta”, Kinsta. [Online]. Tillgänglig vid: https://kinsta.com/se/kunskapsbas/vad-node-js/. [Åtkomstdatum: 09 maj 2022]

[12] ”Introduktion till JSON”. [Online]. Tillgänglig vid: https://www.json.org/json-sv.html [Åtkomstdatum: 13 maj 2022]
[13] A. Boduch, React and React Native: use React and React Native to build applications for desktop browsers, mobile browsers, and even as native mobile apps. Birmingham Mumbai: Packt Publishing, 2017.
[14] H. Götesson, ”Utmaningar och möjligheter med att använda skärmläsare”, Enskede, nov. 2019 [Online]. Tillgänglig vid: https://www.srf.nu/globalassets/intressepolitiska-dokument/utmaningar-och-mojligheter-med-att-anvanda-skarmlasare_tillganglig_v2.pdf. [Åtkomstdatum: 06 april 2022]
[15] ”React Components”. [Online]. Tillgänglig vid: https://www.w3schools.com/react/react_components.asp. [Åtkomstdatum: 13 maj 2022]
[16] ”React SPA for Increased Performance & CX”, Royal Cyber Blog, 12 januari 2021. [Online]. Tillgänglig vid: https://www.royalcyber.com/blog/ecommerce/react-spa-for-increased-performance-and-customer-experience/. [Åtkomstdatum: 20 april 2022]
[17] ”Overview | craft.js”. [Online]. Tillgänglig vid: https://craft.js.org//docs/overview. [Åtkomstdatum: 21 mars 2022]
[18] ”What is Material UI in React?”, Educative: Interactive Courses for Software Developers. [Online]. Tillgänglig vid: https://www.educative.io/edpresso/what-is-material-ui-in-react. [Åtkomstdatum: 21 mars 2022]
[19] “About us - MUI”. [Online]. Tillgänglig vid: https://mui.com/about/. [Åtkomstdatum: 06 april 2022]
[20] ”Scrum Guide | Scrum Guides”. [Online]. Tillgänglig vid: https://scrumguides.org/scrum-guide.html. [Åtkomstdatum: 18 februari 2022]
[21] Atlassian, ”What is Jira Software used for?”, Atlassian. [Online]. Tillgänglig vid: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for. [Åtkomstdatum: 21 mars 2022]
[22] ”Minimum Viable Product (MVP)”. [Online]. Tillgänglig vid: https://www.productplan.com/glossary/minimum-viable-product/. [Åtkomstdatum: 13 maj 2022]
[23] ”Why Visual Studio Code?” [Online]. Tillgänglig vid: https://code.visualstudio.com/docs/editor/whyvscode. [Åtkomstdatum: 08 april 2022]
[24] ”Git - user-manual Documentation”. [Online]. Tillgänglig vid: https://git-scm.com/docs/user-manual. [Åtkomstdatum: 18 februari 2022]
[25] ”HTML Web Storage API”. [Online]. Tillgänglig vid: https://www.w3schools.com/html/html5_webstorage.asp. [Åtkomstdatum: 13 maj 2022]
[26] ”javax.swing (Java Platform SE 7)”. [Online]. Tillgänglig vid: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html. [Åtkomstdatum: 19 maj 2022]

Institutionen för Data- och informationsteknik
Chalmers tekniska högskola
Göteborg, Sverige
www.chalmers.se

CHALMERS