This is the MSDOS system design document

Skapad av

Erik Bengtsson
Erik Anttila Ryderup
Jonas Nordin
Axel Hertzberg
Theodor Lyrheden
Oliver Österberg
Filip Hansson

1. Introduktion

Under detta avsnitt beskrivs applikationens syfte och bakgrund samt en förklarande lista med viktiga definitioner som används i dokumentet.

1.1 Syfte

Dokumentet har som syfte att beskriva bokningssystemet MS DOS kod-arkitektur, grafiska gränssnitt och bakomliggande datastrukturer. Bokningssystemet är till för att användas i en bostadsrättsförening eller hyreshus och ska göra det möjligt för boende att boka gemensamma lokaler/prylar via en webbapplikation. Bokningssystemet är beskrivet mer detaljerat i **Business Model Canvas.**

1.2 Omfång

Dokumentet kommer främst att gå igenom de arkitektur/designmässiga delarna av applikationen och kommer inte att demonstrera hur användandet av applikationen fungerar.

1.3 Definitioner

CI - Continous Integration. En byggserver som automatiskt testar koden när det laddas upp kod på git-repot.

2. Designöverblick

I denna delen kommer applikationens övergripande design gås igenom.

2.2 Problembeskrivning

Samhället har idag blivit mer digitaliserat än någonsin. Trots digitaliseringen har många enkla digitala lösningar i just flerbostäder länge lyst med sin frånvaro. Beställaren har i projektet efterfrågat en enkel webbapplikation där det går att boka husets gemensamma faciliteter digitalt, istället för dagens analoga lösning där exempelvis tvättider bokas via en tavla.

2.3 Använda teknologier

Webbapplikationen skrivs i språket JavaScript med hjälp av det grafiska ramverket React. För att använda programmet krävs det att användaren har en webbläsare installerad med stöd för JavaScript. För testning har jest använts vilket berörs mer under 4.4.

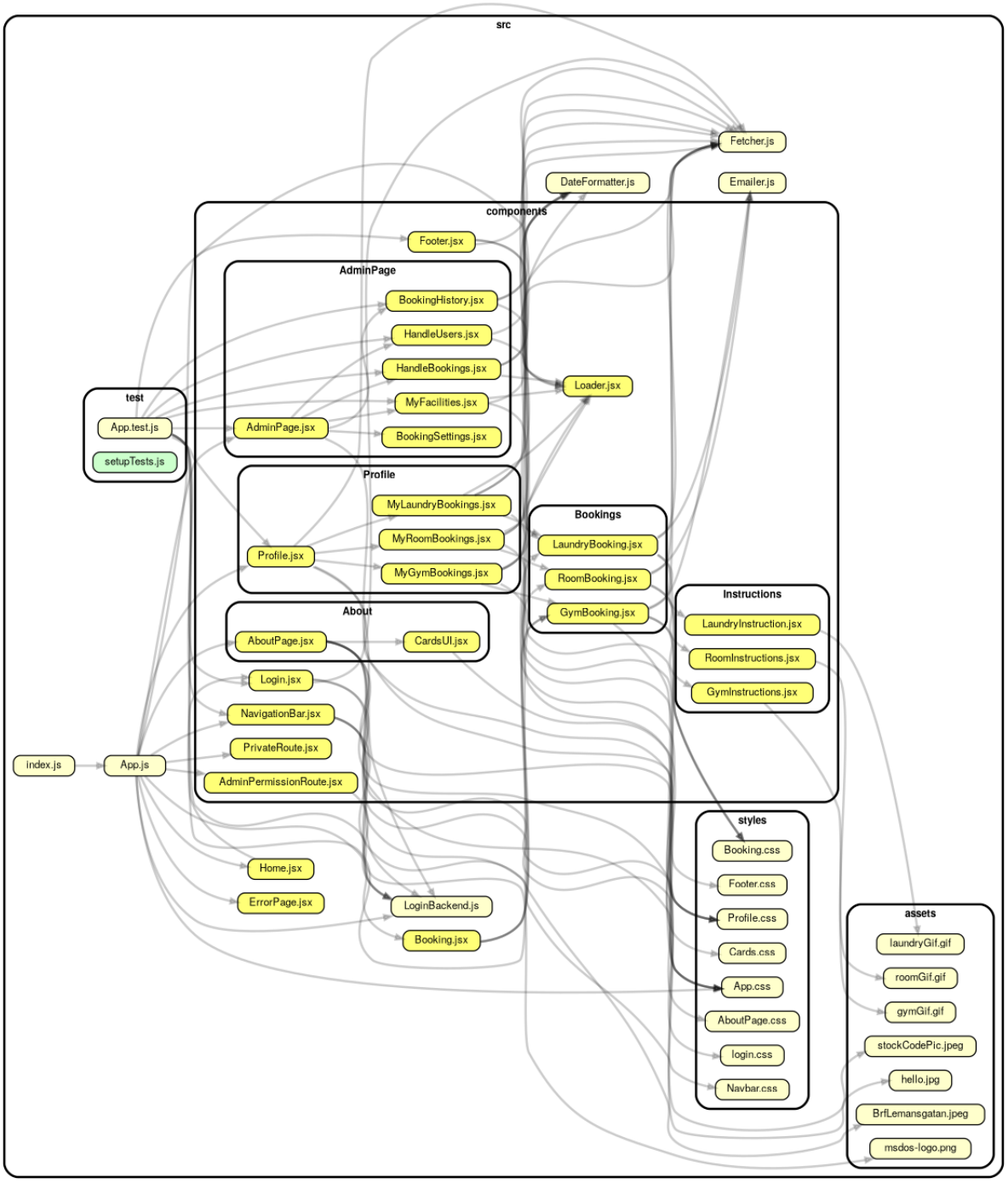
2.4 Övergripande systemarkitektur

Applikationen och databasen (ref 4.1) körs för tillfället lokalt på datorn av utvecklings-tekniska skäl. Det är ett beslut som tagits av gruppen för att det ska vara enklare att lägga till nya funktioner till applikationen utan att störa driften.
Applikationen är uppbyggd av dessa två huvudsakliga delar:

- Spring-Boot server
- App.js som läser in alla React-komponenter.

Alla React-komponenter ansvarar mer eller mindre självständigt för både logik och utseende vilket inte alltid är helt optimalt ifrån en arkitektur-mässig synvinkel, något som kommer diskuteras senare i dokumentet.
När en särskild komponent laddas in på sidan så hämtar komponenten innehållet dynamiskt information från databasen och visar upp det grafiskt på skärmen.

Spring-Boot server är ett java bibliotek som tillhandahåller ett RESTful api. Mer konkret innebär detta att man med hjälp av en GET kan hämta till exempel alla användare eller alla bokningar. Det går också genom POST att lägga upp exempelvis nya bokningar till servern.



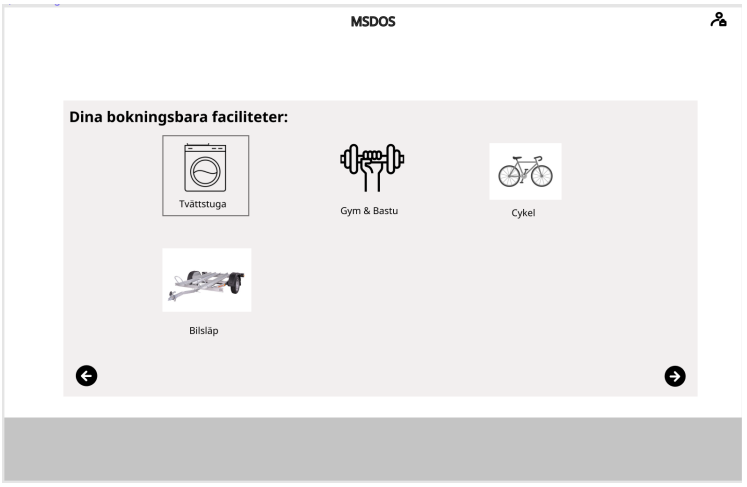
2.5 Designmönster

Detta har redan tagits upp

3. Webbapplikations grafiska gränssnitt

I den här sektionen beskrivs webbapplikations grafiska gränssnitt. I Mockup (3.1) illustreras hur applikationens grafiska gränssnitt var tänkt att se ut. Under 3.2 beskrivs de design-avvägningar som gjorts.

3.1 Mockup



3.2 Design av användargränssnittet

Tanken med användargränssnittet är att den ska vara lätt att använda. Ambitionen är att det ska vara enkelt för någon som inte är van vid datorer att använda mjukvaran. Detta är en följd av applikationens målgrupp är svårdefinierad då människor i alla åldrar och med olika datorvana kan tänkas bo i samma hus.

Längst upp i applikationen finns en ruta med länkar som är den primära navigationen i appen. Detta känns igen från flera olika hemsidor då det är ett vanligt sätt att lägga upp navigationen. I denna komponent har även ett så kallat Escape Hatch implementerats. Det syftar till att användaren ska känna sig trygg när denne navigerar på webbsidan och på ett knapptryck återvända hem. I vår applikation är det logotypen i vänstra hörnet som fungerar som denna.
I webbapplikationen finns mängder av klickbara komponenter och knappar som användaren ska interagera med. Det har gjort att fokus i gränssnitts-designen har legat på att visa vilka av dessa komponenter som användaren dynamiskt har möjlighet att interagera med. Exempel på detta är att knapparna vid inloggning är gråmarkerade fram tills dess att användaren har skrivit in något i textfältet då den byter färg till blå.

4. Datalagring, säkerhet och tester

Den här sektionen beskriver hur programmet hanterar data, säkerhet och hur testning av funktionaliteten sker.

4.1 Datalagring

För att spara data används programmet JSON server som i applikationen fungerar som en enkel databas.

4.2

4.3 Security

Eftersom det är en webbapplikation bör säkerhetsaspekter tas hänsyn till. Programmet tar dock inte hänsyn till säkerhet mer än att validera lösenord lokalt. Eftersom det sker hos användaren så är det enkelt att undvika kontrollen, vilket är ett problem. Ett ännu större säkerhetsproblem i applikationen är för tillfället är att JSON server inte har någon möjlighet att säkerställa att användaren inte får åtkomst till informationen. I praktiken innebär detta att `/api/users` ger ut all information om alla användare, inklusive administratörer.

Dessutom hämtar webbapplikationen lösenordet i klartext för det angivna lägenhetsnumret när användaren klickar på ‘logga in’, även om lösenordet faktiskt är fel. Detta eftersom kontrollen om lösenordet stämmer sker i användarens webbläsare.

Lösenordet hashas inte på clientside för att det inte nämnvärt ökar säkerheten för projektets stakeholder. Det hade varit bättre för användaren med tanke på att samma lösenord kanske används på flera hemsidor.

Inloggningssystemet borde skrivas om till att hasha lösenordet på servern och sedan spara det hashade lösenordet. När inloggning sker borde lösenordet skickas på nytt till servern och sedan borde hash jämföras.

Eftersom webbsidan i dagsläget inte skickar data via ssl är den också sårbar för en ‘Man in the middle’-attack eftersom paketen skickas okrypterade. Utveckla möjlig \ref{https://www.namecheap.com/security/do-i-need-ssl-certificate/}

4.4 Testing

Testning sker via testbiblioteket Jest. Vilket valdes eftersom Reacts hemsida hänvisade till detta biblitotek för tester. \ref{https://reactjs.org/docs/test-utils.html}

I Jest finns det möjlighet att spara en snapshot på hur koden som bygger upp en webbsida ser ut och sedan jämföra den koden med det som faktiskt genereras. Detta fick gruppen inte riktigt att fungera med CI eftersom det finns en sida som ändrar sig varje minut och därför behöver en snapshot varje minut. Alternativen för att lösa detta är antingen att undanta den sidan eller att generera snapshots på byggservern. Det första alternativet har nackdelen att den sidan kan förstöras utan att det märks. Det andra alternativet har problemet att CI inte hittar fel med snapshots överhuvudtaget.