Stockholm Science & Innovation School
Gymnasiearbete
Handledare: Joakim Ågren

UI Designer
- Diploma project for the academic year 2023-2024

Uljas Wennström

Abstract

In today's rapidly evolving digital landscape, intuitive and collaborative user interface (UI) design tools are crucial. This report details the development of a web-based UI design application that supports real-time collaboration. The application leverages modern technologies such as React, Express.js, socket.io, and Prisma. Its architecture is divided into two distinct software segments: frontend and backend. React enhances the frontend development experience, while Express.js, socket.io, and Prisma manage the backend for real-time data synchronization and storage. The focus is on seamless integration between frontend and backend to facilitate designing, saving, and user collaboration. This report not only explores the inherent challenges of implementing a live collaboration system but also addresses critical aspects like object snapping, data storage, and optimizing data transfer for scalable use. The result of the project is a functional UI design application that supports real-time collaboration.

Innehållsförteckning

Abstract 2
Innehållsförteckning 3
1. Inledning 4
1.1 Bakgrund 4
1.2 Syfte, mål och frågeställning 4
2. Resultat 5
2.1 Arkitektur 5
2.1.1 Frontend 5
2.1.2 Backend 6
2.2 Kommunikation mellan klient och server 8
2.3 Autentisering och användarhantering 8
2.3.1 AuthService 9
2.3.2 Verify mellanprogrammen: 9
2.4 Spara, rendera och fleranvändare 10
2.4.1 Spara och rendera: 10
2.4.2 Samarbete och fleranvändare 10
2.5 Snapping 11
3. Diskussion & slutsats 12
3.1 Idéer och problem 12
3.2 Slutsats 13
Källförteckning 14

1. Inledning

1.1 Bakgrund

Användargränssnitt spelar en avgörande roll i den digitala värld vi lever i idag. De utgör inte bara det första intrycket av en applikation utan fungerar också som en bro mellan användaren och tekniken. Ett bra gränssnitt är inte bara estetiskt tilltalande, utan också intuitivt och användarvänligt, vilket underlättar människors interaktion med teknik varje dag.

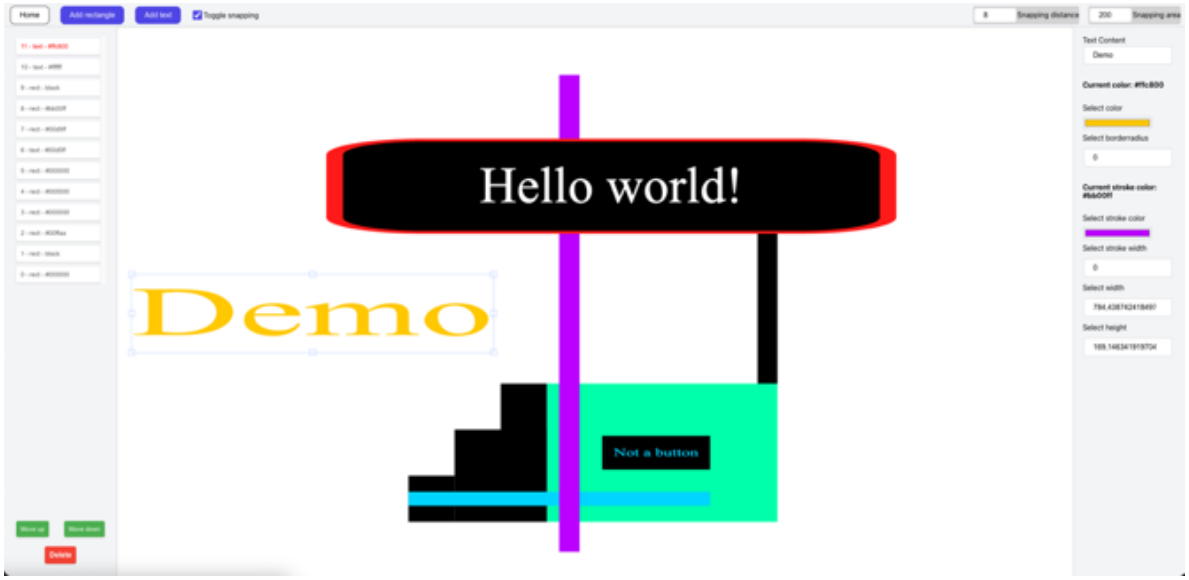
Gränssnittsdesignsverktyg är till för att underlätta för designers att skapa gränssnitt. I stället för att skriva kod tillåter dessa verktyg att designers snabbt och smidigt kan skapa prototyper för utvecklare att senare implementera.

1.2 Syfte, mål och frågeställning

Syftet med detta arbete är att bredda mina egna kunskaper och färdigheter i frontend och backendutveckling genom att skapa ett eget gränssnittsdesignsverktyg. Målet är att utveckla ett fullt fungerande gränssnittsdesignsverktyg med stöd för realtidssamarbeten med andra användare. Att utveckla ett sådant verktyg är ett omfattande och tidskrävande arbete som väcker flera frågor som ska besvaras i rapporten. För det första, hur kan ett realtidssamarbetssystem implementeras, kanske med hjälp av WebSockets? För det andra, på vilket sätt kan en databas användas för att effektivt spara designen samtidigt som rendering sker på frontend? Slutligen, hur kan datatransfer optimeras för att skala användningen av applikationen?

2. Resultat

Resultatet blev ett fungerande gränssnittsdesignverktyg i form utav en webb-applikation. Applikationen erbjuder en intuitiv användarupplevelse med stöd för realtidssamarbete. Bilden nedan visar applikationens designyta.



Figur 1: Skärmdump på designverktyget

2.1 Arkitektur

Applikationen är i grunden uppdelad i fyra delar: WordPress-sidan, webb-klienten, servern och databasen. De två första delarna, WordPress-sidan och webb-klienten, utgör frontend och är de delar användarna ser och interagerar med. Backend består av servern och databasen, som användarna inte direkt interagerar med. Alla fyra delar är placerade på separata webbhotell och måste interagera effektivt med varandra för att möjliggöra att användarna sömlöst kan utföra designarbete, spara sina projekt och samarbeta med varandra.

2.1.1 Frontend:

WordPress-sidans huvuduppgift är att informera om tjänsten och skicka vidare användarna till Webb-klienten. Det är alltså främst en marknadsföringsplatform. WordPress har bra SEO (Search Engine Optimization), vilket innebär att sökmotorer enklare kan hitta sidan. Dessutom, till skillnad från att programmera en webbsida så går WordPress-sidor mycket snabbt att sätta upp

Webb-klienten eller bara klienten är byggd i React och utgör hjärtat av applikationen. Det är här användarna interagerar med verktyget. Allt som har med designande att göra ska hanteras av klienten. Appen kan egentligen fungera med endast klienten utan servern. Dock kräver samarbete och sparande en server och databas.

Anledningen till att React valdes är p.g.a. dess komponenbaserade arkitektur. Det gör det möjligt att dela upp gränssnittet i återanvändbara byggstenar där varje del ansvarar för sin uppgift.

Klientens komponenter är indelat i följande kategorier eller mappar. **Pages**, vilket är de olika sidorna. Alltså hemskärmen, designverktyget och inbjudningssidan. **Features** som är delar på de olika sidorna som kräver en egen komponent, till exempel ”canvasen” (ytan där designen visas) eller ”lagervyn” på

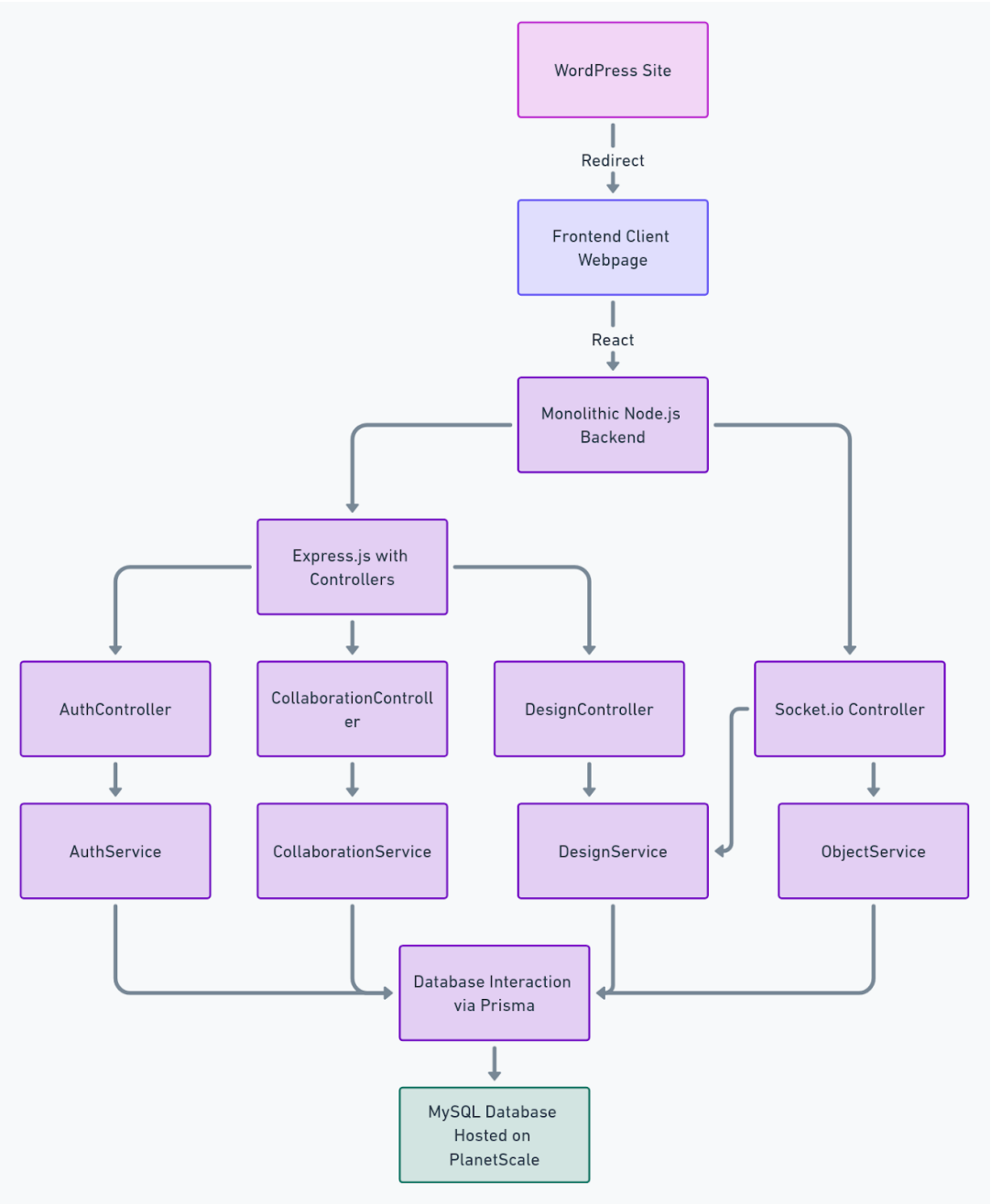
designsidan. Mappen **components** innehåller komponenter som återanvänds ofta. Det inkluderar färdigdesignade knappar och inputfält som kan placeras var som helst i klienten.

2.1.2 Backend:

Servern och databasen är också två mycket viktiga delar. Servern hanterar alla processer som inte kan utföras lokalt på klientens sida. Till exempel **fungerar** den som en mellanhand mellan användare under samarbete. Servern ser även till att användarnas arbete inte bara sparas säkert i databasen, utan också att det kan hämtas effektivt vid efterfrågan. Den hanterar också användarnas konton och ser till att alla har tillgång till sina egna designs och inte andras utan tillstånd.

Servern är byggd i Node.js vilket är en javascriptmotor för att köra javascriptkod på server, vilket normalt sett endast körs i webbläsaren. Serverarkitekturen är monolitisk men servicebaserad. Det innebär att koden inkapslad och uppdelad i mindre ”tjänster” som ansvarar för sitt eget respektive uppdrag. Tjänsterna kan även direkt prata med databasen. Men eftersom systemet är monolitiskt, körs alla dessa tjänster på en och samma server, till skillnad från en mikrotjänstarkitektur där tjänsterna är distribuerade över flera servrar. Tjänsterna kan inte kommunicera själva med klienten utan mellan varje tjänst och klienten finns en ”controller”. Varje ”controllers” ansvarar för att ta emot anrop via nätverksprotokollen http eller webb-socket. Därefter föra vidare informationen till en service och sedan skicka tillbaka ett svar till klienten.

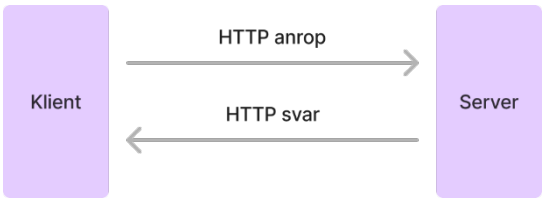
Databasen har ingen logisk funktion utan är endast till för att spara och hålla reda på all data som krävs för att köra applikationen. Det innefattar användare, designs, objekt i designen, inbjudningar, samarbetspartners mm.



Figur 2: Schematisk bild över applikationen med fokus på backend

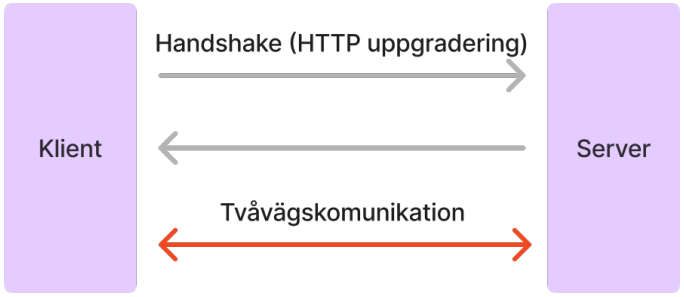
2.2 Kommunikation mellan klient och server

Huvuddelen av kommunikationen sker via nätverksprotokollet HTTP (Hypertext Transfer Protocol). HTTP är standardprotokollet för överföring av data på internet, bland annat hela webbsidor men också andra typer av data. Processen fungerar så att klienten, till exempel en webbläsare, skickar en förfrågan till servern. Denna förfrågan kan vara en begäran om att hämta en webbsida, vilket då skickas tillbaka till webbläsaren (Ferguson, 2021). I fallet med designapplikationen så skulle klienten kunna skicka en begäran om att ladda in en lista på designs. Servern bearbetar förfrågan och svarar med en lista över tillgängliga designprojekt som klienten kan interagera med.



Figur 3 : Hur fungerar HTTP

Den delen som hanterar sparning av objekt i designen och sammarbetsuppdateringar är inte byggd med HTTP utan i stället WebSocket via biblioteket socket.io. WebSocket är en teknik som möjliggör tvåvägskommunikation i realtid mellan en webbserver och en klient. WebSocket skiljer sig från det traditionella HTTP-protokollet, som är enkelriktat och kräver att klienten initierar alla förfrågningar. När en WebSocket-anslutning etableras, sker en så kallad "handshake" via HTTP. Därefter uppgraderas kommunikationen till WebSocket-protokollet, vilket tillåter kontinuerlig och interaktiv kommunikation (Socket.io, 2024). Det betyder att servern kan skicka data till klienten utan att klienten först behöver skicka en förfrågan, vilket är användbart för applikationer som kräver snabba uppdateringar i realtid, exempelvis chattapplikationer eller onlinespel. I den här applikationen används det för att snabbt kunna spara ändringar i designen. Dessutom för att kunna uppdatera samarbetspartnerns gränssnitt när en ändring i designen.



Figur 4:Hur fungerar Websocket

2.3 Autentisering och användarhantering

Autentisering och användarhantering är en mycket kritisk del av applikationen och möjliggör att användarnas sparade projekt/designer endast finns tillgängligt för ägaren och de användare skaparen tillåter. Nästan allt i appen kräver autentisering och därför är det viktigt att det fungerar på ett säkert och smidigt sätt.

Autentisering och användarsystemet är uppdelat i tre delar. ”AuthService” som ansvarar för registrering, inloggning och borttagning av konto. ”Verify-mellanprogrammen” har till uppgift att kontrollera ifall användaren är inloggad och isåfall vilken användare. ”Autherize” tjänsten innehåller funktioner för att verifiera ifall användaren har tillgång till en specifik design, dels ifall hen är ägare eller ifall ägaren tillåtit användaren åtkomst.

2.3.1 AuthService:

Registreringen fungerar så att användaren matar in epost och ett lösenord. Först så verifieras eposten med hjälp av regex (textfiltreringssystem) för att bekräfta att den inmatade texten är i format av en e-postadress. Därefter söks databasen igenom efter adressen för att se ifall användaren redan är registrerad. Om testerna går igenom så hashas och saltas lösenordet via Bcrypt och en ny användare läggs till i databasen med epost salt och det hashade lösenordet.

Hashning är när en matematisk enkelriktad funktion används för att skapa en krypterad version av lösenordet. Denna version består av till synes slumpmässiga tecken som inte kan tydas tillbaka till lösenordet. Det leder till att lösenorden blir svårkapade ifall en cyberkriminell får tillgång till databasen (Grigutytė, 2023). Varje gång ett och samma lösenord går igenom hashingfunktionen återskapas exakt samma kombination av siffror och bokstäver. Det är ett problem eftersom om ett lösenord kapas kan alla andra likadana lösenord kapas. Därför används ”salting”. ”Salting” är när slumpmässiga siffror läggs på det originella lösenordet innan de hashas. Det leder till att lösenord som registreras och egentligen är likadana inte har samma hash (Grigutytė, 2023).

Vid inloggning tas e-post och lösenord in. Databasen söks igenom efter användare med samma e-post. Ifall en sådan finns så läggs de sparade saltet från den användaren till på det intagna lösenordet. Därefter hashas lösenordet och de nya hashet jämförs med hashet av den upphittade användaren. Ifall hashen matchar godkänns inloggningen och en JWT (json web token) genereras och skickas tillbaka i http svaret. Klienten sparar sedan JWT:n i lokal storage (lagringsplats i webbläsaren). JSON web tokens är ett kompakt och säkert sätt att överföra information. De används ofta för att bekräfta identiteten på användare eftersom de kan bära användarinformation på ett smidigt sätt och är mycket svåra att återskapa. Själva JWT:n är endast en serie siffror och bokstäver men kan avkodas till viktig information med hjälp av en annan hemlig nyckel på servern. (Jwt.io, u.d.)

2.3.2 Verify mellanprogrammen:

Nästan alla anrop till servern kräver autentisering. Autentisering handlar om att säkerställa att rätt person har tillgång till rätt resurser. Det genomförs med hjälp av JWT:n (som förklaras på 2.3.1) och verify mellanprogrammen. Mellanprogram är funktioner som bearbetar inkommande förfrågningar innan de når den slutliga ändpunkten. Vid ett anrop skickar klienten med JWT:n som last. Innan förfrågan når den specifika ändpunkten i servern, passerar den genom mellanprogrammen. "Verify" mellanprogrammet kontrollerar JWT:n för att bekräfta användarens autentisering. Om JWT:n är giltig, tillåts anropet fortsätta till den avsedda ändpunkten; om inte, kan mellanprogrammet stoppa förfrågan och skicka tillbaka ett felmeddelande till klienten.

I och med att vissa delar av applikationen körs via websockets andra http så krävs två liknande verifieringsmellanprogram. Dessa fungerar huvudsakligen likadant bara att informationen tas emot och skickas ut på olika sätt. Båda mellanprogrammen börjar med att JWT:n först extraheras från den inskickade förfrågningen. Vid giltigt token valideras det mot en hemlig nyckel som sparas i servern. Om jwt inte hittas eller verifieringen misslyckades avbryts processen och ett felmeddelande skickas. Om token är giltigt, söker programmet efter användaren i databasen. Ifall användaren hittas läggs användarinformation till i förfrågan, och flödet fortsätter till nästa steg i applikationen och användaren är autentiserad.

2.4 Spara, rendera och fleranvändare

2.4.1 Spara och rendera:

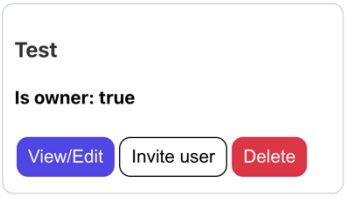
När användare skapar en design via hemskärmen så registrerar servern en ”design” i databasen. Den ”designen” är kopplad till skaparen och innehåller bland annat id, namn, tomma listor för samarbetspartners, inbjudningar och objekt.

Så fort användaren öppnar en design etableras en websocket förbindelse. Via förbindelse hämtar klienten sedan en lista på alla objekt som tidigare skapats. Varje objekt innehåller information om dess position, storlek, typ, färg, z-index (lager) och annan data som krävs för korrekt visning. Därefter renderas informationen på skärmen via biblioteket fabric canvas.

När användaren placerar eller ändrar objekt i designen och sedan avmarkerar eller väljer ett annat objekt, så konverteras de berörda objekten till json (textbaserat format som används för att utbyta data). Därefter anropas servern med uppgift att uppdatera designen i databasen. Detta sker med information om tillhörande design, användare och berörda objekt i json. Alltså uppdateras inte designen i databasen direkt vid ändring utan istället sker uppdatering när arbetet med ett objekt är slutfört för att motverka överbelastning. Ett flertal objekt kan dessutom markeras och flyttas samtidigt. Därför kan servern ta emot en lista på fler objekt att uppdatera samtidigt.

2.4.2 Samarbete och fleranvändare

För att samarbeta med andra användare i samma design krävs först att ägaren bjuder in samarbetspartners till designen. Det går till så att ägaren väljer knappen *invite user* på den designen på hemskärmen. Därefter så skriver ägaren in e-postadressen på den användaren som ska bjudas in. Severn anropas då med information om designen, ägaren och användaren. Servern skapar sedan en inbjudan i designens inbjudningslista i databasen.



Figur 5 : Bild av design på hemskärmen

När den inbjudne användaren öppnar hemskärmen markeras ringklockan (knappen för att öppna notisfältet) och texten *New invite* visas. Användaren får då välja att klicka på ringklockan och acceptera inbjudan. Ifall inbjudan accepteras så anropas servern igen och inbjudan i databasen tas bort och ett en ny *collaborator* läggs till i designens *collaborator lista* i databasen. Nu är användaren registrerad som samarbetspartner och designen visas på hemskärmen och går att öppna. Dock saknas knapparna *invite user* och *delete* eftersom endast ägaren har dessa befogenheter.

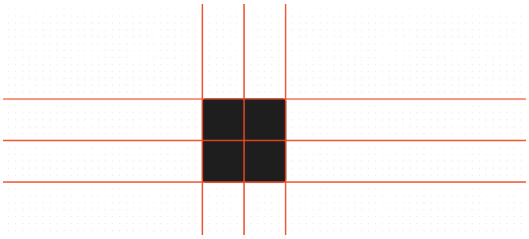
Sparsystemet för objekt fungerar likadant oavsett en eller fler användare arbetar i samma design samtidigt. Systemet som tidigare beskrivs sparar endast ändringar när objekt avmarkeras. Dessutom får andra användares klienter aldrig reda på att designen uppdaterats och att de behöver hämta nya ändringar från servern.

För att lösa detta problem finns ett helt annat system implementerat. Så ett eller fler objekt flyttas eller ändras skickas en websocket meddelande till servern med de aktiva. Med aktiva objekt menas de objekt som användaren har markerade. Det betyder att ändringar skickas mycket mer frekvent än endast när användaren arbetet klart med ett objekt och avmarkerat det.

Servern tar emot de aktiva objekten och direkt skickar ut eller reflekterar objekten till alla andra användare som arbetar i samma design. Detta sker utan att röra databasen eller andra tunga funktioner för att spara på tid och datakraft. Användarnas klienter tar emot och renderar ändringarna.

2.5 Snapping

Snapping är en mycket viktig funktion som hjälper användare att enkelt positionera sina objekt i designen genom att de snappar (alltså hoppar) till rätt position. Snapping funktioner finns implementerade i många olika typer av dataprogram och fungerar på olika sätt. Vanligt är att objekt snappar till olika förbestämda linjer, punkter eller mitten av arbetsytan.



Figur 6 : Illustration av snappinglinjer

I UI designerapplikationen snappar de aktiva objektet till närliggande objekt (markerade objektet). Varje gång muspekaren flyttas så kontrollerar programmet ifall användaren har ett aktiv objekt. Ifall detta är sant så sker en serie beräkningar. För alla objekt inom en förbestämd area kalkyleras sex virtuella linjer. Av dessa är tre linjer vertikala och resterande tre horisontella vilket visas i grafiken intill. Därefter så jämförs det aktiva objektets linjer med varje annat närliggande objekts linjer. Ifall någon av det aktiva objektets linjer är inom en förbestämd distans till ett annat objekts linje i plan korrigeras det aktiva objektens position så att linjernas koordinat är samma. Denna korrigering kallas för snapp då det aktiva objektet nu har ”hoppat” eller snappat till det närliggande objektet. Linjerna beräknas endast virtuellt och visas normalt inte för användaren. För att förbättra användarupplevelsen ritas dock de involverade linjerna ut när en snapp sker. Detta ger en visuell bekräftelse och visar användaren vart och varför objektet snappade.

3. Diskussion & slutsats

3.1 Idéer och problem

Projektet resulterade i en fungerande applikation för UI-design som stöder realtidssamarbete och erbjuder en intuitiv användarupplevelse. Applikationens alla delar, tekniker och ramverk samverkar för att möjliggöraeffektiv designprocess.

Under projektets gång har ett flertal problem och svårigheter uppstått. Ett av de mer framträdande problemen som uppdagades var en utmaning relaterad till användningen av Fabric.js för hantering av designobjekt. Ursprungligen saknade canvas biblioteket Fabric.js objekt en hantering av lager och id. Det är avgörande för att kunna spara designer och rendera dessa med samma utseende som de sparas.

För att lösa detta problem utökades Fabric.js-objekt med ytterligare egenskaper, såsom id och layerIndex enligt nedan.

```
interface CustomFabricObject extends fabric.Object {

  id: string;
  layerIndex: number;
}
```

Dock löste detta inte problemet helt eftersom funktionen för att konvertera Fabric.js-objekt till JSON format ”sakande kunskap” om de nya förändringarna. På grund av detta registrerades inte de nya variablerna i json filen. För att lösa detta problem anpassades funktionen toObject enligt följande:

```
fabric.Object.prototype.toObject = (function (toObject) {
  return function (this: CustomFabricObject) {
    return fabric.util.object.extend(toObject.call(this), {
      id: this.id,
      layerIndex: this.layerIndex,
    });
  };
})(fabric.Object.prototype.toObject);
```

En annan teknisk utmaning var hur applikationen hanterade situationer där två användare arbetade med samma objekt samtidigt. När två användare arbetar med samma objekt uppstår konflikter pga dubbelkommando. Lösningen på detta problem blev att implementera en mekanism som blockerar andra användare från att interagera med ett redan aktivt objekt. Detta säkerställer att designprocessen förblir konsistent och fri från oväntade överlappningar eller dataförlust.

Trots framgångarna med projektet, fanns det idéer som behövde överges, huvudsakligen på grund av tidsbegränsningar. Bland dessa var utvecklingen av ett system för skärmmallar och inramningar av designerna för att simulera hur de skulle se ut på olika enheter, såsom telefoner eller surfplattor. Dessutom förkastades idén om e-postutskick för lösenordsåterställning och inbjudningar på grund av potentiella kostnader, tidsåtaganden och risken att e-posten skulle markeras som skräppost.

För framtiden öppnar projektet upp för en rad förbättringsmöjligheter. Att integrera funktioner såsom zoom och stöd för designmallar skulle ytterligare förbättra applikationens användbarhet och flexibilitet. Även om projektet inte kommer att fortsätta i sin nuvarande form, lägger det en solid grund för framtida utveckling ifall projektet skulle återupptas.

3.2 Slutsats

Detta projekt har inte bara tjänat som ett sätt att utöka kunskaper inom fullstackutveckling utan också belyst viktiga överväganden och utmaningar som kommer med utvecklingen av samarbetsverktyg för design.

Ett fullt fungerande gränssnittsdesignverktyg med stöd för realtidssamarbeten med andra användare har under projektets gång skapats. Det innebär att målet är uppnått. Projektet har även lett fram till svar på de frågor som belystes i inledningen. Realtidssamarbetet kan implementerades via websocket. Det sker genom att två system samverkar sömlöst i en websocketförbindelse med servern för att spara ändringar respektive uppdatera samarbetspartners gränssnitt. Denna arkitektur, där systemen är åtskilda, är viktig för att uppnå en högkvalitativ användarupplevelse genom ständiga och oavbrutna uppdateringar. För att optimera databasens effektivitet och minska dess belastning, sparas ändringar med en fördröjning. Detta beskrivs i avsnittet *spara, rendera & fleranvändare 2.4.*

Databasen användning och dataöverföringens effektivitet maximeras genom att endast spara ändringar när arbetet med ett objekt är slutfört. När användaren placerar eller ändrar objekt i designen och sedan avmarkerar eller väljer ett annat objekt uppdateras databasen. Alltså uppdateras inte designen i databasen direkt vid ändring utan i stället sker uppdatering när arbetet med ett objekt är slutfört för att motverka överbelastning. Liksom websocketförbindelsen beskrivs även det här i avsnittet *spara, rendera & fleranvändare 2.4*.

Genom att navigera genom tekniska utmaningar och hitta innovativa lösningar har en fullständig applikation kunnat skapas som inte bara underlättar designprocessen för designers men också främjar samarbete och kreativitet bland användarna. Framtida arbete kan utvidga applikationens funktionalitet och användbarhet, men den nuvarande versionen utgör en solid grund för vidare utveckling och innovation.

Källförteckning

Ferguson, W. C. (2021, mars). HTTP (Hypertext Transfer Protocol) . Hämtad från TechTarget: https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol

Socket.io. (2024, 1 februari). How it works. Hämtad från socket.io: https://socket.io/docs/v4/how-it-works/

Grigutytė, M. (2023, 16 juni). What is Bcrypt and how does it work? Hämtad från Nordvpn: https://nordvpn.com/sv/blog/what-is-bcrypt/

Jwt.io. (n.d.). Introduction to JSON Web Tokens. Hämtad från Jwt.io: https://jwt.io/introduction