CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Crossportal: Design och implementation av en webbaserad korsordsapplikation

Kandidatarbete inom Data- och Informationsteknik

Ludvig Hammarstedt
Fredrik Nyhlén
Maida Sahiti
Albert Sjöman
Oscar Nurmi Torslund

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2026

Kandidatarbete 2026

Crossportal

Design och implementation av en webbaserad korsordsapplikation

Ludvig Hammarstedt
Fredrik Nyhlén
Maida Sahiti
Albert Sjöman
Oscar Nurmi Torslund

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs universitet
Göteborg, Sverige 2026

Crossportal
Design och implementation av en webbaserad korsordsapplikation

Ludvig Hammarstedt
Fredrik Nyhlén
Maida Sahiti
Albert Sjöman
Oscar Nurmi Torslund

© Ludvig Hammarstedt, Fredrik Nyhlén, Maida Sahiti, Albert Sjöman, Oscar Nurmi Torslund 2026.

Handledare: Peter Ljunglöf, Institutionen för Data- och informationsteknik
Medrättande lärare: Aarne Ranta, Institutionen för Data- och Informationsteknik
Examinator: Patrik Jansson, Institutionen för Data- och Informationsteknik

Kandidatarbete 2026
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola och Göteborgs universitet
412 96 Göteborg
Telefon: 031-772 1000

Typeset in LaTeX
Göteborg, Sverige 2026

Crossportal
Design och implementation av en webbaserad korsordsapplikation
Ludvig Hammarstedt, Fredrik Nyhlén, Maida Sahiti, Albert Sjöman, Oscar Nurmi Torslund

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola och Göteborgs universitet

Abstract

Digitalization has created new opportunities for the development of interactive and user adapted web applications. Within the field of crossword creation, there is a limited selection of available tools, particularly for Swedish crosswords. This project report describes the development of a crossword web application, called Crossportal, which allows users to create, solve, and share crosswords. The target audience are users who want to create their own crosswords as well as solve those created by others. The project was also limited by excluding mobile adaptation of the web application. Crossportal supports both Swedish and English crosswords, but does not support hybrid crosswords that combine characteristics from both types.

The result is a functional web application that includes several core features such as crossword creation and solving, image import, and management of user information through registration, login, and profiles. The project was developed using React for the user interface, TypeScript for type safety and Tailwind CSS for flexible design management. Firebase was used for authentication and data storage. Furthermore, the application provides opportunities for future development and additional functionality.

The group worked according to an agile approach inspired by Scrum. This methodology was chosen to create a flexible and iterative development process. By categorizing features into must-have, should-have, and could-have requirements, a clear specification was established. Trello was used to visualize and structure the workflow. Tasks were divided and organized into a product backlog according to their priority.

Sammandrag

Digitalisering har skapat nya möjligheter för utveckling av interaktiva och användaranpassade webbapplikationer. Inom området för korsordsskapandet finns det begränsat utbud av verktyg, särskilt för svenska korsord. Denna rapport beskriver utvecklingen av en webbaserad korsordsapplikation, kallad Crossportal, som gör det möjligt för användare att skapa, lösa och dela korsord. Målgruppen är användare som vill skapa egna och lösa andra användares korsord. Projektet har även avgränsats genom att mobilanpassning inte implementerats för webbapplikationen. Crossportal stödjer både svenska och engelska korsord, men inte hybridkorsord som kombinerar egenskaper från båda typer.

Resultatet är en funktionell webbapplikation som innehåller flera centrala funktioner som skapandet och lösning av korsord, import av bilder, hantering av användaruppgifter vid registrering, inloggning och profiler. Projektet har utvecklats med React för gränssnittet, TypeScript för typsäkerhet och Tailwind CSS för bättre och flexibel hantering av design. Firebase användes för autentisering och datalagring. Det finns även möjligheter för att utveckla programmet vidare.

Gruppen har arbetat agilt inspirerad av Scrum. Detta arbetssätt val- des för att skapa en flexibel och iterativ utvecklingsprocess. Genom att kategorisera funktioner som måste ha, borde ha och kan ha skapades en tydlig kravspecifikation. Trello användes för att visualisera och strukturer arbetet. Arbetet delades upp i uppgifter och placerades i produkt backloggen efter prioriteringsordning.

Förord

Kandidatarbetet gjordes under vårterminen 2026 vid Institutionen för data- och informationsteknik på Chalmers Tekniska Högskola.

Gruppen vill tacka vår handledare Peter Ljunglöf, universitetslektor i data- och informationsteknik, för allt stöd, alla råd och vägledning under projektets gång.

Vi vill även rikta ett tack till Max Moreau för hans värdefulla bidrag till både projektplanen, Figma-designen och utvecklingen av applikationen under hans tid i projektet.

Ludvig Hammarstedt, Fredrik Nyhlén, Maida Sahiti, Albert Sjöman, Oscar Nurmi Torslund, Göteborg, juni 2026

Innehåll

1

Inledning

Tack vare digitaliseringen har nya möjligheter skapats för utvecklingen av interaktiva applikationer. Detta projekt fokuserar på att utveckla webbapplikationen Crossportal, som gör det möjligt för användare att enkelt skapa egna korsord och lösa andra användares korsord på samma plattform. Detta kapitel presenterar bakgrunden till ämnesvalet, projektets syfte samt de avgränsningar som har gjorts inom projektet.

1.1 Bakgrund

Intresset för korsord har ökat stadigt de senaste åren (Hermez & Tobiasson, 2025). Försäljningen av korsordstidningar har ökat, i en marknad där tidskriftstidningar sett breda nedgångar under flera år (Tidskrifter, 2021). Utvecklingen tyder på ett fortsatt växande intresse, vilket skapar förutsättningar att skapa en digital lösning som tillmötesgår korsordsintresserades behov. En webbaserad lösning för att skapa egna korsord kan tillgodose det nyligen ökade intresset. Webblösningen tillåter användare att skapa egna korsord efter sina önskemål, som sedan kan delas med andra korsordsintresserade för att lösas.

Att skapa ett korsord från grunden är en komplex uppgift. I synnerhet om korsordet måste uppnå särskilda villkor, såsom att ord ska passa vertikalt och horisontellt (Beacham m. fl., 2001). Utöver detta kan rutnätets storlek ändras och bilder kan läggas in i korsordet, vilket ytterligare komplicerar uppgiften. Rutnätets storlek har en stor inverkan på komplexiteten, då antalet kombinationer ökar i mycket snabb takt (Beacham m. fl., 2001). Generering av korsord blir således ett kombinatoriskt problem, vilket kräver algoritmer för att lösas (Beacham m. fl., 2001).

Det valda ämnet utforskar hur algoritmer kan implementeras för att skapa en funktionell och användarvänlig webblösning för att skapa egna korsord. Att generera ett korsord baserat på en ändlig mängd ord med varierande villkor kan formuleras som ett constraint satisfaction problem (CSP) (Beacham m. fl., 2001; Poole & Mackworth, 2023). Ett constraint satisfaction problem är ett beräkningsproblem där alla involverade variabler tilldelas värden inom ett system på ett sådant sätt att alla specificerade strukturella och logiska krav uppfylls (Poole & Mackworth, 2023, Chapter 4). Algoritmiska lösningar är centrala för att effektivt kunna navigera och lösa sådana problem (Beacham m. fl., 2001).

Ur akademisk synvinkel är det relevant att undersöka, utveckla och väga algoritmer utifrån kriterier som tidskomplexitet och funktionalitet. Val av algoritmer kommer styras av deras möjlighet att tillhandahålla nödvändiga funktioner, samtidigt som algoritmernas tidskomplexitet ligger på en acceptabel nivå. Vidare får inte de underliggande algoritmerna påverka användarupplevelsen för produkten, utan måste vara väl avvägda för att garantera både tidseffektiv och korrekt generering av korsord.

Dessutom är det akademiskt relevant att undersöka och utveckla ett gränssnitt som är både funktionellt och användarvänligt. Frågeställningen rör människa-datorinteraktion (HCI), vilket studerar samspel mellan system och användare. I den här lösningen är det av särskild relevans, då algoritmernas komplexitet måste vägas mot krav på användarvänlighet. Även en ovan internetanvändare ska kunna skapa sina egna korsord. Det kräver ett tydligt och genomtänkt gränssnitt, utan att det påverkar produktens faktiska användbarhet.

1.2 Syfte

Syftet med projektet är att utveckla en webbaserad korsordsapplikation, kallad Crossportal, som gör det möjligt för användare att skapa personliga korsord på både svenska och engelska, samt att tillhandahålla funktionalitet för delning av dessa korsord så att de kan lösas av andra användare.

Crossportal förväntas innehålla intuitiva verktyg som användare, oavsett teknisk bakgrund, ska kunna använda och förstå. Därutöver förväntas Crossportal kunna föreslå ord baserat på tema samt generera ledtrådar baserade på specifika ord.

1.3 Avgränsningar

Webblösningen förväntas inte stödja mobila enheter, då tidsåtgången för att implementera sådan funktionalitet ansågs vara för hög och resurserna bör i stället prioriteras till andra delar av projektet.

Crossportal ska stödja både svenska och engelska korsord, men det primära målet är att stödja svenska korsord, då det finns få befintliga verktyg för att skapa sådana. Möjligheten att bygga ett hybridkorsord som kombinerar egenskaper från både engelska och svenska korsord kommer inte tillhandahållas. Att implementera ett sådant korsord ökar komplexiteten avsevärt, då ytterligare begränsningar och krav måste beaktas. Algoritmer för att lösa sådana utökade constraint satisfaction problems blir oproportionerligt komplexa i förhållande till projektets omfattning.

Vidare kommer ordcensur av lexikon och ledtrådar inte att implementeras i applikationen av liknande skäl. I stället förutsätts att lämpliga lexikon och ledtrådar brukas vid användningen av webblösningen.

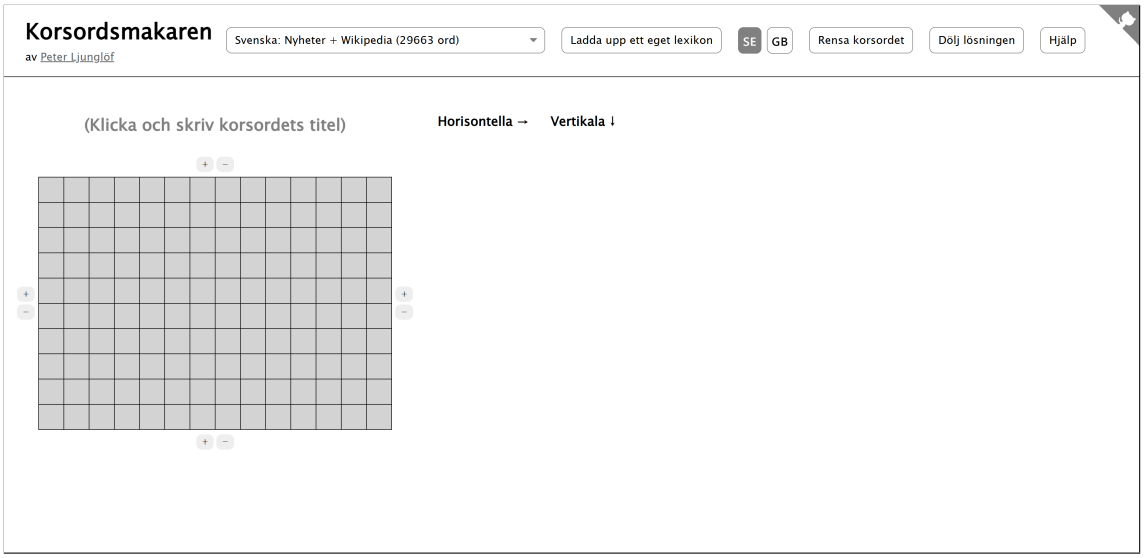
2

Problem och utmaningar

Utveckling av ett system för att skapa korsord innebär flera tekniska och användarrelaterade utmaningar. Det primära problemet i detta projekt handlar om hur man kan konstruera ett system som kombinerar avancerade algoritmer för korsordsgenerering med ett intuitivt och användarvänligt gränssnitt. Även om den grundläggande uppgiften kan beskrivas som att skapa korsord, visar en djupare analys att problemställningen omfattar betydligt fler dimensioner än själva genereringen.

2.1 Befintlig prototyp

Detta projekt utgår från en befintlig teknisk prototyp, se figur 2.1, som kan betraktas som ett fungerande konceptbevis, då den bygger på en tydligt strukturerad kod för konstruktion av korsord. Prototypen är dock implementerad i JavaScript, vilket försvårar vidareutveckling, refaktorering samt hanteringen av mer komplexa funktioner. I takt med att systemet växer och fler element, såsom databasstöd, användarkonton och avancerade algoritmer, integreras, ökar risken för svårupptäckta fel och oklara beroenden i kodbasen.



Figur 2.1: Den ursprungliga prototypen som ligger till grund för projektet.

2.2 Algoritmiska utmaningar

Dessutom saknar prototypen flera viktiga funktioner som är nödvändiga för att skapa ett fullständigt och praktiskt tillämpbart system. Det finns ingen databaslösning för att spara korsord och användardata, ledtrådar hanteras separat i en lista snarare än genom att integreras i rutnätet, och systemet tillhandahåller begränsat stöd för att slutföra fullständiga korsord med flera samverkande ord och ledtrådar. Sammantaget visar detta att prototypen främst fungerar som en demonstration av teknisk möjlighet snarare än att vara ett skalbart och långsiktigt hållbart system.

För att lösa det övergripande problemet kan det vara fördelaktigt att dela upp uppgiften i flera mindre delproblem. Ett av de mest centrala delproblemen gäller algoritmer och fokuserar på hur systemet effektivt kan föreslå flera möjliga lösningar för att fylla i ett korsord. Eftersom många ord korsar varandra skapas olika beroenden inom rutnätet, vilket gör att problemet liknar klassiska constraint satisfaction problem (CSP). Det krävs algoritmer som på ett korrekt sätt kan hantera dessa beroenden samtidigt som de håller processtiden på en rimlig nivå.

2.3 Automatisk generering av ledtrådar

Ett annat algoritmiskt delproblem rör möjligheten att automatiskt skapa eller föreslå ledtrådar för specifika ord, utan att använda sig av en färdig lista med fördefinierade lösningar. Att skapa meningsfulla och korrekta ledtrådar utgör en i synnerhet svår utmaning, eftersom det innebär att man måste hantera språkliga uppgifter, semantiska samband samt relationer mellan olika ord. Uppgiften innebär betydande språktekniska utmaningar, som kan kräva förenkling eller avgränsning för att uppnås inom den angivna tidslinjen för projektet.

2.4 Temahantering

Utöver detta förekommer delproblem relaterade till temahantering, där systemet ska kunna identifiera ord som tillhör samma tematiska kategori och därefter rekommendera ytterligare ord inom samma tema. Detta ställer krav på hur ord relateras till varandra, hur tematiska samband modelleras samt hur dessa förmedlas till användaren på ett begripligt och korrekt vis.

2.5 Användargränssnitt och interaktionsdesign

Ett av de viktigare delproblemen rör användargränssnitt och interaktionsdesign. Systemet ska tillhandahålla avancerad funktionalitet, såsom automatiserade förslag, redigering av korsord samt redigering av ledtrådar. Detta ska presenteras på ett begripligt och lättanvänt vis för användare oavsett teknisk bakgrund. Den centrala utmaningen ligger i att dölja avancerade algoritmer med hög komplexitet bakom ett intuitivt och tydligt gränssnitt.

2.6 Användarhantering och datadelning

Uppdraget innefattar även hantering av användarkonton och användarspecifik information, såsom sparade korsord och inställningar. Detta har en nära koppling till processen för skapande, lagring och delning av korsord på ett smidigt sätt. Därigenom möjliggörs publicering av färdiga korsord online, vilket tillåter andra användare att ta del av och lösa dem.

Sammantaget kan projektet betraktas som en relevant akademisk frågeställning, då det innefattar flera lager av utmaningar. Bland dessa finns algoritmiska utmaningar kopplade till constraint satisfaction och optimering, strukturella utmaningar rörande hur korsord representeras och lagras, samt problem relaterade till användarinteraktion och gränssnittsdesign. Denna samverkan mellan olika dimensioner gör projektet till ett tydligt och välavgränsat exempel på integrationen av teori och praktik inom datavetenskap.

3

Verktyg

Detta avsnitt presenterar vilka verktyg som har använts vid utvecklandet av korsordsapplikationen.

3.1 Figma

Figma är ett molnbaserat designverktyg för att designa gränssnitt och skapa prototyper. Till skillnad från traditionella designprogram är Figma helt i webbläsaren, vilket möjliggör samarbete i realtid på samma projekt. Genom att dela en länk till Figma-projektet kan gränssnittet granskas direkt i webbläsaren, vilket underlättar kontinuerlig feedback utan krav på export av enskilda filer (Designlab, u.å.). Under projektets gång användes Figma både för att skapa låg- och högfidelitets prototyper, vilket möjliggjorde skapandet av interaktiva flöden där funktionalitet och användarvänlighet kunde testas innan programmeringsfasen inleddes. Detta minskade risken för logiska fel i gränssnittet och fungerade som en visuell specifikation under utvecklingen. Figma utgjorde därmed en central punkt för projektets visuella identitet och underlättade kommunikationen inom gruppen.

3.2 Firebase

Firebase är en ”Backend-as-a-Service” (Baas) skapad av Google som tillhandahåller färdiga molntjänster för app- och webbutveckling (Doug Stevenson, 2018). Projektet har främst använt tre delar:

- Cloud Firestore: En NoSQL-databas som lagrar data i dokument och samlingar. Den valdes för sin förmåga att hantera realtidssykronisering. När en användare ändrar en bokstav i ett korsord kan ändringen omedelbart distribueras till databasen och andra anslutna klienter utan att sidan behöver laddas om

- Firebase Authentication: Hanterar säker inloggning och användarhantering, vilket möjligör att användare kan spara sina egna korsord kopplade till ett unikt konto. Då projektets fokus inte låg på att utveckla ett eget system för inloggning och säkerhet, användes Firebase för att få tillgång till en färdig och säker infrastruktur. Detta gjorde det möjligt att implementera inloggning via

Google-konton utan omfattande egen utveckling. På så sätt kunde projektet dra nytta av en beprövad säkerhetslösning för hantering av användardata utan att behöva lägga resurser på att bygga systemet från grunden.

- Hosting: En tjänst för att snabbt och säkert publicera webbapplikationen globalt med automatiskt SSL-stöd (HTTPS). Genom att använda Firebase Hosting eliminerades behovet av att sätta upp och underhålla en egen server (så kallad self-hosting). Detta innebar att applikationen blev tillgänglig för andra personer via en offentlig webbadress direkt efter publicering, utan att projektgruppen behövde hantera tekniska aspekter som nätverkskonfiguration eller serverdrift.

3.3 Trello

Trello är ett online verktyg som används för att visualisera och strukturera ett teams arbete enligt Kanban metoden. Arbetet struktureras i digitala ”kort” som representerar enskilda uppgifter. Genom att dela upp utvecklingen i kolumner skapas en tydlig överblick av projektets status. Detta hjälper gruppen att identifiera flaskhalsar och säkerställa att inga uppgifter faller mellan stolarna. Arbetet kan fördelas mellan gruppmedlemmarna. Deadlines kan sättas och uppgifter kan prioriteras (Trello, u.å.).

Grundidén för Kanban metoden är att dela upp arbetet i uppgifter och lägga dem i produkt backloggen efter prioriteringsordning. Varje kort representeras som en uppgift i trello. Korten flyttas mellan olika kolumner som representerar olika steg i processen: att göra, pågår och klart (Microsoft, 2025). På det sättet kan alla i gruppen hålla koll på vad som behöver göras, vad som görs och vad som är klart.

3.4 TypeScript

TypeScript är ett programmeringsspråk som bygger vidare på JavaScript genom att lägga till stöd för statiska typer. Idag är JavaScript ett av världens mest använda språk, men en betydande nackdel är att dess flexibilitet ofta leder till att programmerare gör typfel som inte upptäcks förrän programmet körs. Genom att använda TypeScript säkerställs det att typerna i programmet är korrekta redan under utvecklingsfasen, då koden kontrolleras innan den kompileras.

Valet av TypeScript gjordes för att minska risken för buggar och för att möjliggöra utvecklingen av ett mer komplext program än vad som vore praktiskt med enbart JavaScript. I en korsordsapplikation, där stora mängder sammankopplad data (som koordinater, bokstäver och korsande ord) ständigt hanteras, underlättar TypeScript genom att ge tydliga felmeddelanden om data hanteras på fel sätt (Microsoft TypeScript-team, 2026). Detta gör koden mer strukturerad, lättare att underhålla och fungerar som en extra säkerhet när flera personer samarbetar i samma kodbas.

3.5 JSON

JSON (JavaScript Object Notation) är ett textbaserat och språkoberoende format som används för att lagra och utbyta data. Dess huvudfunktion i detta projekt är att fungera som bryggan mellan applikationens backend (Firebase) och frontend. Backend ansvarar för hantering av data och kommunicering med databasen, medan frontend ansvarar för användargränssnittet och användarinteraktioner. JSON valdes eftersom det är industristandard för API:er (applikationsprogrammeringsgränssnitt) och gör det möjligt att strukturera stora mängder information på ett sätt som är lättläst för både människor och maskiner.

I praktiken används JSON för att representera korsordets hela struktur som ett objekt. Detta inkluderar information om varje rutas position, dess innehåll och dess relation till andra rutor. Eftersom formatet är så lättviktigt kan dessa komplexa datamängder skickas snabbt över nätverket, vilket är avgörande för att applikationen ska kännas responsiv. Utan ett strukturerat format som JSON skulle hanteringen av korsordets alla beroenden och regler bli betydligt svårare att organisera och kommunicera mellan systemets olika delar.

3.6 React

React är ett JavaScript-bibliotek för att bygga användargränssnitt, med fokus på en modulär och komponentbaserad arkitektur. Istället för att se webbsidan som ett enda dokument, delas gränssnittet upp i mindre, återanvändbara komponenter. Detta är särskilt fördelaktigt för en korsordsapplikation där varje ruta eller ledtråd kan hanteras som en egen logisk enhet. Detta innebär att varje del fungerar som en självständig byggsten som sköter sitt eget utseende och sin egen funktion.

En av de främsta anledningarna till att React valdes framför liknande ramverk som Vue är dess omfattande ekosystem och flexibilitet. Medan Vue ofta använder en mallbaserad struktur, använder React JSX, vilket gör att JavaScript och HTML kan kombineras lätt. Detta ger utvecklare större kontroll och gör det enklare att integrera logik direkt i komponenterna. Eftersom projektet använder TypeScript, väger Reacts stöd för typsäkerhet tungt, då det underlättar felsökning och gör koden mer förutsägbar jämförbar med Vues mer flexibla struktur. Dessutom innebär Reacts marknadsledande position att det finns ett enormt bibliotek av färdiga lösningar och dokumentation att tillgå.

3.7 Tailwind CSS

Tailwind CSS är ett ramverk som effektiviserar stylingprocessen genom ett så kallat ”utility-first” tänk. Istället för att skriva långa, separata CSS-filer, stylas elementen direkt i koden med hjälp av fördefinierade klasser. Detta säkerställer en konsekvent och responsiv design som automatiskt anpassar sig efter olika skärmstorlekar.

Vi valde att använda Tailwind CSS istället för traditionell CSS eller ramverk som Bootstrap för att öka utvecklingstakten och hålla koden ren. Med Tailwind slipper man att lägga tid på att hitta på egna namn för CSS-klasser, vilket ofta är tidskrävande. Det get också större frihet att designa applikationer precis som man vill jämför med Bootstrap, som ofta har ett mer förutbestämt utseende.

3.8 Versionshantering

Versionshantering möjliggör att kunna spåra och hantera ändringar. Flera personer i en grupp kan samarbeta parallellt utan att påverka varandras arbeten vid utveckling av mjukvara. Man kan även återställa tidigare versioner. Git är ett versionshanteringssytem som används i plattformen GitHub.

3.8.1 Git

Git är ett versionshanteringssytem som är gjort för ej linjär utveckling av mjukvara. Den spårar förändringar i filer och den är väldigt användbar när flera ändrar samma fil på samma tid. Det betyder att kodgrenar kan skapas och flera kan jobba på sin egen gren utan att påverka huvudkoden. Sedan kan man sömlöst slå ihop kodgrenarna med huvudkoden (GitHub, Inc., 2026).

3.8.2 GitHub

GitHub är en plattform där du kan spara, dela, skriva kod tillsammans med andra (GitHub, Inc., 2026). Koden kan sparas i ett repository (ett arkiv där kod lagras och versionshanteras) på GitHub för att dela eller samarbeta med andra. Den har Git inbyggt, vilket gör att förändringar kan spåras och tidigare versioner återställas.

4

Arbetsmetod

I detta kapitel beskrivs de metoder för kommunikation och arbete som har använts under projektets gång.

4.1 Intern kommunikation

Intern kommunikation inom gruppen skedde via Discord. Flera kanaler i Discord-gruppen användes för att strukturera olika typer av ämnen. Kanalerna användes för kommunikation med handledaren, information om kommande möten, intern kommunikation inom gruppen samt för delning av länkar och dokument.

Discord är en plattform som används för kommunikation. Användare kan samtala med varandra via röst, text eller video. Som användare kan man ansluta sig eller skapa egna grupper (Buffy, 2025).

4.2 Agilt arbetssätt och Scrum

Projektet tillämpade en agil arbetsmetodik inspirerad av Scrum (Scrum, u.å). Detta ramverk valdes för att skapa en flexibel och iterativ utvecklingsprocess. För att hantera och prioritera projektets krav användes MoSCoW-metoden. Genom att kategorisera funktioner som måste ha, borde ha och kan ha skapades en tydlig kravspecifikation. Måluppfyllelsen för projektet definierades av att samtliga krav i kategorin måste ha fungerade korrekt och att mjukvaran kunde exekveras utan krascher eller andra kritiska fel.

Systemet konstruerades utifrån en modulariserad arkitektur, där projektet delades upp i tydliga och avgränsade moduler, såsom frontend, backend och databas. Denna struktur valdes för att minimera beroenden mellan olika delar av koden och möjliggöra parallellt arbete.

Utvecklingsarbetet strukturerades i form av tidsavgränsade iterationer, så kallade sprintar som vanligtvis sträckte sig över tre veckor. Varje sprint inleddes med ett planeringsmöte där gruppen definierade vilka uppgifter som skulle prioriteras. Genom att arbeta iterativt kunde gruppen kontinuerligt utvärdera och anpassa produkten efter nya insikter och tekniska utmaningar som uppstod under utvecklingens

gång. Vid slutet av varje sprint genomfördes en retrospektiv för att reflektera över gruppens arbetsprocess och identifiera potentiella förbättringsområden inför nästkommande iteration.

4.3 Roller inom gruppen

Roller i projektet fördelades inom gruppen och roterades var tredje vecka. Det var för att alla gruppmedlemmar ska få möjlighet att prova på alla roller förutom produktägarrollen. Handledaren Peter valdes till produktägare. I rollen som produktägare ansvarade han för att besvara frågor kring prioriteringar, design och produktkrav. Nedan följer en kort beskrivning av roller inom gruppen:

- **Ordföranden/projektledaren** ansvarade för att hålla i mötena, leda arbetet och säkerställa att projektet gick framåt.
- **Scrum-mästaren** ansvarade för att gruppen följde Scrum reglerna samt hjälpte gruppen vid behov. Ansvaret för produkt backloggen och att bedöma vad som kunde hinnas göras klart var också en del av uppgifterna.
- **Sekreteraren** dokumenterade gruppmöten, handledarmöten, möten med fackspråksavdelningen samt andra typer av anteckningar och beslut.
- **Utvecklarna** arbetade agilt enligt 4.2 och ansvarade för utvecklingen av projektet.

4.4 Genomförande

Genomförandet av applikationen Crossportal delades upp i tre faser: design, refaktorisering och systemarkitektur samt implementering av funktioner.

4.4.1 Fas 1: Design

Den visuella utformningen inleddes med framtagning av översiktliga pappersprototyper, följt av mer detaljerade skisser i designverktyget Figma. Designarbetet startade individuellt för att generera en variation av kreativa koncept, vilka därefter sammanfördes till ett gemensamt designsystem som gruppen enades om. Detta underlag utgjorde den visuella specifikationen för den slutgiltiga webbapplikationen.

4.4.2 Fas 2: Refaktorisering och systemarkitektur

Den befintliga prototypen, ursprungligen utvecklad i JavaScript, skulle genomgå en omfattande refaktoriseringsprocess för att migreras till TypeScript. Detta språkval implementerades för att introducera statisk typning, vilket ökade kodbasens robusthet avsevärt, samtidigt som det underlättade felhantering under utvecklingsprocessen.

För att strukturera applikationen integrerades biblioteket React, vilket möjliggjorde en modulär och komponentbaserad frontend-arkitektur. Det grafiska gränssnittet implementerades med hjälp av Tailwind CSS, ett ramverk som effektiviserade stylingprocessen och säkerställde en responsiv design. Som backend-lösning implementerades Firebase, vilket hanterade databaslagring samt autentisering och möjliggjorde uppdatering av systemet i realtid.

4.4.3 Fas 3: Implementering av funktioner

Implementationsfasen bedrevs både individuellt och i mindre grupper. För att visualisera arbetsflödet och kontinuerligt följa upp tidsplanen användes Trello som en digital Kanban-tavla. På denna fördelades ansvarsområden, och uppgifternas status övervakades och uppdaterades löpande.

Utvecklingsarbetet och versionshantering skedde via GitHub. För att undvika konflikter och bibehålla en stabil huvudversion utvecklades nya funktioner uteslutande i separata brancher. När en funktion var färdigutvecklad skapades en pull request. Innan denna slogs samman med huvudprojektet genomfördes en kodgranskning av minst en annan gruppmedlem. Denna process säkerställde god kodkvalitet och minimerade risken för integrationsproblem.

4.5 Användartester

För att verifiera hur väl projektets syfte hade uppnåtts, med särskilt fokus på användargränssnittet, tillämpades formativa användartester. Dessa tester baserades på ”tänka högt”– metoden, vilket syftade på att fånga användarnas omedelbara tankeprocesser och kognitiva belastning under den första interaktionen med systemet.

4.5.1 Genomförandet av testerna

Testerna genomfördes genom att deltagarna fick utföra olika valda uppgifter som exempelvis registrera ett konto, skapa och lösa ett korsord. Under testets gång uppmanades deltagarna att tänka högt och beskriva sina tankar, exempelvis beskriva vad de försökte göra, vad de förväntade sig skulle hända samt om de upplevde några svårigheter eller något de inte förstod.

4.5.2 Resultat av testerna

Resultaten från de formativa testerna gav ovärderliga insikter som ledde till flera viktiga designiterationer i projektet. En tydlig trend under testerna var att användarna generellt hade lätt att förstå de grundläggande interaktionerna, såsom att markera rutor och skriva in text. Det identifierades vissa kognitiva friktioner kopplade till systemets mer avancerade funktioner.

Ett konkret problem som framkom var relaterat till markeringen av riktning vid skapande av nya ord. I en tidig version av gränssnittet upplevde flera testdeltagare

frustration när de råkade markera ord baklänges, vilket resulterade i felaktig textinmatning. Baserat på denna feedback implementerades en automatisk standardisering av ordets riktning. Systemet uppdaterades med en logik som automatiskt byter plats på start och slutpunkt om användaren drar markören i fel riktning. Vilket omedelbart minskade antalet felinmatningar och ökade den upplevda användarvänligheten avsevärt.

Ett annat identifierat förbättringsområde rörde bildimporten. Testanvändarna uttryckte viss osäkerhet kring huruvida bilden faktiskt hade kopplats till rätt ord i rutnätet. För att åtgärda detta itererades designen i sidopanelen för att tydligare visualisera kopplingen mellan ett ord och en bild. Genom att visa en miniatyrbild direkt intill det specifika ordet i listan minskade den kognitiva belastningen, och testanvändarna uppgav att de kände sig betydligt mer trygga med gränssnittet.

5

Slutprodukt och implementeringar

I detta kapitel presenteras slutprodukten samt dess implementeringar som gjorts. Först beskrivs hur produktens olika delar fungerar och det följs upp med hur dessa funktioner har implementerats tekniskt.

5.1 Översikt av systemet

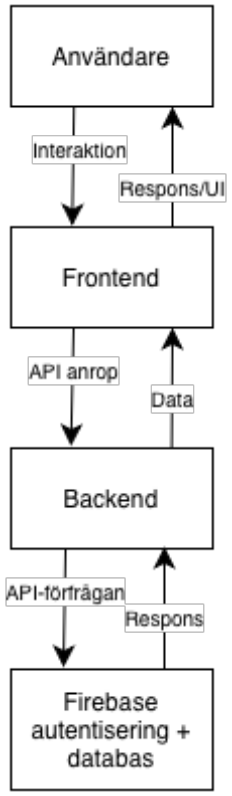
Systemet som har utvecklats är ett webbaserad korsordsapplikation där användare ska kunna skapa, lösa och dela korsord. Målgruppen är användare som både vill både skapa egna korsord och lösa andras korsord. Korsordsapplikationen består av flera centrala funktioner, förutom att skapa och lösa, kan användare importera bilder för att göra det mer visuellt anpassade, skapa ett eget konto, hantera sin profil och sina egna korsord.

I figur 5.1 visas hur systemets arkitektur ser ut. Användaren interagerar med frontend, som ansvarar för användargränssnittet och hanteringen av interaktioner. Frontend kommunicerar med backend genom API–anrop för att hämta data. Backend arbetar i bakgrunden och ansvarar för att hantera data samt kommunicera med databasen. Backend kommunicerar även vidare med Firebase genom API–förfrågningar för att hantera autentisering, användarprofiler och datalagring. Den slutliga versionen av koden finns publicerad som en pre-release på GitHub, version v0.1 (LudHamma, 2026).

5.2 Spela korsord - grundfunktionalitet

När en användare interagerar med systemet för att lösa ett korsord erbjuds en dynamisk och responsiv upplevelse, med ett tydligt fokus på användarvänlighet och smidig navigering. Upplevelsen bygger på en kombination av intuitiv rutnätsinteraktion, visuell tydlighet och realtidsfeedback.

Ur ett användarperspektiv sker den primära interaktionen direkt i korsordets rutnät. Användaren kan klicka på valfri bokstavsruta för att markera den och därefter omedelbart börja skriva in sitt svar. För att minska den kognitiva belastningen och underlätta förståelsen markeras det valda ordet visuellt i rutnätet. Denna visuella indikation hjälper användaren att snabbt identifiera exakt vilka celler som tillhör



Figur 5.1: Översikt av systemets struktur och dataflöde mellan användare, frontend, backend och Firebase.

det aktuella ordet som håller på att fyllas i. Systemet är utformat för att hantera traditionella horisontella och vertikala inmatningsriktningar, men erbjuder även ett specialanpassat stöd för svenska korsord. I dessa korsord stöds böjda ord, vilket innebär att ordet kan byta riktning mitt i inmatningen, och systemet hanterar två distinkta böjningsvarianter för att återskapa den klassiska svenska korsordskänslan.

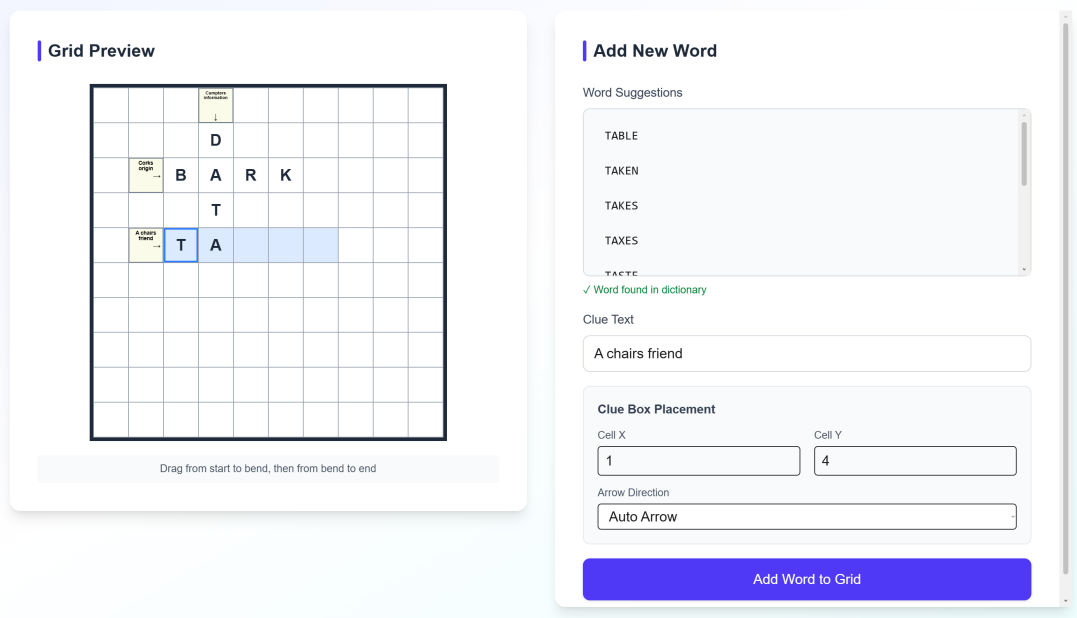
Navigeringen i rutnätet har utformats för att vara så flexibel och friktionsfri som möjligt. Användaren kan välja att förflytta sig mellan rutorna med hjälp av musen genom direkta klick, eller via tangentbordets piltangenter (upp, ner, vänster, höger). För att ytterligare effektivisera ifyllandet av korsordet tillämpar systemet automatisk navigation. När en användare skriver in en bokstav flyttas markören per automatik vidare till nästa lediga ruta i ordets riktning. Systemet är även förlåtande vid felinmatningar; om användaren trycker på backstegstangenten när markören befinner sig i en tom ruta, hoppar markeringen automatiskt tillbaka till den föregående rutan i sekvensen, vilket gör det enkelt att radera och skriva om ett ord.

Ur teknisk synvinkel kompletteras denna interaktionsmodell av ett kraftfullt system för validering och återkoppling i realtid. Under spelets gång validerar systemet kontinuerligt användarens inmatningar genom att jämföra dem med den korrekta bakomliggande lösningen. Resultatet av denna validering kommuniceras omedelbart till användaren via färgkodning: felaktigt inmatade bokstäver markeras tydligt med röd färg, medan de korrekta bokstäverna lyser grönt. Denna direkta feedback loop

gör att användaren omedelbart blir medveten om sina gissningars korrekthet, vilket skapar en mer interaktiv och engagerande lösningsprocess.

5.3 Skapa korsord

Funktionen för att skapa korsord är den centrala delen av systemet och är utformad för att ge användaren ett strukturerat arbetsflöde vid konstruktion av ett nytt korsord. Skapandet sker i en redigeringsvy som visas i figur 5.2 där användaren stegvis definierar korsordets grundläggande egenskaper, såsom titel, språk och rutnätsstorlek. Därefter kan användaren lägga till ord genom att markera en rad rutor i rutnätet. Till varje ord kopplas även en ledtråd, vilken senare används av den person som löser korsordet.

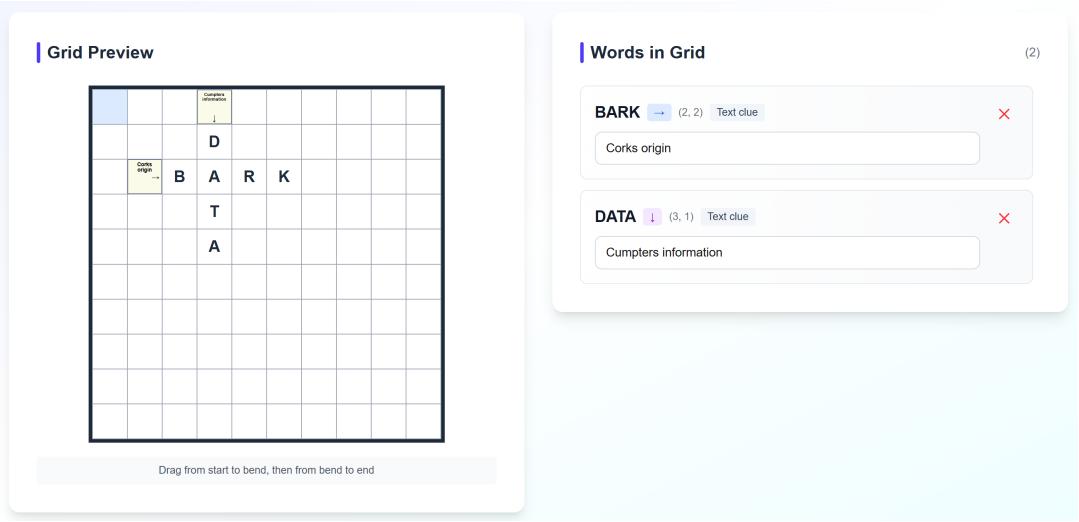


Figur 5.2: Redigeringsvyn där användaren kan lägga till ord genom att markera en rad rutor.

Ur ett användarperspektiv är redigeringsvyn uppbyggd för att stödja ett naturligt arbetsflöde där korsordet kan utvecklas successivt. Användaren kan först skapa strukturen i rutnätet och därefter fylla det med ord som passar in i den valda layouten. För att underlätta konstruktionen finns stöd för förhandsvisning av ordplacering, vilket gör det möjligt att kontrollera hur ett ord kommer att passa in innan det läggs till permanent. På så sätt minskas risken för felaktig placering och onödiga ändringar i ett senare skede.

En viktig del av funktionen är stödet för ordlistor. Genom att välja en ordlista kan användaren få hjälp att bedöma om ett ord är giltigt i förhållande till det valda språket. Detta bidrar till bättre kvalitet i det färdiga korsordet och gör det enklare att arbeta konsekvent. Ordlistorna används även som underlag för att ge förslag och kontrollera inmatade ord i realtid, vilket förbättrar redigeringsupplevelsen.

Tekniskt är funktionen uppdelad i en vy- och en logikdel, där redigeringsgränssnittet separeras från den tillhörande tillståndshanteringen. Detta gör implementationen mer överskådlig och underlättar vidare utveckling. Tillståndet för korsordet omfattar bland annat ord, ledtrådar, rutnätsdimensioner, vald ordlista och nuvarande markering i rutnätet. När användaren interagerar med gränssnittet uppdateras detta tillstånd kontinuerligt, vilket gör att redigeraren alltid återspeglar den aktuella versionen av korsordet.



Figur 5.3: Korsordsvy när användaren har lagt till ord i korsordsrutnätet

5.3.1 Ordlistor

Datan som lagras i ordlistorna utgör en viktig grund vid skapandet av korsord och används både vid valideringen av giltiga ord och vid förslag av möjliga ord. Varje ordlistas titel startar med vilket specifikt språk det innehåller följt av orden som är tagna från.

De befintliga ordlistorna är implementerade som JavaScript-filer och är hämtade från den befintliga prototypen. Den första raden i en ordlist-fil innehåller ordlistans titel, vilken används som identifierare i dropdown menyn där man väljer vilken ordbok som ska användas vid konstruktion av korsord. Under titeln listas varje ord på egen rad tillsammans en unik sträng av bokstäver och siffror. Denna unika textsträng används inte av systemet i nuläget, men ligger som grund för implementering av ord teman i framtiden.

Systemet ger även användaren möjlighet att importera egna ordlistor. Detta gör det möjligt att anpassa tillåtna ord efter specifika ämnesområden, språk eller personliga ordsamlingar. Systemet tillåter import av JavaScript- och textfiler, vilka förväntas följa samma struktur som de befintliga ordlistorna. Om filen saknar korrekt formatering för ordlistans titel används istället filnamnet som titel i systemets dropdownmeny. Importerade ordlistor markeras dessutom med en asterisk (*) symbol i slutet av titeln för att skilja dem från de förinstallerade ordlistorna.

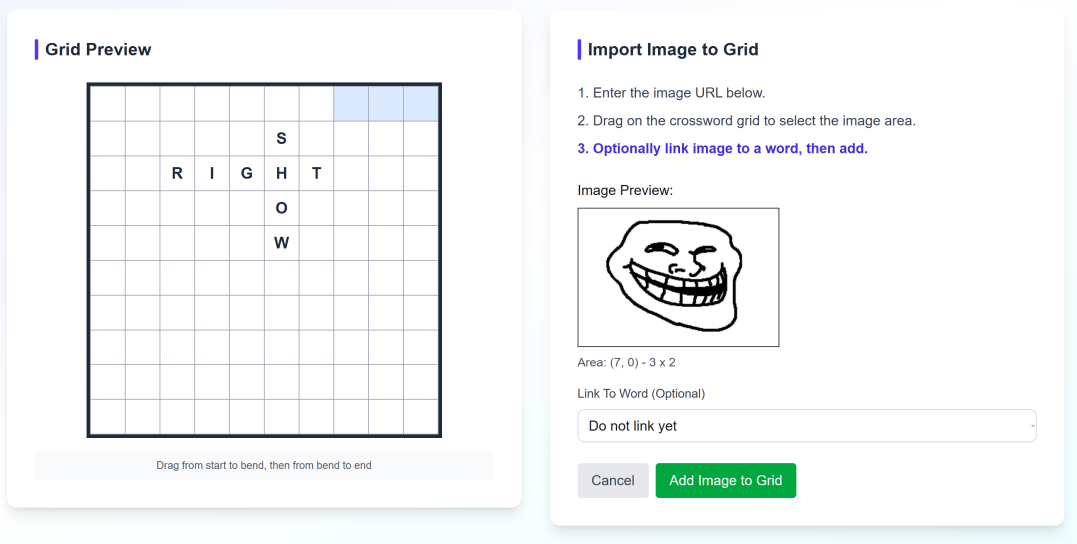
Orden i den importerade ordlistan hämtas från det första ordet på varje rad efter titelraden. Detta gör att importfunktionen kan hantera både filer som följer prototypens mer utökade struktur och enklare ordlistor.

De importerade ordlistorna lagras temporärt under användarsessionen och sparas inte permanent. Systemet stödjer endast en importerad ordlista åt gången, vilket innebär att en ny import ersätter den tidigare importerade ordlistan.

5.4 Import av bilder i korsord

För att utöka korsordets visuella uttryck finns en funktion för att importera bilder och placera dem i rutnätet. Funktionen är avsedd att stödja korsord där bildmaterial används som komplement till textbaserade ledtrådar eller som en del av korsordets visuella struktur. Bildimporten sker i samma redigeringsmiljö som ordskapandet, men hanteras i ett separat arbetsflöde för att de båda momenten inte ska påverka varandra.

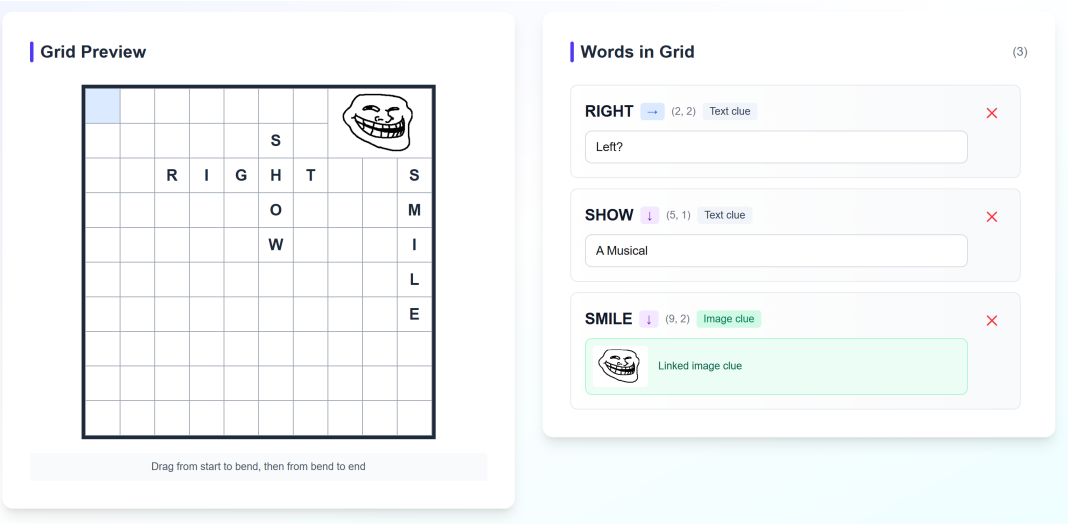
Användaren genomför bildimporten i tre steg. Först anges en bildadress, därefter markeras det område i korsordets rutnät där bilden ska placeras, och slutligen bekräftas importen. Innan bilden läggs till visas en förhandsgranskning tillsammans med den markerade ytan, vilket ger användaren möjlighet att kontrollera att placeringen blir rätt. I nästa steg kan bilden, om så önskas, kopplas till ett specifikt ord i korsordet, vilket visas i figur 5.4. Detta gör att bilden inte bara fungerar som ett dekorativt inslag utan också kan användas funktionellt i relation till korsordets innehåll.



Figur 5.4: Gränssnitt för bildimport med förhandsgranskning samt verktyg för att länka bilden till ett specifikt ord

Ur en teknisk synvinkel lagras varje importerad bild som en separat del av korsordets data, där både bildens adress och dess position samt storlek i rutnätet sparas. Detta innebär att bilden kan återges på exakt samma plats när korsordet senare visas

eller ska lösas av en slutanvändare (se figur 5.5). Om användaren behöver göra ändringar kan bilden också tas bort från korsordet utan att resten av innehållet påverkas. Lösningen är därför konstruerad för att vara flexibel och samtidigt hålla datamodellen tydlig.



Figur 5.5: Korsordsvyn med den importerade bilden på plats i rutnätet. I sidolistan till höger syns även att ordet är kopplat till bilden

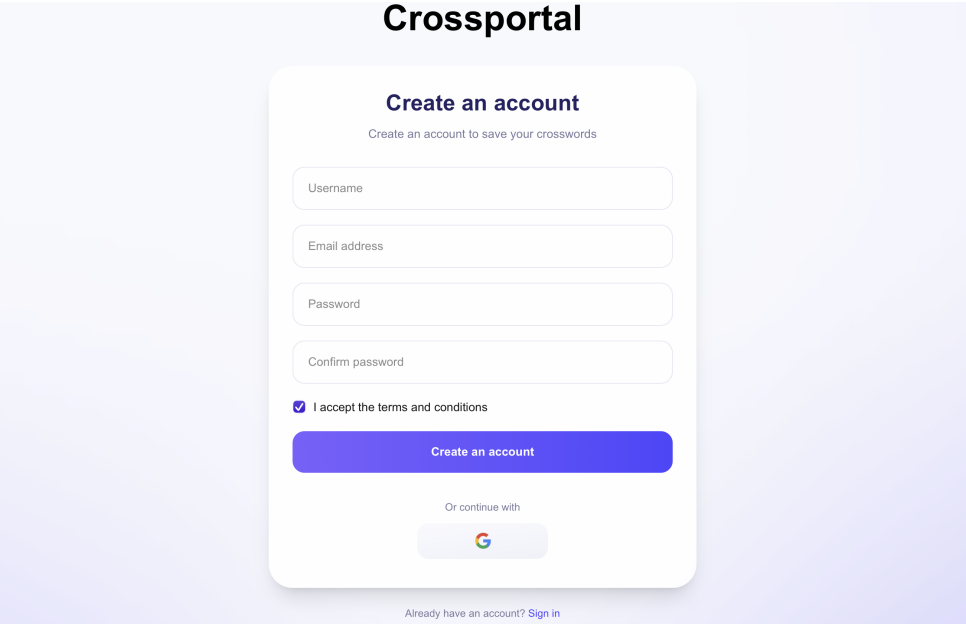
Sammantaget bidrar bildimporten till att korsordet kan anpassas efter olika typer av innehåll och användningsområden. Funktionen stärker därmed systemets uttrycksmässiga möjligheter, samtidigt som den är integrerad i samma strukturerade redigeringsflöde som övriga delar av korsordsskapandet.

5.5 Registrering och inloggning

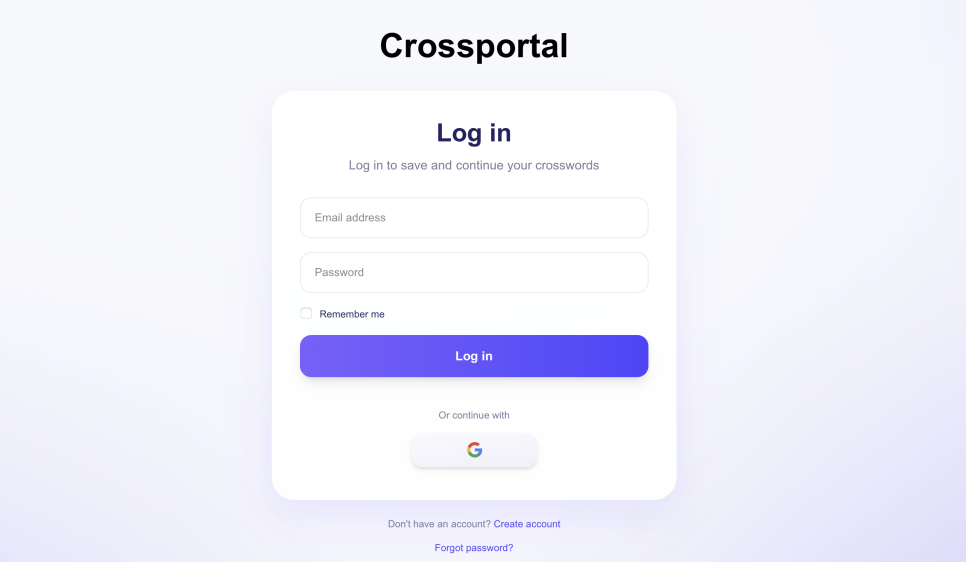
Användaren kan skapa ett konto genom att fylla i användarnamn, mejladress och lösenord, vilket visas i figur 5.6. Om användaren redan har ett konto kan den logga in med sin mejladress och lösenord som visas i figur 5.7. Det finns även möjlighet att registrera sig och logga in med sin Google mejladress. Dessutom kan lösenordet återställas om användaren har glömt det.

För registrering och inloggning används Firebase Authentication som hanterar autentisering. När en användare registrerar sig så skapas ett konto med deras uppgifter genom att frontend skickar användarens uppgifter till Firebase Authentication via ett API-anrop. Firebase skapar ett unikt användar–ID (UID) som används för att identifiera användaren i systemet. När användaren har skapats lagras användarnamnet och profiluppgifterna i databasen som är kopplad till Firebase.

Vid inloggning skickas användarens mejladress och lösenord från frontend till Firebase Authentication för verifiering. Uppgifterna som användaren fyllt i kontrolleras att de stämmer överens med dem som finns lagrade i databasen. Om de är korrekta och verifieringen lyckas, genereras en autentiseringstoken som skickas tillbaka



Figur 5.6: Registreringssidan där användaren kan skapa ett konto med sin mejladress eller Google–mejl.

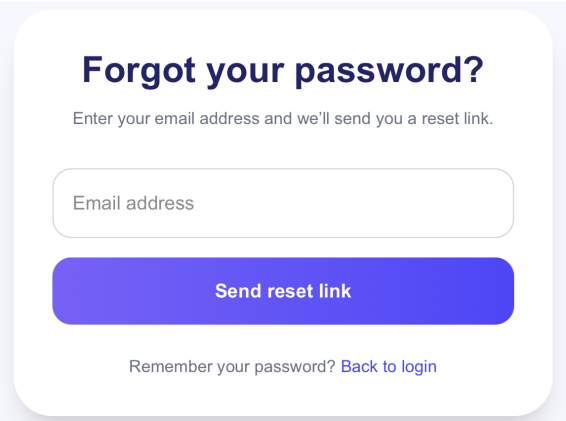


Figur 5.7: Inloggningssidan där användaren kan logga in med sin mejladress och lösenord eller via Google.

till frontend. Denna token används för att verifiera användarens ID och logga in användaren.

Inloggning med Google är implementerad genom Firebase, vilket gör att användare kan logga in utan att registrera sig manuellt. Om autentiseringen med Google–mejl lyckas första gången användaren loggar in, hämtar Firebase användarens grundläggande profilinformation och skapar automatiskt ett konto.

När användaren klickar på ”Forgot password?” i figur 5.7, skickas användaren till återställningsvyn, vilket visas i figur 5.8. Funktionen för glömt lösenord använder Firebase återställningstjänst för att återställa lösenordet genom att användaren först anger sin mejladress. Därefter skickas en säker återställningslänk till användarens mejladress för att skapa ett nytt lösenord.



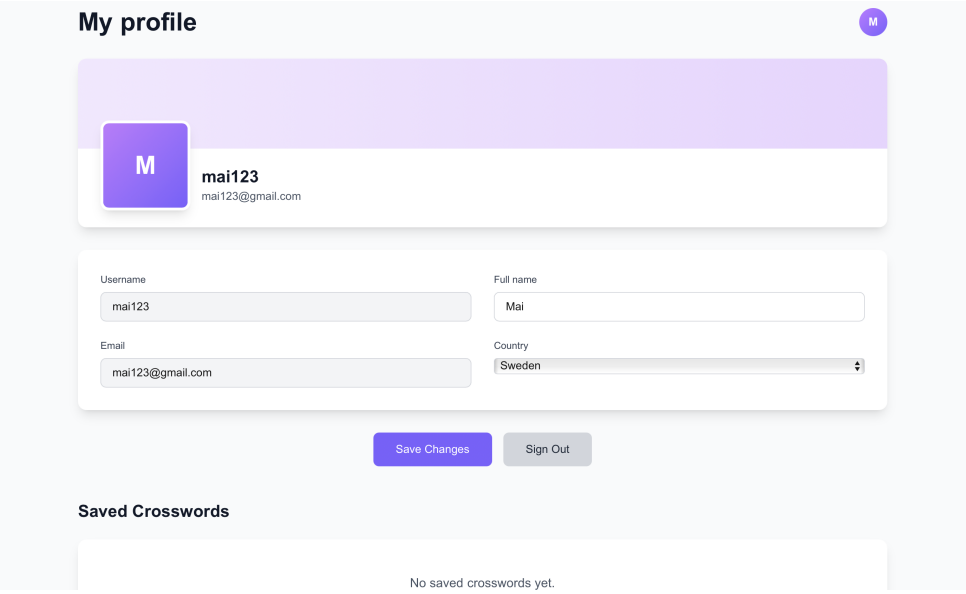
Figur 5.8: Återställningsvyn där användaren anger sin mejladress för att få en återställningslänk och skapa ett nytt lösenord.

5.6 Profiler

Profilsystemet fungerar som applikationens centrala nav för användarhantering, identitet och datalagring. Härifrån får användaren en samlad och överskådlig vy över sitt konto, sina sparade korsord samt sina sociala interaktioner inom systemet.

Gränssnittet för användarhantering erbjuder flera viktiga funktioner. Från profilsidan kan användaren se central information såsom sin registrerade e-postadress samt datumet då kontot ursprungligen skapades. Användaren ges också möjlighet att personifiera sitt konto genom att byta användarnamn; detta görs enkelt genom att klicka direkt på användarnamnsrutan som visas i figur 5.9. När redigeringen är klar sparas de nya uppgifterna automatiskt i systemet utan att användaren behöver navigera till separata inställningssidor. Vidare finns tydliga funktioner för att logga ut, vilket säkert returnerar användaren till startsidan, samt ett alternativ för att permanent radera sitt konto efter att en säkerhetsbekräftelse har genomförts.

Utöver den personliga kontohanteringen är profilsystemet djupt integrerat med applikationens vänskapssystem som visas i 5.10, vilket möjliggör datadelning och social



Figur 5.9: Profilvy för en användare.

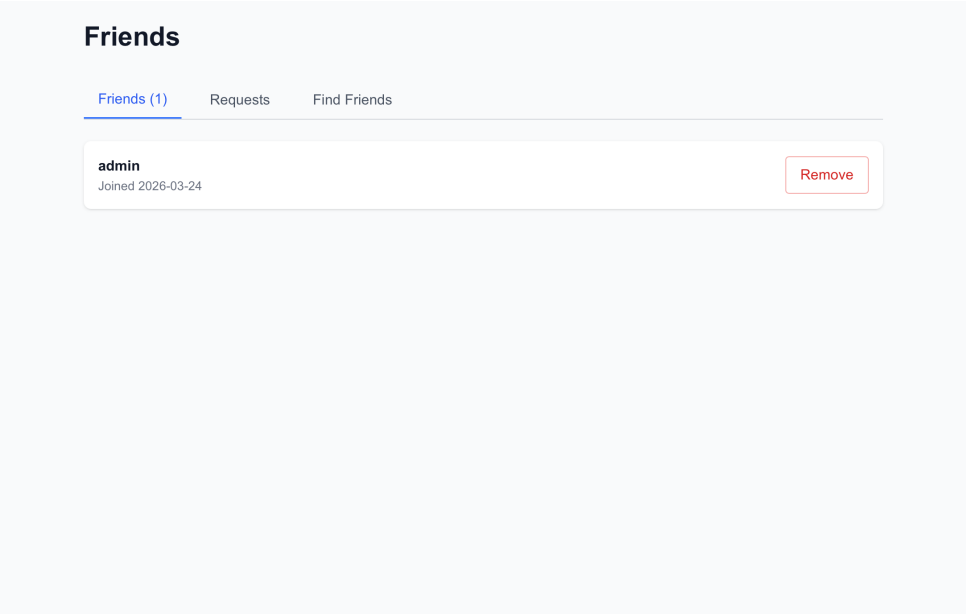
interaktion. Användare kan söka upp och skicka vänskapsförfrågningar till andra, samt administrera inkommande förfrågningar genom att antingen acceptera eller avvisa dem. Denna vänskapsintegration är avgörande för systemets delningsfunktionalitet, eftersom den tillåter användare att publicera sina egendesignade korsord och göra dem tillgängliga för vänner att se och lösa.

Ur ett integrations- och arkitekturperspektiv vilar profilsystemet på en robust och modern molninfrastruktur. För inloggning och säker identifiering av användare utnyttjas Firebase Authentication. All data som rör profilen, liksom användarens bibliotek av sparade korsord, lagras i Firebase Realtime Database. Datan är logiskt strukturerad och separerad under sökvägarna users/uid/profile/ och users/uid/savedCrosswords/ för att optimera hämtning och säkerhet. Säkerheten i databasen garanteras genom strikta Firebase-regler som säkerställer att varje specifik användare enbart kan läsa och modifiera sin egen data.

För att underlätta interaktionen mellan frontend-applikationen och databasen har logiken abstraherats i React. Detta har åstadkommits genom utvecklingen av en dedikerad React-hook, useProfileController. Denna hook kapslar in all komplex databasinteraktion och förser applikationens övriga komponenter med ett enhetligt och lätthanterligt gränssnitt för att läsa och uppdatera profildata. Kombinationen av dessa tekniker skapar en säker, responsiv och underhållbar profilupplevelse.

5.7 Ledtrådar i rutor

För att stödja skapandet av traditionella svenska korsord har ett system för att placera ledtrådar direkt i rutnätet implementerats. Till skillnad från klassiska engelska korsord, där ledtrådarna presenteras i separata listor vid sidan av rutnätet, integreras ledtrådarna här i dedikerade rutor inuti själva spelplanen (se figur 5.3).



Figur 5.10: Vänskapsvyn där användaren kan hantera sina vänner, söka efter andra användare och hantera vänförfrågningar.

Funktionaliteten tillåter användaren att koppla textledtrådar till specifika ord. Dessa ledtrådar placeras automatiskt i anslutning till ordets startpunkt. En enskild ledtrådsruta kan innehålla en eller flera ledtrådar, beroende på hur många ord som utgår från rutan. Om en ruta innehåller flera ledtrådar separeras dessa visuellt med hjälp av en avdelande linje för att tydliggöra gränssnittet för användaren.

En central del av implementationen är den dynamiska hanteringen av utrymme och riktningspilar. För att säkerställa att ledtrådarna förblir läsbara oavsett längd, tillämpar systemet en algoritm för dynamisk textskalning. Textstorleken beräknas utifrån teckenantalet samt antalet ledtrådar som delar på rutans utrymme. Långa texter skalas ner och rad bryts automatiskt.

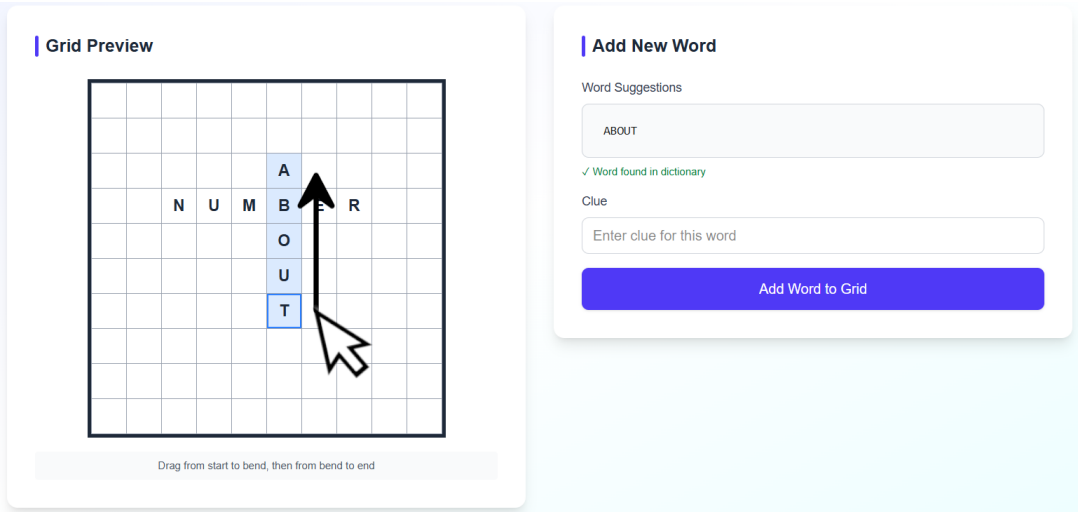
Vidare genererar systemet riktningspilar som höger, nedåt eller böjda pilar för ord som byter riktning. Riktningspilarna hjälper till att visuellt guida användaren till vilket ord ledtråden tillhör. Vid fallet av ord som byter riktning så placeras en böjd pil inuti bokstavsrutan där själva ordet byter riktning. Det finns även andra varianter av svenska korsord där pilar inte används, så användaren kan även välja ledtrådar utan pilar. Detta möjliggör ett system som kan tillfredsställa flera olika varianter av korsord beroende på vad användaren vill skapa.

Ur ett tekniskt och strukturellt perspektiv hanteras detta genom att rutnätets datamodell skiljer på olika celltyper, i huvudsak tom, bokstav och ledtråd. Under renderingen av gränssnittet, både i redigerings vyn och spelläget, utvärderas varje cell. Om cellen identifieras som en ledtrådsruta extraheras dess data, varpå React komponenterna dynamiskt ritar ut rätt textstorlek, placering och tillhörande pilar. Detta skapar ett responsivt och flexibelt system som kan hantera den visuella komplexiteten som krävs för svenska korsord.

5.8 Konsekvent ordriktning

I mål att förbättra läsbarheten och göra korsorden enklare att tolka så finns en funktion som standardiserar ordets riktning. Målet med denna process är att säkerställa att alla ord läses från vänster till höger eller uppifrån och ned, oavsett hur användaren initialt markerar dem. Detta skapar en mer konsekvent struktur för korsordet vilket underlättar användarens förmåga att tolka orden.

Funktionen anropas efter att användaren har markerat start- och slutpunkten för ett nytt ord, men innan systemet föreslår möjliga ord. När funktionen anropas så jämförs ordets start- och slutpunkt, om ordets startpunkt är under eller till höger om slutpunkten så byts positionerna på ordets start- och slutpunkt med varandra. Detta tillåter användaren att markera ord från höger till vänster eller nedifrån och upp utan att orden blir fel riktade som visas i figur 5.11, vilket leder till ett mer användarvänligt system.



Figur 5.11: Ord som läses uppifrån och ned även om de markerades nedifrån och upp.

Figur 5.12: Tillåtna former av böjda ord.

För att möjliggöra att böjda ord ska kunna fylla dessa krav begränsas de möjliga

böjningarna till de varianterna som visas i figur 5.12, då de resterande inte kan innehålla kontinuerliga ord om samma formatering appliceras på dem. Dessa böjningar, likt raka ord, har också möjligheten att markeras höger till vänster och nedifrån och upp utan att ordets läsriktning invalideras.

6

Diskussion och slutsats

I detta avsnitt diskuteras resultatet, val av lösningar, fördelar och nackdelar med arbetet, eventuella förbättringar som kan göras, integritet och dataskydd, och slutsatser.

6.1 Val av lösningar

Vid val av programmeringspråk, valdes TypeScript i frontend istället för JavaScript, som den befintliga prototypen var implementerad i. Detta val var viktigt för att projektets komplexitet inte skulle begränsas.

Även om ett projekt som syftar till att utveckla en webbapplikation för generering av korsord vid första anblick kan uppfattas som relativt okomplicerat, aktualiserar det flera samhälleliga och etiska aspekter som beaktades. En av de mest centrala frågorna rör partiskhet i de ordlistor som applikationen baseras på. Partiskhet kan ta sig uttryck i både exkludering och inkludering av specifika ord, vilket i förlängningen kan bidra till att förstärka eller motverka rådande normer, stereotyper och fördomar. Denna problematik belyses bland annat i Hjertström och Zelander (2011), där de undersöker hur fult språk kan spridas samt hur sociala normer både påverkar och påverkas av sådan spridning. Därför finns det bara möjlighet att använda ordlistorna som finns för att motverka stereotyper och fördomar.

I utvecklingen av designen för applikationen har samhälleliga och etiska aspekter tagits hänsyn till för att öka tillgängligheten. För att säkerställa att applikationen kan användas av en bred användargrupp har den utformats i enlighet med principer för tillgänglig design. Detta innefattar val av färgscheman, kontrastnivåer, textstorlekar och enkel gränssnitt. Dessa designval är särskilt relevanta för användare med olika funktionsnedsättningar, såsom synnedsättningar.

6.2 Fördelar

En samhällelig aspekt som valts att belysa är hur korsord kan användas för att stärka inlärandet av språk. Speglat i Jonsson (2003), Tengelin (2012) och Gustavsson (2014) examensarbeten kan vi se hur användandet av korsord stärker lärandet hos elever i skolan. Nationalencyklopedin (u.å.), förkortat NE, drar samma slutsats men trycker

på att det är viktigt för personer i alla åldrar. Korsord har en bevisad positiv effekt för hjärnan och som NE skriver så ”håller det hjärnan alert”. Detta understryker värdet i projektet ytterligare.

6.3 Nackdelar och begränsningar

En svaghet är att applikationen saknar särskild implementation för mobila enheter, därför begränsas tillgängligheten till användare som använder större skärmar. För att förbättra tillgängligheten skulle en mobilanpassad version kunna utvecklas i framtiden.

En begränsning är att systemet är beroende av Firebase för datalagring och autentisering. Detta minskar flexibiliteten då man måste använda en extern tjänst och påverkar möjligheten att ändra systemet i framtiden.

En annan begränsning är att temahanteringen hann inte implementeras i systemet inom projektets tidsram. Idén var att systemet skulle kunna identifiera ord som tillhör samma tematiska kategori och därefter rekommendera ytterligare ord inom samma tema. Detta hade kunnat lösas genom att utveckla algoritmer eller användas AI för att analyserna samband mellan ord. Funktionen kan implementeras vid vidareutveckling av systemet.

6.4 Framtida förbättringar

För att gå från den här första versionen till en app som verkligen kan konkurrera på marknaden behöver vi titta på hur vi kan göra den smartare. Idag är verktyget ganska manuellt, men vi vill att det ska fungera mer som en medskapare som hjälper användaren när det tar stopp. Samtidigt vill vi bygga ut de sociala delarna, så att man enkelt kan dela och lösa korsord tillsammans med både vänner och främlingar.

6.4.1 Mobilanpassning och ökad tillgänglighet

Något som verkligen skulle lyfta användarupplevelsen är att anpassa applikationen för mobiler och surfplattor. Som det ser ut just nu är verktyget helt utformat för stora skärmar, vilket gör det smidigt att jobba på en dator men nästan omöjligt på en telefon. Eftersom de flesta idag använder mobilen som sin huvudsakliga enhet för både nöje och enklare skapande, missar vi en stor målgrupp genom att vara låsta till skrivbordet.

Att göra appen mobilvänlig handlar om mer än att bara förminska allt så det får plats. Det krävs en helt ny genomgång av hur designen reagerar på olika skärmstorlekar, så kallad responsiv design. Den stora utmaningen här är hur man får plats med ett helt korsordsrutnät på en liten skärm utan att det blir plottrigt och svårt att klicka. Man skulle behöva titta på lösningar som smart inzoomning, där skärmen automatiskt fokuserar på den ruta man skriver i, och se till att knappar och menyer är anpassade för fingrar istället för en liten muspekare.

Det handlar också om att lösa hur det digitala tangentbordet fungerar ihop med rutnätet så att det inte täcker för det man försöker se. Om vi lyckas göra appen tillgänglig på mobilen förvandlas den från att vara ett ”program” man måste sitta ner vid en dator för att använda, till en rolig app man kan plocka fram när som helst. Det skulle göra att betydligt fler personer faktiskt börjar använda verktyget i vardagen.

6.4.2 Manuell konstruktion till automatiserad Constraint Solving

En stor förbättring som vi faktiskt siktade på redan från början är automatiserat skapande av korsord. Som det ser ut just nu vilar hela ansvaret för strukturen på användaren själv; man får manuellt dra ut längden på varje ord och sedan bläddra i listan för att hitta något som råkar passa. Det gör att det krävs en hel del tålamod för att få ihop ett bra pussel.

En enorm förbättring vore därför att bygga in en så kallad Constraint Solver. Det låter kanske avancerat, men i praktiken handlar det om att ge appen en egen ’hjärna’ som förstår hur alla rutor hänger ihop. Istället för att användaren ska gissa sig fram, kan systemet räkna ut exakt vilka bokstäver som måste sitta var för att hela korsordet ska gå ihop logiskt.

Istället för att användaren testar ord för ord skulle systemet kunna räkna ut hela lösningen på en gång. Den tittar då på hela rutnätet samtidigt och ser till att alla korsningar stämmer matematiskt. Det här skulle spara massor av tid. Istället för att bråka med att få bokstäverna att gå ihop kan användaren bara bestämma ett tema och låta appen sköta det tråkiga pusslandet. Det är här vi verkligen skapar värde, genom att göra det enkelt för vem som helst att bygga ett proffsigt korsord utan att vara expert.

6.4.3 Interaktivt samarbete med Auto-fill

Något som känns som ett mer naturligt steg än att automatisera precis allt är att börja med en Auto-fill funktion. Det fungerar som en slags smart assistent. Om du har byggt halva korsordet men fastnar i ett hörn för att inga ord verkar passa, ska du bara kunna trycka på en knapp för att få hjälp.

För att det här ska funka krävs det att algoritmen tänker flera steg i förväg, något man kallar för ”look-ahead”. Systemet kollar inte bara vad som passar i just de rutorna du jobbar med, utan räknar ut om det ordet kommer ställa till problem för de andra rutorna längre fram. Det handlar om att hjälpa användaren att behålla sitt flow. Istället för att behöva radera hälften av de orden man gjort för att man råkat ”måla in sig i ett hörn”, kan appen guida en genom de svåraste delarna. Det gör att skapandet känns mer som ett samarbete än en kamp mot logiken.

6.4.4 AI-genererade ledtrådar och semantiska kvalitetsspärrar

En annan spännande vidareutveckling vore att integrera AI som kan ge förslag på ledtrådar i korsordet. Om en användare till exempel placerar ordet ”Kaffe”, skulle systemet direkt kunna föreslå allt från enkla beskrivningar som ”Svart dryck” till mer kluriga varianter som ”Böna som piggar upp”. Tanken är att användaren ska kunna välja en stil eller svårighetsgrad som passar just deras korsord, vilket gör att man slipper fastna när fantasin brister.

Men för att detta ska fungera på en öppen plattform krävs det också att vi har kontroll på vad som skrivs. Vi skulle behöva bygga in semantiska filter som känner av och flaggar olämpligt innehåll. Istället för att bara ha en lista på ”förbjudna ord” kan AI hjälpa till att förstå sammanhanget och rensa bort sådant som är kränkande eller rent sagt dåligt. Det handlar om att skapa en trygg och trevlig miljö för alla användare, oavsett om man delar sitt korsord med vänner eller publicerar det för främlingar. Genom att automatiskt flagga för tveksamma ordval ser vi till att plattformen håller en god nivå utan att vi behöver sitta och moderera allt manuellt.

6.4.5 Ordlistornas struktur och importering

Det nuvarande systemet för ordlistor utvecklades tidigt under projektets utvecklingsfas med mål på att snabbt möjliggöra grundläggande funktionalitet för validering och hantering av ord. Lösning har därefter inte genomgått någon större omstrukturering eller vidareutveckling vilket har behållit vissa tekniska kompromisser och lösningar som inte är optimala ur ett långsiktigt perspektiv.

En tydlig begränsning är att de permanenta ordlistorna fortfarande lagras som JavaScript-filer direkt hämtade från det tidigare projektet. Eftersom resten av systemet är implementerat i TypeScript används dessa filer i praktiken endast som strukturerade textkällor snarare än som faktisk JavaScript-kod. Detta skapar en viss inkonsekvens i projektets arkitektur och gör att ordlistorna tar mer plats än de skulle göra med ett bättre anpassat filformat.

En möjlig vidareutveckling är därför att ersätta de nuvarande JavaScript-filerna med ett enklare och mer passande filformat, exempelvis rena textfiler, JSON eller andra strukturerade dataformat. En sådan ändring skulle göra det möjligt för ordlistorna att bli enklare att läsa, underhålla och validera, samtidigt som det skulle förbättra kompatibiliteten med TypeScript-baserad logik. Det skulle också vara passande att i samband med detta att ändra strukturformatet i filen så att exempelvis orden listas på ett mer effektivt sätt och att orden enklare kan kopplas till teman.

Importfunktionen för ordlistor implementerades relativt sent under produktens utveckling och är därför mindre utvecklad än andra delar av systemet. Funktionen fungerar i sin nuvarande form främst som stöd för import av mindre ordlistor för specifika ord och kräver att användaren har en förståelse över hur filen hanteras internt. Det finns därför flera möjliga förbättringsområden. En möjlig utveckling är

att göra importören mer flexibel genom att stödja fler filformat och mindre strikt formatering.

En annan förbättring vore att implementera en hjälpfunktion som automatiskt bearbetar och normaliserar importerade filer innan de används i systemet för att möjliggöra importen av större oprocessade filer. En sådan funktion skulle kunna analysera innehållet, identifiera giltiga ord, filtrera bort ogiltiga rader samt omvandla data till ett standardiserat internt format. Detta skulle göra importprocessen mer robust och användarvänlig samtidigt som det minskar risken för inkompatibla ordlistor.

Ytterligare framtida utveckling skulle även kunna inkludera permanent lagring av importerade ordlistor i ett kompendium där användaren sedan kan favorisera ordlistorna som de är intresserade av, stöd för flera aktiva ordlistor samtidigt samt möjligheten att lägga till specifika nya ord utan att behöva importera en helt ny ordlista.

6.5 Integritet och dataskydd med Firebase och cookies

Firebase användes för autentisering och lagring av data. Firebase tjänsten tillhandahålls av Google. Vid användning av Googles tjänster är det viktigt att följa dataskyddsregler, som GDPR (General Data Protection Regulation), för att säkerställa att personuppgifter hanteras på ett säkert och korrekt sätt. Projektet har tagit hänsyn till integritetsaspekter för att hantera användardata på ett säkert sätt. Applikationen erbjuder även möjligheten att radera sitt konto helt.

Ingen direkt implementation av samtyckeshantering för cookies har utvecklats inom projektet, men det är en viktig aspekt som kan tas med i framtida vidareutveckling av systemet.

6.6 Slutsats

Under de två läsperioderna vi har jobbat har vi med hjälp av en befintlig prototyp snabbt kunnat utveckla en fungerande applikation som faktiskt går att använda i praktiken. Att ha den utgångspunkten vi har haft, har gjort att vi kunde lägga vår tid på att utveckla appens kärnfunktioner och skapa faktiskt värde, snarare än att fastna i en komplicerad uppstartsfas. Resultatet är en plattform där de centrala delarna fungerar och som tydligt bevisar att konceptet är genomförbart.

Vi är särskilt stolta över att ha skapat en lösning som kombinerar kreativitet med teknisk stabilitet. Kodbasen och arkitekturen utgör nu en robust grund för framtida utveckling. Strukturen är så pass genomtänkt att nya funktioner kan integreras utan att man behöver börja om från noll. Vi ser detta projekt som ett starkt första steg, där vi inte bara har lärt oss enormt mycket, utan också levererat en produkt som faktiskt fungerar i praktiken.

Givetvis återstår en stor del arbete innan applikationen är redo för en riktig lansering. En viktig insikt rör valet av Firebase som backend. Medan det var ovärderligt i början för att snabbt komma igång, ser vi utmaningar på sikt. Om applikationen skulle kraftigt växa medför plattformen en stor risk för svårhanterliga driftskostnader. För en uppskalning i framtiden krävs därför en mer långsiktig strategi, så att systemet kan växa hållbart utan att kostnaderna blir ett problem.

Sammanfattningsvis är vi mycket nöjda med vad vi har åstadkommit under den tid vi haft. Vi har gått från noll till en fungerande applikation, stakat ut en tydlig riktning framåt och byggt en stabil bas. Även om det är flera steg kvar till en lansering, bevisar projektet att vår grundidé är stark och att potentialen finns där för att skapa något ännu bättre i framtiden.

Referenser

Beacham, A., Chen, X., Sillito, J., & van Beek, P. (2001). Constraint Programming Lessons Learned from Crossword Puzzles. https://cs.uwaterloo.ca/~vanbeek/Publications/cai01a.pdf

Buffy. (2025). *Nybörjarguide för Discord*. Hämtad 2 maj 2026, från https://support.discord.com/hc/sv/articles/360045138571-Nyb%C3%B6rjarguide-f%C3%B6r-Discord

Designlab. (u.å.). *Introduction to Figma*. Hämtad 1 maj 2026, från https://designlab.com/figma-101-course/introduction-to-figma

Doug Stevenson. (2018). *What is Firebase? The complete story, abridged.* Hämtad 8 maj 2026, från https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

GitHub, Inc. (2026). *About GitHub and Git*. Hämtad 30 april 2026, från https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

Gustavsson, R. (2014). *Digitala verktyg i en inkluderande undervisning?* [Examensuppsats, Högskolan Kristianstad]. DiVA.

Hermez, M., & Tobiasson, E. (2025). Korsord allt mer populärt – även bland yngre. https://www.sverigesradio.se/artikel/korsord-allt-mer-populart-aven-bland-yngre

Hjertström, A., & Zelander, E. (2011). *Fult språk och språknormer* [examensuppsats, Malmö Högskola]. DiVA.

Jonsson, C. (2003). *Ordinlärning i främmande språk* [examensuppsats, Linköpings universitet]. DiVA.

LudHamma. (2026). *Korsordsmakaren v0.1*. Hämtad 2 juni 2026, från https://github.com/maidads/korsordsmakaren/releases/tag/v0.1

Microsoft. (2025). *Vad är Kanban?* Hämtad 8 maj 2026, från https://learn.microsoft.com/sv-se/devops/plan/what-is-kanban

Microsoft TypeScript-team. (2026). *The TypeScript Handbook*. Hämtad 1 maj 2026, från https://www.typescriptlang.org/docs/handbook/intro.html

Nationalencyklopedin. (u.å.). Korsordshjälpen. *Nationalencyklopedin*. https://www.ne.se/info/privatpersoner/korsordshjalpen/

Poole, D. L., & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press. https://artint.info/3e/html/ArtInt3e.Ch4.html

Scrum. (u.å). *Vad är Scrum?* Hämtad 18 maj 2026, från https://www.scrum.org/learning-series/what-is-scrum/

Tengelin, K. (2012). *Undervisa ord eller undervisas av ord* [examensuppsats, Linköpings universitet]. DiVA.

Tidskrifter, S. (2021). Så går det för korsordstidningarna. https://sverigestidskrifter.se/nyheter/sa-gar-det-for-korsordstidningarna/

Trello. (u.å.). *Komma igång med Trello*. Hämtad 8 maj 2026, från https://trello.com/guide/trello-101