IT-HÖGSKOLAN

Här startar din IT-karriär.

Reseplaneringsverktyg

En mer personlig reseplaneringsupplevelse för resenärer i Stockholm med omnejd

Datum: Maj 2023
Studerande: Svante Jonsson
Utbildning: JavaScript-utvecklare, 400 YH-poäng
Kurs: Examensarbete, 20 YH-poäng
Examinator: Niklas Hjelm

Sammanfattning

Denna rapport presenterar utvecklingen av en användarvänlig och effektiv reseplanerare och avgångstavla för kollektivtrafiken i Stockholm med omnejd. Projektets huvudmål var att förse användare med ett pålitligt och lättanvänt planeringsverktyg med funktionalitet såsom reseförslag och avgångstider i realtid samt en interaktiv karta för att visualisera resvägen. Verktyget skulle också göras tillgängligt för en bred publik genom att erbjuda flera färgteman och översättningar som användarna själva kan välja, utifrån sina egna preferenser. Projektet inkluderade även ett inloggningssystem som skulle göra det möjligt för användarna att spara deras favoritstationer och resor, men denna funktionalitet kunde tyvärr inte slutföras helt inom den givna tidsramen.

Utvecklingsprocessen följde en tydlig och väl strukturerad planering och ett noggrant urval av lämpliga teknologier och verktyg. Trots en oförutsedd IT-attack vilket ledde till att API:n för att hämta trafikinformation behövde bytas ut, som orsakade en försening i utvecklingen, uppnådde projektet ändå majoriteten av de förutbestämda målen. Resultatet blev alltså ett omfattande och personligt reseplaneringsverktyg med ett intuitivt användargränssnitt, vilket gör det till en effektiv lösning för kollektivtrafikanvändare i Stockholmsområdet.

Abstract

This report presents the development of a user-friendly and efficient journey planner and departure board for public transport in the Stockholm area. The main goal of the project was to provide users with a reliable and easy-to-use planning tool with functionality such as travel suggestions and real-time departure times as well as an interactive map to visualize the travel routes. The tool would also be made accessible to a wider audience by offering multiple color themes and translations that users can choose for themselves, based on their own preferences. The project also included a login system that would allow users to save their favorite stations and journeys, but unfortunately this functionality could not be fully complete within the given time frame.

The development process followed a clear and well-structured project plan and a careful selection of suitable technologies and tools. Despite an unforeseen IT attack that required the API retrieving traffic information to be replaced, which caused a delay in the development, the project still achieved the majority of the predetermined goals. The result was a comprehensive and personal travel planning tool with an intuitive user interface, which makes it an efficient solution for public transport users in the Stockholm area.

Innehållsförteckning

1. Introduktion

I dagens samhälle är resor med kollektivtrafik väldigt vanliga, för både jobbpendling och fritidsresor. Som en webbutvecklare och användare av kollektivtrafik upptäckte jag en efterfrågan på ett verktyg som kunde underlätta planeringsprocessen samt göra den mer tillgänglig och effektiv. Det här projektet gav mig möjligheten att skapa ett användarvänligt och effektivt reseplaneringsverktyg samt testa mina kunskaper och färdigheter inom webbutveckling.

Projektets främsta mål var att ge användarna en pålitlig och tydlig reseplanerare, avgångstavla och interaktiv karta för att hjälpa dem enkelt planera sina resor. Projektet var också översatt till både svenska och engelska. Det erbjöd även användarna valet mellan ett mörk och ett ljust färgtema. Allt det här för att ta hänsyn till så många individer som möjligt. Inkluderat i projektet var också ett inloggningssystem som gjorde det möjligt för användare att spara sina preferenser kring färgtema och språk samt favoritstationer och resor.

Jag utvecklade det här projektet som mitt examensarbete. Det gav mig möjligheten att pröva mina färdigheter samtidigt som jag försökte skapa något som skulle kunna användas i verkligheten. Projektet var designat för att ge en snabbare och mer personlig reseplaneringsupplevelse för resenärer i Stockholm med omnejd.

2. Syfte och mål

2.1. Syfte

Syftet är att designa och skapa ett webbaserat reseplaneringsverktyg för kollektivtrafik. Det här verktyget ska innehålla både en reseplanerare och avgångstavla som ska ge användarna möjligheten att planera sina resor, se avgångar och utforska en interaktiv karta som visuellt presenterar föreslagna resvägar. Det kommer också finnas flera färgteman; exempelvis mörkt och ljust tema. För att göra reseplaneringsverktyg så tillgänglig som möjligt kommer den att finnas tillgänglig på både svenska och engelska.

Dessutom ska det finnas ett login system som ger användarna möjligheten att spara sitt föredragna färgtema och språk. De kommer också kunna spara deras favoritstationer och resor vilket kommer förbättra användarupplevelsen och göra reseplaneringsverktyg mer personligt.

Kort sagt är projektets huvudsyfte att ge användarna ett omfattande och personligt reseplaneringsverktyg som fungerar både innan och när man är ute i trafiken. Användarna ska kunna enkelt planera deras resväg, få uppdaterad trafikinformation och se en visuell representation av resan från punkt A till punkt B. Genom att erbjuda flera färgteman och språk kommer projektet att kunna nå en bredare publik.

2.2. Mål

Huvudmålet med projektet är att skapa en reseplanerare som ska underlätta planeringen av resor med kollektivtrafik. Användarna får information om de bästa rutterna och transportmedlen för deras resmål genom att använda filter såsom föredragen avgångstid och start- och slutstation.

Ett andra mål är att skapa en avgångstavla som visar information om stationers avgångar och ankomster i realtid. Denna funktion ska hjälpa användarna att hålla sig informerade kring kommande avgångar och eventuella tidsförändringar såsom förseningar.

Projektet ska också innehålla en interaktiv karta som visuellt presenterar resultaten från både reseplaneraren och avgångstavlan. Förhoppningen är att kartan ska hjälpa användarna att bättre förstå rutterna och resealternativen de har att välja mellan.

Hela projektet ska också finnas tillgängligt i flera färgteman, exempelvis ett mörkt och ett ljust tema. Målet med denna funktion är att användarna ska kunna välja ett färgtema som matchar deras preferens och på så sätt göra användarupplevelsen mer personlig.

Projektet siktar även på att vara tillgängligt på både svenska och engelska. Det här för att ta hänsyn till olika användares språkkunskaper och preferenser. Tanken är att implementera det här så att projektet ska kunna översättas till fler språk i framtiden.
Slutligen ska projektet inkludera ett inloggningssystem som gör det möjligt för användare att spara sina preferenser för färgtema och språk. De ska också kunna spara deras favoritstationer och resor. Målet med denna funktion är att ge användarna en personlig och smidig upplevelse av projektet.

3. Metod

3.1. Planering

Planeringssteget var en väldigt viktigt del som la grund för ett lyckat genomförande och slutförande av projektet. En genomtänkt projektplan togs fram efter ett flertal faktorer såsom tidsallokering av tid för varje del, önskade funktioner, existerande alternativ och passande teknologier vägdes in i beslutsprocessen. En noggrann planering säkerställde att projektet skulle genomföras smidigt och effektivt, samt resultera i ett högkvalitativt resultat.

Tidsramen för projektet bestämdes till 3 månader med en deadline i slutet av april. Det här tillät att tillräckligt med tid gavs till varje del av projektet. Designsteget gavs 2 veckor till följd av utvecklingssteget som fick 2 hela månader att utföras. Slutligen skulle 1-2 veckor ägnas för att testa och fixa eventuella fel på resultatet från det tidigare steget.

Sedan gjordes en analys av liknande projekt, till exempel Västtrafik och SL:s egna reseplaneringsverktyg för att identifiera deras styrkor och svagheter. Det här steget blev betydande för projektet när det kom till att identifiera områden som kunde förbättras och funktioner som kunde introduceras så att det här projektet skulle stå ut och bli bättre än de som redan fanns på marknaden.

Därefter togs besluten om vilken funktionalitet som skulle inkluderas i projektet. Utifrån den förutbestämda tidsramen och analyser av liknande verktyg beslutades det att inkludera följande funktioner: en reseplanerare, avgångstavla, interaktiv karta, ljusa och mörka färgteman, översättning till svenska och engelska, inloggningssystem samt möjlighet att spara favoritstationer och resor. Målet med denna uppsättning funktioner var att skapa ett välutvecklat projekt som också kunde bli färdigställt inom den uppsatta tidsramen.

Teknikerna som valdes att användas i projektet baserades utifrån flera faktorer, till exempel tidsramen, tidigare erfarenheter och vad som lämpades bäst till uppgiften. Efter en noggrann undersökning valdes MySQL som databas för projektet medan NestJS i TypeScript valdes som API ramverk och React i TypeScript valdes som klient ramverk. För hämtning av trafikinformation valdes Västtrafiks API som senare byttes ut till SL:s API då Västtrafik stängde ner deras utvecklarportal på obestämd tid. Turborepo valdes slutligen som bas för projektets struktur samtidigt som Docker och Nginx skulle användas för att underlätta framtida distribution av projektet.

Sammanfattningsvis var planeringssteget ett genomtänkt och viktigt steg i projektprocessen som gjorde de resterande delarna av projektet mycket lättare att utföra. Genom att noggrant överväga tidsramen, önskad funktionalitet och passande teknologi så pass tidigt var projektet direkt på väg mot ett högkvalitativt resultat.

3.2. Design

Designsteget började med att återigen analysera liknande projekt, såsom SL och Västtrafiks egna reseplaneringsverktyg för att identifiera deras styrkor och svagheter. Analysen ledde till upptäckten av flera bästa praxis. Den kunde även användas som designspiration för projektet, eftersom designstyrkorna hos liknande projekt kunde skiljas från deras svagheter.

Med analysen gjord och önskad funktionalitet bestämd var det dags att formgiva projektets utseende. Med hänsyn till funktionaliteten och användarupplevelsen bestämdes det tidigt att den interaktiva kartan skulle få ett stort fokus i designen. Det här skulle inte endast underlätta implementationen av funktionaliteten i projektet utan också förbättra projektets utseende.

Efteråt med en generell design i åtanke påbörjades en wireframe som sedan fungerade som en ritning för projektets struktur och layout. Denna på många sätt simpla rendering av slutresultatet gav en tydlig bild av det tänkta flödet och komponenter vilket säkerställer att all funktionalitet var inräknad och välintegrerad.

När wireframen var färdigställd var det dags att skapa en detaljerad mockup som skulle ge en mer fullständig visuell representation av det slutgiltliga resultatet. Det här steget inkluderade att välja typografi, lämpliga färgteman och andra designelement som skulle bidra till projektets estetik. Valen gjordes noggrant utifrån designprinciper, tidigare erfarenheter och den föregående analysen. Målet var att uppfylla användarnas preferenser och förväntningar för ett projekt som det här.

Kort sagt var designsteget en grundläggande och välplanerad del av projektet som lade grunden för ett fortsatt bra arbete. Designstegets fokus var på att analysera liknande projekt, önskad funktionalitet och användarupplevelse. Det avsattes även tillräckligt med tid för att skapa ordentliga wireframes och mockups. Det här säkerställde i sin tur att projektet skulle vara både visuellt tilltalande och funktionellt effektivt.

3.3. Utveckling

Utvecklingssteget var den sista delen av projektet och dess mål var att utföra planeringen och implementationen av designen framgångsrikt samt testa och finslipa projektet inom den avsedda tidsramen. För att lyckas med det här krävdes en välorganiserad arbetsprocess och projektstruktur, där alla aspekter av projektet hanterades effektivt och presenterades tydligt.

3.3.1. Arbetsprocess

Utvecklingssteget var uppdelat i 1 veckas etapper, liknande de sprintar som ofta syns i projekt på agila arbetsplatser. Varje vecka påbörjades med att planera vad som behövde göras och hur lång tid en uppgift förväntades att ta. Om det fanns uppgifter kvar från

föregående vecka prioriterades det att färdigställa dessa före planeringen av nya. Det här bestämdes för att undvika att uppgifter per vecka skulle öka exponentiellt i de fall att en uppgift tog mer tid än planerat. Uppgifterna hanterades genom GitHub issues och deras status av GitHub Projects. Efter planeringen ägnades resten av veckan till att lösa de planerade uppgifterna samt att testa applikationen bitvis och i sin helhet.

Det fanns flera anledningar till att använda GitHubs egna projektplaneringsverktyg under projektet. En av anledningarna var att utvecklaren hade tidigare använt verktyget, vilket innebar att gränssnittet och funktionaliteten redan var bekant. Detta bidrog till att projektet snabbt kunde komma igång, utan att behöva ödsla tid på att lära sig ett nytt verktyg. Eftersom projektet versionshanterades med hjälp av Git och lagrades hos GitHub, var deras planeringsverktyg också mycket lättillgängligt. Att ha både verktygen och källkoden på samma plats, samt att de erbjöd flera smidiga integrationer direkt med projektet underlättade avsevärt för att hålla koll på statusen för olika uppgifter.

För att sammanfatta kan det fastställas att projektets struktur med veckoetapper och användningen av Github bidrog till en effektiv och välorganiserad arbetsprocess. Genom att använda bekanta verktyg med smidig integration sinsemellan minimerades lärandekurvan och hanteringen av uppgifterna förenklades.

3.3.2. Projektstruktur

Projektets struktur utformades för att skapa en effektiv och smidig utvecklingsprocess. Det här gjordes genom implementationen av Turborepo, TypeScript och användningen av delad kod mellan API:n och klienten.

För det första använde projektet Turborepo, ett modernt byggsystem optimerat för JavaScript och TypeScript. Turborepo gjorde det möjligt att implementera en monorepo struktur för projektet, vilket betydde att API:n och klienten smidigt kunde integreras i ett och samma repository. Sammanslagningen ledde till en mer organiserad och enkel utvecklingsmiljö som i sin tur resulterade i en konsekvent och robust kodbas.

Utöver Turborepo använde sig projektet TypeScript för att ge kodbasen struktur och läsbarhet. TypeScripts typsystem blev också avgörande när det kom till att förhindra buggar och säkerställa att koden betedde sig förutsägbart, vilket ökade projektets kodkvalité. Genom att använda TypeScript kunde författaren skriva mer lättunderhållen och skalbar kod som följaktligen resulterade i en mer robust och stabil produkt.

Slutligen gynnades projektet mycket av att kunna dela kod mellan API:n och klienten, vilket gjordes möjligt av Turborepo. Att kunna dela kod mellan applikationerna innebar exempelvis att endast ett exemplar av en TypeScript typ behövde skapas när den behövdes för båda applikationerna, istället för att behöva skapa två separata men identiska typer. Det här säkerställde konsekvens och enhetlighet mellan olika de olika delarna av projektet och minskade risken för fel som härstammar från duplicerad kod.

Sammanfattningsvis var projektets struktur, som byggdes med Turborepo, TypeScript och möjligheten att dela kod mellan applikationer, utformad för att skapa en så smidig utvecklingsprocess som möjligt och därmed resultera i en högkvalitativ produkt.

3.3.3. Auth0

För att uppnå målet att tillåta projektets användare att logga in och spara sina preferenser var implementationen av ett effektivt och säkert autentiseringssystem avgörande. Med dessa krav beslutades det att använda Auth0, en flexibel och stabil lösning för att integrera autentisering och auktorisering tjänster i webbappar.

Auth0 valdes att användas i projektet på grund av flera anledningar. För det första, efter att ha haft erfarenhet av tjänsten i tidigare projekt, hade den visat sig vara en pålitlig och säker lösning. Bekantskapen med Auth0:s funktioner underlättade integrationen i projektet, vilket säkerställde att målet för ett inloggningssystem och smidig användarupplevelse kunde uppnås.

En annan stor fördel med Auth0 är dess lätta konfigurering och snabba integration. Tjänsten och plattformen är designade för att vara en drop-in lösning, vilket minskar tiden för implementationen och tillåter en snabbare utvecklingsprocess. Tack vare detta kunde projektet snabbt gå vidare till att implementera annan funktionalitet, samtidigt som det kunde erbjuda ett robust och säkert autentiseringssystem.

Utöver den lätta implementationen erbjuder Auth0 inte bara möjligheten att skapa konton på traditionellt vis, genom email och lösenord, utan också genom redan existerande konton på andra plattformar. Det här ger alltså projektets användare alternativet att exempelvis logga in med ett Google-konto. Som ett resultat av denna funktion blev projektet mer komplett och användarupplevelsen ökade avsevärt.

Kort sagt var valet av Auth0 ett strategiskt beslut för att säkerställa en säker och användarvänlig autentiseringsprocess, samtidigt som tiden för implementationen kunde förkortas.

3.3.4. Västtrafik och SL:s API:er

Eftersom projektets huvudmål var att skapa ett reseplaneringsverktyg för resenärer i kollektivtrafiken var det avgörande att ha tillgång till den senaste trafikinformationen. Därför ansågs det nödvändigt att använda en extern API för att hämta trafikinformation.

Inledningsvis var projektet planerat att använda Västtrafiks API, eftersom det ger heltäckande information om kollektivtrafiken i Göteborg med omnejd. API:n erbjuder värdefull information om stationer, avgångar, reseförslag och mycket mer, vilket gjorde den till ett perfekt val för projektet. Men runt mitten av april utsattes Västtrafik för en IT-attack,

vilket resulterade i att deras API och utvecklarportal blev otillgängliga en period. API:n var snabbt tillbaka, men deras utvecklarportal, som innehöll API dokumentationen, förblev nere då Västtrafik hade planerat att ersätta portalen senare på året. På grund av detta blev det snabbt tydligt att API:n behövde bytas ut till en annan med tillgänglig dokumentation om projektet skulle hinna bli klart i tid.

Efter en ingående undersökning och tester av liknande API:er beslutades det att projektet skulle använda SL:s API istället. SL:s API omfattar information om kollektivtrafiken i Stockholm och dess kranskommuner, likt datan Västtrafiks API erbjöd. Medan denna förändring krävde vissa justeringar i projektet verkade det vara en hållbar lösning i slutändan. SL:s API tillät enkel hämtning av väsentlig trafikinformation, såsom stationer, avgångar och reseförslag, vilket var avgörande för att projektet skulle kunna fortsätta och resultera i en färdig produkt inom den givna tidsramen.

3.3.5. Databasen

Projektet använde MySQL, en open-source relationsdatabas för datalagring och hantering. MySQL:s effektiva datahantering- och hämtningsfunktioner bidrog till skapandet av en pålitlig, skalbar och högpresterande databas som i sin tur underlättade implementationen av projektets primära funktioner.

Databasen lagrade och hanterade effektivt flera olika datatyper, såsom stationer och information kring dessa samt Auth0 referenser. Dessutom skapades flera relationer inom databasen för att koppla användare till deras favoritstationer och resor, vilket i slutändan förbättrade användarupplevelsen och den övergripande funktionaliteten av projektet.

För att sammanfatta valdes MySQL som databas på grund av dess öppna källkod, robusta funktionalitet och effektiva relationsdatahantering. Det här beslutet resulterade i att projektet kunde framgångsrikt lagra, relatera och hantera all väsentlig information, till exempel stationer dess data.

3.3.6. API:n

Projektets API hade som huvudmål att skapa en smidig koppling mellan klienten och alla underliggande datakällor, såsom databasen och SL:s API. För att lösa denna uppgift valdes serverramverket NestJS eftersom det hade ett flertal fördelar, till exempel ett robust modulsystem och tydlig dokumentation.

En av huvudanledningarna till varför NestJS valdes var ramverkets kompatibilitet med TypeScript. Det här gav projektet flera fördelar, exempelvis ökad kodkvalité, robust typkontroll och en minskad mängd fel vid runtime, som alla resulterade i mer pålitlig och lätthanterad kodbas.

Dessutom var NestJS:s modulära arkitektur mycket passande för projektet. Genom att dela upp kodbasen i mindre och mer lätthanterliga moduler kunde projektet enkelt skalas upp när det var dags att implementera ny funktionalitet samtidigt som projektet hölls lätt underhållet.

NestJS erbjöd också inbyggda moduler för jobb med till exempel databaser, hantering av säkerhet etc. Dessa funktioner underlätta en snabb utvecklingsprocess, vilket tillät fokuset att läggas på att skapa en bra produkt istället för att fastna i krångligheterna med att hantera infrastrukturen på klientsidan.

Sedan var även NestJS:s CLI (Command Line Interface) verktyg väldigt trevligt när det kom till repetitiva uppgifter, såsom att skapa controllers, services och modules. Att ha tillgång till det här verktyget påskyndade utvecklingsprocessen avsevärt, vilket gjorde att tidsramen för projektet blev mer flexibel.

Sammanfattningsvis visade NestJS sig vara ett mycket bra val för utvecklingen av projektets API. Ramverkets kompatibilitet med TypeScript, modulära arkitektur och omfattande funktioner och verktyg gjorde utvecklingsprocessen snabbare, säkerställde hög kodkvalité och gjorde API:n mer skalbar. Genom användningen av NestJS fick projektet en robust och pålitlig API med all önskad funktionalitet och flexibiliteten att enkelt kunna lägga till ny funktionalitet i framtiden.

3.3.7. Klienten

Klientens huvudmål var att förse projektets användare med ett snyggt och smidigt reseplaneringsverktyg. För att lyckas med detta behövde klienten effektivt kunna hämta och visa data från API:n, tillåta säker autentisering och ha ett visuellt tilltalande gränssnitt. Utifrån dessa kraven besultades det att klienten skulle skrivas med React i TypeScript med Vite som byggverktyget.

React valdes som front-end-ramverk på grund av dess komponentbaserade arkitektur, som ofta resulterar i mer modulär och återanvändbar kod. Genom att följa strukturen React tillåter blev det mycket lätt att effektivt organisera gränssnittet i distinkta komponenter, vilket i sin tur gjorde det enkelt att hantera och underhålla kodbasen. Reacts användning av en virtuell DOM (Document Object Model) gör också att ändringar i gränssnittet sker snabbt och smidigt på ett intelligent sätt, vilket säkerställer att användarupplevelsen blir bra.

Beslutet att skriva React med TypeScript grundades delvis i dess typsystem, vilket gav flera fördelar jämfört med vanlig JavaScript. Genom att hålla klienten typad under utvecklingen minskade inte bara risken för fel vid runtime utan bidrog också till högre kodkvalité. En annan anledning till att TypeScript valdes var för att hålla klienten och API:n lika på ett sådant sätt att kod enkelt skulle kunna delas sinsemellan, vilket resulterade i en effektiviserad utveckling och kommunikation mellan applikationerna.

Användningen av Vite istället för standard verktyget CRA (Create React App) grundades i verktygets fokus på att skapa en snabbare och smidigare utvecklingsupplevelse. Vites utvecklingsserver erbjuder flera bra funktioner, exempelvis HMR (Hot Module Replacement) som även kallas för hot reloading, vilket tillåter utvecklaren att se ändringar i realtid utan att behöva ladda om hela applikationen. Denna funktion förbättrar utvecklingsprocessen avsevärt och accelererar projektets tillväxt, vilket minskar trycket på den bestämda tidsramen.

För att sammanfatta var beslutet att använda React, TypeScript och Vite grundat i deras sammanställda förmåga att leverera en robust, pålitlig och lätt underhållen applikation. Dessa teknologier underlättade utvecklingen avsevärt och resulterade i ett responsivt och visuellt tilltalande front-end.

3.3.8. Distribution

För att effektivt kunna distribuera projektet krävdes en uppsättning metoder som kunde hantera applikationens tjänster och deras dependencies. Tekniken som valdes att användas behövde se till att projektet skulle köra smidigt och pålitligt i olika datormiljöer samtidigt som en effektiv och konsekvent distributionsprocess bibehölls. Efter noggrant övervägande valdes teknikerna Docker, Docker Compose och Nginx ut för att hjälpa projektet uppnå dessa krav.

Genom användningen av Docker kunde det skapas så kallade containrar för projektets olika delar, alltså, en för klienten, en för API:n och en för databasen. Dessa containrar omfattar applikationskoden, programmeringsmotorn, systemverktygen, dependencies och eventuella inställningar som krävs för att delarna ska fungera korrekt. Denna paketering av koden och dess dependencies i containrar tillåter applikationen att köras konsekvent och pålitligt, oberoende av den underliggande datormiljön.

Docker Compose användes för att hantera de olika containrarna i projektet. Genom en gemensam konfigurationsfil definierades projektets tjänster, deras relationer och dependencies. Denna fil fungerade som en ritning för hela projektet, vilket underlättade hanteringen och uppläggningen av containrarna och distributionen i helhet. Tack vare Docker Compose räckte det med ett kommando för att starta alla projektets delar utifrån specifikationerna i Docker-filerna och den gemensamma konfigurationsfilen.

Slutligen användes Nginx för att hantera HTTP-trafiken på klientsidan. Som webbserver bearbetade Nginx effektivt förfrågningar och skickade nödvändiga resurser till klienten när de efterfrågades. Dessutom fungerade den som en reverse proxy för att vidarebefordra förfrågningar från klienten till API:n.

Sammanfattningsvis löstes distributionen av projektet, efter noggrant övervägande med hjälp av Docker, Docker Compose och Nginx. Dessa teknologier underlättade hanteringen

av projektet och dess olika delar samtidigt som de säkerställer att allt alltid kommer att starta och köra på samma sätt, oberoende av den underliggande datormiljön.

4. Resultat och analys

4.1. Resultat

Projektet resulterade i ett fungerade reseplaneringsverktyg med en smidig reseplanerare och avgångstavla som uppnår majoriteten av de förutbestämda målen. Bland de uppnådda målen hittas möjligheten att välja mellan olika färgteman, vilket ger användaren mer kontroll över sin upplevelse. Dessutom finns stöd för flera språk, såsom svenska och engelska implementerat, vilket gör produkten mer tillgänglig och anpassad för en bredare publik.

En annan viktig funktion som framgångsrikt implementerades var den interaktiva kartan som hjälper användare att lättare förstå resultaten från reseplaneraren och avgångstavlan. Genom kartan kan användarna enkelt få en överblick över resultaten och utifrån det göra välinformerade beslut.

Slutligen har användarna möjligheten att logga in i produkten genom att antingen skapa ett nytt konto eller använda ett befintligt Google-konto. Tyvärr hann inte funktionaliteten för användare att spara deras favoritresvägar och -stationer riktigt färdigställas. Implementationen var fullbordad i API:n, medan kopplingen från klienten inte hann slutföras dessvärre. Det resulterade i att funktionen inte kunde göras tillgänglig för användarna inom den utsatta tidsramen och målet kan därför inte ses som avklarat.

4.1.1. Auth guard

Den följande bilden är av en NestJS guard. En guard skyddar vissa delar av applikationen från obehörig åtkomst och kontrollerar om användaren har de nödvändiga behörigheterna för att få tillgång till en viss rutt. Det här säkerställer att endast behöriga användare kan använda de skyddade delarna av applikationen.

```
@Injectable()
export class AuthGuard implements CanActivate {
    async canActivate(context: ExecutionContext): Promise<boolean> {
        try {
            const request = context.switchToHttp().getRequest();

            const token = request.headers['authorization']?.split(' ')[1];

            if (!token) throw new Error();

            const client = new JwksClient({
                jwksUri: `https://${env.AUTH0_DOMAIN}/.well-known/jwks.json`
            });

            const decoded = decode(token, { complete: true });

            if (!decoded?.payload?.sub) throw new Error();

            // Save the user object to the request
            request.user = decoded.payload;

            const key = await client.getSigningKey(decoded.header.kid);

            const verifiedToken = verify(token, key.getPublicKey(), {
                algorithms: ['RS256']
            });

            if (!verifiedToken) throw new Error();
            else return true;
        } catch (error) {
            throw new UnauthorizedException({
                success: false,
                message: 'Unauthorized'
            });
        }
    }
}
```

Denna specifika guarden kontrollerar att det inkommande anropet har en giltig JWT (JSON Web Token) i Authorization headern. Om den är giltig tillåter guarden anropet att fortgå, men om den istället är ogiltig eller saknas avbryter guarden anropet och skickar avsändaren svaret att hen är oautentiserad och har därför inte åtkomst till denna rutt.

Då de flesta av rutterna i API:n var helt öppna, alltså användarna behövde inte vara autentiserade för att använda dem, behövdes endast denna guard för rutter som rörde funktionalitet för inloggade användare, exempelvis rutten för att hämta en användares favoritstationer. Eftersom favoritstationer är unika för varje konto får endast den inloggade användaren hämta sina egna favoriter, det här görs genom att avkoda JWT:n för att identifiera vilken användare det är som gör anropet och endast hämta deras favoritstationer. Alltså, för att hämta en användares favoritstationer räcker det inte att ha deras unika identifikationsnummer, utan man behöver faktiskt vara inloggad med dennes konto.

4.1.2. Docker Compose

```yaml
version: '3.7'
name: exarb
services:
    db:
        build:
            context: ./
            dockerfile: apps/db/Dockerfile
    api:
        env_file: ./apps/api/.env
        environment:
            - NODE_ENV=production
        build:
            context: ./
            dockerfile: apps/api/Dockerfile
        ports:
            - '3000:3000'
        depends_on:
            - db
    client:
        env_file: ./apps/client/.env
        environment:
            - NODE_ENV=production
        build:
            context: ./
            dockerfile: apps/client/Dockerfile
        ports:
            - '80:80'
```

Det här är projektets Docker Compose-fil som används för att definiera och köra Docker-applikationer med flera containrar. Filen definierar vilka tjänster applikationen består av och hur de kommunicerar med varandra. I denna fil är det 3 tjänster definierade: db (databasen), api (API:n) och client (klienten). Varje tjänst definieras med sin egen Dockerfile som används för att bygga en image för tjänsten. Dockerfilen anger dependencies och eventuella konfigurationer för tjänsten, medan Docker Compose-filen binder samman allt och ser till att de kommunicerar med varandra på rätt sätt.

För att köra ett projekt baserat på flera applikationer i Docker-miljö är Docker Compose inte nödvändigt, men det underlättar konfiguration och utförande avsevärt. Docker Compose förenklar processen genom att tillåta hanteringen av alla tjänster att ske från en enda fil och att starta och stoppa dem med ett enda kommando. En annan fördel med Docker Compose är att den erbjuder även annan funktionalitet såsom automatiskt nätverksskapande och Service Discovery, vilket kan vara användbart i stora och komplexa projekt.

4.1.3. useApi

```
function useApi<Req, Res>(method: RequestInit['method'], path: string, body?: Req, queries?: Req):
ApiResponse<Res> {
    const [data, setData] = useState<Res>({} as Res);
    const [error, setError] = useState<Error | null>(null);
    const [loading, setLoading] = useState<boolean>(false);

    const { getAccessTokenSilently, isAuthenticated } = useAuth0();

    const controller = new AbortController();

    useEffect(() => {
        (async () => {
            const config: RequestInit = {
                method,
                signal: controller.signal,
                headers: {
                    'content-type': 'application/json'
                }
            };

            if (isAuthenticated) {
                const token = await getAccessTokenSilently();
                config.headers = {
                    Authorization: `Bearer ${token}`
                };
            }

            if (body) {
                config.body = JSON.stringify(body);
            } else if (queries) {
                const query = new URLSearchParams(queries as Record<string, string>).toString();
                path = `${path}?${query}`;
            }

            try {
                setLoading(true);
                const response = await fetch(`${env.API_URL}/${path}`, config);

                if (!response.ok) {
                    throw new Error(response.statusText);
                }

                const result: Res = await response.json();

                setData(result);
            } catch (e) {
                setError(e as Error);
            } finally {
                setLoading(false);
            }
        })();

        return () => controller.abort();
    }, [getAccessTokenSilently, method, path, body, queries]);

    return { error, loading, ...data };
}

export default useApi;
```

Den ovanstående bilden är en React hook som används för att göra anrop till API:n. Den har ett flertal parametrar, såsom HTTP-metoden, adressen och eventuella body- eller query-filter för anropet. Hooken returnerar ett objekt som innehåller anropssvaret, felmeddelanden och en boolean för att indikera om anropet fortfarande bearbetas. Om användaren är inloggad på klienten sätts också Authorization headern för att möjliggöra anrop till rutter skyddade av den tidigare nämnda guarden.

Användningen av en hook för att göra anrop till API:n förenklade koden och processen att hämta data avsevärt. Genom att abstrahera bort detaljerna av hur anropet görs kunde hooken användas på flera ställen utan att behöva repetera samma kod. Sedan underlättade även hooken hanteringen av anropssvaret, felmeddelanden och laddstatusen när den används i andra komponenter.

4.1.4. Avancerade komponent props

```
const styles = {
    base: 'rounded-md py-2 px-3 text-sm font-bold uppercase cursor-pointer
disabled:cursor-not-allowed focus:input_focus',
    transition: 'input_transition',
    variant: {
        text: 'text-blue-600 hover:bg-blue-600/10 active:bg-blue-600/20 disabled:text-
gray-500 disabled:bg-transparent dark:text-blue-400 dark:hover:bg-blue-400/10
dark:active:bg-blue-400/20 dark:disabled:text-gray-500 dark:disabled:bg-transparent',
        contained:
            'text-white shadow-sm bg-blue-600 hover:bg-blue-700 hover:shadow active:bg-
blue-500 active:shadow-md disabled:bg-gray-500 disabled:shadow dark:shadow-blue-500/25
dark:bg-blue-500 dark:hover:bg-blue-400 dark:hover:shadow-white/10 dark:active:bg-blue-600
dark:active:shadow-white/20 dark:disabled:bg-gray-500 dark:disabled:shadow',
        outlined:
            'border-2 border-blue-600/75 text-blue-600 hover:border-blue-600/100 hover:bg-
blue-600/10 active:bg-blue-600/20 disabled:border-gray-500 disabled:text-gray-500
disabled:bg-transparent dark:border-blue-400/75 dark:text-blue-400 dark:hover:border-
blue-400/100 dark:hover:bg-blue-400/10 dark:active:bg-blue-400/20 dark:disabled:border-
gray-500 dark:disabled:text-gray-500 dark:disabled:bg-transparent'
    },
    size: {
        small: 'py-1 px-2 text-xs',
        medium: 'py-2 px-3 text-sm'
    }
};

interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
    className?: string;
    children: React.ReactNode;
    variant?: keyof typeof styles.variant;
    size?: keyof typeof styles.size;
}

function Button({ className, children, variant = 'text', size = 'medium', ...props }:
ButtonProps) {
    const classes = clsx(className, styles.base, styles.transition,
styles.variant[variant], styles.size[size]);

    return (
        <button className={classes} {...props}>
            {children}
        </button>
    );
}
```

Det här är en React-komponent som skapar en stylad knapp baserad på standard HTML (HyperText Markup Language) elementet button. Den tar in props såsom className, children, variant och size, etc. för att avgöra utseendet och beteendet av knappen. variant

propen bestämmer stilen på knappen, medan size definierar storleken. className möjliggör ytterligare styling att läggas på knappen.

Att använda extends och keyof typeof är fördelaktigt av flera anledningar. Genom att basera komponenten på den riktiga HTML knapp-komponenten erbjuder extends en möjlighet att automatiskt ta del av alla standard-props. Tack vare det här kan den nya knapp-komponenten användas precis som en vanlig knapp utan att behöva implementera standard funktionaliteten manuellt, vilket underlättar och snabbar på utvecklingsprocessen.

Samtidigt bidrar keyof typeof till att förhindra ogiltiga värden vid användning av komponenten. Det här åstadkoms genom att endast tillåta alternativ som finns i styles.variant respektive styles.size objekten. Alltså säkerställs det att komponenten alltid kommer fungera som förväntat, även om objekten skulle uppdateras med nya alternativ eller när befintliga alternativ tas bort.

4.2. Analys

När projektet nådde sin deadline i slutet av april hade det lyckats uppnå majoriteten av sina förutbestämda mål. Resultatet blev ett fullt fungerande reseplaneringsverktyg med intuitiva filter och detaljerade resultat som förtydligas ytterligare med hjälp av en interaktiv karta. Några av de filter som erbjuds är start- och slutstation samt önskad ankomst- och avgångstid. Informationen som visas i resultaten är bland annat avgångstider, stationer, linjer och fordon, medan kartan visar hur alla delar hänger ihop i stadsbilden.

Utöver att vara en fungerande reseplanerare och avgångstavla hade projektet som mål att vara anpassad och tillgänglig till så många individer som möjligt genom att ge dem mer kontroll över deras användarupplevelse. Det här målet uppnåddes genom att erbjuda hela användargränssnittet på både svenska och engelska, med tanken och strukturen att enkelt kunna lägga till fler översättningar i framtiden. En annan aspekt av projektet som ökar tillgängligheten och ger användaren mer kontroll över sin upplevelse är implementationen av flera färgteman. I detta stadie erbjuds två färgteman, ett ljust och ett mörkt, men även denna implementationen har utformats på så sätt att det enkelt ska gå att skapa fler färgteman senare.

En tredje funktion som skulle förbättra användarupplevelsen och göra användningen av reseplaneringsverktyget smidigare var ett inloggningssystem, vilket skulle göra det möjligt för användarna att spara sina preferenser för färgtema och språk samt deras favoritstationer och resor. Tyvärr kunde all funktionalitet runt inloggningssystemet inte färdigställas och därför sparas endast preferenserna för färgtema och språk lokalt på användarens maskin, medan funktionen att sätta favoritstationer och resor inte kunde erbjudas över huvud taget.

Trots att alla mål inte kunde uppfyllas anses projektets syfte som avklarat. Eftersom huvudsyftet var att ge användarna ett omfattande och personligt reseplaneringsverktyg som

fungerar både innan och när man är ute i trafiken, vilket projektet resulterade i. Användarna kan enkelt planera deras resväg, få uppdaterad trafikinformation och se en visuell representation av resan från punkt A till punkt B. Samtidigt får användarna en personlig upplevelse av projektet tack vare valen av färgtema och språk som de har att välja mellan.

Projektet hade goda förutsättningar för att uppnå projektets syfte och mål. Många av dessa låg i förarbetet och planeringen av projektet. Genom att analysera liknande verktyg samt undersöka och testa olika teknologier för att i ett tidigt skede hitta den väsentliga funktionaliteten och lämpliga teknologier för att implementera dessa var grundidén av projektet tidigt definierad. Sedan med hjälp av en tydligt definierad tidsplan, för både projektet i helhet men också för varje individuell del kunde projektet anpassas för att inte bli för stort. Alltså kunde överflödig eller tidskrävande funktionalitet direkt tas bort från planen istället för att behöva justera den allteftersom, vilket hade gjort den otydlig och oanpassad.

Många förutsättningar för projektet låg även i själva arbetet. Att ha tydliga tidsramar för varje del ledde till att ingen tid gick till spillo. Designsteget fick tillräckligt med tid för att kunna producera en tydlig och väl genomtänkt mockup, vilket ledde till att i utvecklingssteget fanns det ingen tvekan om vad som skulle implementeras eller hur det skulle se ut. Sedan var även utvecklingssteget genomfört med en tydlig och smart arbetsprocess, vilket gjorde den förutbestämda tidsramen mer hanterbar.

Som tidigare nämnt var alla verktyg och teknologier som användes i projektet noggrant testade och utvalda utifrån deras förmåga att tillsammans skapa en produkt som skulle uppfylla projektets syfte och mål. Men trots noggrann förundersökning och planering går vissa händelser inte att förutse eller förhindra, en av dessa var IT-attacken Västtrafik utsattes för, vilket resulterade i att projektet behövde ersätta deras API med SL:s. Denna incident ledde till en betydande försening i projektets utveckling, eftersom det krävdes tid för att kontakta Västtrafik angående hur de skulle åtgärda problemet. När beskedet kom att det skulle ta dröja innan en lösning fanns tillgänglig behövde ytterligare tid läggas på att hitta och implementera en passande ersättning för den ursprungliga API:n.

5. Slutsatser

Med projektet nu avslutat kan jag konstatera att en välplanerad och tydlig planering, kombinerat med en strukturerad och effektiv arbetsprocess, ledde till en produkt som blev väldigt lik min ursprungliga vision, trots oförutsedda motgångar. Trots att inte alla mål för projektet uppnåddes, anser jag resultatet som en succé eftersom den grundläggande funktionaliteten finns på plats och produkten är fullt användbar, vilket uppfyller projektets huvudsakliga syfte.

Under arbetet med projektet fungerade det mesta mycket smidigt, vilket till stor del kan attribueras till den tid och omsorg jag ägnade åt att noggrant välja de teknologier och verktyg som passade bäst för uppgiften. Men oavsett hur mycket tid och omsorg som ges till planeringen kan man aldrig vara förberedd för allt och oförutsägbara händelser kommer alltid att ske. I det här projektet var det plötsliga bytet från Västtrafik till SL som var väldigt oväntat, och om projektet inte hade en deadline, så hade bytet troligen inte skett. Trots att SL:s API i många aspekter hade samma funktionalitet och struktur som Västtrafiks API, var det detaljerna och bristen på personlig kontakt med datan som gjorde bytet störande. I vissa fall var SL:s API mycket mindre smidig än Västtrafiks när det gällde att hämta viss information, såsom linjernas färger. Dessutom var det mindre stimulerande att utveckla ett reseplaneringsverktyg för Stockholmsområdet eftersom jag själv är baserad i Göteborg.

Om det här projektet endast hade varit ett fritidsprojekt utan någon koppling till min utbildning hade jag antagligen pausat utvecklingen i väntan på att Västtrafiks API hade blivit tillgänglig för mig igen. Men eftersom jag valde att genomföra det här projektet som en sista prövning på utbildningen, bestämde jag mig för att byta till SL:s API för att kunna utveckla en fullständig produkt inom den givna tidsramen. Om jag i framtiden arbetar med ett projekt som involverar en extern API, likt det här projektet, skulle jag redan i planeringen förbereda alternativ i förebyggande syfte, för att undvika att behöva pausa utvecklingen och leta efter ett passande alternativ då.

Den största saken jag kommer ta med mig från det här projektet är vikten av att ha en genomtänkt och tydlig planering. I tidigare projekt har jag medvetet ägnat mindre tid åt planering eftersom jag inte trott att det skulle göra någon större skillnad. Därför bestämde jag mig för att i det här projektet ifrågasätta det tankesättet och göra en ordentlig planering för att se om det skulle göra någon skillnad under arbetsprocessen och om det skulle påverka det slutgiltiga resultatet.

Jag är glad att kunna konstatera att resultatet av denna förändring var väldigt positiv. Tack vare att jag ägnade mer tid i början till att planera och strukturera upp projektet blev fördelarna snabbt tydliga. Utvecklingsprocessen blev betydligt smidigare och mindre stressig eftersom jag visste precis vad som skulle göras hela tiden och behövde därför inte spendera tid på vad som skulle komma härnäst. Dessutom kunde jag se en positiv ökning i kvaliteten på det slutgiltiga resultatet, även om inte alla mål för projektet kunde uppnås.

Eventuellt kommer jag att färdigställa projektet efter utbildningens avslut genom att lägga till den planerade funktionaliteten, så att projektets alla mål till sist uppnås. Det är dock osannolikt att jag fortsätter mer än så, eftersom trafikinformationen är relaterad till Stockholmsområdet vilket inte är användbart för mig.