Collecting, Optimization and Conversion Kit

En dynamisk webblösning med API-ändpunkter i Entity Framework Core

Slutrapport för Programvaruprojekt, 15 HP, VT-2023

Författare: Adam El Soudi, Franco Kovacevic-Gahne & Carl-Johan Johansson

Handledare: Martin Goblirsch

Sammanfattning

Rapporten beskriver utvecklingen av en dynamisk .NET-applikation för att arkivera data i en relationsdatabas. Inmatningen sker i form av kalkylark från Microsoft Excel och informationen kan sedan efterfrågas av externa programvaror via ett REST-baserat API. Genom rapporten ges också en översikt av vilka alternativa lösningar det finns och relaterad litteratur som bidragit med relevanta perspektiv och kunskap för problemklassen.

Nyckelord: Data wrangling, transformering av data, .NET, dynamisk webbapplikation, REST-baserat API

Abstract

This paper describes the development and design cycles of a dynamic web application in Entity Framework 6 for the purpose of wrangling, archiving and extracting data. The application reads worksheets from Excel, transforms the data and sorts it into a relational database from where it can later on be extracted through http requests via a RESTful API. The paper also gives a brief overview of existing solutions to similar problems and offers a review of available literature that is of interest to the problem domain.

Keywords: Data wrangling, data transformation, .NET, dynamic web application, RESTful API

Förord

Vi vill tacka Therese Hadland på Universitets- och högskolerådet för ett roligt uppdrag och för alla möten och material som legat till grund för arbetet. Vi vill uttrycka en särskild tacksamhet till Martin Goblirsch för en stadig handledning och utmärkta tips och feedback, utan vilka vi inte skulle lyckats navigera det här projektet.

Innehållsförteckning

1. Introduktion

Rapporten beskriver ett projekt som genomförts med syftet att bättre kunna lagra och transportera data för ett handläggningssystem som är i behov av att visualisera statistik. Kapitel 1 definierar uppdragsgivaren, det praktiska uppdraget, och de avgränsningar som gjorts för att kunna producera en lösning.

1.1 Uppdragsgivare

Projektet genomfördes på uppdrag av Universitets- och högskolerådet (UHR) i Visby.

UHR är en statlig myndighet som har flera uppdrag inom utbildningssektorn. Dessa uppdrag har gemensamt att de stödjer utveckling via utbildning och internationella utbyten. UHR ansvarar för handläggning och samordnad av högskoleutbildningar i Sverige, samt anmälningar till högskoleprovet, vilket verksamheten ansvarar för att ta fram. UHR jämför även utländska utbildningar och examensintyg med dess svenska motsvarigheter; detta för att förenkla för högskole- och arbetssökande till Sverige som har utländska utbildningar/examen.

Avdelningen Internationellt Samarbete inom UHR hjälper skolor och organisationer att genomföra internationella utbytesprogram och resor via utlysningar som de ansvarar för. Organisationer kan ansöka om bidrag för att genomföra ett projekt och blir beviljade pengar om deras ansökan uppfyller kriterierna samt blir godkända för att få ekonomiskt stöd. Det är denna avdelning som agerat uppdragsgivare.

RAUK är ett internt utvecklat handläggningssystem i form av en webbapplikation som UHR använder för att hantera ansökningar. Applikationen består av en användardel och ett handläggningssystem. Handläggare på myndigheten kan logga in och bedöma ansökningar och läsa rapporter som skickats in för avslutade projekt där UHR betalat ut bidrag. Dessa rapporter beskriver bland annat hur många som deltog i projekten i slutändan och om det finns bidragspengar kvar som inte spenderades.

1.2 Praktiskt uppdrag och breddning

Den efterfrågade lösningen skulle göra det lättare för handläggare att hämta data för att med den som utgångspunkt kunna generera visuella representationer i form av stapeldiagram eller linjediagram, för att exempelvis kunna skapa en översikt av antalet män och kvinnor som deltar i de projekt som UHR betalat ut bidrag till. Eftersom tillgång inte givits till varken RAUK eller UHR:s databas utvecklades en separat applikation för datainläsning och dataleverans.

Som underlag för projektet levererades totalt fyra dataset för tre olika UHR-program: Atlas Partnerskap, Atlas Praktik och MFS Stipendier. Dessa dataset innehöll information om gjorda ansökningar mellan 2015-2020 som skulle arkiveras. Ett nytt databasschema behövde skapas för att stoppa in datan och sedan göra den tillgänglig för frontend-applikationer.

En enkel frontend-lösning behövdes för att kunna ladda upp Excelfiler via fält på en webbsida och sedan automatiskt sortera in datan i en databas. Ett REST-baserat API behövde designas för leverans av data till visualiseringsverktyg. Lösningen skulle med andra ord innehålla tre delar:

- Ett webbaserat UI med filinläsning
- En relationsdatabas för att lagra den inlästa datan
- Ett API från vilket datan kan hämtas ut via specifika ändpunkter

1.3 Förkortningar och namnförtydliganden

I rapporten förekommer vissa tekniska termer, förkortningar och akronymer oftare än andra. För att underlätta för läsaren presenteras här i korthet en lista över dessa uttryck och deras inbördes definitioner:

- Structured Query Language (SQL) är ett frågespråk för att läsa och skriva information i relationsdatabaser med hjälp av datadefinitionsfrågor, även kallade queries. (Microsoft, 2023a)
- Entity Framework 6.0 (.NET) är ett ramverk för att skapa dynamiska webbapplikationer som använder sig av objekt-relationell mappning (ORM). (Microsoft, 2023b)
- MVC Core, alt. ASP.NET MVC eller .NET Core, är en treskiktad arkitektur inom Entity Framework där applikationens affärslogik separeras från dess gränssnitt. (Microsoft, 2023c)
- Relationsdatabas syftar till en databas där datarader lagras i tabeller med unika nycklar. Tabellerna har relationer till andra tabeller i samma databas genom olika bindningar (constraints). (Oracle, 2023)
- Universitets- och högskolerådet (UHR) är uppdragsgivaren för projektet (se 1.1 Introduktion).
- RAUK är applikationen som UHR använder för att registrera och handlägga ansökningar. Namnet är en akronym som står för Rättssäkert, Användarvänligt, Uppföljning, Kvalitet.
- ER-Diagram, eller Entity Relationship Diagram (ERD), är en modell som visar de relationer som existerar mellan olika typer av data som lagras i en relationsdatabas. Projektets ERD är ritat i Chen-notation. (Moss, 2012)
- Unified Modelling Language (UML) är ett notationsspråk som används för att modellera klasser och relationer mellan olika moduler i ett program. (Booch, Rumbaugh & Jacobson, 2005)
- Objekt-relationell mappning (ORM) är ett sätt att koppla samman databaser med programkod, där tabeller motsvaras av specifika klasser och kolumner av deras variabler. Informationen mappas mellan databas och program via objekten. (Microsoft, 2023e)
- En POCO-klass är en klassfil som motsvarar en tabell i en SQL-databas. POCO är en akronym som står för “Plain Old CLR Object”. (EntityFrameWork Tutorial)
- Ett API (Application Programming Interface) är protokoll och standarder som beskriver hur mjukvarukomponenter kan interagera med varandra för att utbyta data. Inom kontexten för den här rapporten åsyftas ett REST-baserat (Representational State Transfer) API som skapats för att kunna transportera lagrad data ut ur applikationen till externa anropare. (Ranga & Soni, 2019)

1.4 Avgränsningar

Ett specificerat krav var att denna data skulle struktureras upp och sorteras in i en SQL-databas. En datamodell för hur UHR:s dåvarande schema såg ut fanns som underlag för att skapa ett nytt schema (se extern bilaga 1).

Skicket och formatet på flera av kolumnerna i Excel-arken begränsade hur vi kunde lagra datan och efterfråga information. Eftersom SQL och .NET var ett uttryckt krav kunde inte NoSQL-baserade lösningar såsom MongoDb användas, vilka annars hade kunnat underlätta arbetet med datalagring och extraktion (MongoDb, 2023).

De tekniska avgränsningar som varit av relevans för uppdraget handlade om utvecklingsmiljöer och tekniker för leverans av de slutgiltiga leverablerna. För utvecklingen av applikationen användes Microsoft ASP.NET MVC Core som är en populär miljö för att skapa dynamiska

webbapplikationer. UHR själva använde .NET 6.0 på sina servrar och hade formulerat ett krav om att arbetet skulle ske inom samma ramverk.

Lösningen använde sig av Entity Framework-tillägget EPPlus för att importera datan från Excel-filer till applikationen. Detta begränsade de filtyper som kunde laddas upp till kalkylblad från Microsoft Excel.

En snabb analys av alternativa verktyg för utveckling gjordes, där bland annat JetBrains Rider undersöktes som en möjlig utvecklingsmiljö. För författaren som använde sig av Mac OS visade sig detta vara ett bättre alternativ, medan författarna som använde Windows valde att arbeta kvar i Visual Studio eftersom Rider saknade vissa inbyggda verktyg för att arbeta med migrationer och databaser i Entity FrameWork Core.

Versionshantering i projektet sköttes med hjälp av Git och Bitbucket, och eftersom valet av IDE inte påverkade eller försvårade den här aspekten av arbetet gjordes ingen ytterligare analys av alternativ programvara.

Leverans av leverablerna skedde via ett REST-baserat API som utgjorde den enda ingångspunkten i systemet genom vilken information kunde efterfrågas från databasen. Författarna gjorde bedömningen att frontend-applikationer inte borde ha tillgång till en större del av datan än vad som krävs för den statistikvisualisering som UHR efterfrågat.

En annan projektgrupp fick i uppdrag att parallellt utveckla en frontend-lösning för hur man visualiserar statistik i form av diagram, och hur datan ska visualiseras i slutändan är följaktligen inte någonting som har varit relevant för författarna.

1.5 Författarens eller delförfattarnas bidrag

Författandet av denna rapport har fördelats mellan gruppmedlemmarna. Eftersom alla tre även bidragit i samtliga aspekter i designstadiet har ingen ytterligare specificering av arbetsfördelningen gjorts.

2. Problematisering och forskningsanknytning

Kvalitetssäkring handlar om att säkerställa att kvaliteten i en produkt "är tillräckligt hög" (Gustavsson & Görling, 2019, s. 13). Denna något godtyckliga definition innebär dock att det är svårt att precisera exakt vad som menas innan en produkt faktiskt existerar. I stället blir kvaliteten en sorts efterkonstruktion, när man med efterklokhet kan reflektera över saker som man upplever saknas i informationssystemet. Gustavsson & Görling (2019) observerar att:

> "Anledningen till att många IT-satsningar i dag misslyckas är inte att det saknas teoretiska modeller och ramverk för hur vi borde agera. Anledningen är snarare att vi ofta har problem med att tillämpa dem på verkligheten med all den komplexitet som omger oss." (Gustavsson & Görling, 2019, s.13)

En kvalitetsbrist som handledaren på UHR upplevde i handläggningssystemet för RAUK var exempelvis bristen på statistiska sammanställningar för rapporter över avslutade projekt. Eftersom inbyggda lösningar saknades för att ta fram informationen behövde uppdragsgivaren gå alternativa vägar via extern programvara i form av Excel för att generera statistik som sedan kunde visualiseras manuellt i dessa program i form av diagram och modeller.

Det saknades viss dokumentation över systemets delar. Det fanns exempelvis en datamodell, men den visade sig vara komplicerad och något rörig, och hade förlängts på icke-optimala vis genom åren. Ett enklare sätt att lagra och uthämta den information som användes för statistikvisualisering efterfrågades därför.

2.1. Problemformulering

I dagens digitala samhälle har tillgänglighet och användning av data tagit en central plats. Data kan ge en värdefull insikt och påverka beslutsfattandet för en organisation. Därför är det viktigt att informationstillgångarna lagras på rätt sätt och är anpassade för organisationens syfte (Mandinach, 2012).

Avsaknad av snabb åtkomst till data kan på samma vis ha stora konsekvenser för en organisation. Det kan bland annat leda till förlorade möjligheter, ineffektiva arbetsprocesser, och att felaktiga beslut fattas på dåliga grunder eftersom viktig information inte kan bearbetas och analyseras. Detta kan leda till att organisationer inte kan utnyttja sina dataresurser till sin fördel **(**DeLone & McLean, 2004**).**

Dokumentationsbrist för informationssystem är ett annat stort problem som företag och myndigheter ofta brottas med i takt med att digitaliserade informationssystem tar allt större plats i deras organisationer (Chomal & Saine, 2014). Kod bör vara skriven på ett vis som gör den lätt att förlänga (Martin, 2000), men om dess funktionalitet inte är tydligt beskriven blir den ändå svår att bygga ut. Felaktig, inkomplett eller utdaterad dokumentation leder ofta till sämre mjukvarukvalitet (Chomal & Saine, 2014).

2.1.1 Huvudproblem

Det praktiska uppdragets huvudproblem är ett exempel på problemklasser som handlar om datalagring, dataåtkomst och datamanipulation, vilka kan summeras under paraplybegreppet data wrangling. Med data wrangling åsyftas en process som går ut på att samla in, välja ut och transformera data i något slags analytiskt syfte (Thurber, 2018). Officiell svensk terminologi saknas som ofta är fallet med anglifierad IT-lingo, men begreppet syftar till att man "brottas med" eller "kämpar mot" datan för att på så vis kunna göra den användbar. Författarna har tagit beslutet att från och med nu översätta begreppet till 'dataförädling' på svenska.

En myt bland utvecklare gör gällande att professionella dataanalytiker kan spendera 50-80% av sin tid på dataförädling (Piringer & Endel, 2015) vilket leder till att det inte finns mycket kvar

att spendera på utveckling, testning och experimentering (ett påstående som författarna av den här rapporten anser vara korrekt utifrån erfarenheten med uppgiften).

2.1.2 Delproblem

Utöver huvudproblemet identifierades ytterligare delproblem som relaterade till uppgiften:

- Leverans av data till externa applikationer via API
- Dokumentation och notationsmodeller för informationssystem

2.2. Liknande system och aktuella ramverk

För att få en bättre förståelse för hur transformationen av data skulle kunna ske gjordes vissa efterforskningar av existerande verktyg med liknande funktionalitet. Eftersom uppdraget var av en teknisk natur riktades fokuset mot system som andra utvecklare talade gott om på Stack Overflow, Youtube och programmeringsinriktade webbsidor snarare än programvara som kunde identifieras i akademiska forskningsartiklar:

- KNIME (Konstanz Information Miner), en open source-lösning för dataanalys (Knime, 2023). Några av kraven från UHR på applikationen som utvecklades var att den skulle samla in, bearbeta och presentera data för en frontend-applikation som sedan visualiserade den på en webbsida. KNIME gav insikt i hur dessa funktioner skulle kunna se ut på ett tekniskt plan.
- Datameer X, en plattform för datainsamling och datamanipulation. Användare kan direkt ladda upp data eller använda unika datalänkar för att lyfta in rådata och sedan bryta ner den i mindre delar (Datameer, 2023). Inledningsvis kändes det här som ett intressant system men visade sig senare vara skapat främst för Big Data-transformering.
- Apache Nifi är ett öppet källkodsramverk som ger användaren möjligheten att samla in data från olika källor för att sedan bearbeta och rensa innan det skickas vidare till front-end-applikationen. Rensning är en process att identifiera, korrigera eller ta bort felaktig, ofullständig, duplicerad eller irrelevant data (Apache, 2023).
- Talend Open Studio är likt Apache Nifi en plattform för dataintegration, datatransformation, och rensning av smutsig rådata (Talend, 2023).

2.3. Relaterad forskning och litteratur

För att identifiera lämpliga källor som kunde stödja arbetsprocessen användes sökmotorer, bibliotekstjänster och språkmodeller där sökningar gjordes efter nyckelord och fraser som relaterade till de identifierade problemklasserna. Även litteratur från tidigare kurser under programmet har lyfts in vid behov. Resultaten nedan är de som ansågs vara mest relevanta för uppgiftens natur och omfattning.

Piringer och Endel (2015) betonar att dataförädling ofta är en tråkig och långsam men samtidigt nödvändig aktivitet inom varje organisation där det sker någon form av dataanalys. De menar att även om verktygen och teknologin ständigt förändras så krävs det att organisationen ägnar tid åt att tolka och förbereda sina dataset för vidare användning. Mandinach (2012) tar på liknande vis upp exempel på hur beslut som utgår från data påverkar organisationer och företag.

Syftet med dataförädling är att höja kvaliteten på datan som ett system begagnar sig av (Piringer & Endel, 2015). Kvalitet i informationssystem är i gengäld ett begrepp som kan delas in i flera mindre delar (McConnell, 2004). En generell regel inom mjukvaruutveckling är att intern kvalitet förbättras genom kodutveckling (McConnell, 2004, s. 464). McConnells forskning identifierar flera interna egenskaper såsom underhållbarhet, flexibilitet och återanvändbarhet

som ansågs vara intressanta för lösningen utifrån de identifierade problemklasserna (se 2.1.1 och 2.1.2).

API:er (Application Programming Interface) är grundstenarna för kommunikation i moderna webbapplikationer (Fay, 2022). API låter tjänster kommunicera mellan varandra och utbyta information via frågor-och-svar-strukturer utan att någon mänsklig anropare behöver vara inblandad. De två huvudsakliga teknologier som existerade när det här projektet genomfördes var REST (Representational State Transfer) och SOAP (Simple Object Access Protocol). Även Friesen (2019) har skrivit om API:er och deras popularitet på webben.

Dokumentation i mjukvaruutveckling är ofta avgörande för att en applikation eller kodmiljö ska kunna förlängas och modifieras (Chomal & Saine, 2014). Sommerville (2001) beskriver fyra krav för bra dokumentering som beskriver mjukvara. Sommerville menar att för att uppfylla dessa krav krävs olika former av dokument, från informella arbetsdokument till användarmanualer. Booch, Rumbaugh & Jacobson (2005) beskriver modelleringsspråket UML som kan användas för att skapa delar av denna dokumentation.

Javascript används allt oftare i webblösningar för att göra dem mer dynamiska. Lindley (2013) förmedlar instruktioner och information om hur Document Object Model fungerar. Officiell dokumentation för .NET användes för att guida det övriga utvecklingsarbetet (Microsoft, 2023a - 2023e), men även Okur et al. (2014) var av intresse vad gällde asynkron programmering.

2.4. Implikationer för uppdraget

Matrisen nedan ger en översikt över de litteraturkällor som använts och deras kopplingar till uppgiftens lösning:

Litteraturkälla	Relevanta sökbegrepp	Koppling till projektet
Att arbeta med Systemutveckling (Gustavsson & Görling, 2019)	Systemutveckling, kvalitetssäkring, kvalitetsegenskaper	Boken ger en bra översikt av systemutveckling och teorier som knyter an till informationssystem. Rent praktiskt har den använts för att skapa en bättre förståelse för ämnet och arbetsprocesser, samt i inspirationssyfte.
Code Complete (McConnell, 2004)	Quality Assurance, internal and external system qualities, maintainability, flexibility, reusability	Boken definierar begrepp som interna och externa egenskaper i avseende på kvalitetssäkring och systemkvalitet. Arbetet har strävat efter att uppfylla de tre interna kvalitetegenskaper som nämns i 2.3, men också de aspekter som McConnell beskriver i kapitel 5: “Desirable characteristics of a design”.

Data Wrangling: Making data useful again (Piringer, H. & Endel, F., 2015)	Data wrangling, dirty data, data analysis	Piringer & Endel beskriver flera metoder och begrepp som arbetet med analys och förädling av data har utgått från.
API Features Individualizing of Web Services: REST and SOAP (Ranga, V. & Soni, A., 2019)	API, RESTful, SOAP, API design, stateless transfer	Artikeln beskriver REST och SOAP som koncept och har använts för beslutsfattande kring vilken sorts API-design applikationen skulle implementera.
Significance of Software Documentation in Software Development Process (Chomal & Saine, 2014)	software documentation, software development, poor system quality	Rapporten diskuterar relationerna mellan modeller, dokument, källkod och dokumentation och utgör tillsammans med Sommerville (2001) en grund för hur författarna av den här rapporten tänkt kring dokumentering av sitt arbete.
Software Documentation (Sommerville, 2001)	software documentation, documentation requirements	Sommervilles fyra krav för dokumentation av mjukvara låg till grund för den dokumentation som skapades under och efter arbetet.
DOM Enlightenment (Lindley, 2013)	-	Boken har delvis använts för att skapa förloppsindikatorn på filuppladdningssidan.
ASP.NET Documentation (Microsoft, 2023d)	ASP.NET, .NET, MVC Core	ASP.NET är det ramverk som används för applikationen. Den tekniska dokumentationen har använts direkt i utvecklingen för att skriva kod och strukturera applikationens olika lager.
Features and Technical Overview (EPPlus, 2023)	Excel extensions for .NET	EPPlus är ett gratistillägg (extension) i .NET som använts för att läsa in Excel-data i lösningen som skapats.

2.5. Förväntade kunskapsbidrag

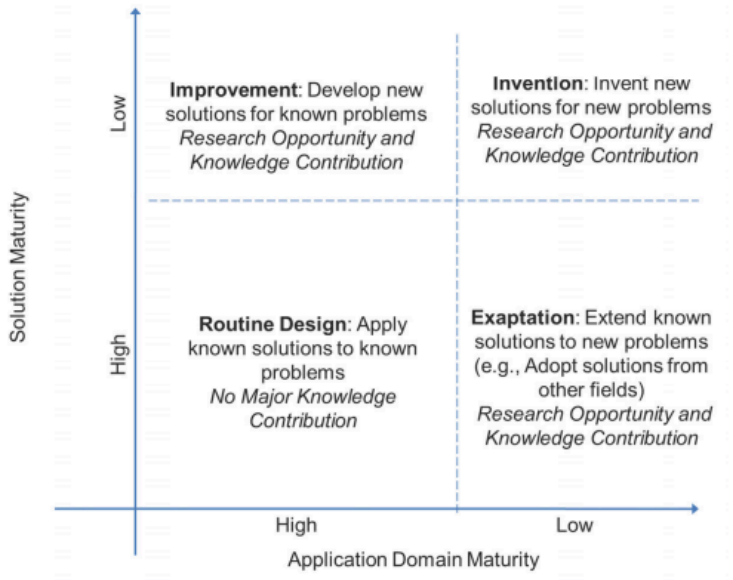
Kunskapsbidraget till UHR ligger i utformningen av applikationens funktioner. Lösningen har dokumenterats noggrant för framtida utvecklare i form av statiska, dynamiska och konceptuella

datamodeller, samt detaljerade beskrivningar av funktionalitet som relaterar till filinläsning och API-ändpunkter.

Den forskning som bedrivits är ett exempel på design science research, eller designforskning. I en vetenskaplig mening definieras designforskning som en research-baserad approach som samtidigt har en konkret designtillämpning (Carstensen & Bernhard, 2018, s.1)

Gregor & Hevner (2013, s. 346) skriver att målen med designforskning inom informationssystem även innefattar skapandet av sociotekniska lösningar som exempelvis beslutssystem, modelleringsverktyg och utvärderingsmodeller. Lösningarna används för att förbättra produkter, processer, tjänster eller teknologier på något vis. Gregor & Hevner (2013, s. 343) skriver också att ett fundamentalt problem med kunskapsbidrag från designforskning är att ingenting inom informationssystem egentligen är nytt. De menar att ett bättre sätt att bedöma bidraget är att utgå från projektets mognad, vilket i den aktuella kontexten kan översättas till hur mycket applicerbar forskning det redan finns, i två kategorier. De placerar lösningens mognad på en Y-axel och designdomänens mognad på en X-axel (se fig. 3).

Den designforskning som bedrivits i det genomförda projektet passar in i den övre vänstra halvan av den här matrisen, där en hög Application Domain Maturity möter en låg Solution Maturity. Projektet erbjöd en ny lösning för ett känt problem, relaterat till hur man hämtar data för att kunna visualisera diagram, och skapades inom ett känt ramverk (.NET Entity Framework 6.0) för dynamiska webbapplikationer. Kunskapsbidraget är främst värdefullt för kunden självt.



Figur 1: DSR Knowledge Contribution Framework (Gregor & Hevner, 2013, s. 345)

Förhoppningen är samtidigt att lösningen kan användas av framtida studier där andra utvecklare försöker bygga liknande artefakter. Ågerfalk (2014) nämner att empiriska bidrag behöver en teoretisk grund att utgå från, men att syftet med själva rapporten och utvecklingen av artefakten inte behöver vara några teoretiska bidrag av vikt:

> "If we choose to regard empirical contributions as being on a par with theoretical contributions rather than as subordinate (in a given study), then we can think of contributions as existing on a continuum – less emphasis on theoretical contributions provides space for more elaborate empirical contributions, and vice versa." (Ågerfalk (2014, s. 595)

3. Forsknings- och utvecklingsprocessen

I det här kapitlet beskrivs planeringar, metoder och forskningsdomäner som använts för att strukturera arbetet och tydliggöra kopplingen till akademisk forskning.

3.1. Projektöversikt

Under projektets gång hölls möten kontinuerligt med produktägarna på UHR för att hålla dem uppdaterade om projektet. Milstolpar, loggar, och arbetsmetoder hjälpte till att ytterligare strukturera arbetet. Syftet med dessa var att identifiera vilket jobb som behövde utföras och att säkerställa att det faktiskt utfördes inom tidsramarna för projektet.

Tabellen nedan visar en mer detaljerad översikt av planeringen. Den visar veckorna där de olika delarna av projektet har arbetats med, samt hur strukturen för arbetsprocessen sett ut:

Tabell. 1. En övergripande veckoplanering

Veckonummer	12	13	14	15	16	17	18	19	20	21	22
Sökning efter forskningslitteratur	■	■	■	■							
Möte med UHR	■		■		■		■			■	
Möte med Handledare			■		■		■			■	
Jobba med API							■	■	■		
Skriva på Rapporten			■	■	■	■		■		■	
Kanban-aktiviteter	■	■	■	■	■	■	■	■	■	■	■

Ett GANTT-schema skapades som innehöll en kalender kopplad till planeringen. Schemat uppdaterades kontinuerligt för att säkerställa att kalendern inte blev inaktuell. Pellerin och Perrier (2018) beskriver vikten av att ha en flexibel planering för ett projekt. De menar att även om de flesta forskare presenterar schemaläggning som en engångsföreteelse så är det sällan så i verkligheten (Pellerin & Perrier, 2018, s. 2161).

Milstolpar i kombination med en tydlig tidsplanering är ett sätt att säkerställa att ett projekt når sitt mål inom den fastställda tidsramen (Lindkvist, Söderlund, och Tell, 1998). För projektet gavs det kontinuerlig feedback i form av handledarmöten och kamratrespons via universitetet. Enligt Paasivaara och Lassenius (2003, s. 197) är synkronisering av milstolpar någonting som företag eller aktörer som samarbetar bör anamma tidigt i arbetsprocessen.

Under projektets gång fördes en intern logg där tankar och aktiviteter som rörde arbetet dokumenterades med tidsstämplar och namn. Goldkuhl (2019) noterar att loggning av aktiviteter är ett enkelt och snabbt sätt att dokumentera arbetet som gör arbetsprocessen mindre riskabel. Rees och Prando (2001) stärker detta påstående genom att fastställa att stark dokumentation kan bidra till att bespara en organisation stress, tidsbrist och andra potentiella problem. Att ha en stark dokumentation förebygger även potentiella juridiska konsekvenser som kan uppstå (Rees & Prando, 2001).

3.2. Metodstöd

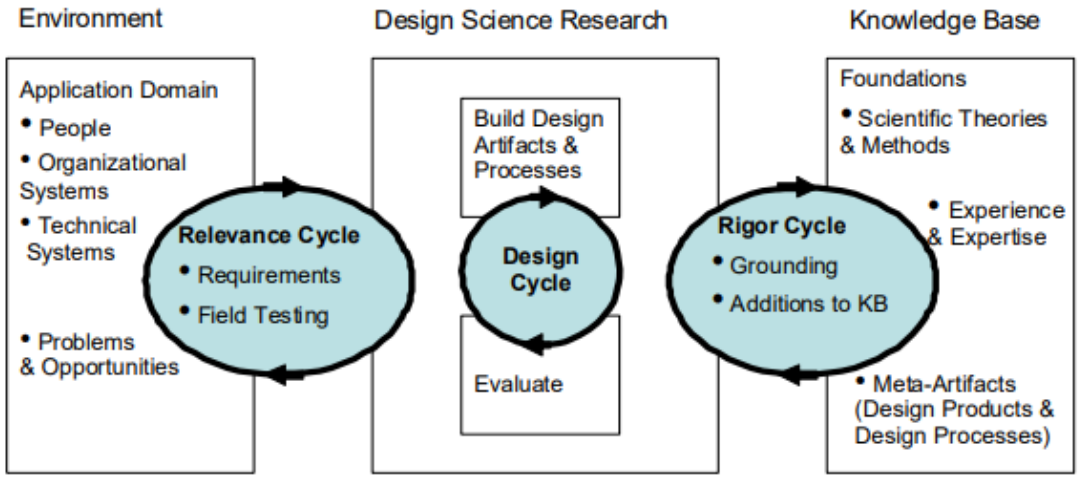
Nedan beskrivs de metoder som använts för att vägleda och utforma forsknings- och utvecklingsprocesserna i projektet.

3.2.1 Metodstöd för forskning

Förhållandet mellan design och efterforskning i design science research är någonting som Baskerville et al (2004) intresserat sig för. Deras publicering nämner tre huvudsakliga approacher inom området: Design With Research, där målet med att bygga någonting är för att producera en lösning och efterforskningen används mer som ett stöd; Research Into Designing, vars syfte är att skapa djupare akademisk förståelse för designprocesser; och Design As Research Methodology, som går ut på att skapa ny design som bidrar med bättre förståelse för forskningsdomänen (Baskerville et al, 2011, s. 8).

Författarna av den här rapporten anser att dessa kategorier är något grovkorniga och har dimensioner som då och då tycks överlappa varandra. Främst är det dock metodiken Design With Research som har efterföljts i arbetet. Baskerville et al (2011, s. 9) skriver att syftet med Design With Research vanligtvis är att bygga någonting för praktiska eller sociala ändamål eller också för att man vill fördjupa förståelsen för designen, vilket korrelerar till det huvudsakliga kunskapsbidraget för projektet som bedömdes vara en förbättring av någonting existerande (se 2.5 Förväntade kunskapsbidrag). Efterforskningar utförs inom denna metodik för att skapa en vetenskaplig grund för de principer som själva designen sedan baseras på.

Enligt Hevner (2007) kan designforskning delas in i tre cykler: en relevanscykel, en designcykel och en rigörcykel (se fig. 2 nedan).



Figur 2. "Design Science Research Cycles" (Hevner, 2007, s. 2)

Relevanscykeln initierar enligt Hevner (2007, s. 3) designforskningen genom en kontext som definierar premissen för projektet, dess tekniska specifikationer och vilka problem som behöver lösas. Resultatet av designforskningen returneras till kontexten där den kan utvärderas och studeras.

Rigörcykeln bidrar enligt Hevner (2007, s. 4) med etablerad kunskap. Det är upp till efterforskarna själva att göra grundliga efterforskningar av den etablerade kunskapsbasen för att på så vis kunna garantera att designen som produceras innehåller någon slags forskningsbidrag.

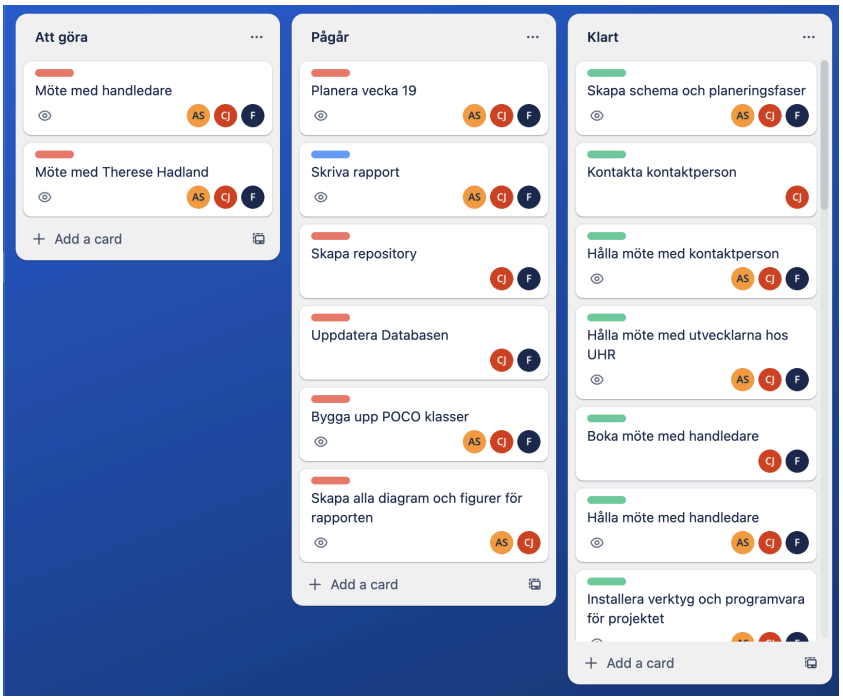
Designcykeln definierar Hevner (2007, s. 4) som hjärtpunkten i projektet. Den itererar mellan skapandet av en artefakt, dess utvärdering, och feedbacken på den för att på så vis kunna förfina designen.

3.2.2 Metodstöd för utveckling

Under projektets gång har Kanban tillämpats som arbetsmetod. Kanban är ett agilt sätt att arbeta med mjukvaruutveckling som definierar ett arbetsflöde i tre steg: Att göra, Pågår och Klart (Radigan, 2022).

Kanban är flexibelt och lämpar sig för både stora och små team (Kanban, 2023). Arbetet är uppdelat i objekt. Dessa objekt kan representeras av exempelvis post-it-liknande lappar på en virtuell anslagstavla som Trello (Trello, 2023). Dessa lappar innehåller information om vem som ansvarar för arbetsobjektet som lappen refererar till. De innehåller också information om det specifika arbetsobjektet, en beskrivning av det pågående arbetet, samt dess förväntade längd.

Filosofin bakom Kanban är att prioritera arbetsobjekt baserat på deras värde; det vill säga, de arbetsuppgifter som är viktigast. Prioriteringen av arbetsuppgifter sker genom att visualisera arbetsflödet och begränsa mängden arbete som sker samtidigt. Detta gör det lättare att identifiera och åtgärda flaskhalsar i arbetet.



Figur 3: Trello-board med lappar.

3.3. Dataanalys

UHR har flera olika program där folk och organisationer kan ansöka om bidrag. Av dessa program förmedlade produktägaren ansöknings- och rapportdata från tre stycken som relaterade till avslutade utlysningar.

Piringer och Endel (2015. s. 112) beskriver “dirty data”, förorenad data, som data där det trots utförlig dataförädling inte alltid är tydligt hur den ska kunna sorteras och användas. Eftersom rådatan som låg till grund för projektet kom från Excelfiler och i flera fall saknade identifierande nyckelvärden, relationer och enhetliga datatyper blev det svårt att både behålla

relationerna mellan datavärdena och att skapa en databas som uppnådde tredje normalformen, vilken kännetecknar en god struktur för relationsdatabaser (Padron-McCarthy & Risch, 2018).

Diarienummer från tidigare ansökan
År1:200226840 och År2:200228009
398-2014
290-2014
200228656
15132013
403-2014
200228232
397-2014
IPK2012:1645

Figur 4: Exempel på förorenad data som gjorde det svårt att skapa logiska relationer i en databas.

Processen bakom dataförädling handlar om att samla in ostrukturerad data och förbereda den för visualisering, modellering eller permanent lagring (Piringer & Endel, 2015). Samtliga tre aspekter knöt an till målen för lösningen. En kompromiss angående datalagringen fick göras som utgick från datasetens storlek och beskaffenhet (se 3.5.1 Databas).

3.4. Forskningetik och sekretess

Under arbetet med projektet har vi förhållit oss till Vetenskapsrådets forskningsetiska principer (Vetenskapsrådet, 2017). Vi har säkerställt att vi uppfyller dessa etiska krav i avseende på:

- Transparens & Ärlighet:
  - Källor och bidrag har krediterats, och vi har inte plagierat något tidigare arbete. Arbetet har genomförts på ett transparent och ärligt vis.
- Öppenhet & Tillgänglighet:
  - Genom att dokumentera hur arbetet under projektet utförts har det gjorts möjligt för andra att granska resultatet.
- Objektivitet:
  - Ingen påverkan av egenintressen har skett i våra beslut inom projektarbetet, vilket hjälper att hålla en objektiv inställning till uppdraget.
- Ansvar:
  - Vi har tagit ansvar för vårt arbete och respekterat lagar, regler och riktlinjer som ställts av uppdragsgivaren och forskningsområdet..
- Respekt:
  - Hänsyn till de konsekvenser som kan uppstå från vårt arbete och respekt för de inblandade personer och organisationer som kan påverkas.

Datan som använts för projektet innehåller personuppgifter, och dessa kommer att lagras i UHR:s system enligt de lagar som gäller. I övrigt har inga särskilda åtgärder vidtagits vid hantering av dataseten eftersom UHR är en myndighet och allt material som rör ansökningar i deras system därmed är offentligt. Alla kopior av dataseten raderades efter genomförande av projektet.

3.5. Antaganden

Ett antal antaganden om lösningens tekniska beskaffenheter gjordes utifrån formulerade krav och beskrivningar. Leverans av leverabler via API var ett sådant, grundat i faktumet att applikationen utvecklades separat från RAUK självt och använde en annan databas. Andra antaganden gjordes kring lösningens flexibilitet baserat på information om att dataseten antingen innehöll rapporter för slutförda uppdrag eller ansökningar som fått avslag. Ett exempel på ett sådant antagande är att det inte skulle tillkomma eller redigeras information i dessa dataset. Ett relaterat antagande gjordes även baserat på reflektionen att lösningen borde fungera för att arkivera även framtida rapporter, och därför behövde en dynamisk design för att kunna hantera lätta modifieringar av schemat för applikationens databas.

3.5.1 Databas

Eftersom detaljerad information saknades om hur scheman för tabellerna i databasen skulle se ut, samt vilka tabeller som över huvud taget behövdes, gjordes ett antal antaganden baserade på analysen av den data som fanns i Excel-underlaget:

- Eget databasschema, vilket gjorde det lättare och mer effektivt att hantera datan.
- En gemensam databas skapades för alla program för att förenkla hanteringen och delningen av data mellan dem.. Bedömningen gjordes att det skulle vara mer fördelaktigt för lösningen.
- Gemensamma tabeller för alla UHR-program, för att förenkla statistikframtagning och datajämförelser mellan olika program.
- Brott mot tredje normalformen ansågs acceptabla eftersom datan som lagrades relaterade till avslutade uppdrag och inte skulle komma att förändras. Dubbellagrad data på vissa ställen utgjorde därför inget hot mot datans beständighet.

4. Designresultat

Lösningen är skapad i Entity Framework Core och programspråket C#, med vissa css- och html-element som appliceras på de visuella frontend-aspekterna av programmet. Även Javascript och Bootstrap används i dessa delar.

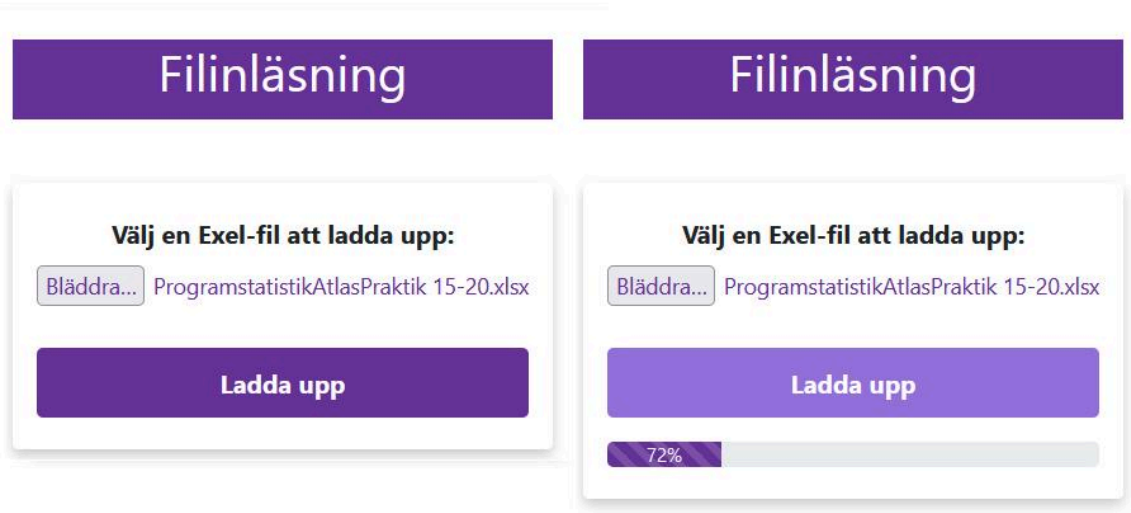
Lösningen låter en användare läsa in en Excelfil från en webbsida (se 4.1). Koden extraherar och sorterar datan från Excelfilen genom objekt-relationell mappning (se 4.2) och sparar sedan datan till lämpliga tabeller i en relationsdatabas (se 4.3). Kommunikationen med själva applikationen för att hämta och leverera data sker via ett REST-baserat API som designades specifikt för detta ändamål (se 4.4).

4.1. Filinläsning

Lösningen har ett enkelt webbaserat användargränssnitt för filuppladdning. Sidan skapades i första hand för att underlätta testningen av filuppladdningen men fungerar även som en ingångspunkt om man vill använda lösningen för arkivering bortom det material som legat till grund för projektet.

Färgerna som används följer UHR:s färgtema för RAUK. Utsikten består av tre huvuddelar. Det finns en knapp som låter användaren välja den Excel-fil som ska laddas upp. Koden validerar också att filen är i .xml eller .xmls-format innan inläsningen initieras för att säkerställa att inget ovidkommande laddas upp till databasen.

Ytterligare en knapp existerar som startar inläsningsproceduren för den valda Excel-filen. Slutligen finns det en förloppsindikator som visualiserar inläsningsprocessen för att ge en indikation om hur mycket tid som återstår.



Figur 5: Filinläsning via webbgränssnitt.

4.1.1 Bootstrap, JQuery och Document Object Model

Förloppsindikatorn (se högra bilden i fig. 5) är byggd med två inlänkade Javascript-bibliotek, JQuery och Bootstrap, som hostas av Google respektive Stackpath. Dessa två bibliotek länkas in via script-taggar i vyn för filinläsningssidan. Bootstrap använder sig även av en extern css-fil från Stackpath.

Javascript-koden använder sig av DOM (document object model), ett programmeringsinterface för webbapplikationer som representerar strukturelement på en webbsida i form av objekt i ett

hierarkiskt träd (Lindley, 2013). Med hjälp av DOM adderas en eventhanterare till form-elementet för filinläsningen. När användaren trycker på knappen märkt ‘Ladda upp’ aktiveras eventhanteraren. Metainformation om filen såsom dess namn, storlek och typ samlas in och skickas sedan som en så kallad AJAX Request till den JQuery-modul som länkats in på klientsidan.

Ett XMLHttpRequest-objekt skapas därefter som lägger på en eventlyssnare som följer filens uppladdningsstatus. När filuppladdningen är färdig avfyras den och ett meddelande dyker upp som bekräftar att uppladdningen lyckades.

Förloppsindikatorn representerar med andra ord en grov uppskattning av hur lång tid filen borde ta att ladda upp och uppdateras inte i realtid relativt till faktiska överföringshastigheter. Den ger ändå en indikation om att någonting händer i applikationen under filinläsningen, utan vilken användaren lätt kan bli förvirrad och ledas till att tro att processen misslyckats.

4.2. Dataformatering

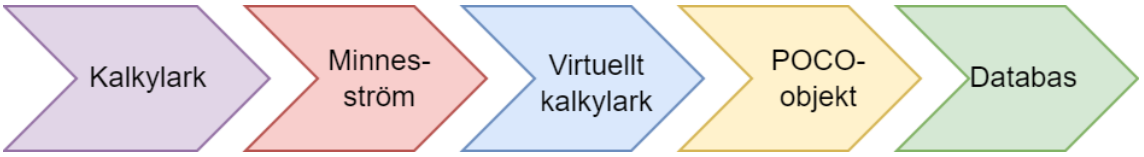
Filinläsningprocessen inleds efter att användaren laddat upp en Excel-fil via uppladdningssidan. Den uppladdade Excel-filen bearbetas och importeras till applikationens POCO-klasser. Den representerar en automatiserad dataförädlingsprocess där koden genom sin generaliserade natur försöker förebygga de problem som existerar i de otvättade dataseten (Piringer & Endel, 2015).

Filinläsningsmetoden är asynkront programmerad och körs på en separat tråd (Okur, et al., 2014). Eftersom filen inte ska sparas till servern öppnas i stället en minnesström och Excel-filen kopieras över till strömmen för att kunna bearbetas.

4.2.1 EPPlus

För att kunna läsa datan i Excel-filerna används EPPlus; ett tillägg för .NET-applikationer som innehåller färdiga bibliotek för konvertering av Excel-kolumner till datastrukturer som programspråk som C# kan förstå och använda. (EPPlus, 2023)

EPPlus underlättar arbetet med dataförädling genom att lösa problemet med att konvertera filer mellan olika format. Datan från minnesströmmen sparas ner i ett virtuellt Excel-ark i stället och programmet itererar sedan genom detta rad för rad.

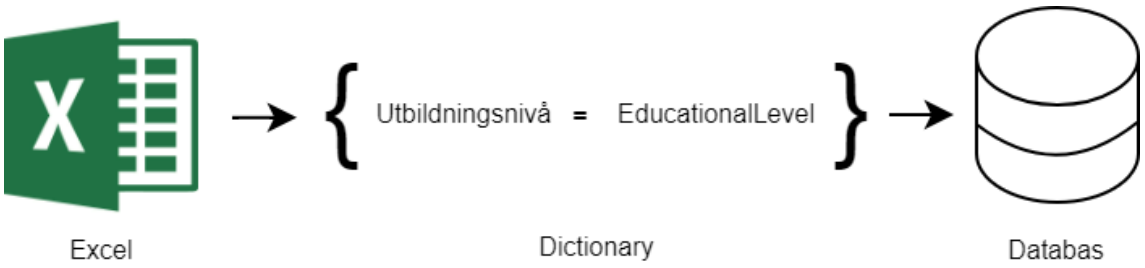


Figur 6: Kedjan visar hur informationen vandrar mellan olika datastrukturer innan den slutligen når databasen.

4.2.2. Objekt-Relationell Mappning

Den här filinläsningsprocessen är ett exempel på objekt-relationell mappning (ORM) där kolumner och celler i en databas motsvaras av klasser och fält i programkod. (Microsoft, 2023e)

Klasserna, så kallade POCO-klasser (EntityFrameWork Tutorial), instansieras till objekt vilka datan sorteras in i för att sedan skrivas till databasen vid något tillfälle.



Figur 7: Dictionary-klass används som ett uppslagsverk för att matcha namn på kolumner i Excel mot motsvarande kolumner i en databas.

Genom att matcha Excel-kolumnerna mot databasen kan programmet bestämma under körtid vilken typ av objekt som datan ska skrivas till. Det här är en utgångspunkt för att skrivandet av data till databasen ska kunna ske i lösningen eftersom koden inte vet på förhand vilken tabell (och därmed inte heller vilken motsvarande POCO-klassinstans) datan ska sparas i. Det innebär också att lösningen blir mer flexibel och återanvändbar (McConnell, 2004) (se 5.2 Tolkande utvärdering).

Koden använder sig även av koncept här som är relaterade till Reflection i C#; en komplex programmeringsteknik som gör det lättare att generalisera kod. Semantiskt syftar ordet till "spegling". Reflektion i C# innebär att man kan manipulera data för typer, objekt och metoder under körtid. (Microsoft, 2021). Det finns vissa kontroversiella aspekter i detta (se 5.2.1 Tolkande utvärdering).

Efter varje rad i Excel-arket bearbetats läggs alla objekt som inte är tomma till i listor som initialiserades i början av filinläsningen. När alla rader i Excel-arket itererats genom sparas dessa listor med objekt till de tabeller i databasen som motsvarar POCO-klasserna.

4.3. API

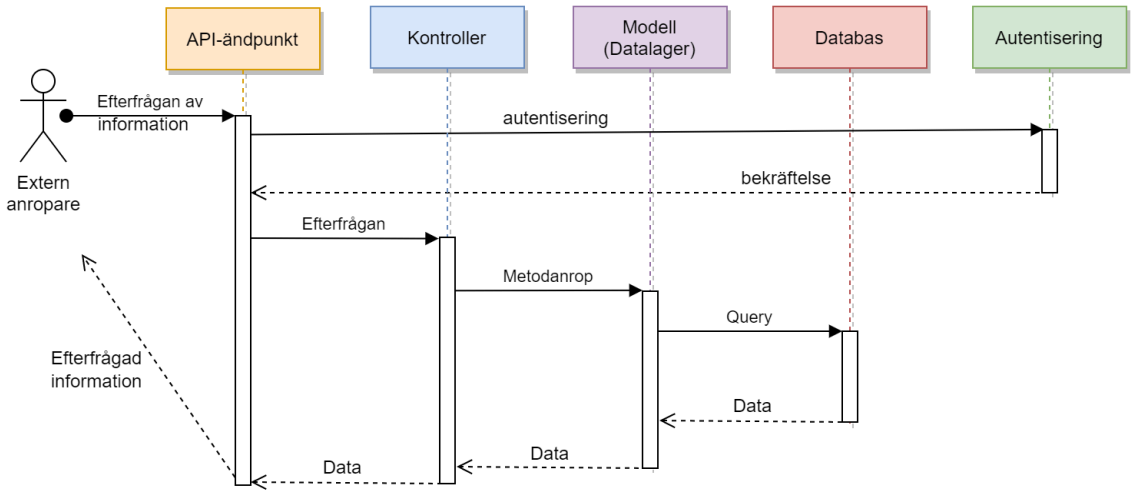
Efterfrågningar och utlämningar av data sker via ett REST-baserat API. REST står för Representational State Transfer och beskriver en arkitektur där kommunikation sker mellan webbtjänster; så kallad M2M (maskin-till-maskin). Det är ett snabbt och effektivt sätt att transportera data och hantera förfrågningar på (Ranga & Soni, 2019) som bygger på principen om att all data inte behöver finnas på ett och samma ställe. Även monolitiska lösningar kan med fördel använda sig av API-kommunikation för att uppnå en högre flexibilitet i systemet och för att definiera tydligare ansvarsområden för olika komponenter.

API:et används för att hantera tillståndslös kommunikation via http-protokoll i lösningen (Ranga & Soni, 2019). Med tillståndslös menas inom programmering att en applikation inte sparar data som genereras under en session för att använda på nytt i en annan. Varje anrop är nytt och genererar ny information (Ranga & Soni, 2019).

API:et är webbaserat och har en ändpunkt som specificerats för anroparen på förhand. Genom att en hämtningsförfrågan skickas till ändpunkten aktiveras olika processer i applikationen som resulterar i ett dataflöde genom dess olika skikt.

Datan returneras i form av ett JSON-objekt (JavaScript Object Notation), vilket är det mest populära formatet för webbaserade API-tjänster (Friesen, 2019). I praktiken innebär det att datan ligger lagrad som nyckel-värde-par inuti ett omslutande objekt vilket gör den lätt att sortera och organisera.

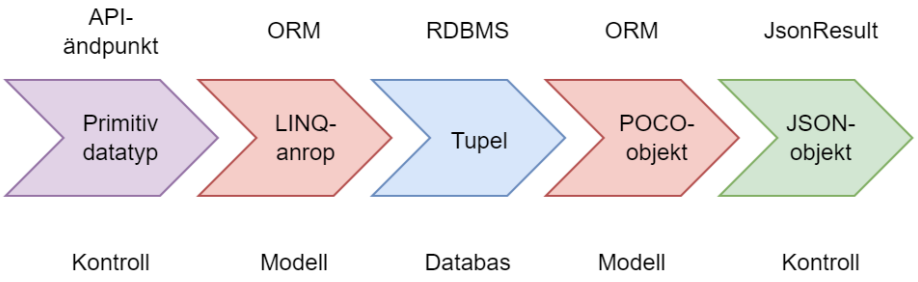
Vid ett API-anrop kan informationsflödet i applikationen illustreras på följande vis (se nästa sida):



Figur 8: Sekvensdiagram för API-anrop.

API-ändpunkten definieras av en webblänk till applikationens adress följt av en /api-ändelse. Via ändpunkten kan ett anrop slussas vidare till en av flera metoder i kontrollerskiktet av applikationen som i sin tur anropar logik från datalagret för att formulera en queryfråga som hämtar ut data ur relationsdatabasen. Queryfrågan använder sig av LINQ (Language Integrated Query) för att precisera vilka tabeller och kolumner datan ska hämtas från.

ASP.NET Core innehåller en fördefinierad klass döpt till JsonResult som på automatisk väg konverterar instansobjekt till JSON-objekt, vilket innebär att ingen extra tid behöver läggas på att paketera datan inför transport.



Figur 9: Olika moduler och lager som samverkar vid ett API-anrop.

Nedan följer en lista på de API-ändpunkter som är definierade i systemet, deras anropsadresser, och vilka argument de accepterar:

Tabell 2: API-ändpunkter

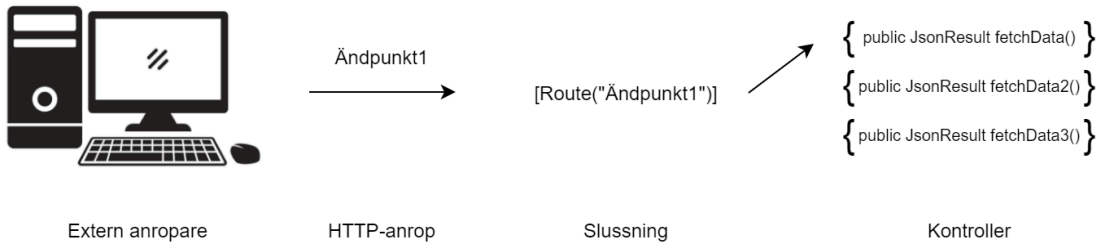
Ändpunkt	**Gör**	**Parametrar**	**Exempelanrop**
GetDnrAtlasPartnerskap(dnr)	Returnerar utvald data för en ansökan med specificerat dnr-nummer för programmet Atlas	dnr: En sträng med det efterfrågade dnr-numret	`http://{hostingadress}/api/GetDnrAtlasPartnerskap?dnr={dnr-numre`

	Partnerskap		t}
GetPeriodAtlasPartnerskap(fromPeriod, toPeriod)	Returnerar utvald data för alla ansökningar som gjorts inom ett visst spann för programmet Atlas Partnerskap	fromPeriod: Startperioden toPeriod: Slutperioden	http://{hostingadress}/api/GetDnrAtlasPartnerskap?fromPeriod={startperioden}&toPeriod={slutperioden}
GetDnrAtlasPraktik(dnr)	Returnerar utvald data för en ansökan med specificerat dnr-nummer för programmet Atlas Praktik	dnr: En sträng med det efterfrågade dnr-numret	http://{hostingadress}/api/GetDnrAtlasPraktik?dnr={dnr-nummer}
GetPeriodAtlasPraktik(fromPeriod, toPeriod)	Returnerar utvald data för alla ansökningar som gjorts inom ett visst spann för programmet Atlas Praktik	fromPeriod: Startperioden toPeriod: Slutperioden	http://{hostingadress}/api/GetDnrAtlasPraktik?fromPeriod={startperioden}&toPeriod={slutperioden}
GetDnrMFSStipendier(dnr)	Returnerar utvald data för en specifik ansökning som gjorts inom ett visst spann för programmet MFS Stipendier	dnr: En sträng med det efterfrågade dnr-numret	http://{hostingadress}/api/GetDnrMFSStipendier?dnr={dnr-nummer}
GetPeriodMFSStipendier(fromPeriod, toPeriod)	Returnerar utvald data för alla ansökningar som gjorts inom ett visst spann för programmet Atlas Praktik	fromPeriod: Startperioden toPeriod: Slutperioden	http://{hostingadress}/api/GetPeriodMFSStipendier?fromPeriod={startperioden}&toPeriod={slutperioden}

Eftersom databasen inte är menad att modifieras efter att informationen arkiverats finns det inga API-ändpunkter som möjliggör andra former av CRUD-operationer (Copy, Read, Update, Delete) än läsning.

.NET stödjer möjligheten att skapa REST-baserade API:n genom att definiera kontrollerklasser som ärver från basklassen ControllerBase i stället för Controller. ControllerBase definierar en kontroller utan någon Vy-support (Microsoft, 2023e) som aktiveras och expedieras per anrop.

Kontrollerklassen fungerar som en normal kontroller i övrigt och kan ta in både en databaskontext och ett repository i sin konstruktor för att utföra operationer mot datalagret. Själva API-ändpunkterna som anropas i URL:en definieras av en Route-notation kopplad till den metod som ska exekveras:



Figur 10: Kontrollerlogiken använder notationer ('Route') för att slussa webbanrop till rätt metod.

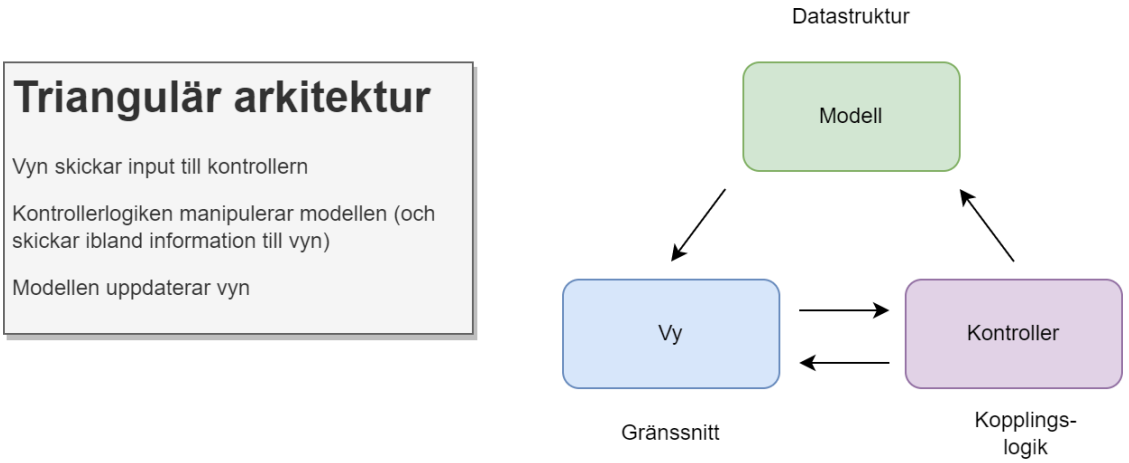
API-leveransen är, precis som kodgeneraliseringen, en aspekt som är designad för att förbättra de interna kvalitetsegenskaper som McConnell (2004) menar påverkar systemkvaliteten (för diskussion se 5.2 Tolkande utvärdering).

4.4. Konceptuella datamodeller

En konceptuell datamodell har en viktig roll i en utvecklingsprocess av databasapplikationer. Den ger en övergripande teknisk bild över dataflöden och relationer mellan olika entiteter och dess attribut. Modellerna är fristående; det vill säga, de är inte bundna till specifika teknologier, programmeringsspråk, databasstrukturer eller en särskild applikation. De fungerar som en ritning för systemet och används för att modellera och förstå hur data relaterar till varandra på en hög nivå, och är följaktligen användbara för mycket av den systemdokumentation som Sommerville (2001) beskriver som exempel på god dokumentation.

4.4.1 Triangulär arkitektur

Med tanke på den teknologiska stacken som används för detta uppdrag är det relevant att nämna MVC (Model View Controller) och treskiktad arkitektur. I MVC-projekt separeras applikationen i tre komponenter. Vissa typer av treskiktad arkitektur är av linjär art där lager åtföljer lager i en hierarki. MVC-arkitekturen är i stället triangulär:



Figur 11: MVC-arkitektur.

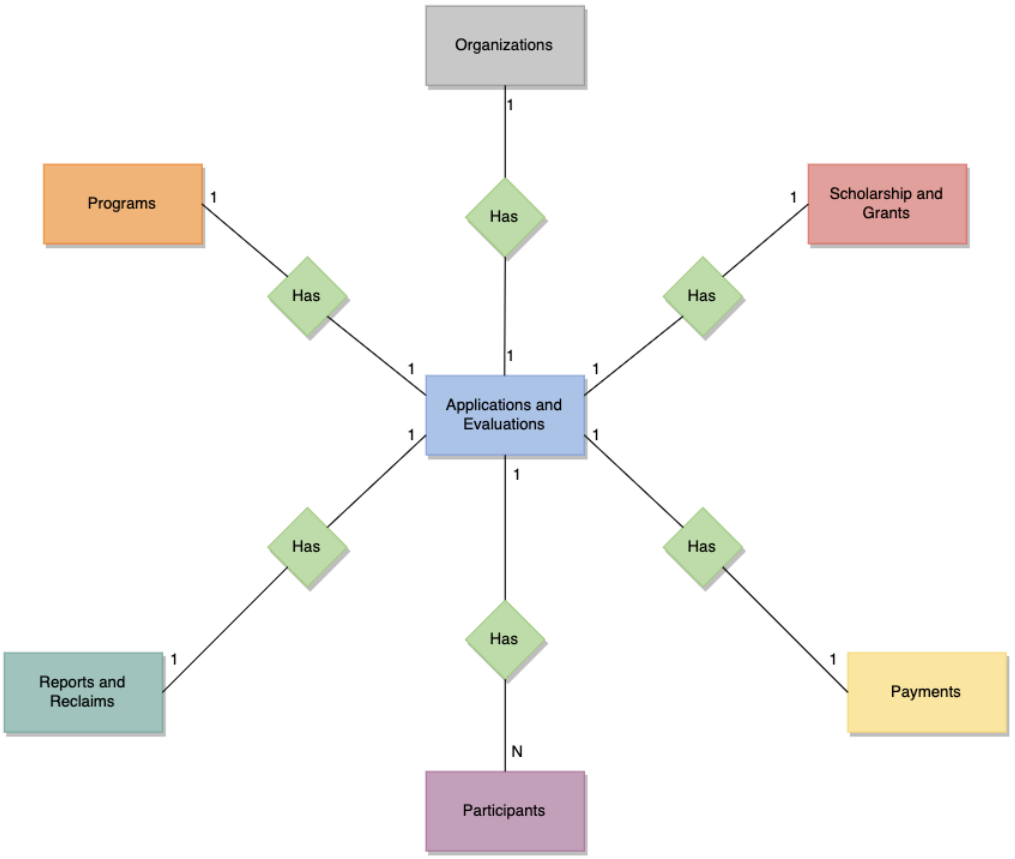
De tre lagren har olika ansvarsområden i applikationen:

- Modellen ansvarar för den underliggande datastrukturen i en applikation. Innehåller oftast ren data och funktioner för att uppdatera, ändra och hämta data till/från/i databasen.
- Vyn ansvarar för att visuellt presentera data för användaren. Den innehåller logiken för att kunna presentera data för användaren på ett begripligt sätt.
- Kontrollen agerar som en koppling mellan modellen och vyn. Den tar emot användarens interaktion med användargränssnittet från vyn, vilket kan vara att begära uppdateringar från modellen, för att sedan returnera uppdaterad data till vyn.

Genom att använda MVC-arkitekturen skapas mer robusta och skalbara applikationer som är lättare att underhålla (McConnell, 2004). I projektet finns det bara en vy: den som definierar uppladdningssidan för filinläsningen (se 4.1). Den styrs av en HomeController som innehåller logik för att manipulera vyn. HomeController anropar metoder i Repository-klassen i Modellagret (se 4.5.2); en implementering av IRepository som innehåller exekveringsinstruktioner för all kod som är kopplad till CRUD-operationer mot databasen.

4.4.2 ERD

Ett ERD hjälper till att förtydliga och illustrera uppbyggnaden av det rationella SQL-databasschemat för projektet. Avsikten är att ge en tydlig bild av hur de olika tabellerna interagerar och samverkar, vilket gör databasschemat lättare att läsa och förstå. ER-diagrammet



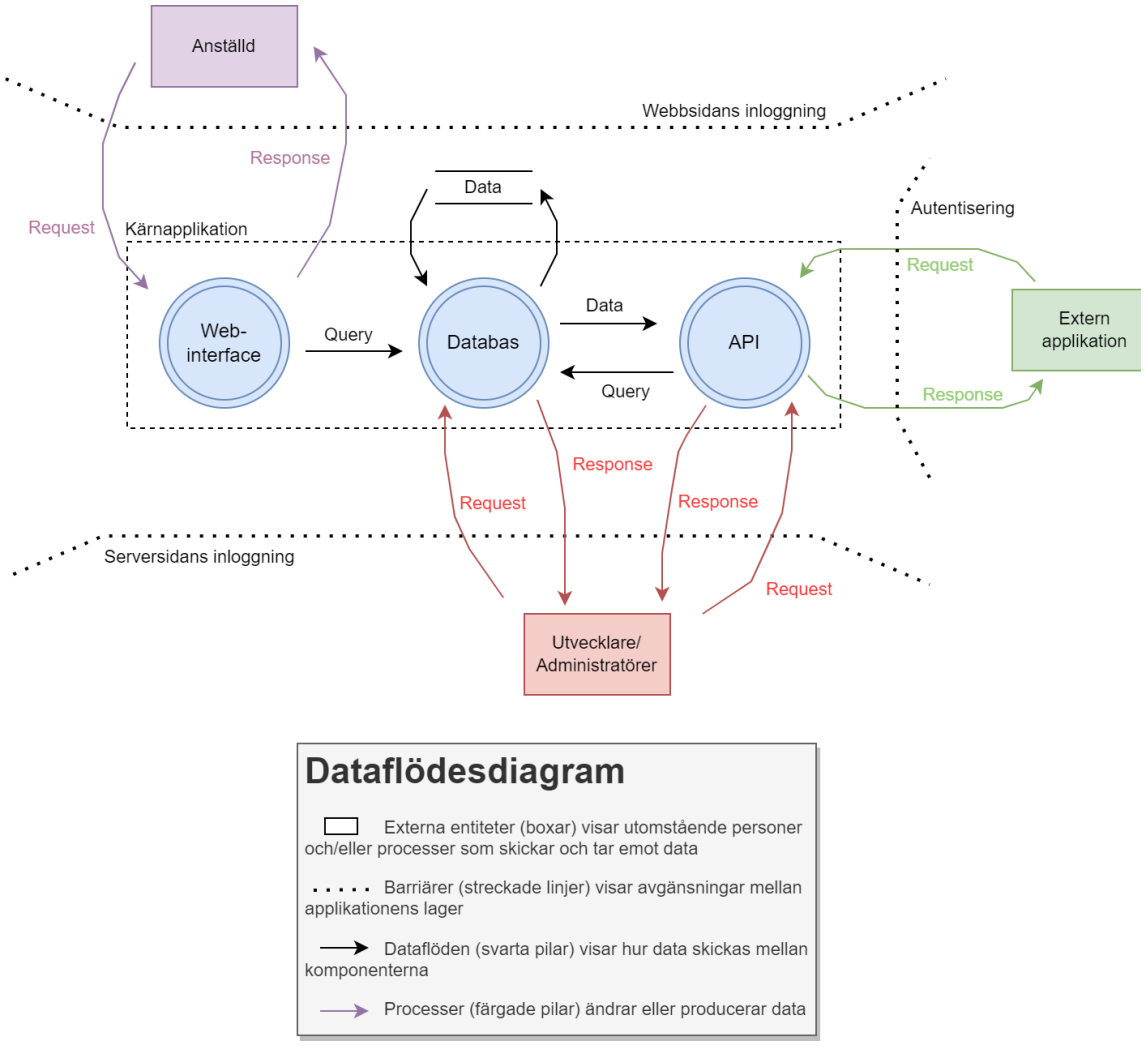
Figur 12: ER-diagram för databasen i applikationen.

är ritat i Chen-notation (Rodina, 2023) men har uteslutit attributen för att kunna ge en bättre överblick. Diagrammet representerar den databas som skapats för den aktuella lösningen och inte den som UHR använder sig av i sin nuvarande miljö.

4.4.3 Dataflöden

Ett dataflödesdiagram är en dynamisk datamodell som illustrerar informationsflöden mellan komponenterna i ett system. Det använder olika notationer för att beskriva kopplingar, in- och utmatningar av data, och de skyddsbarriärer som finns (exempelvis inloggning). Dessa skyddsbarriärer är inte implementerade i lösningen eftersom ingen tillgång har givits till UHR:s inloggningssystem.

Applikationen innehåller följande flöden (se nästa sida):



Figur 13. Dataflödesdiagram för lösningen.

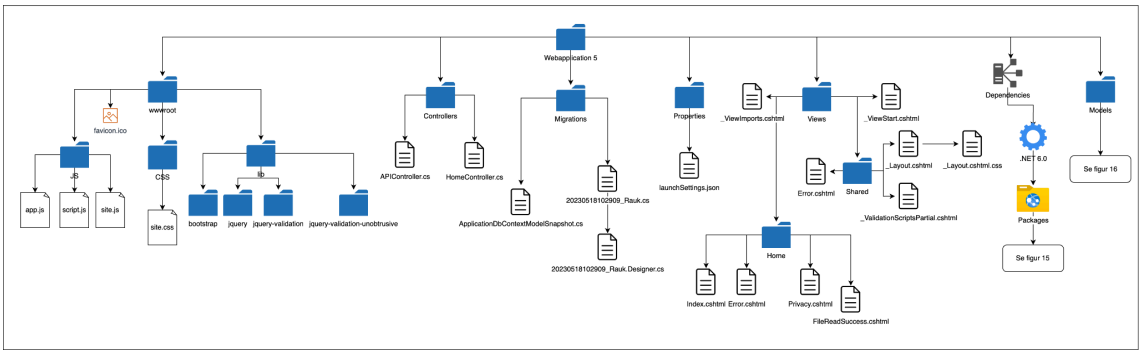
Användare som vill ladda upp filer bör först verifiera sin identitet genom autentisering. I applikationen existerar inte denna avgränsning eftersom vi inte haft tillgång till UHR:s inloggningssystem. Inloggningssidan slussar vidare användaren till en sida där denne kan utföra en request genom att ladda upp en datafil (se 4.1). Responsen som levereras är i form av ett meddelande som informerar om huruvida filinläsningen lyckats.

Filen som skickas till webbservern extraheras till en dataström. Applikationen sorterar in datan från kolumnerna i objekt med hjälp av objekt-relationell mappning (se 4.2), en objektorienterad programmeringsteknik där tabeller och kolumner i en relationsdatabas representeras av klasser och instansvariabler i ett datorprogram (Telerik, 2022). Dessa klasser instansieras till objekt där datan lagras innan den skrivs till databasen.

Efterfrågningar av data från databasen sker via det internt designade API:et och levereras via ändpunkter till externa applikationer eller andra processer som vill nyttja den (se 4.3).

4.5. Projektstruktur

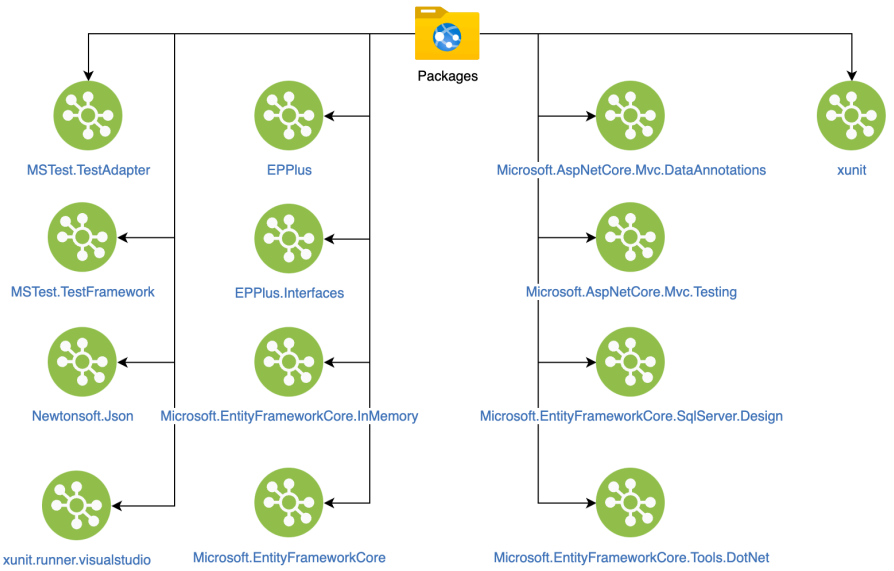
Figuren nedan ger en översikt av projektstrukturen. Mapparna som är av huvudsakligt intresse för funktionaliteten i lösningen är wwwroot, Controllers, Models och Packages. Rootmappen innehåller de delar av programmet som anpassar utseendet på vyn (det grafiska UI som visas i webbläsaren, se 4.1 Filinläsning). Kontrollerfilerna hanterar kommunikationen mellan olika delar av applikationen:



Figur 14. Generell mappstruktur för projektet.

4.5.1 Tillägg

Packages ger en översikt av projektets externa beroenden (de bibliotek som hämtats in i form av tillägg). De kan delas in i tre grupper: de som har att göra med testning av lösningen (xUnit, In

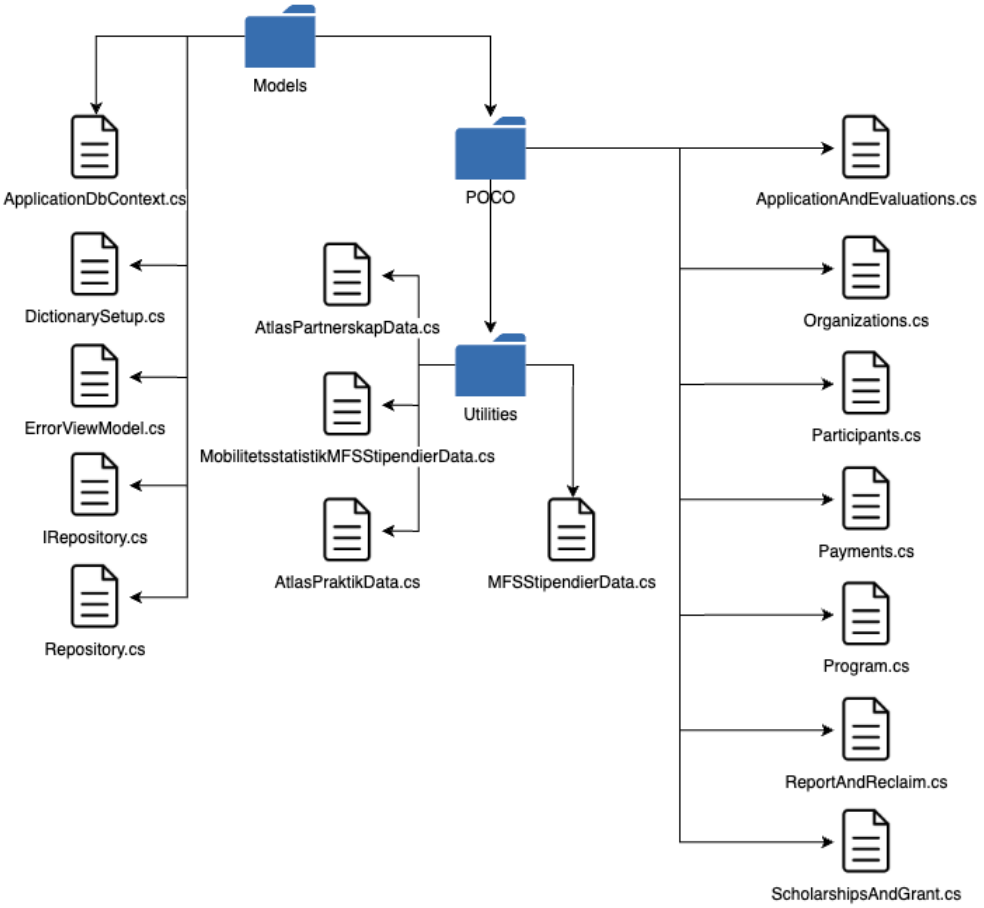


Figur 15. Beroenden för projektet.

Memory, MSTest), de som har att göra med filinläsning (EPPlus, Newsonsoft) och de som har att göra med objekt-relationell mappning (SqlServer, DataAnnotations, Tools.Dotnet). De tillägg som relaterar till filinläsningen är direkt nödvändiga för lösningens funktionalitet.

4.5.2 Modeller

Models-mappen innehåller fem .cs-filer (C# är programmeringsspråket som används och alla klassfiler har filändelsen .cs) och två undermappar: POCO-mappen och Utilities-mappen. Filerna som finns i POCO-mappen representerar tabellerna och deras struktur utifrån vilka en migration till en databas sedan skapas. Inuti POCO-mappen finns också Utilities-mappen som innehåller hjälpklasser som används för att hämta ut data vid API-anropen:



Figur 16. Innehållet i Modellmappen.

5. Utvärdering

I det här kapitlet presenteras de olika utvärderingsformer som använts för att verifiera, testa och tolka den lösning som designades.

5.1. Demonstrativ utvärdering

Applikationen demonstrerades för uppdragsgivaren, utvecklingschefen, och ytterligare en utvecklare hos UHR. Under mötet presenterades koden och funktionaliteten i lösningen under en öppen demonstration där frågor och funderingar besvarades. Efter presentationen förmedlades ytterligare synpunkter på lösningen. Feedbacken var generellt väldigt positiv.

Positiva synpunkter:

- Utvecklingschefen uppskattade lösningen där leverablerna levereras via ett rest-baserat API och kommenterade att det ofta är så UHR arbetar i praktiken. API-lösningen möjliggör att informationen inte behöver finnas inom samma applikation eller ens på samma server som den frontend-tillämpning som har i uppgift att visualisera datan.
- Utvecklingschefen var även positivt inställd till att Postman användes för att testa API-ändpunkterna eftersom samma program används av dem själva i deras utvecklingsarbete.
- Uppdragsgivaren noterade att det var bra att utvecklarna var på plats för att kunna hantera frågor som handlade om de mer tekniska aspekterna av projektet. Hon var imponerad över hur långt projektet hade kommit på ganska kort tid och betonade att hon gärna jobbar med alla inblandade igen i framtiden. Hon kommenterade också att projektet gjort henne positivt inställd till framtida samarbeten med Campus Gotland.
- Kommentarer gavs om att koden såg ut att hålla en hög kvalitet.

Övriga synpunkter:

- Det påpekades att applikationen skapats med MVC Core som är en miljö UHR inte använder för närvarande. Dock finns det ingenting i själva lösningen som sådan som är specifikt för ASP.NET MVC, och författarna av den här rapporten menar att det därför går att med lätthet flytta projektet till .NET 6 utan att förlora någon funktionalitet.
- UHR visade förståelse för att tiden inte hade räckt till för att kunna koppla samman lösningen med det frontend-projekt som en annan projektgrupp skapat åt UHR, och på så vis demonstrera ett komplett dataflöde från backend till frontend-visualisering.

5.2. Tolkande utvärdering

Alla krav och funktioner som beskrevs i det praktiska uppdraget (se 1.2) implementerades i lösningen, antingen delvis eller fullt ut. Genom detta anser författarna att de producerat en lösning som uppfyller de mål som existerat för projektet.

Det fanns inga tydliga specifikationer om hur uppdragsgivaren ville att lösningen för datahanteringen skulle se ut eller hur datan skulle föras över från Excel till relationsdatabasen. Det fanns inte heller något formulerat krav på hur schemat för databasen skulle vara strukturerad. En bedömning gjordes därför som utgick dels från datans kvalitet, dels från den information som var aktuell att leverera via API, att det var bättre att använda några få tabeller för att lagra datan även om det innebar att normaliseringen av dessa tabeller blev lidande.

UHR:s datamodell innehöll 99 tabeller, och ett liknande schema skulle inneburit en enorm mängd JOIN-operationer för att hämta ut data. Ett annat problem som beskrivits tidigare i dataanalyskapitlet (se 3.4) var att kvaliteten på datan i Excel-arken generellt klassades som låg.

Eftersom det saknades tydliga relationer mellan många av kolumnerna var det svårt att skapa en normaliserad lösning där data kunde läsas in från flera olika filer.

Det ansågs därför att det inte fanns någon vinst i den ökande komplexiteten som ett bättre normaliserat schema skulle medfört. Resultatet var en databas för applikationen där viss information lagras dubbelt och null-värden ofta förekommer, men författarna ansåg detta acceptabelt eftersom datan som låg till grund för projektet var avsedd för arkivering och aldrig behövde ändras.

Eftersom SQL och relationsdatabaser var ett formulerat krav var det aldrig aktuellt att lagra dataseten i en NoSQL-dokumentdatabas som exempelvis MongoDb, och ingen formell utvärdering av en sådan lösning gjordes därför även om författarna närde misstankar om att den skulle passat UHR:s data bättre.

Resultatet var att databasen i applikationen endast innehöll sju tabeller, men de kunde lagra data på ett konfliktfritt vis från alla tre UHR-programmen.

5.2.1 Reflektion i .NET

Reflektion är ett användbart verktyg under rätt omständigheter (Microsoft, 2021), men det innebär också kostnader när det kommer till prestanda. Det är en komplex programmeringsteknik som kan innebära att problem uppstår under körtid om man inte är införstådd med hur koden fungerar.

Funktionaliteten och dynamiken i applikationen har vägts mot de potentiella riskerna och slutsatsen har varit att fördelarna ändå är långt större än nackdelarna. De flesta av problemen med reflektion uppstår när man försöker få åtkomst till objekt som förändras av någon annan process samtidigt under körtiden. Vi vet på förhand att det inte finns något i de objekt vi använder oss av som kan förändras av någon annan process. På grund av dess natur kan reflektion också överskriva åtkomstmodifierare och få tillgång till fält som är märkta som privata, vilket bryter mot idén om inkapsling som är en huvudprincip inom objektorienterad programmering. I de objekt som används för att kommunicera med databasen i applikationen finns emellertid inga privata fält.

5.2.2 API-design

Valet föll på en API-lösning för leverans av information av flera anledningar. Med ett API kan exempelvis mängden data som en utomstående applikation har tillgång till begränsas. Om projektgruppen som arbetat med frontend-lösningen enbart behöver vissa av datakolumnerna för att visualisera sin statistik bör den inte ha tillgång till allting annat som också hamnar i databasen för arkivering.

Resultatet av dessa val under utvecklingen är en lösning som kan användas för fler ändamål, och av fler frontend-applikationer för att visualisera data i framtiden, än den som beskrevs i det praktiska uppdraget (se 1.2).

Ett REST-baserat API valdes över ett SOAP-baserat eftersom REST inte har några restriktioner vad gäller sitt representativa format medan SOAP är XML-baserat (Ranga & Soni, 2019, s.2 / 619). Ett REST-baserat API ansågs lämpligare att implementera och på så vis utnyttja JSON-formatets nyckel-värde-natur (Ranga & Soni, 2019, s.3 / 620) för att leverera information till frontend-moduler. Ett REST-API var slutligen lättare att skapa och implementera, och passade därför också bättre in i utvecklingsprocessens något snäva tidsramar.

5.2.3 Dokumentation

Lösningens dokumentation skedde enligt de fyra krav som Sommerville (2001, s. 2) beskriver för bra dokumentering av programvara:

1. Dokumenten bör fungera som ett medium för kommunikation mellan medlemmarna i utvecklingslaget.
2. De ska kunna fungera som ett informationsarkiv för underhåll och drift
3. De ska kunna bidra med information för ledningen för att hjälpa dem planera, budgetera och schemalägga utvecklingsprocesser, och
4. En del av dokumenten ska kunna beskriva för användare hur de använder systemet

I rapportens designresultat (se 4.1-4.7) används både konceptuella datamodeller, ingående beskrivningar av källkoden samt en översikt av API-funktionalitet för att dokumentera den lösning som tagits fram. Själva programkoden är också noggrant kommenterad för att framtida utvecklare ska kunna orientera sig genom den. Författarna anser därigenom att de uppfyllt kraven som både Sommerville och Chomal & Saine (2014) gestaltar för god dokumentation på ett tillfredsställande vis - inom ramarna för projektets omfattning och tidplan. Mer omfattande dokumentation skulle tagits fram i större mån av tid, men precis som den framarbetade lösningen bör dokumentation ses som ofärdig.

5.2.4 Interna kvalitetsegenskaper och goda designprinciper

Designen av lösningen inspirerades av de interna kvalitetsegenskaper som McConnell (2004) definierar för informationssystem; sådana som påverkar systemkvaliteten i någon utsträckning. Tre av dessa interna kvalitetsegenskaper i synnerhet har ansetts relevanta för att utvärdera lösningens kvalitet: Återanvändbarhet, flexibilitet och underhållbarhet (McConnell, 2004, s. 264):

- **Underhållbarhet** definieras som hur lätt det är att modifiera ett system för att ändra eller förlänga dess funktionalitet.
- **Flexibilitet** innebär att ett system går att modifiera för fler miljöer och appliceringsområden än de som det ursprungligen var designat för.
- **Återanvändbarhet** beskrivs som ett mått på hur lätt det är att använda delar av ett system i andra system.

Designfasen influerades av dessa kvalitetesgenskaper för att generera en lösning som matchar deras beskrivning. Tidigare nämnda dokumentation som producerats (se 5.2.3) anses höja kvaliteten på systemets underhållbarhet. API-leverans och kodgeneralisering (se 5.2.2 och 5.2.1) anses höja systemets flexibilitet och återanvändbarhet. McConnell beskriver också i kapitlet “Desirable Characteristics of a Design” hur interna designaspekter som lösa kopplingar och “förlängningsbarhet” (eng. “extensibility”) är en indikation på god design. En lös eller mjuk koppling i programmering innebär att man försöker hålla förbindelserna mellan de olika delarna i ett program till så få som möjligt (McConnell, 2004, s. 80 / 117).

Ett konkret exempel på mjuka kopplingar i lösningen är hur kodrepositoryt Repository är en implementation av interfacet IRepository. Alla kontrollerklasser som använder sig av repositoryt tar in ett IRepository i sina konstruktorer och undviker därmed hårda beroenden.

5.3. Utvärdering genom designprocessen

Eftersom Hevners tre designcykler (Hevner 2007) använts som metodstöd under arbetet (se 3.1 Metodstöd för forskning) har lösningen som producerats testats kontinuerligt under utvecklingens gång, och författarna menar därför att den experimentella utvärderingen som utförts också är en typ av formativ utvärdering. För tydlighetens skull har de dock brutits upp i separata underkategorier under samma huvudkategori.

5.3.1 Formativ utvärdering

Designprocessen har utförts enligt utsatta milstolpar. För varje milstolpe som fullföljts har författarna av rapporten reflekterat över det utförda arbetet för att gemensamt godkänna resultatet, eller också göra förändringar i de delar som ansetts kräva en förbättring.

Kanban-metoden har följts under projektet för att fördela arbetet i "tasks" och distribuera dessa mellan gruppmedlemmarna. Git har använts för att enkelt kunna dela förändringar och ha en versionshantering. Arbetet har generellt granskats och godkänts av andra gruppmedlemmar innan en commit och push skett till huvudgrenen i repositoryt.

Varje arbetsdag inleddes med en kort diskussion kring hur långt arbetet kommit och vad som behövde göras under dagen och veckan, och GANTT-schemat uppdaterades som ett resultat.

Dataförädlingsfasen hade en relativt stor påverkan på resultatet av designprocessen och den slutgiltiga lösningen. Under förädlingsarbetet upptäcktes många brister i den data som hade levererats som underlag. Formatet var inte enhetligt för alla rader i en kolumn och olika filer hade typer som skilde sig för likartade kolumner (såsom exempelvis datum och referensnummer). Dataanalysen påverkade lösningens funktionalitet konkret i avseendet att ökad funktionalitet för att kunna förstå, läsa och sortera data krävdes. Kraven på lösningens flexibilitet blev i och med detta högre och samtidigt mer tekniskt komplicerade.

5.3.2 Experimentell utvärdering

Under designfasen har lösningens funktionalitet testats och utvärderats i takt med att den definierats (och omdefinierats) i programkod. Författarna har valt att se på testningen inte som ett separat åtagande utan som en ständigt pågående förbättringsprocess i linje med Hevners (2007) iterativa cykler.

Den experimentella utvärdering som ägnades mest tid åt var CRUD-operationer för databasen samt de definierade GET-funktioner i API-ändpunkterna som via LINQ i datalagret hämtade ut, paketerade och transporterade information till externa anropare.

För att simulera externa anropare användes programmet Postman. Test utfördes inledningsvis även med xUnit (se bilaga 5. Kod för Unit-testning av API-ändpunkt) men miljön övergavs senare eftersom positiva och negativa tester inte ansågs bidra med någonting värdefullt inom kontexten för projektet. Lösningen krävde att det gick att validera både vilken sorts information som returnerades och hur den var strukturerad, och inte bara *att* ett anrop lyckades hämta information från datalagret.

Postman å sin sida kan exekvera HTTP-requests och är därför en utmärkt testmiljö för API-anrop. I illustrationen nedan simuleras ett anrop till databasen i testmiljön (en lokal sql-server på utvecklarnas dator) för att kunna utvärdera vad som returneras i anropet:

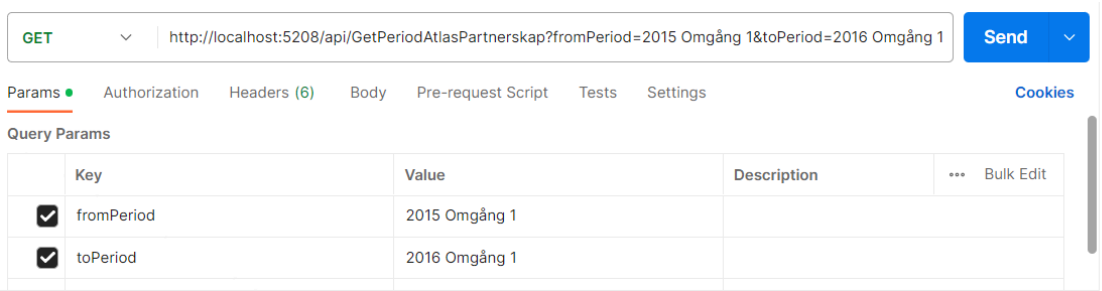


Fig. 17: Anrop till testmiljön via Postman.

Datan som returneras går att strukturera i en mängd olika format och former i programmet. För teständamål har JSON-format använts eftersom det är det som API-et skickar ut (se bilaga 6: JSON-resultat). Datapunkter ligger lagrade i nyckel-värde-par inuti en godtycklig mängd objekt, paketerade i en array.

Möjligheten att köra anrop till ändpunkterna och fånga upp responsen innebar att Postman inte bara användes för att testa API:et utan också att databaskopplingar och operationer i datalagret (såsom formulerade LINQ-frågor) fungerade korrekt.

5.4. Sammanfattande karakterisering och reflektion

Uvärdering av arbetet gjordes i form av en målbaserad utvärdering (Cronholm & Goldkuhl, 2003), där lösningen utvärderats baserat på förutbestämda mål. Cronholm & Goldkuhl (2003, s. 66) skriver att de definierade målen kan vara både mänskliga och organisatoriska, men att den traditionella tolkningen av målbaserad utvärdering oftast går att relatera till konkreta, mätbara mål.

Kritik har riktats mot den här typen av utvärdering eftersom den bekymrar sig mer för tekniska och ekonomiska aspekter (Cronholm & Goldkuhl, 2003, s. 66), men då lösningen som producerats är en teknisk tillämpning utan mycket mänsklig interaktion framstår en sådan utvärderingsform som lämplig i det här fallet. Styrkan i den valda utvärderingsformen är att den i gengäld kan användas för att bedöma om målen för uppdraget har uppnåtts eller ej. Målen utgörs inom ramarna för projektet av de huvud- och delproblem som identifierats i rapporten (se 2.1 och 2.2), och eftersom projektet även haft ett begränsat forskningsvärde blir lösningen av dessa problem den bästa måttstocken att använda för en utvärdering.

Designen anses uppfylla de mål som kan identifieras. Applikationen fungerar och erbjuder, så långt som funktionaliteten hunnit implementeras, en lösning för de problem som UHR formulerade kring dataframtagning i sitt handläggningssystem (se 1.2 Praktisk uppdrag).

Ett självkritiskt perspektiv är att för mycket tid slösades bort i och med att nyckeldesignbeslut klubbades sent (exempelvis användning av ett API) vilket krävde en omfattande upprepning av relevanscykeln som Hevner (2007) nämner för att omdefiniera miljön för projektet. Även författarnas magra erfarenhet av den rent tekniska miljön var en bidragande faktor till att lösningen aldrig hann sammankopplas med den lösning som producerades av frontend-gruppen som arbetade parallellt mot UHR. Samma brist på erfarenhet låg också bakom avsaknaden av reflektion som styrde valet av MVC Core.

6. Diskussion och abstraktion

Under projektarbetet har det tagits flera designbeslut som vi anser har varit av stor vikt. Valet av en relationsdatabas var dock ett kluvet. Den data som togs emot ansågs lämpligare att lagra i en NoSQL-lösning, men kraven från uppdragsgivaren specificerade att SQL skulle användas. Detta innebar att ingen utredning gjordes där alternativa lagringsmetoder jämfördes med SQL. En sådan utredning skulle förvisso krävt ytterligare tid, men hade samtidigt kunnat göra både forskningsbidraget och kunskapsbidraget i arbetet större.

Flera olika designkoncept undersöktes i början av projektet eftersom ett formulerat uppdrag saknades. Eftersom formulerade riktlinjer även saknades rent generellt tvingades författarna att utforma sina egna riktlinjer för lösningen. Allt detta tog lång tid och påverkade hur mycket tid som fanns kvar för själva designfasen. Trots tidsbristen och den generella bristen på riktlinjer anser författarna dock att kvaliteten i den producerade lösningen är hög och att artefakten uppfyller de mål som definierades för projektet (se 1.2 Praktisk uppdrag).

Programmet är anpassat för UHR, men själva typen av lösning i sig är applicerbar för andra system av andra utvecklare, så till vida att den illustrerar tydligt hur man kan bygga en liknande lösning. Mandinach (2012) lyfter fram data-driven decision making; det vill säga, dataanalys av en organisations informationsresurser. Hur dessa resurser är lagrade och hur pass tillgängliga de är spelar en betydande roll för kvaliteten i analysen. Dataförädling är ett sätt att höja kvaliteten på och tillgängligheten för dessa resurser rent generellt, men är bara effektiv till en viss gräns. Bättre indata resulterar ofta i bättre utdata. Datakvaliteten i dataseten hade på liknande vis ett direkt inflytande på designen av själva artefakten: en högre datakvalitet skulle resulterat i bättre generaliserad kod och ett mer robust program.

Likt fallet med NoSQL-lösningen är det främst författarnas brist på erfarenhet som gjort att detta inte diskuterats med produktägaren, och författarna själva reflekterar självkritiskt över sin insats i det här avseendet. Universitetets utformning av kursen innebar också en brist på tid för analys i ett inledande skede av projektet - analys som skulle kunnat leda till omdefinierade krav.

Principer från lösningen bör kunna appliceras på andra problemområden och domäner. Författarnas åsikt är att andra utvecklare kan dra nytta av att tolka och använda de konceptuella datamodeller och dataflöden som dokumenterats eftersom de beskriver processer och designidéer som syftar till att höja den interna systemkvaliteten för informationssystem (McConnell, 2004). Dataförädling är ett brett område som går att förankra i flera olika typer av dataanalys och olika omständigheter. Mandinach (2012) utgår exempelvis från utbildningssystem i sin publicering, men slutsatserna är applicerbarbara på systemutveckling rent allmänt.

Informationskvalitet är en av de tre dimensioner som allting utgår från i Delone och McLeans (2004) ISS-modell, som beskriver hur framgångsrika informationssystem byggs upp. Informationskvalitet definieras som kvaliteten i den information som systemet kan lagra, leverera och producera. Lösningen som producerats i projektet är intimt kopplad till den här dimensionen så till vida att den försöker förbättra den.

Principiella lärdomar från analys- och designfaserna handlar främst om hur viktig kvaliteten på grunddatan är, och kan ur det ljuset ses som en validering av ISS-modellen. Dataförädling är en nödvändig process inom varje organisation som sysslar med dataanalys (Piringer & Endel, 2015) men författarna av denna rapport vill också betona att det är viktigt att information samlas in och lagras på ett korrekt vis från början eftersom det inte finns någon självklar lösning på många av de problem som skapas när så inte sker:

> “it can be shown that data wrangling is still a defiant process. Although a variety of solutions for single problems and special situations exist, hardly any of them cover all raised considerations.” (Piringer & Endel, 2015, s. 112)

7. Slutsatser

Nedan presenteras sammanfattande tankar kring uppdragets praktiska del och de metoder som användes, samt rekommendationer om fortsatt forskning- och utvecklingsarbete.

7.1. Slutförande av uppdraget

Arbetet för projektet resulterade i en dynamisk .NET-applikation som ska användas för dataarkivering i en relationsdatabas. Den data som databasen innehåller, matas in via uppladdning av Excel-filer. Åtkomst till applikationen och dess data sker via ett REST-API.

Det anses att den demonstrativa och tolkande utvärderingen har påvisat att lösningen uppfyller uppdragets mål: att lagra och leverera specifik data som ska kunna användas som statistik för att generera diagram och/eller modeller. Urvalet av data sker direkt i lösningen. API-metoderna för respektive ansökningsprogram är förkonfigurerade för att bara hämta ut data från specifika kolumner enligt UHR:s önskemål. Detta begränsar vilken data anroparen kan efterfråga från systemet.

Resultatet är en lösning som gör det möjligt för UHR att senare bygga eller implementera en frontend-lösning som kan hämta och visualisera data. Den kan också bidra till att uppdragsgivarens datahanterings- och transfomeringsprocess effektiviseras, samt underlätta deras arkivering av gamla rapporter i systemet för avslutade utlysningar.

Leverabler är det som levereras är själva lösningen, samt dokumentation i form av den här rapportens designresultat. Dokumentationen anses uppfylla de krav som Sommerville (2001) formulerat för dokumentbeskrivning av mjukvara så långt det är möjligt. På grund av tidsbrist existerar dock ingen separat dokumentation utöver rapportens beskrivningar.

De sekundära leverabler som lösningen i sin tur levererar är informationen som kommer ut ur ändpunkterna via API-förfrågningar. Det är dessa sekundära leverabler som sedan kan användas som statistik för att definiera visualiseringen någon annanstans.

7.2. Fortsatt praktiskt arbete och vidareutveckling

Nedan tas exempel upp på vad som kan utvecklas vidare om produktägaren väljer att arbeta vidare med lösningen.

- **Säkerhet:** Ett viktigt område med utrymme att vidareutvecklas är säkerheten för våra sekundära leverabler, dvs. den information som skickas via våra API ändpunkter. Det skulle behövas en utredning om vilka säkerhetsåtgärder är passande, och hur implementationen av dessa kommer ske baserat på lösningens användningsyfte och den sekretessnivå informationen i leverablerna kommer ha.

  Den potentiella kunskapsutvecklingen inom det här området innefattar kunskaper om bästa praxis för kryptering, autentisering och auktorisering och skulle kunna leda till värdefulla erfarenheter av att utveckla säkerhetsprotokoll.

- **Standardisering:** Ett annat viktigt område med stort utrymme för förbättring är standardiseringen av formatet på den data som laddas upp till lösningen.

  Genom att ha en enhetlig standard för formatet på den data som laddas upp kommer många potentiella fel att åtgärdas och arbetet för att rensa data kommer att minska. Vad vi menar med att standardisera formatet är att, för ett Excel-kalkylark, ha regler för kolumnrubriker och ha dessa rubriker för alla kolumner som innehåller samma typ av data. Detta bör implementeras genom alla program så att en gemensam arkivering kan ske felfritt. Detta åtagande skulle innebära att det krävs färre upprepade rader i applikationens kod, och det kommer att underlätta dataextraheringen.

Den fortsatta forskningen för detta område relateras till behovet av en standardisering av dataformat och kan exempelvis innefatta studier om bästa praxis för datastandardisering och verktyg, samt metoder som ser till att data som uppladdas håller dessa standarder.

- **Validering:** Detta är ett område som författarna skulle implementerat själva ifall tiden hade räckt till. Validering av filformat är ett viktigt område och bör utvecklas vidare för att förhindra stora säkerhetsbrister som skadliga kodinjektioner. I nuläget skulle exempelvis en illasinnad användare kunna ladda upp en fil med skadlig kod som sedan behandlas och exekveras av applikationen, så länge den uppfyller ett simpelt filändelsekrav, vilket kan leda till skador för databasen, för applikationen och även för UHR:s informationstillgångar.

  Det skulle behövas en utredning om vilka filformat UHR jobbar med och vilka de skulle vilja använda med lösningen för att sedan implementera dessa valideringskrav i applikationen. Det kan också vara värt att utreda hur identitetshanteringen ska skötas, och om alla användare verkligen ska ha uppladdningsrättigeter eller inte.

Slutligen finns det några ofärdiga API-ändpunkter som behöver implementeras för en mer komplett funktionalitet, varpå nästa logiska steg sedan blir att koppla samman lösningen med frontend-modulen som utvecklats på annan plats.

7.3. Forskningsbidrag

Det förväntade kunskapsbidraget är en förbättring av den problemklass som identifierades, vilket inom ramarna för projektet givit lärdomar som andra kan använda sig av i fältet. Detta är i första hand av relevans för uppdragsgivaren (UHR), och förhoppningen är att det kan förbättra vad Mandinach (2012, s.71) beskriver som data-driven decision making (DDDM): systematisk insamling, analysering och tolkning av data med syftet att bättre kunna informera policy och beslutsfattande i en organisation. Detta kunskapsbidrag anses dock också relevant för andra organisationer och utvecklare som är intresserade av automatiserad dataförädling för att kunna höja kvaliteten på sin DDDM.

Lösningen bidrar med en prototyp vars forskningsbidrag främst är programkoden självt, samt den dokumentation som skapats i form av den här rapporten. God dokumentation kan öppna upp möjligheter för andra utvecklare som vill ha inspiration för att skapa liknande eller mer avancerade lösningar som grundar sig i liknande problemklasser.

7.4. Fortsatt forskning

Ett område för fortsatt forskning och kunskapsutveckling skulle kunna vara att undersöka det färdiga systemet ur ett användningsperspektiv. De kvalitetsegenskaper som anses ha förbättrats i lösningen (McConnell, 2004) leder indirekt till en förbättrad användbarhet, genom att möjliggöra frontend-modulen som användaren interagerar med. Med användbarhet menas det som Nielsen (1993) beskriver som en produkt av två underkategorier vilka kan översättas till nytta (utility) och användarvänlighet (usability) på svenska:

> "Usefulness is the issue of whether the system can be used to achieve some desired goal. It can again be broken down into the two categories of utility and usability, where utility is the question of whether the functionality of the system in principle can do what is needed, and usability is the question of how well users can use that functionality." (Nielsen, 1993, ss. 24-25)

Användbarhet är också ett ord som ofta figurerar i litteratur som behandlar informationssystem och informatik. I Delone & McLeans Information Systems Success Model (DeLone & McLean,

2004) tas hänsyn till flera faktorer som kan påverka framgången för ett IS-system; däribland användbarhet, som är en faktor som styr användarupplevelsen.

Utifrån Nielsens definition anses lösningen i det här projektet ha bidragit med bättre nytta (utility). Frontend-modulen som en annan grupp byggde parallellt var på liknande vis tänkt att bidra med bättre användarvänlighet (usability). Det skulle vara intressant i synnerhet att, i ett system där två sådana här lösningar kopplats samman för att tillsammans höja användbarheten, undersöka relationen mellan förbättrad användbarhet och det som Mandinach (2012) beskriver som data-driven decision making, och huruvida det går att identifiera ett samband mellan ökad användbarhet och bättre DDDM i allmänhet.

7.5. Metodreflektion

Användningen av Kanban (se 3.2.2 Metodstöd för utveckling), i kombination med en virtuell anslagstavla, underlättade planeringen av arbetet. Anslagstavlan användes för att enkelt kunna lägga till och fördela uppgifter inom projektet. Att använda Kanban som arbetsmetod har även inneburit en mer effektiv tidsfördelning i projektet. En viktig aspekt av filosofin bakom Kanban är att gruppen gemensamt bestämt vilka uppgifter som ska ingå i varje sprint. Innan en sprint avslutats påbörjas därför inga nya uppgifter. Detta gav en mer logisk följd åt arbetsuppgifterna, vilket passade projektet bra eftersom implementationen av viss funktionalitet, som exempelvis designen av API:et, byggde på att en databas redan existerade. Detta har generellt inneburit att flaskhalsar undvikits i designfasen där viss funktionalitet inte kunnat implementeras på grund av att andra delar av lösningen saknats. Det har samtidigt inneburit att det varit lättare att identifiera och åtgärda ineffektivitet i arbetsprocessen.

Referenser

Apache (2023). *Features*, apache.org. Tillgänglig: https://nifi.apache.org/ [2023-05-26]

Baskerville, R. & Kaul, M. & Storey, V.. (2011). *Unpacking the Duality of Design Science*, International Conference on Information Systems 2011, ICIS 2011.
Tillgänglig: https://uppsala.instructure.com/courses/73534/files/4802934?wrap=1 [2023-04-17]

Baskerville, R., Kaul, M. & Storey, V. M. (2011). *Unpacking the Duality of Design Science*, Thirty Second International Conference on Information Systems, Shanghai 2011.
Tillgänglig via Uppsala Universitets intranät:
https://uppsala.instructure.com/courses/73534/files/4802934?wrap=1 [2023-05-30]

Booch, G., Rumbaugh, R. & Jacobson, I. (2005). *The Unified Modeling Language User Guide,* 2a upplagan, Addison-Wesley Professional.

Carstensen, A-K. & Bernhard, J. (2018). *Design science research – a powerful tool for improving methods in engineering education research*, European Journal of Engineering Education, Volume 44, Issue 1-2, 2019. Tillgänglig:
https://www.tandfonline.com/doi/citedby/10.1080/03043797.2018.1498459?scroll=top&needAccess=true&role=tab [2023-04-17]

Chomal, V. S. & Saine, J. R. (2014). *Significance of Software Documentation in Software Development Process*, International Journal of Engineering Innovation & Research, Volume 3, Issue 4, ISSN: 2277 – 5668. Tillgänglig:
https://www.researchgate.net/profile/Jatinderkumar-Saini/publication/281965276_Significance_of_Software_Documentation_in_Software_Development_Process/links/55ffdc6008aeba1d9f841187/Significance-of-Software-Documentation-in-Software-Development-Process.pdf
[2023-05-25]

Cronholm, S. & Goldkuhl, G. (2003). *Strategies for Information Systems Evaluation- Six Generic Types*, Electronic Journal of Information Systems Evaluation Volume 6 Issue 2(2003) 65-74

Datameer (2023). *Datameer X*, datameer.com. Tillgänglig: https://www.datameer.com/dmx/ [2023-05-26]

Delone, W. & McLean, E. (2004). *Measuring e-Commerce Success: Applying the DeLone & McLean Information Systems Success Model*. International Journal of Electronic Commerce. 9. 31-47. 10.1080/10864415.2004.11044317.

EPPlus (2023). *Features and Technical Overview*, epplus.com.
Tillgänglig: https://www.epplussoftware.com/en/Developers/Features [2023-04-24]

Fay, G. (2022). *APIs Are A Building Block—Don't Make Them A Stumbling Block*, forbes.com. Tillgänglig:
https://www.forbes.com/sites/forbestechcouncil/2022/03/28/apis-are-a-building-block-dont-make-them-a-stumbling-block/ [2022-05-25]

Friesen, J. (2019). *Java XML and JSON: Document Processing for Java SE*, 2:a upplagan, Apress.

Gregor, Shirley & Hevner, Alan. (2013). *Positioning and Presenting Design Science Research for Maximum Impact*. MIS Quarterly. 37. 337-356. Tillgänglig:
https://www.researchgate.net/publication/262350911_Positioning_and_Presenting_Design_Science_Research_for_Maximum_Impact [2024-04-24]

Gustavsson, T. & Görling, S. (2019). *Att arbeta med systemutveckling*, Studentlitteratur AB Lund, Upplaga 1:2.

Hevner, A. (2007). Issue 2 Article 4 1-1-2007 Recommended Citation Hevner. Scandinavian Journal of Information Systems, [online] 19(2).
Tillgänglig: https://www.uio.no/studier/emner/jus/afin/FINF4002/v13/hefner-design.pdf [2023-04-18]

Kanban (2023). *How the kanban methodology applies to software development*, kanban.com. Tillgänglig: https://www.atlassian.com/agile/kanban [2023-05-30]

Knime (2023). *KNIME analytics platform: Allowing anyone to build and upskill on data science*. Tillgänglig: https://www.knime.com/knime-analytics-platform [2023-04-17]

Lindkvist, L., Soderlund, J. and Tell, F. (1998). *Managing Product Development Projects: On the Significance of Fountains and Deadlines*, Organization Studies, 19(6), pp.931–951. Tillgänglig: https://doi.org/10.1177/017084069801900602 [2023-04-17]

Lindley, C. (2013). *DOM Enlightenment*, O'Reilly Media, Inc. ISBN: 9781449342845

Mandinach, E.B. (2012). *A Perfect Time for Data Use: Using Data-Driven Decision Making to Inform Practice*, American Psychological Association. ISSN: 0046-1520 print / 1532-6985 online. Tillgänglig: https://www.tandfonline.com/doi/full/10.1080/00461520.2012.667064 [2023-05-28]

Martin, R. C. (2000). *Design Principles and Design Patterns*, Object Mentor.
Tillgänglig: http://staff.cs.utu.fi/~jounsmed/doos_06/material/DesignPrinciplesAndPatterns.pdf [2023-05-09]

McConnell, S. (2004). *Code Complete*, Microsoft Press, Andra upplagan.
Tillgänglig: https://uppsala.instructure.com/courses/65158/files/3736397?wrap=1 [2023-04-12]

Microsoft (2021). *Reflection in .NET*, microsoft.com. Tillgänglig: https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection [2023-01-11]

Microsoft (2022). *What is Power BI? - Power BI*. learn.microsoft.com. Tillgänglig: https://learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview [2023-04-17]

Microsoft (2023a). *SQL i Access: grundläggande begrepp, vokabulär och syntax*, microsoft.com. Tillgänglig: https://support.microsoft.com/sv-se/office/sql-i-access-grundl%C3%A4ggande-begrepp-vokabul%C3%A4r-och-syntax-444d0303-cde1-424e-9a74-e8dc3e460671 [202305-28]

Microsoft (2023b). *Entity Framework documentation hub*, microsoft.com. Tillgänglig: https://learn.microsoft.com/en-us/ef/ [2023-05-28]

Microsoft (2023c). *Overview of ASP.NET Core MVC*, microsoft.com. Tillgänglig: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0 [202305-28]

Microsoft (2023d). *ASP.NET Documentation*, microsoft.com. Tillgänglig: https://learn.microsoft.com/en-us/aspnet/core/?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0 [2023-04-18]

Microsoft (2023e). *Create web APIs with ASP.NET Core*, microsoft.com. Tillgänglig: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0 [2023-05-29]

MongoDb (2023). *Advantages of MongoDb*, mongodb.com. Tillgänglig: https://www.mongodb.com/advantages-of-mongodb [2023-05-22]

Moss, K. (2012). *The Entity-Relationship model*, Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON). Tillgänglig: https://ieeexplore.ieee.org/document/6201182 [2023-05-28]

Nielsen, J. (1993). *Usability engineering*, Academic Press, Första upplagan. ISBN: 1-12-518406-9

Oracle (2023). *What is a Relational Database (RDBMS)?*, oracle.com. Tillgänglig: https://www.oracle.com/database/what-is-a-relational-database/ [2023-05-28]

Okur. S., Hartveld, D., Dig, D., van Deursen, A. (2014). *A study and toolkit for asynchronous programming in C#.* Tillgänglig: https://dl.acm.org/doi/abs/10.1145/2568225.2568309 [2023-05-31]

Padron-McCarthy, T. & Risch, T. (2018). *Databasteknik*, Studentlitteratur, 2:a upplagan

Paasivaara, M. and Lassenius, C. (2003). *Collaboration practices in global inter-organizational software development projects*, Software Process: Improvement and Practice, 8(4), ss.183–199. Tillgänglig: https://doi.org/10.1002/spip.187 [2024-04-17]

Pellerin, R. and Perrier, N. (2018). *A review of methods, techniques and tools for project planning and control*, International Journal of Production Research, [online] 57(7), pp.2160–2178.
Tillgänglig: https://doi.org/10.1080/00207543.2018.1524168 [2023-04-17]

Piringer, H. & Endel, F. (2015). *Data Wrangling: Making data useful again*. IFAC Papers OnLine,Volume 48, Issue 1. Tillgänglig: https://www.sciencedirect.com/science/article/pii/S2405896315001986?via%3Dihub [2023-05-23]

Radigan, D. (2022). *What is Kanban?* Atlassian.
Tillgänglig: https://www.atlassian.com/agile/kanban [2023-04-17]

Ranga, V. & Soni, A. (2019). *API Features Individualizing of Web Services: REST and SOAP*, International Journal of Innovative Technology and Exploring Engineering (IJITEE). ISSN: 2278-3075, Volume-8, Issue-9S. Tillgänglig: https://www.researchgate.net/publication/335419384_API_Features_Individualizing_of_Web_Services_REST_and_SOAP [2022-05-25]

Rees, B. and Prando, P. (2001). *Documentation and Log Keeping: Ensuring Your Work Does What You Intend It to Do*, Health Physics, Health Physics Society, pp.265–268.
Tillgänglig: https://journals.lww.com/health-physics/Abstract/2001/09000/Documentation_and_Log_Keeping__Ensuring_Your_Work.7.aspx [2023-04-17]

Sommerville, I. (2001). *Software Documentation*, Lancaster University, UK Issue 3, August 2000. Tillgänglig: http://www.literateprogramming.com/documentation.pdf [2023-05-29]

Talend (2023). *Take Your Data For A Spin*, talend.com. Tillgänglig: https://www.talend.com/products/talend-open-studio/ [2023-05-26]

Telerik (2022). *.NET Basics: ORM (Object-Relational Mapping)*, telerik.com.
Tillgänglig: https://www.telerik.com/blogs/dotnet-basics-orm-object-relational-mapping [2023-05-09]

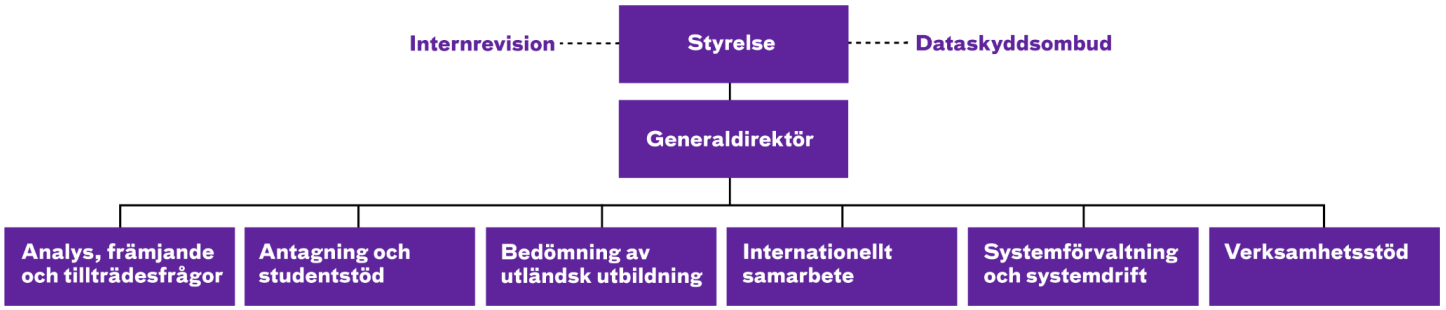
Thurber, M. (2018). *What is Data Wrangling and Why Does it Take So Long?* ElderResearch.com. Tillgänglig: https://www.elderresearch.com/blog/what-is-data-wrangling-and-why-does-it-take-so-long/ [2023-05-22]

Trello (2023). Tillgänglig: https://trello.com/sv [2023-05-30]

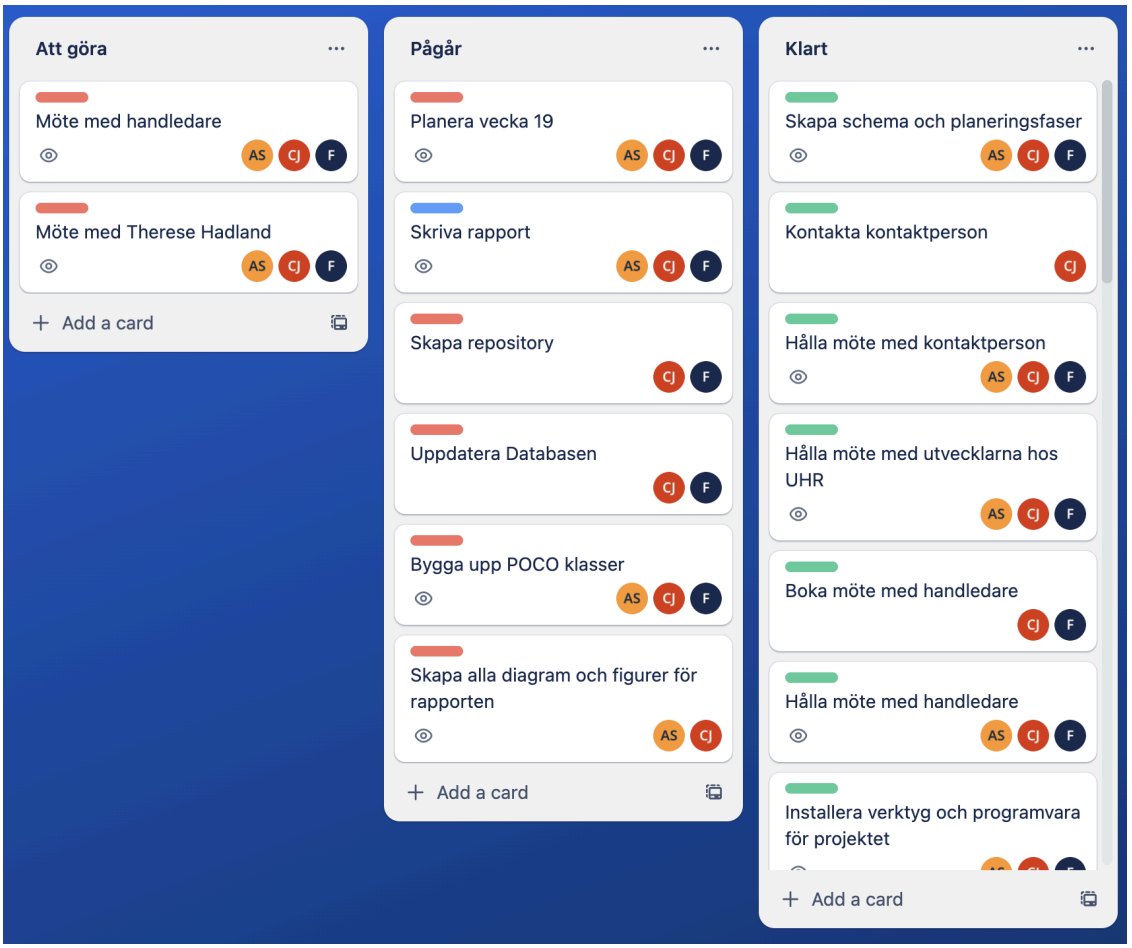
Rodina, D. (2023). *Chen ER Diagram - Entity-Relationship Diagram in Chen Notation,* Software Ideas Modeler. Tillgänglig: https://www.softwareideas.net/chen-er-diagram-erd [2023-05-29].

Ågerfalk, P. (2014). *Insufficient theoretical contribution: a conclusive rationale for rejection?*, European Journal of Information Systems, 23(6), pp. 593-599

Bilaga 1: Organisationsstruktur för UHR



Bilaga 2 - Trellotavla



Bilaga 3 – Användarfall

Användningsfall 1: Ladda upp datafil

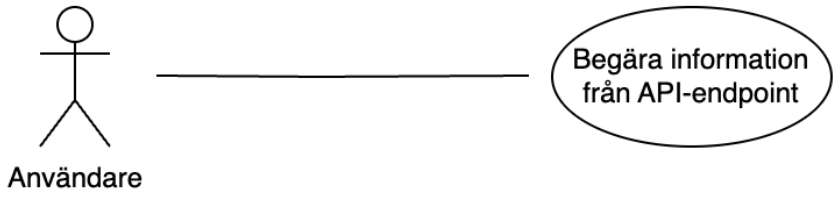
Primära aktörer: Användare
Förutsättningar: Användaren är inloggad, datafil finns tillgänglig för uppladdning.
Lyckat resultat: Användaren laddar upp datafilen, och data lagras i databasen.
Grundfall:

1. Användaren tar fram sidan för uppladdning av datafil.
2. Systemet visar upp en filuppladdningsvy.
3. Användaren väljer och laddar upp önskad datafil.
4. Systemet validerar filformatet och läser in data från filen till de relevanta tabellerna i databasen.
5. Användaren får en bekräftelse när uppladdningen och inläsningen av data är klar.

Undantag:

6. Steg 4. Om filformatet är ogiltigt eller innehållet inte kan läsas, visas ett felmeddelande och användaren uppmanas att ladda upp en giltig fil.

Användningsfall 2: Begära information från API-endpoint



Primära aktörer: Användare
Förutsättningar: Användaren är inloggad och har behörighet att begära information från API-endpoint.
Lyckat resultat: Användaren efterfrågar information från API-endpoint och får den önskade informationen levererad.
Grundfall:

1. Användaren navigerar till sidan för att begära information från API-endpoint.

2. Systemet visar upp en vy för att ange vilken information som efterfrågas och i vilket format (t.ex. JSON eller XML).
3. Användaren anger de parametrar som krävs för att begära den önskade informationen och väljer format.
4. Systemet behandlar förfrågan och hämtar relevant information från databasen baserat på de angivna parametrarna.
5. Systemet returnerar den efterfrågade informationen i det valda formatet till användaren.
6. Användaren tar emot och använder den levererade informationen.

Undantag:

7. Steg 4. Om förfrågan inte kan behandlas på grund av ogiltiga parametrar eller annat fel, visas ett felmeddelande och användaren uppmanas att korrigera förfrågan.

Bilaga 4: Kod för Unit-testande av API-ändpunkt

Konstruktor och klassvariabler:

```
namespace APITest
{
    public class APIUnitTest
    {

        private readonly APIController controller;
        private readonly ApplicationDbContext context;


        //KONSTRUKTOR
        public APIUnitTest()
        {
            var options = new DbContextOptionsBuilder<ApplicationDbContext>()
                .UseSqlServer("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=RaukDb;Integrated Security=True;Connect
                 Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi Subnet
                 Failover=False")
                .Options;


            context = new ApplicationDbContext(options);
            controller = new APIController(context);
        }
```

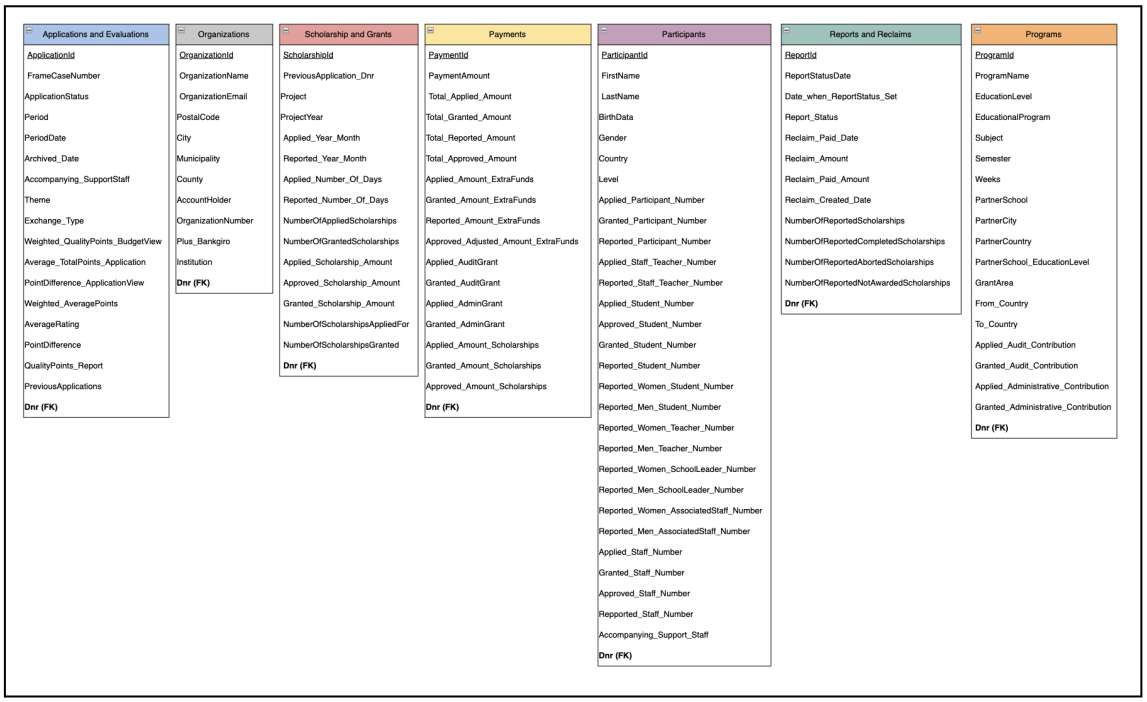
Testmetod:

```
        //Testmetod
        [Fact]
        public void TestGet()
        {
            // Arrange
            int id = 2;
            var organizationName = context.Organizations
                                    .Where(o => o.OrganizationId == id)
                                    .Select(o => o.OrganizationName)
                                    .FirstOrDefault();

            // Act
            var result = controller.Get(id);

            // Assert
            Assert.IsType<JsonResult>(result);
            var organization = Assert.IsType<Organization>(result.Value);
            Assert.Equal(organizationName, organization.OrganizationName);
        }
```

Bilaga 5: Entity Attribute Box



Bilaga 6: JSON-resultat

Exempel på att hämta ut individuell rapport via ett API-anrop:


```
{
    "dnr": "3.3.1.42.226-2015",
    "period": "2015 Omgång 1",
    "applicationStatus": "Beviljad",
    "total_Granted_Amount": 81000,
    "total_Approved_Amount": 81000,
    "applied_Student_Number": null,
    "approved_Student_Number": null,
    "granted_Participant_Number": 9,
    "reported_Participant_Number": 12,
    "reported_Women_Student_Number": null,
    "reported_Men_Student_Number": null,
    "reported_Women_Teacher_Number": 6,
    "reported_Men_Teacher_Number": 3,
    "reported_Women_SchoolLeader_Number": 3,
    "reported_Men_SchoolLeader_Number": null,
    "reported_Women_AssociatedStaff_Number": null,
    "reported_Men_AssociatedStaff_Number": null
},
```