FångstDataBanken

En administrativ webbplats för Sportfiskarna

FångstDataBanken

An administrative web site for The Swedish Anglers Association

Jesper Hellberg
Petter Jönsson

Fakulteten för hälsa, natur- och teknikvetenskap

Datavetenskap

C-uppsats 15 hp

Handledare: Donald F. Ross

Examinator: Thijs Jan Holleboom

Oppositionsdatum: 2015-06-02

Sammanfattning

Föreningen Sportfiskarna samlar in statistik över fisket i Sverige för användning inom forskning och förvaltning. I och med lanseringen av applikationer för smarta telefoner har antalet inrapporterade fångster ökat och ett behov av att hantera data både till och från dessa applikationer har uppstått. Denna uppsats beskriver skapandet av Sportfiskarnas webbplats FångstDataBanken. Webbplatsens huvudsakliga syfte är att administrera informationen som finns tillgänglig i de mobila applikationerna men också att kunna hantera de inrapporterade fångsterna för statistiska ändamål.

Projektets resultat är en webbplats skriven i ASP.NET MVC som tillfredsställer Sportfiskarnas behov. Webbplatsen utgör kärnan i Sportfiskarnas system för att hantera fångstrapportering och kommer att tas i bruk under sommaren 2015.

Abstract

The Swedish Anglers Association (Sportfiskarna) collects statistics on fishing in Sweden for use in research and management. With the launch of applications for smart phones, the number of reported catches has increased and the need to manage data both to and from these applications has arisen. This dissertation describes the creation of Sportfiskarna's website FångsDataBanken. The main purpose of the site is to administer the information provided in the mobile applications but also to deal with the reported catches for statistical purposes.

The result of the project is a website written in ASP.NET MVC that satisfies the customer's requirements. The site is the core of Sportfiskarna's system to manage the catch reporting and will become operational in the summer of 2015.

Förord

Många personer har hjälpt oss på olika sätt under detta examensarbete. Vi vill tacka personalen på Sogeti i Karlstad som tagit sig tid att hjälpa oss när det behövts.

Ett extra tack vill vi rikta till Mats Persson på Sogeti som varit vår handledare på plats under projektet och Donald F. Ross på Karlstads Universitet som varit handledare för uppsatsen. Tack också till Robert Åhs som hjälpte oss med att förstå hur databasen, WebApi:t och iOS applikationen hängde samman och Onur Jadermark som granskade vår kod.

Slutligen vill vi tacka vår kund Peter Belin från Sportfiskarna för stort engagemang och för lektionen i flugfiske.

Innehåll

Figurer

Tabeller

1 Introduktion

Föreningen Sportfiskarna för statistik över fisket i Sverige. Detta görs för att skaffa kunskap om till exempel olika artbestånd och miljön i våra sjöar och vatten. All data samlas i databasen ”FångstDataBanken”. Det är Sportfiskarna som sköter insamlandet av data till FångstDataBanken men innehållet är intressant även för andra. Några som får del av statistiken är Havs- och vattenmyndigheten, Sveriges Lantbruksuniversitet och olika Länsstyrelser.

För att göra det enklare för fiskare att rapportera sina fångster skapades under 2014 en applikation för Androidtelefoner och en motsvarande applikation för iOS är under utveckling. Införandet av mobila applikationer har ökat antalet rapporter till FångstDataBanken och Sportfiskarna behöver nu ett sätt att kunna administrera den data som används av applikationerna för att kunna utöka dem med till exempel nya vatten och arter. Man behöver också ett smidigt sätt att hämta ut statistik ur databasen.

1.1 Uppdraget

Detta projekt går ut på att bygga en administrativ webbplats åt föreningen Sportfiskarna. På webbplatsen skall en administratör från Sportfiskarna kunna lägga till eller uppdatera data som till exempel fiskevatten, arter och/eller fångstmetoder. De ändringar som görs på webbplatsen skall uppdatera de mobila applikationerna. Administratören skall också kunna hämta ut all data ur databasen i ett format som kan användas i andra program för statistiska beräkningar, presentationer eller liknande.

Utöver de administrativa delarna skall webbplatsen också kunna användas av fiskare för att se enklare statistik över de fångster de rapporterat in via de mobila applikationerna.

Projektets mål har varit att skapa en fungerande webbplats åt Sportfiskarna. Webbplatsen skall tas i bruk vid projektets slut.

1.2 Summering

Projektet har resulterat i en webbplats som kommer att tas i bruk i samband med lanseringen av applikationen för iOS någon gång under sommaren 2015. Alla krav som ställts på webbplatsen är uppfyllda och all önskad funktionalitet är implementerad.

1.3 Disposition

I kapitel 1 ges en kortare introduktion till projektet.

I kapitel 2 ges en mer detaljerad bakgrund samt en översikt av projektets delar. Kapitlet innehåller också en översikt av de olika tekniker och verktyg som använts.

Kapitel 3 beskriver olika aspekter av projektets design. Här tas webbplatsens design upp ur ett användarperspektiv, såväl den visuella och funktionella designen behandlas. En stor del av kapitlet ägnas åt databasens design. Kapitlet avslutas med en beskrivning av kodstrukturen.

Kapitel 4 ägnas åt implementationsdetaljer. Här tas intressanta delar av webbplatsen upp i detalj tillsammans med kodexempel och förklaringar.

I kapitel 5 utvärderas resultaten av projektet.

Kapitel 6 innehåller en utvärdering av projektet, lärdomar och tankar om webbplatsens eventuella vidareutveckling.

Bilaga 1 innehåller kravspecifikationen för projektet.

Bilaga 2 är ett kodexempel för att jämföra kodmängd för olika implementationer av databaskod.

2 Bakgrund

I kapitel två beskrivs först företaget Sogeti och organisationen Sportfiskarna. Syftet med fångstrapportering tas upp tillsammans med en beskrivning av uppdraget och en genomgång av tillgängligt material. En introduktion till de tekniker och produkter som använts för att utveckla webbplatsen ges också.

2.1 Sogeti

Sogeti är ett helägt dotterbolag till Cap Gemeni SA. Företaget verkar i 15 länder och har ungefär 20 000 anställda. I Sverige finns 1150 anställda fördelat på 21 kontor och på kontoret i Karlstad verkar ett 60-tal medarbetare [1].

Företaget tillhandahåller IT-konsulttjänster på den lokala marknaden medan Cap Gemini tar hand om den globala marknaden.

Kontoret i Karlstad utvecklar lösningar inom industriellt IT, business intelligence, webb och integration, främst med hjälp av Microsofts plattformar .NET och SQL Server.

2.2 Sportfiskarna

Sveriges Sportfiske- och Fiskevårdsförbund, ofta kallat Sportfiskarna, är en ideell organisation vars främsta mål är att främja sportfisket samt värna om rena vatten och friska fiskbestånd. De företräder också sina medlemmar i politiska frågor rörande fiske som till exempel utbyggnaden av vattenkraft [2].

Organisationen består av ca 50 000 medlemmar och har anställd personal fördelad på sju kontor, bland annat ett regionkontor i Forshaga med inriktning på fisket i Klarälven och Vänern.

2.3 Fångstrapportering

Sportfiskarna för statistik över fångad fisk som ett led i sitt mål att ”värna och vårda vatten”. Statistiken ska ge ett underlag för forskning och förvaltning av vatten samt utveckla sportfisket i stort. Data samlas i ”FångstDataBanken” och insamlingen sker i form av fångstrapporter från en applikation för Androidtelefoner som utvecklades som ett examensarbete [3] under våren

2014. En applikation för iOS enheter är under utveckling och kommer att tas i bruk under sommaren 2015.

I applikationerna skapar användaren ett personligt konto och kan sedan rapportera sina fångster tillsammans med relevant information såsom art, längd, vikt, fångstplats och dylikt. För att göra applikationerna mer attraktiva för användare fungerar de, utöver att rapportera fångster till FångstDataBanken, även som fiskedagbok för användaren. I applikationerna kan användaren också få information om fiskarter och märkning samt se vilka fiskeregler som gäller i ett specifikt vatten.

2.3.1 Terminologi

Den största enheten i systemet är ett ”vatten”. Ett vatten kan vara till exempel en sjö eller en del av ett kustområde. Ett vatten kan delas in i olika zoner för att mer exakt kunna avgöra förekomsten av olika arter i olika delar av vattnet. Till varje vatten finns en mängd fiskarter, fångstmetoder och informationstexter kopplade. Exempelvis är sjön Vänern ett vatten bestående av 10 zoner. I Vänern finns arterna Abborre, Asp, Gädda, Gös, Lax och Öring. De tillåtna fångstmetoderna är: fluga, ismete, jerk, mete, pimpel, trolling och vertikal.

Vissa fiskarter odlas och för att kunna identifiera sådana fiskar klipper fiskodlare fiskens fenor. Det finns två olika varianter av fenklippning: fettfeneklippning och bukfeneklippning. I fångsrapporteringen fyller man i klipptyp för att på så sätt kunna spåra fiskens ursprung. En klarälvslax har till exempel klippt fettfena medan en gullspångslax har både klippt fettfena och bukfena.

En ”fångst” är en fisk av en bestämd art, fångad med en fångstmetod i en zon i ett vatten. Fångsten ska ha en längd och kan också ha en vikt. Om fångsten är av en odlad art kan den ha någon eller båda fenklippningsvarianterna. Slutligen kan en fångst vara upptagen eller tillbakasläppt.

2.4 Uppdraget

Uppdraget har varit att bygga en administrativ webbplats för kunden. Sportfiskarna behöver på olika sätt kunna administrera informationen från applikationerna men man har också behov av att kunna hantera information till applikationerna.

Information från applikationerna består av fångstrapporter. Denna information behöver man kunna exportera för att göra tillgänglig för statistik och forskning.

Informationen till applikationerna kan exempelvis bestå av artbeskrivningar eller information om eventuella begränsningar av fisket på en viss plats. Det ska också finnas möjlighet att utöka applikationerna genom att lägga till nya fiskevatten med tillhörande information som till exempel zoner, arter och fångstmetoder i databasen.

De fiskezoner som visas i applikationerna är baserade på koordinater. Kunden har önskemål om ett verktyg för att kunna ändra och skapa nya zoner genom ett grafiskt gränssnitt. Målet är att kunna rita ut rektanglar på en karta och automatiskt få koordinater. Man ska också kunna ändra storlek och flytta dessa rektanglar samt ge dem egna zonnamn.

Webbplatsens huvudsakliga fokus ligger på de administrativa delarna och kräver en användare med administrativa rättigheter. Administratören får exportera data och göra ändringar som påverkar applikationerna. Utöver den administrativa delen skall det också finnas möjlighet för användare av applikationerna att logga in för att besöka sin fiskedagbok och se statistik över sitt eget fiske. Det skall också finnas en bloggfunktion där administratören kan skriva inlägg som kan läsas av användarna. Under projektets gång tillkom önskemål om att ge utvalda användare möjlighet att exportera rådata för specificerade vatten.

I ett tillägg till uppdraget fanns också önskemål om att användare ska kunna se utökad statistik över sitt fiske i jämförelse med andra användare.

2.5 Tillgängligt material

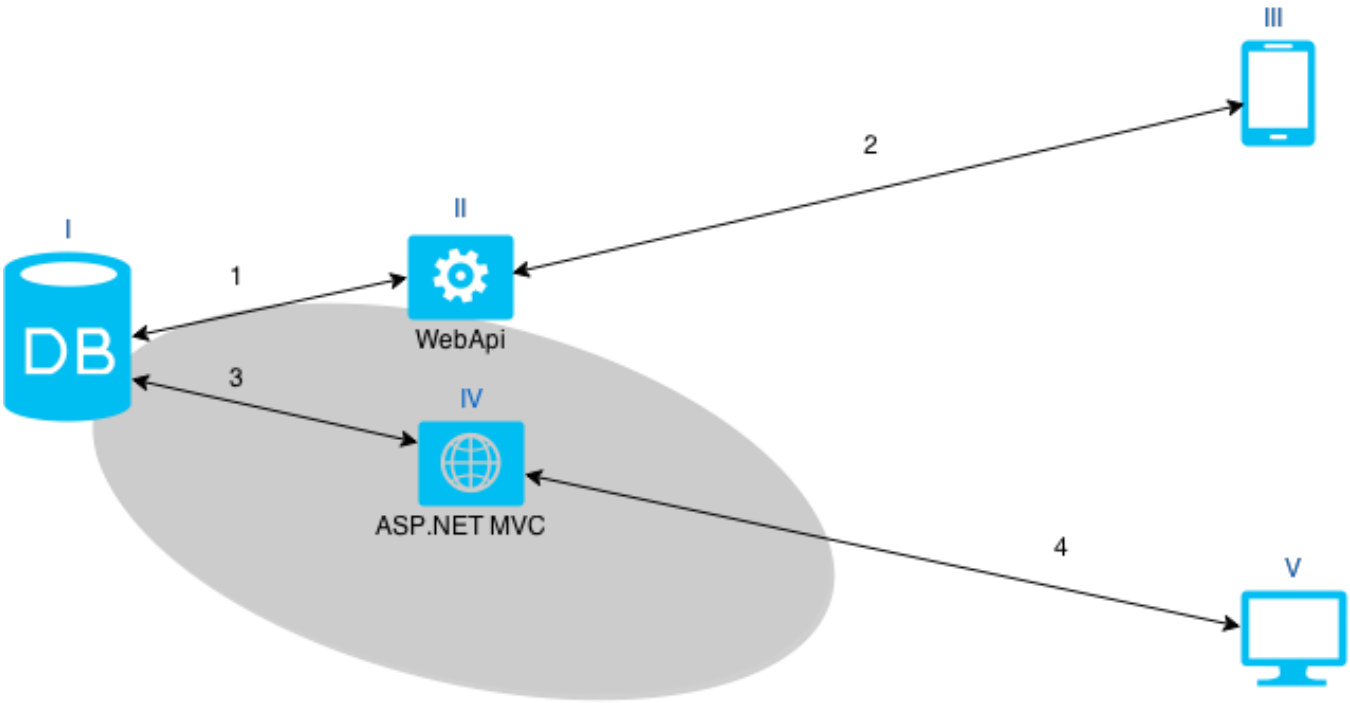
Vid projektets start fanns en fungerande databas som användes av mobilapplikationerna. För att kommunicera med databasen fanns också ett fungerande webbaserat API. Både databasen och API:t behövde byggas ut för att hantera den nya funktionalitet som en administrationsdel tillför.

En kravlista samt analyser av webbplatsens användarvänlighet fanns tillgängliga från ett pågående examensarbete (*Bilaga 1 – Kravspecifikation*).

Rapport och källkod från examensarbetet med Androidapplikationen fanns tillgängliga. En del av materialet var dock utdaterat i och med att Androidapplikationen skall uppdateras till att använda samma databas och API som används av den nya iOS applikationen.

Utvecklingen av iOS applikationen pågick på Sogeti parallellt med utvecklingen av webbplatsen och utvecklingsversioner av applikationen fanns tillgängliga under arbetet.

2.6 Översikt



Figur 2.1: Översikt av systemet

Figur 2.1: Översikt av systemet visar hur systemets delar hänger ihop. Den grå elipsen symboliserar projektets omfattning.

Databasen (del I) är den relationsdatabas [4] som systemet arbetar mot. Del II är API:t som de mobila applikationerna använder för att kommunicera med databasen. Del II ingår som en del i projektet då viss modifiering behöver göras för att tillgodose webbplatsens behov. Del III är en användare via en mobil applikation. Del IV är webbplatsen som utgör kärnan i projektet. Del V representerar användare som loggar in på webbplatsen.

Koppling 1 och 3 använder SQL [5] för att spara och läsa data till och från databasen. 2 och 4 utgörs av HTTP- anslutningar [6].

2.7 Tekniker

Detta avsnitt innehåller kortare beskrivningar av de olika tekniker och designmönster som är relevanta för projektet.

2.7.1 MVC

MVC [7] är ett designmönster där ett program separeras till de tre sammanhängande komponenterna Model, View och Controller. I Model-komponenten finns all logik och data. Om programmet använder en databas hanteras kopplingen mellan programmet och databasen av Model. View-komponenten hanterar användargränssnittet. Controller-komponenten hanterar interaktionen med användaren. Den tar emot information, behandlar den och skickar den vidare till Model eller View.

Genom sin tydliga uppdelning av program drar MVC nytta av designprincipen ”Separation of concerns” [8]. ”Separation of concerns” underlättar utveckling och underhåll av systemet genom att ge utvecklare möjlighet att fokusera på en del av programmet utan att behöva ha full kunskap om hela systemet.

En annan fördel med MVC är att testning av program underlättas jämfört med andra tekniker som till exempel ASP.NET Web Forms [9] [10].

MVC beskrivs i mer detalj i kapitel *4.1 Informationsflöde*

2.7.2 ASP.NET

ASP.NET [9] är Microsofts ramverk för att utveckla dynamiska websidor. ASP.NET använder Common Language Runtime [11] och kan därför programmeras med valfritt .NET språk, till exempel C# [12] eller Visual Basic.NET [13].

2.7.3 ASP.NET MVC Framework

ASP.NET MVC [14] utökar ASP.NET för att hantera designmönstret MVC.

ASP.NET MVC bygger på mjukvarudesignparadigmen ”convention over configuration” [15]. Paradigmen innebär att så länge en utvecklare håller sig inom ramverkets konventioner behövs ingen extra kod skrivas för att koppla samman programmets delar. I ASP.NET MVC yttrar det sig till exempel genom att ett anrop till URI *http://host/Person* anropar metoden Index i

klassen PersonController medan ett anrop till *http://host/Person/Delete/1* anropar metoden Delete i PersonController och skickar med argumentet ”1” till metoden.

MVC och ASP.NET används på serversidan (IV, *Figur 2.1: Översikt av systemet*).

2.7.4 ASP.NET Web API

Ramverket ASP.NET Web API [16] utökar ASP.NET MVC med funktioner för att implementera HTTP-baserade tjänster som kan anropas av webbsidor via AJAX (Asynchronous JavaScript and XML) [17] eller av applikationer på mobila enheter.

ASP.NET Web API gör det enkelt att exponera en databas funktioner för att skapa, läsa, ändra och ta bort poster genom att mappa dessa till metoder i HTTP-protokollet [18]. Dessa operationer går under samlingsnamnet CRUD (Create, Read, Update, Delete) [19]. I kombination med HTTP kallas tekniken REST [20].

Operation	**URI**	**HTTP-metod**	**SQL-metod**
Visa personer	/Person	GET	SELECT
Skapa ny person	/Person	POST	INSERT
Visa person 1	/Person/1	GET	SELECT WHERE 1
Redigera person 1	/Person/1	PUT	UPDATE WHERE 1
Ta bort person 1	/Person/1	DELETE	DELETE WHERE 1

Tabell 2.1: Mappning mellan operationer och metoder

Liksom ASP.NET MVC använder Web API paradigmen ”convention over configuration” för att minska mängden kod som behöver skrivas. Konventionen skiljer sig dock lite eftersom Web API använder sig av fler HTTP-metoder än de vanliga GET och POST. Ett HTTP-anrop av typen GET till URI *http://host/Person/1* anropar till exempel den metod i klassen PersonController som har ett namn som börjar på Get, till exempel GetPerson, och som tar ett argument. Ett HTTP-anrop av typen DELETE till samma URI anropar istället en metod vars namn börjar på Delete och tar ett argument, till exempel DeletePerson(int id).

Web API används mellan de mobila applikationerna och databasen (II, *Figur 2.1: Översikt av systemet*).

2.7.5 Entity Framework

Entity Framework [21] är ett ramverk för objekt/relationsmappning [22] skapat av Microsoft. Objekt/relationsmappare kopplar ihop klasser med bakomliggande databastabeller. Detta innebär att man som utvecklare kan koncentrera sig på att skriva domänspecifika klasser och sedan låta ramverket koppla samman dessa klasser med en underliggande databas.

Med hjälp av ”Code First” [23] är det även möjlig att låta ramverket skapa den underliggande databasen utifrån klasserna definierade i programmet.

Viss Entity Framework-terminologi kan vara bra att lägga på minnet inför kapitel *4.4 Repository*. Ett set i Entity Framework motsvarar innehållet i en tabell i den underliggande databasen. Ett objekt motsvarar en tupel (rad) i den underliggande databastabellen. Objekt är en instans av en modellklass vars egenskaper motsvarar de attribut den underliggande databastabellen har.

Entity Framework används i kopplingen till databasen från webbplatsen och Web API (1 och 3, *Figur 2.1: Översikt av systemet*).

2.7.6 Bootstrap

Bootstrap [24] är ett ramverk för utveckling av användargränssnitt i HTML [25], CSS [26] och JavaScript [27] ursprungligen utvecklat av Twitter. Ramverket underlättar utvecklandet av responsiva webbsidor som fungerar både på datorer och mobila enheter oberoende av skärmstorlek och inmatningsmetod.

2.7.7 Knockout

Knockout [28] är ett JavaScriptbibliotek som bygger på Model-View-ViewModel [29] mönstret. I detta projekt har det använts för dess enkelhet att koppla samman komponenters innehåll, form och funktion genom databindning. Exempel på hur Knockout använts finns i kapitlet *4.6.4 Initiering av Knockout*.

2.7.8 Data-Driven Documents

Data-Driven Documents (D3) [30] är ett JavaScriptbibliotek för manipulering och presentation av data. Data binds till en Document Object Model (DOM) [31] som sedan kan behandlas och

presenteras. För att ge tillgång till mer avancerad grafik än vad som finns tillgänglig i HTML använder D3 Scalable Vector Graphics (SVG) [32] för att visa grafik.

På webbplatsen används D3 för att generera stapeldiagram på statistiksidan (*Figur 3.3: Statistiksidan*). Exempel på implementation finns i *4.7 Statistik*. Det finns en stor variation av presentationsmöjligheter att välja mellan om man i framtiden vill vidareutveckla statistikfunktionerna.

2.7.9 CKEditor

CKEditor [33] är ett JavaScriptbibliotek som används för att möjliggöra enkel redigering av HTML-innehåll. Editorn tillhandahåller WYSIWYG-editering (What You See Is What You Get) av innehåll vilket innebär att användaren kan styra innehållets utseende utan att kunna HTML.

På webbplatsen används CKEditor för redigering av bloggposter (*3.1.4.2 Inlägg*) och infotexter.

Bootstrap, Knockout, D3 och CKEditor exekveras på klientsidan (V, *Figur 2.1: Översikt av systemet*).

2.7.10 Google Maps

Google Maps [34] är en karttjänst från Google som utöver vanliga kartor i form av illustrationer och satellitbilder även innehåller verktyg för ruttplanering samt möjlighet att se platser från gatunivå.

Google tillhandahåller API:er både för mobila plattformar och för webb som gör det möjligt för utvecklare att integrera Google Maps i sina egna produkter [35]. Vid webbutveckling används JavaScript för att nyttja API:t och ge möjlighet att rita på kartan. Det går att rita ut många typer av information från de enklaste intressepunkter [36] och geometriska figurer [37] till avancerade värme- [38] och trafiksituationskartor [39] i ett eller flera lager.

Google Maps är gratis att använda så länge webbsidan inte genererar mer än 25 000 kartvisningar per dag [40].

2.8 Verktyg

Här följer en genomgång av de verktyg som använts i projektet.

2.8.1 Visual Studio 2013

All källkod för projektet skrevs i Visual Studio 2013 [41] som är en integrerad utvecklingsmiljö [42] från Microsoft. Visual Studio kan bland annat användas för utveckling av vanliga program, webb- och molnlösningar. Visual Studio har inbyggt stöd finns för de vanligaste språken inom .NET-utveckling och kan med hjälp av tillägg byggas ut för användning med andra språk.

2.8.2 Team Foundation Server

Team Foundation Server (TFS) [43] är Microsofts produkt för att underlätta utveckling av mjukvara. TFS innehåller flera delar som inte är direkt kodrelaterade som till exempel verktyg för planering, rapportering, testning och versionshantering.

I detta projekt användes TFS tillsammans med Visual Studio för att sköta versionshanteringen av källkoden. TFS användes också för att versionshantera dokument och för att planera projektet.

2.9 Sammanfattning

Detta kapitel har beskrivit uppdraget, bakgrunden, förutsättningar samt tekniker som använts för projektet. En översikt av systemets olika delar har kopplats till dessa beskrivningar.

3 Design

I detta kapitel beskrivs projektets design. Fokus ligger på de delar av systemet som direkt ingår i projektet men även systemet som helhet behandlas. Avsnitt 3.1 handlar om webbplatsens design. Där ges en översiktlig bild av webbplatsens funktionalitet och användargränssnitt. I avsnitt 3.2 presenteras databasdesignen med de olika tabellerna och avsnitt 3.3 behandlar kodstrukturen.

3.1 Webbplatsens design

Webbplatsens design har huvudsakligen utgått från tillgänglig kravspecifikation. Användare kan logga in på webbplatsen med de inloggningsuppgifter de använder på de mobila applikationerna. Det finns tre typer av användarkonton med olika rättigheter: administratör, rådatabehörig och användare. Designmässigt skiljer sig användarkategorierna åt genom att ha olika menyalternativ tillgängliga i navigationslisten.

3.1.1 Icke inloggad användare

Figur 3.1: Meny för icke inloggad användare

En användare som inte har loggat in på webbplatsen kan skapa ett nytt användarkonto, läsa blogginlägg, se en text om syftet med FångstDataBanken samt hitta kontaktuppgifter till Sportfiskarna. För att få tillgång till fler funktioner krävs att användaren loggar in med sina användaruppgifter.

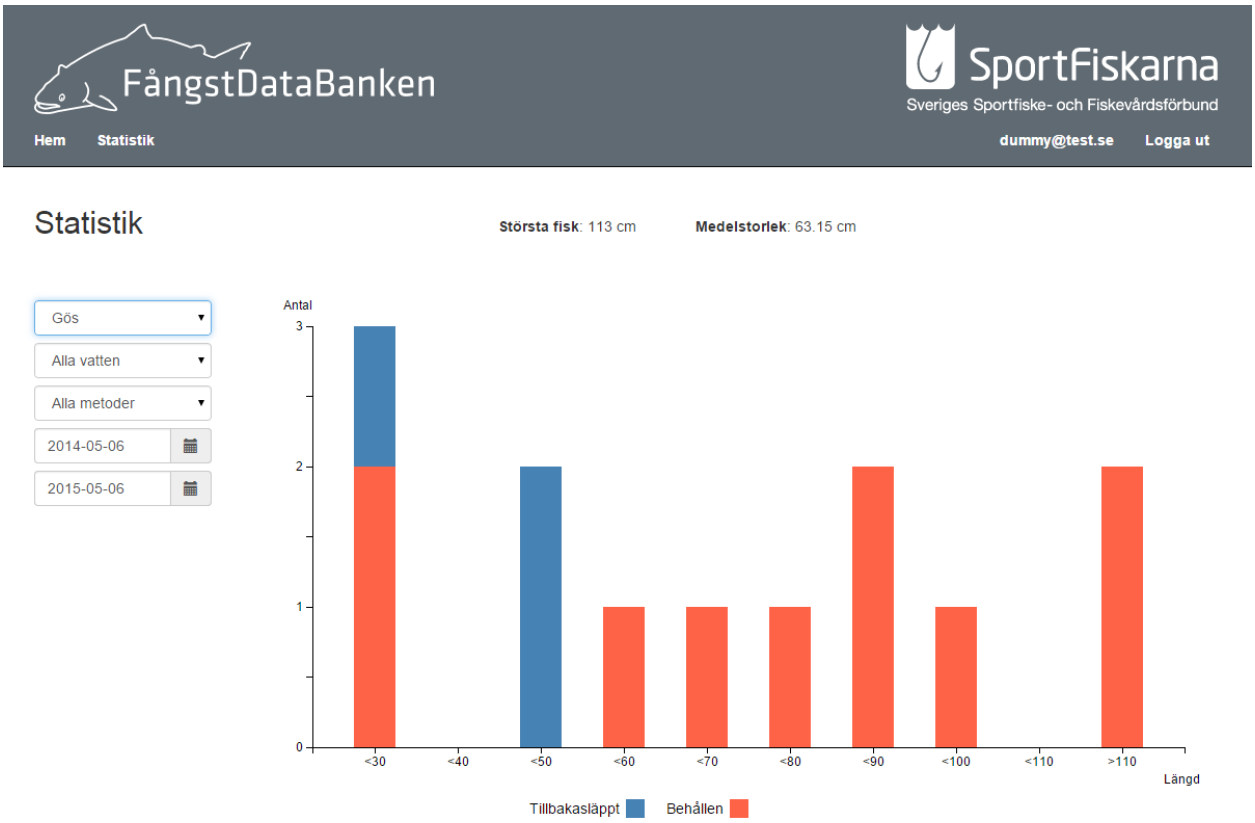
3.1.2 Inloggad användare

En inloggad användare är en användare som loggat in på webbplatsen med sina användaruppgifter. Användaruppgifterna är desamma för webbplatsen som för de mobila applikationerna.

Figur 3.2: Meny för inloggad användare

3.1.2.1 Statistik

Utöver den information en icke inloggad användare har tillgång till, kan en inloggad användare se statistik över sitt fiske. Statistiken visas i ett stapeldiagram med möjlighet att filterera data utifrån art, vatten, fångstmetod och tidsintervall. När användaren ändrar sina val i menyerna ritas diagrammet om och skalan på y-axeln ändras dynamiskt för att motsvara antalet fiskar i det aktuella urvalet. Utöver diagrammet innehåller sidan också information om användarens största fångst samt fångsternas medelstorlek i urvalet.



Figur 3.3: Statistiksidan

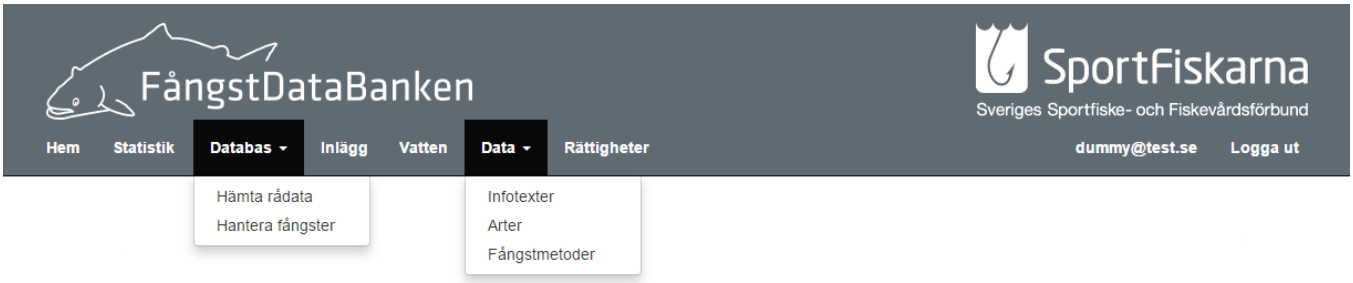
3.1.3 Inloggad rådatabehörig

Figur 3.4: Meny för rådatabehörig användare

En rådatabehörig användare har samma rättigheter som en inloggad användare men har dessutom möjligheten att hämta data direkt ur databasen för ett eller flera specificerade vatten. I den ursprungliga kravspecifikationen skulle endast administratörer ha tillgång till databasen men under projektets gång kom önskemål från kunden att införa ytterligare en användarnivå med möjlighet att kunna hämta rådata för ett specifikt vatten. Detta är en funktion som kan användas när en fiskeklubb vill ha tillgång till rådata från ett lokalt vatten men inte ska ha fullständiga administratörsrättigheter.

Data exporteras i CSV-format [44] och kan användas i olika externa program till exempel för statistikberäkningar.

3.1.4 Inloggad administratör



Figur 3.5: Menyalternativ för administratör

En administratör är en person som har tillgång till alla funktioner på webbplatsen. Detta är tänkt att vara en eller flera representanter från Sportfiskarna. En användare inloggad som administratör har samma menyalternativ som en rådatabehörig användare och dessutom ett flertal möjligheter att uppdatera och förändra information både på webbplatsen och i databasen, och därigenom i de mobila applikationerna. Detta är den centrala delen av projektet eftersom det är genom dessa funktioner som kunden själv kan påverka de mobila applikationerna och hämta information ur databasen.

3.1.4.1 Databas

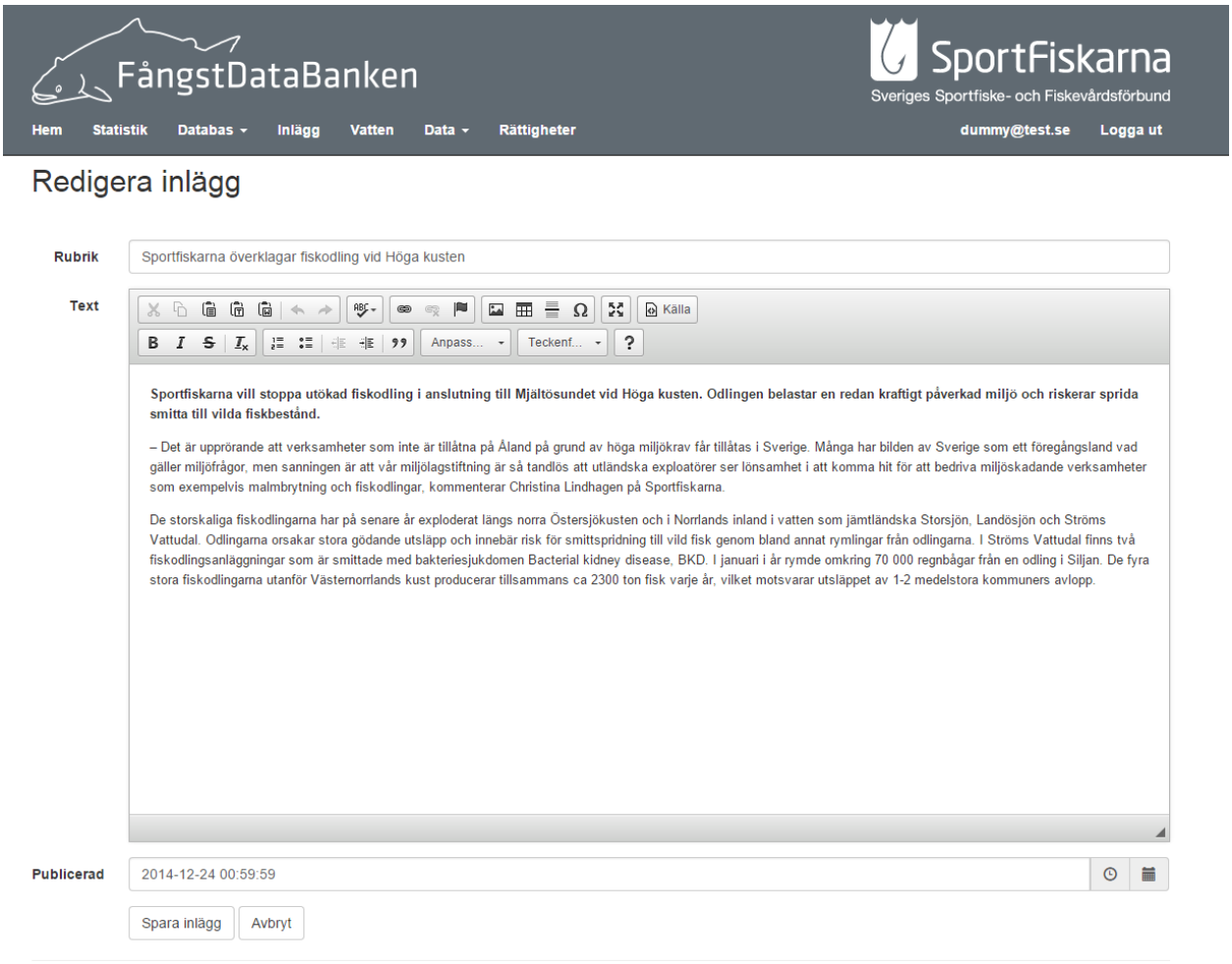
Under menyn ”Databas” finns funktioner för att ”Hämta rådata” ur databasen samt ”Hantera fångster”.

Rådata hämtas på samma sätt som för en rådatabehörig användare (*3.1.3 Inloggad rådatabehörig*) med skillnaden att en administratör hämtar data ur samtliga vatten.

Under ”Hantera fångster” kan administratören se alla fångster i databasen och efter behov redigera eller radera dem. Detta kan vara användbart för att ta bort uppenbart felaktiga fångstrapporter ur databasen.

3.1.4.2 Inlägg

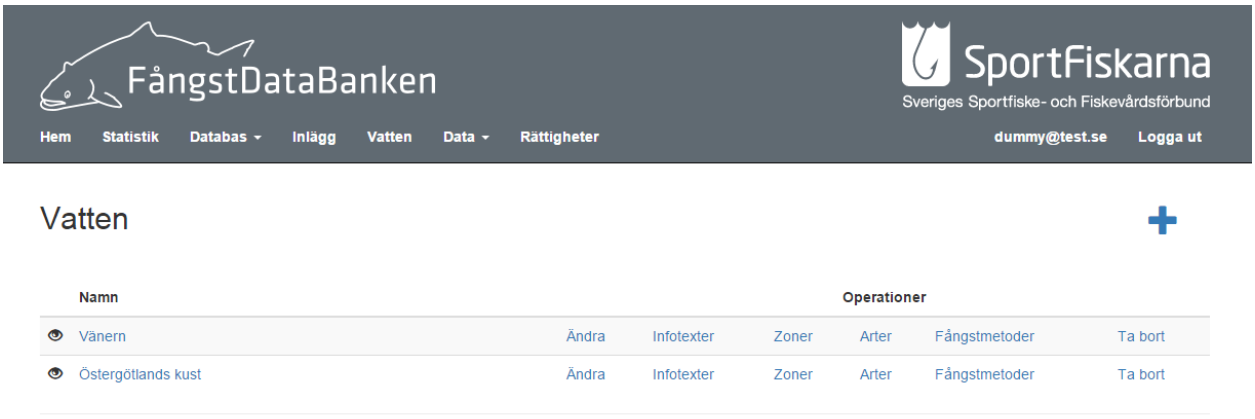
Under menyvalet ”Inlägg” kan administratören hantera bloggen som andra användare kan se på startsidan. Som syns i *Figur 3.6* kan inlägg skapas och redigeras av administratören utan att HTML-kunskap krävs (Se *2.7.9 CKEditor*).



Figur 3.6: Inlägg

Inlägg syns inte på startsidan förrän de blivit publicerade. Detta innebär att en eller flera administratörer kan arbeta på ett inlägg under längre tid och sedan göra det synligt för allmänheten först när inlägget är klart. För att publicera inlägg sätts en publiceringstidpunkt. Inläggen visas inte på startsidan förrän tidpunkten passerats vilket gör det möjligt att automatiskt publicera ett inlägg vid en framtida eventuellt obekväm tid som till exempel midnatt.

3.1.4.3 Vatten

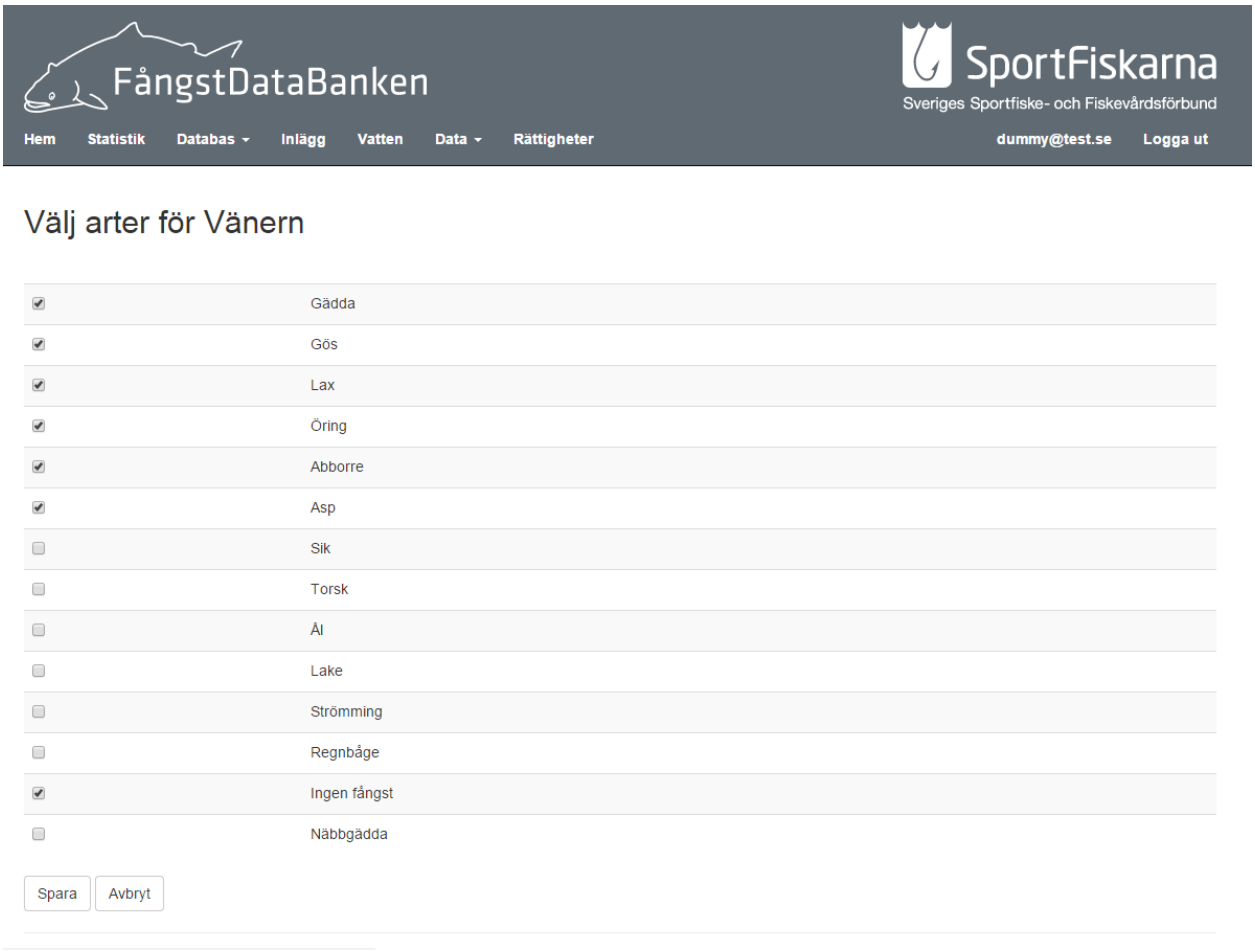


Figur 3.7: Vattenmenyn

En av huvuduppgifterna för en administratör är att hantera de vatten som finns i databasen. Under menyalternativet ”Vatten” (*Figur 3.7: Vattenmenyn*) finns en lista över alla tillgängliga vatten och operationer på dessa. Här kan administratören skapa nya vatten och ändra eller ta bort befintliga vatten. Här finns också länkar för att koppla infotexter, zoner, arter och fångstmetoder till ett vatten.

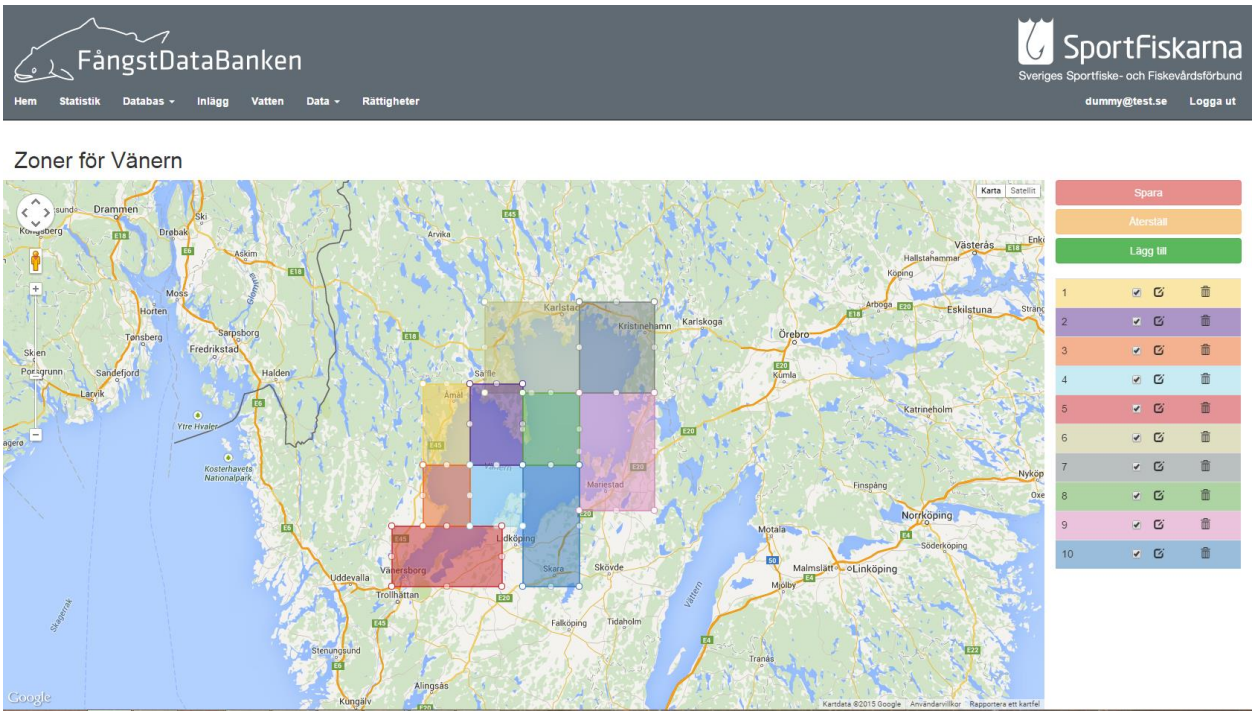
För att ett vatten ska kunna användas till fångstrapportering behöver det minst innehålla fiskarter, fångstmetoder och zoner. Ett vatten kan också innehålla informationstexter men dessa krävs inte för fångstrapporteringen utan ska snarare ses som en informationskanal till användaren.

Befintliga arter, fångstmetoder och infotexter kan läggas till ett vatten genom att följa motsvarande länk från vattenmenyn (*Figur 3.7: Vattenmenyn*). Gränssnittet för att lägga till arter till ett vatten visas nedan (*Figur 3.8: Gränssnitt för att lägga till arter i ett vatten*) och designen är snarlikt för tillägg av fångstmetoder och infotexter.



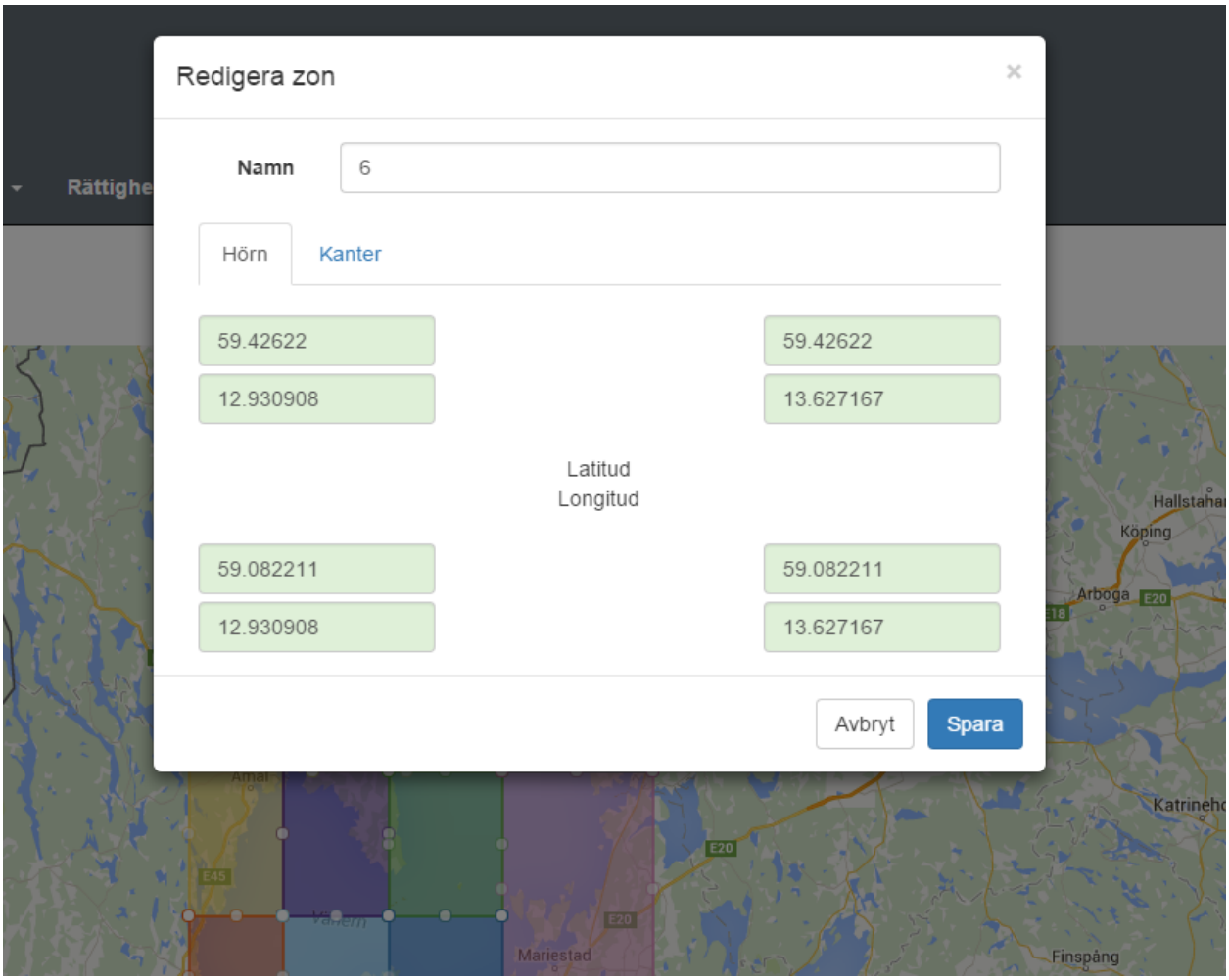
Figur 3.8: Gränssnitt för att lägga till arter i ett vatten

För att hantera ett vattens zoner följer man länken ”Zoner” från vattenmenyn (*Figur 3.7: Vattenmenyn*). Detta öppnar en karta (*Figur 3.9: Gränssnittet för att hantera zoner*) där administratören har möjlighet att rita in nya zoner och ändra storlek, flytta eller ta bort befintliga.



Figur 3.9: Gränssnittet för att hantera zoner

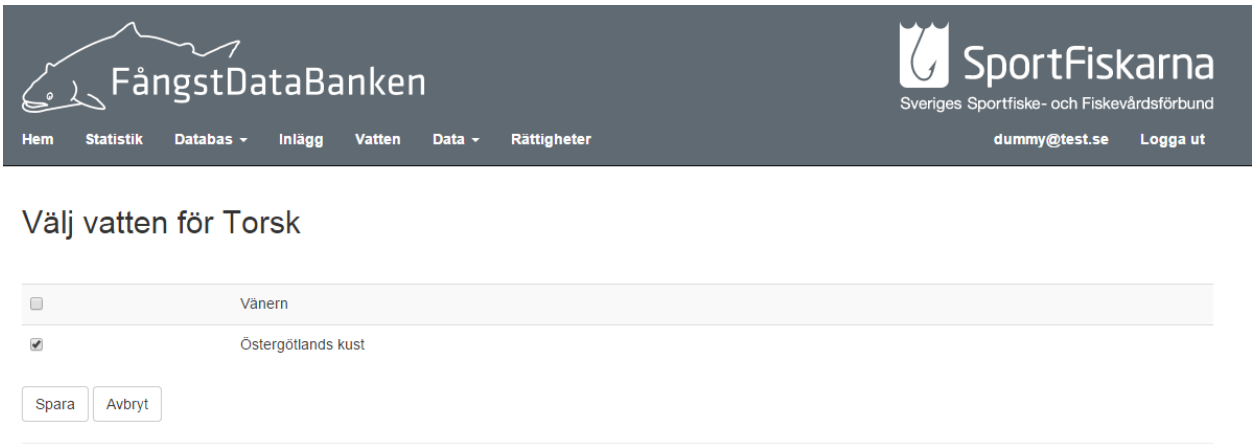
Namnet på en zon kan ändras antingen genom att högerklicka på en zon eller genom att klicka på redigeringsknappen i zonlistan för den zon som skall ändras. I dialogrutan som visas (*Figur 3.10: Gränssnitt för redigering av zondetaljer*) ges administratören även möjlighet att finjustera zonens position och storlek genom att mata in koordinater manuellt.



Figur 3.10: Gränssnitt för redigering av zondetaljer

3.1.4.4 Data

Under dropdown-menyn ”Data” finns separata menyer där administratören kan skapa och hantera ”Infotexter”, ”Arter” och ”Fångstmetoder”. Förutom att skapa nya poster finns även möjlighet att redigera, ta bort och associera posten till ett eller flera vatten (*Figur 3.11: Gränssnitt för artförekomst*). Informationstexterna kan sättas till generella och på så vis kan en informationstext höra till samtliga vatten.



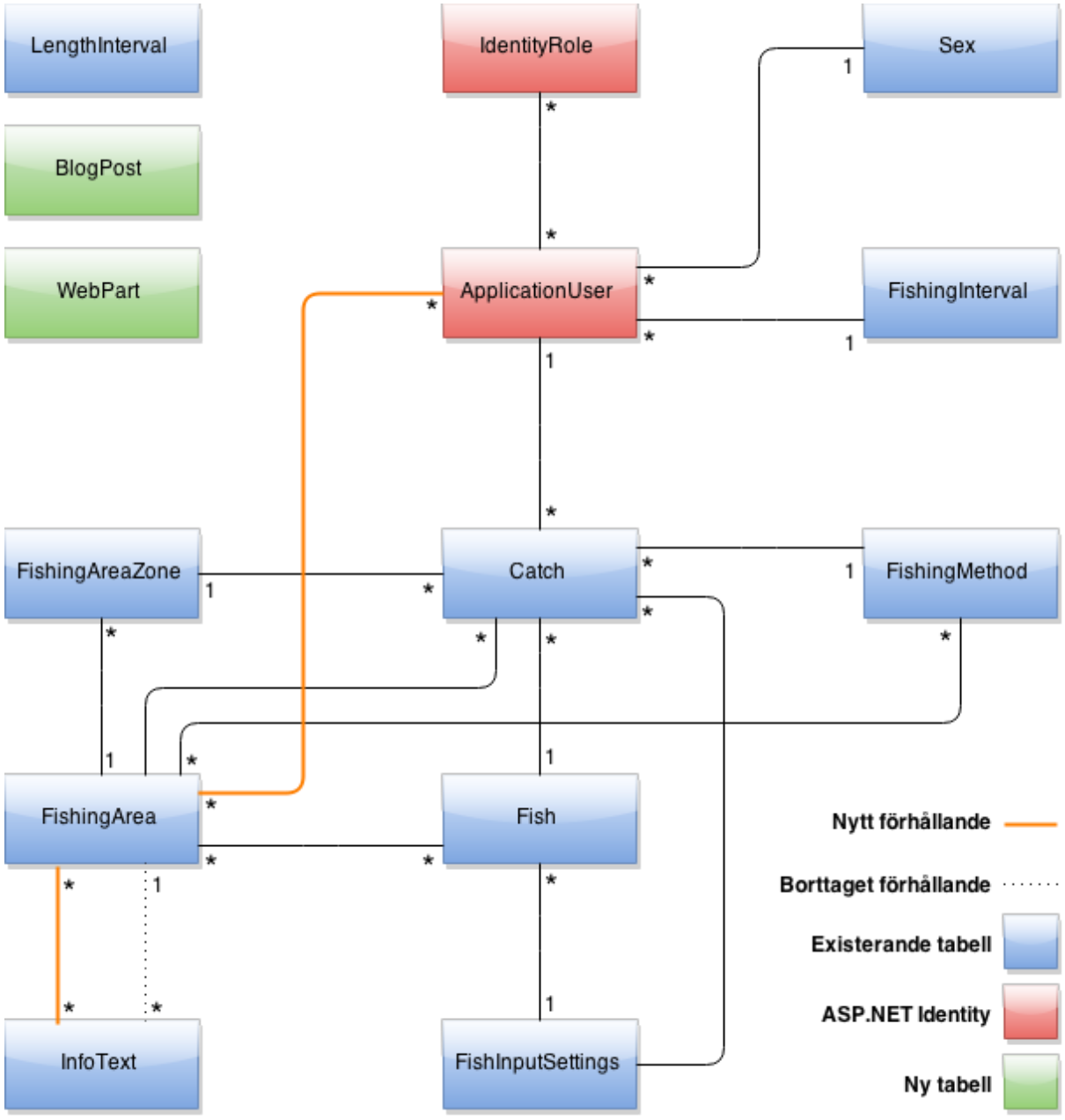
Figur 3.11: Gränssnitt för artförekomst

3.1.4.5 Rättigheter

För att dela ut rådatabehörighet och administratörsrättigheter används menyvalet ”Rättigheter”. Här kan en administratör tilldela befintliga användare rollen ”administratör”, ”rådatabehörig” eller ”ingen roll”. När en användare tilldelas rollen rådatabehörig ges möjlighet att associera användaren med ett eller flera vatten. Processen att dela ut rättigheter beskrivs ingående i *4.1.1 Ett typiskt flöde*.

3.2 Databasdesign

Vid projektets början fanns en fungerande databas skapad för mobilapplikationerna att tillgå. För att implementera webbsidans alla funktioner byggdes denna databas ut med nya tabeller och relationer. Databasen implementerades med hjälp av Entity Framework Code First [21] så en utbyggnad kunde ske genom att skapa nya domänspecifika klasser i C# [12].



Figur 3.12: Databasdesign

3.2.1 Ursprunglig databas

Den ursprungliga databasen innehöll tolv tabeller varav tio användes för fångstdata och två användes av ASP.NET Identity [45] för att lagra användare och roller.

Ytterligare tre tabeller skapades automatiskt av Entity Framework för att möjliggöra många till många förhållanden. Dessa var IdentityUserRole för att koppla ihop användare med roller, FishingAreaFish för att ange vilka fiskarter som finns i vilka vatten samt FishingMethodFishingArea för att ange vilka fiskemetoder som kan användas i vilka vatten. Dessa sammankopplingstabeller visas i *Figur 3.12: Databasdesign* som relationer med * i båda ändarna istället för som tabeller.

3.2.1.1 ApplicationUser

Tabellen ApplicationUser tillhör ASP.NET Identity. Den innehåller information om varje användare i systemet såsom dennes användarnamn, lösenord och e-postadress.

3.2.1.2 IdentityRole

Tabellen IdentityRole innehåller de roller som definierats för webbapplikationen till exempel administratör eller rådatabehörig.

3.2.1.3 Catch

Tabellen Catch innehåller alla fångster som rapporterats i systemet.

3.2.1.4 Fish

Tabellen Fish innehåller de fiskarter som skall kunna rapporteras.

3.2.1.5 FishingArea

Tabellen FishingArea innehåller de vatten som skall kunna väljas vid rapportering.

3.2.1.6 FishingAreaZone

Tabellen FishingAreaZone innehåller de zoner ett vatten delats in i. Zonerna är alltid rektangulära så tabellen behöver bara lagra koordinaterna för zonens nordvästra hörn samt zonens sydöstra hörn.

3.2.1.7 FishingInterval

Tabellen FishingInterval innehåller de olika fiskeintervall en användare kan använda för att beskriva hur ofta han/hon fiskar. Detta används vid registrering av nytt användarkonto. I skrivande stund kan man välja 1 gång per år, 1 gång per månad, 1 gång per vecka och 1 gång per dag.

3.2.1.8 FishingMethod

Tabellen FishingMethod innehåller de olika fiskemetoder som kan rapporteras. Vilka fiskemetoder som kan användas för ett visst vatten styrs via korskopplingstabellen FishingMethodFishingArea.

3.2.1.9 FishInputSettings

Tabellen FishInputSettings styr vilken typ av data som skall vara möjlig att registrera för en specifik typ av fisk. Ett exempel på detta är att det för ädelarter, till skillnad från standardarter, ges möjlighet att rapportera fenklippning. Ett annat exempel är småfisk där det är den totala vikten för fångsten som skall rapporteras, inte varje individuell fisk. Notera att en fisktyp inte är detsamma som en fiskart utan varje art tillhör en viss typ via en främmandenyckel i tabellen Fish.

3.2.1.10 InfoText

Tabellen InfoText används för att lagra informationstexter som visas i mobilapplikationerna. En tupel i InfoText kan antingen innehålla en fullständig text i form av ett HTML-dokument eller en URL till en resurs på Internet. Infotexten kan antingen vara generell och visas för alla vatten eller vara kopplad till ett specifikt vatten via en främmandenyckel.

3.2.1.11 LengthInterval

Tabellen LengthInterval används för att gruppera fångster efter längd för framställning av statistik.

3.2.1.12 Sex

Tabellen Sex definierar vilka kön en användare kan välja mellan. ApplicationUser har en främmandenyckel som refererar till denna tabell.

3.2.2 Tillägg

För att implementera de nya funktioner som önskades skapades två nya tabeller: BlogPost och WebPart. Utöver de nya tabellerna lades det till ett många-till-många förhållande mellan ApplicationUser och FishingArea för att understöda funktionen för rådatauthämtning där användare bara kan plocka ut rådata för de vatten de har tillgång till. Förhållandet mellan InfoText och FishingArea ändrades från en-till-många till många-till-många (se *4.2.1 Infotexter*).

3.2.2.1 BlogPost

Tabellen BlogPost används för att lagra de blogginlägg som kan skrivas av administratörer och som visas på startsidan. Förutom inläggens rubrik och innehåll finns också ett publiceringsdatum som används för att bestämma om ett inlägg skall visas på startsidan (Publiceringsdatum är satt och inte i framtiden) och i vilken ordning inläggen skall visas (Fallande publiceringsdatum).

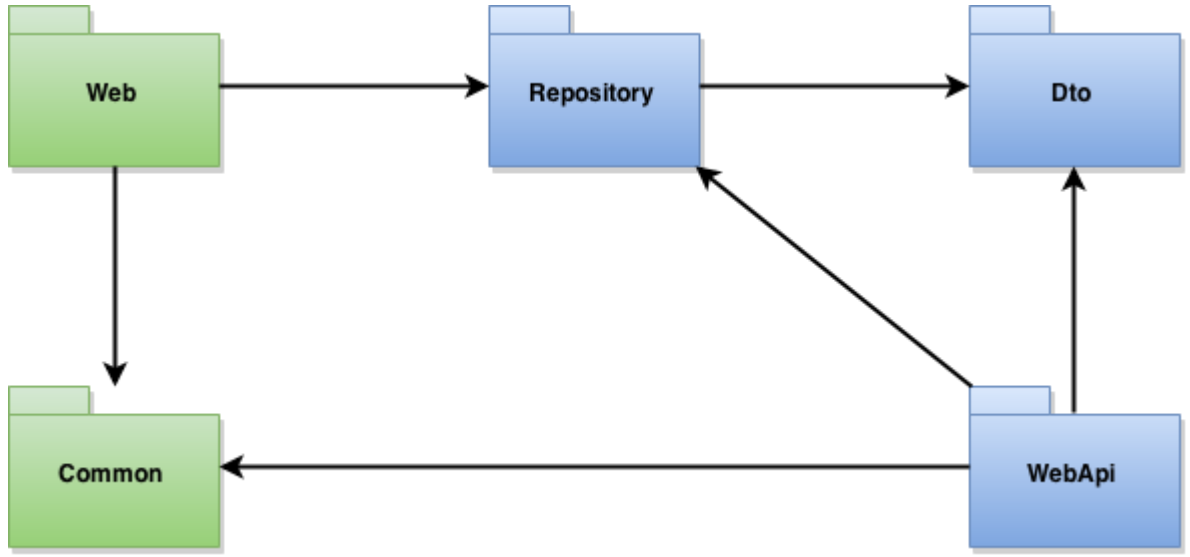
3.2.2.2 WebPart

Tabellen WebPart används för att lagra små fragment av HTML-kod som skall vara inkluderbara i sidor men inte vara fristående sidor i sig själva. Av kravspecifikationens punkter FU14P1 och FU15P1 (*Bilaga 1 – Kravspecifikation*) framgår att webbplatsens skall innehålla syftet med sidan samt kontaktuppgifter till sidans administratör. WebPart lades till främst för dessa punkter men kan mycket väl komma att användas i fler situationer om webbplatsen vidareutvecklas.

3.3 Kodstruktur

Projektets källkod kan delas in i fem olika moduler som presenteras i detta kapitel.

Vid projektets början fanns tre moduler (WebApi, Repository och Dto) implementerade som stöd för mobilapplikationerna. Majoriteten av den kod som skapades under projektets gång kom att vara för den nya modulen Web. I Repository lades nya tabeller till och vissa förhållanden förändrades (Se *3.2.2 Tillägg*). Behov uppstod också av att dela icke databasrelaterad kod mellan Web och WebApi. För detta lades modulen Common till (Se *4.5 ASP.NET Identity och modulen Common*).



Figur 3.13: Översikt av programmoduler och kopplingar

3.3.1 Dto

Dto, ofta skrivet DTO men här enligt .NET:s namnkonvention [46], betyder data transfer object [47]. Modulen innehåller enkla klasser menade att serialiseras [48] till JSON [49] för överföring mellan server och mobila enheter. Tack vare dessa klasser kan den underliggande databasen förändras utan att mobilapplikationerna påverkas så länge som det är möjligt att fylla Dto:ns egenskaper.

I kapitlet *4.2.1 Infotexter* beskrivs hur databasstrukturen för informationstexter förändrades utan att mobilapplikationerna påverkades.

3.3.2 Repository

Modulen Repository innehåller projektets datamodeller samt kod för databasåtkomst.

3.3.3 WebApi

Modulen WebApi innehåller de REST-tjänster som används av de mobila applikationerna. Modulen använder sig av Repository för databasåtkomst och Dto för serialisering av resultat.

3.3.4 Web

Modulen Web innehåller den webbapplikation för administrering av FångstDataBanken som implementerats i detta projekt.

3.3.5 Common

Modulen Common innehåller kod som delas mellan Web och WebApi men som inte passar in i Repository, alltså kod som inte är databasrelaterad. En beskrivning om varför modulen kom till finns i *4.5 ASP.NET Identity och modulen Common*.

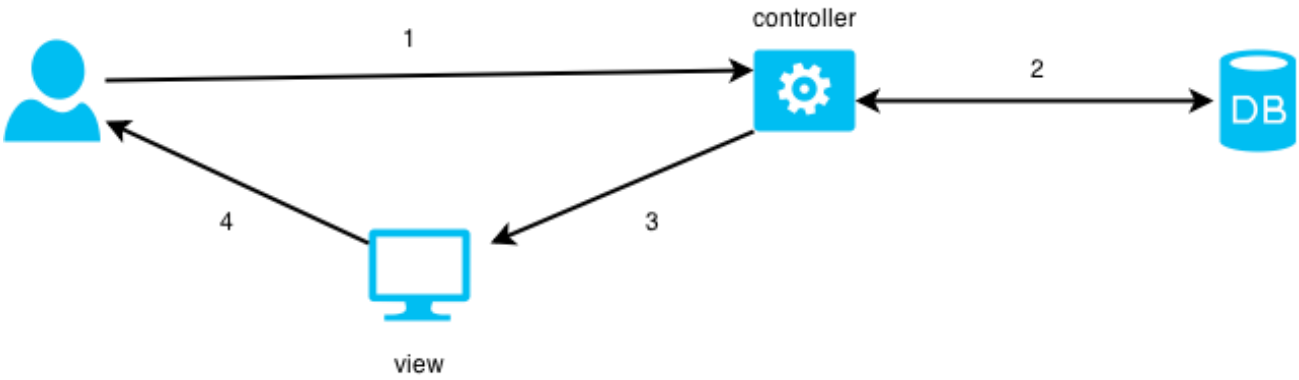
3.4 Sammanfattning

Detta kapitel har beskrivit projektets design ur både visuellt och tekniskt perspektiv. Webbsidans funktioner har förklarats med både text och bilder. Den underliggande databasens tabeller och relationer har beskrivits. Även projektets kodstruktur och moduler har beskrivits.

4 Implementation

Detta kapitel tar upp detaljer runt implementationen av webbplatsen. Kapitlet börjar med att beskriva ett typiskt informationsflöde genom applikationen från användare till server till databas och tillbaka. Det efterföljande avsnittet behandlar hur data synkroniseras mellan webbplatsen och de mobila applikationerna. Tillsammans med avsnittet om tidszoner är det tänkt att ge en uppfattning om bakgrunden till klassen Repository uppkomst. Repository har till uppgift att förenkla databasåtkomsten i applikationen. Avslutningsvis presenteras speciellt intressanta delar av zonediteringen och statistikfunktionen med kodexempel och detaljerade beskrivningar.

4.1 Informationsflöde



Figur 4.1: Hantering av anrop i MVC

När en användare besöker en sida anropas en metod i sidans controller (1, *Figur 4.1*). Beroende på vilken typ av uppgift som skall utföras hämtar eller skickar controllern data från/till databasen (2). Om data skall visas för användaren behandlar controllern informationen efter behov och skickar den sedan till en vy (3) där den presenteras för användaren (4).

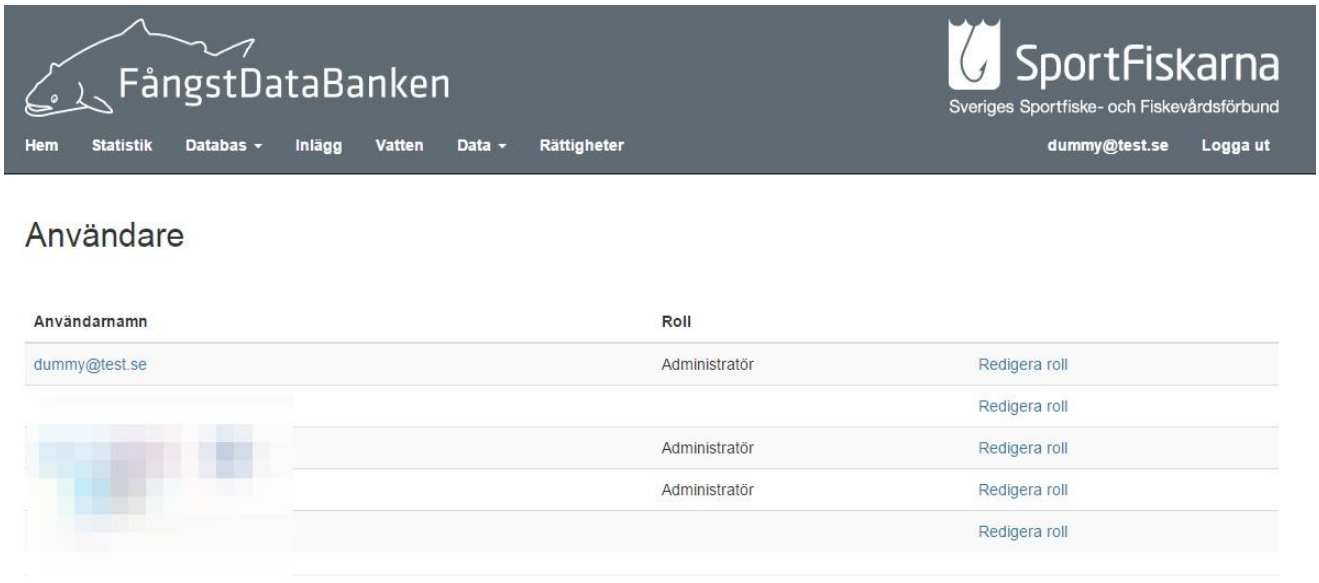
4.1.1 Ett typiskt flöde.

De flesta av webbplatsens sidor fungerar på ett snarlikt sätt. Här används ”Rättigheter”-sidan som illustrerande exempel på arbetsflödet:

En användare (administratör) klickar på ”Rättigheter”-länken i webbplatsens meny. Användarens webbläsare följer länken och skickar en förfrågan till admin/index. Webbplatsen skickar användarens förfrågan till adminController och dess index-metod (1 i *Figur 4.1*).

I index-metoden görs en sökning i databasen för att skapa en lista över alla användare (2). För varje användare skapas en instans av idRoleViewModel. idRoleViewModel är ett objekt som innehåller användarnamn och roll. Detta är en ”view model”, ett objekt med syfte att samla data från olika databastabeller i ett objekt för enklare åtkomst i vyn. En lista bestående av dessa idRoleViewModel returneras sedan till vyn (3).

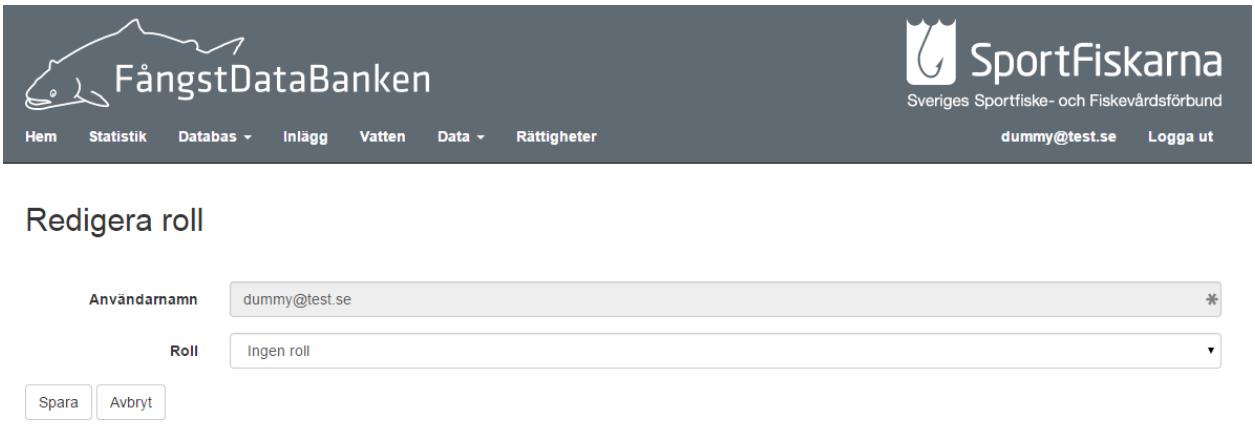
I vyn presenteras informationen för användaren (4). En loop går igenom alla element i viewModel-listan och genererar en lista med användarnamn, roll och "redigera"-länk för varje element (Figur 4.2: Rättigheter-vyn).



Figur 4.2: Rättigheter-vyn

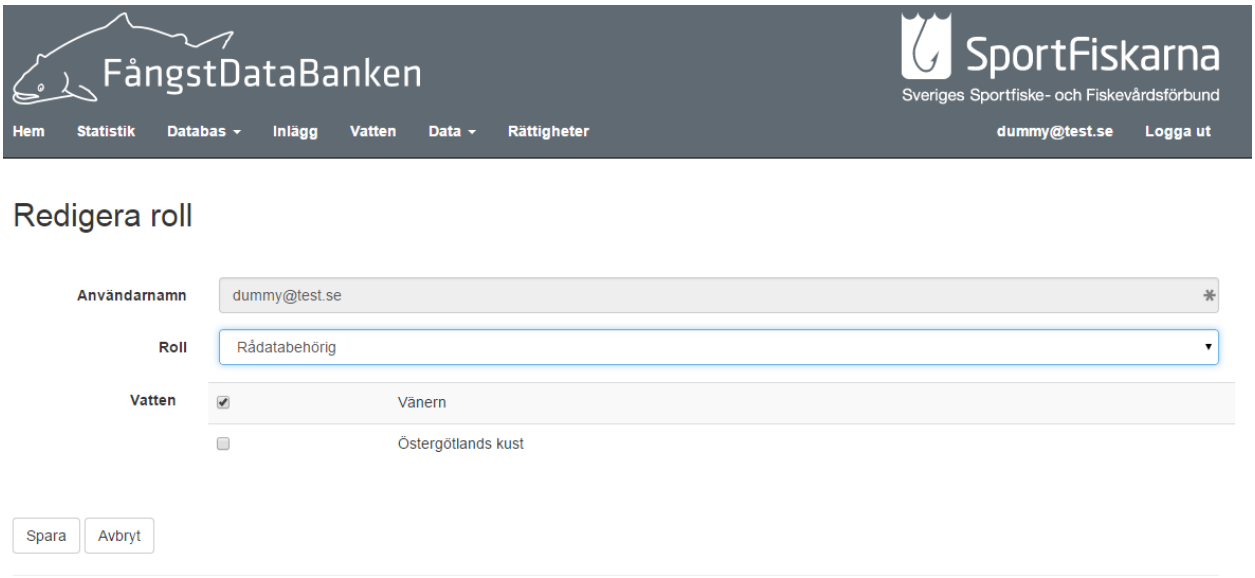
Om administratören klickar på "redigera"-länken för en användare i listan skickar webbläsaren en förfrågan till admin/edit/användarId (1). Förfrågan tas omhand av edit-metoden i adminController. Denna metod tar en sträng med användarId som inparameter. Detta id används för att söka i databasen efter ett användarnamn. Från användarnamnet hämtas en lista över de vatten användaren har rättighet att hämta data ifrån (rawDataWater) (2). Denna info sparas i en ny instans av idRoleViewModel. Edit-metoden sparar dessutom en lista med alla tillgängliga vatten i en ViewBag som tillsammans med idRoleModel skickas till vyn ”edit” (3).

I edit-vyn visas användarnamnet i en textruta och en dropdown-meny genereras med de olika administratörsalternativen "ingen behörighet", "administratör" och "rådatabehörig". En "spara" och en "avbryt" knapp skapas också (Figur 4.3: Redigera roll).



Figur 4.3: Redigera roll (ingen roll)

I denna vy (4) används JavaScript för att dynamiskt hantera de olika menyvalen. Om rådataadministratör väljs visas en lista över alla vatten i systemet (från ViewBag) med checkrutor för att välja vilket vatten behörigheten skall gälla (*Figur 4.4: Redigera roll (rådatabehörig)*). Om något annat alternativ väljs döljs vattenlistan.



Figur 4.4: Redigera roll (rådatabehörig)

Efter att ha gjort sina val klickar användaren på spara-knappen. Webbläsaren skickar ett POST-anrop till adminController/editRole/användarId (1) som läser vilka val som gjorts i vyn och sparar dem i databasen (2).

4.2 Databas och synkronisering

För att hålla de mobila applikationernas data uppdaterad används klassen syncController i modulen WebApi (*3.3.3 WebApi*). Denna controller skapar en instans av syncRepository från modulen Repository (*3.3.2 Repository*).

SyncRepository fyller ett syncOutput-objekt med information som sedan skickas till den anropande applikationen.

SyncRepository anropas med ett datum som parameter. Detta datum motsvarar senaste synkroniseringstillfälle och parametern kallas därför lastSync i koden. För att avgöra vilka datafält den anropande applikationen behöver, går SyncRepository igenom alla datafält och jämför deras UpdatedWhen-fält med lastSync. Om ett fält uppdaterats sedan senaste synkroniseringen sparas fältets data i syncOutput och flaggan syncNeeded sätts till sann. Om ett fält inte har uppdaterats sedan senaste synkroniseringen ignoreras det. Om syncNeeded är sann när alla datafält gåtts igenom returneras syncOutput till anropande applikation.

Följande exempel visar hur fältet Fish sparas i syncOutput om det ändrats sedan senaste synkroniseringen:

```
var fish = context.Fish.FirstOrDefault(f => f.UpdatedWhen > lastSync);
if (fish != null)
{
    syncOutput.FishDto = Mapper.Map<List<FishDto>>(context.Fish.ToList());
    syncNeeded = true;
}
```

Det är de mobila applikationerna som initierar uppdateringen av data genom att anropa API:t. Exakt när detta sker beror på applikationen.

En nackdel med att avgöra synkroniseringsbehovet baserat på jämförelse av datum är att borttagen data inte kan jämföras och således inte heller kan synkroniseras och raderas från applikationerna. Alla fält som är aktuella för synkronisering har dock en Visible-flagga som de mobila applikationerna tar hänsyn till. För att ta bort ett fält från de mobila applikationerna sätter man Visible till falsk. På så vis synkroniseras fältet till applikationen men visas inte. Ett fält måste alltså finnas kvar i databasen för att kunna raderas från applikationerna.

Eftersom en administratör vill kunna styra vilka fält som är synliga för applikationerna kan inte Visible-flaggan användas för att dölja data från webbplatsen. För att en administratör skall kunna ta bort ett fält helt och hållet finns ytterligare en flagga: Deleted. När en administratör väljer att radera ett fält sätts Deleted till sann och Visible till falsk. På så vis kan ett fält döljas både på webbplatsen och i de mobila applikationerna.

4.2.1 Infotexter

Vid projektets början sparades infotexter som generella (en text för alla vatten) eller tillhörande endast ett vatten. Förhållandet mellan infotext och vatten var alltså av typen ett-till-många eller ett-till-ett. Detta var inte en praktisk lösning eftersom det förekommer tillfällen då man vill använda en infotext till flera olika vatten utan att göra den generell. Till exempel kan en infotext gälla för samtliga sötvattensjöar men inte kustvattnen med saltvatten. Infotexterna kan dessutom vara så omfattande att man som administratör inte gärna skriver dem flera gånger.

Den praktiska lösningen var att behålla möjligheten att göra en infotext generell och att dessutom skapa ett många-till-många förhållande mellan vatten och infotexter (se *3.2.2 Tillägg*). På så vis kan samma infotext förekomma i flera vatten.

För att detta skulle fungera med de mobila applikationerna behölls Dto:erna oförändrade men koden i SyncRepository som behandlar infotexter skrevs om.

I ursprungsversionen synkroniserades alla infotexter som förändrats och som hade Visible-flaggan satt till sann. Ändringarna bestod i att lägga till kod för att synkronisera alla infotexter med isGeneral-flaggan satt till sann. Om isGeneral-flaggan inte är sann kontrolleras om infotexten är associerad med några vatten och om så är fallet skickas den till samtliga sina vatten. Infotexter med isGeneral satt till sann skickas alltså en gång och visas i samtliga vatten medan icke generella infotexter skickas en gång per vatten de skall förekomma i. Att skicka samma infotext flera gånger kan tyckas onödigt och resurskrävande men då i stort sett alla servrar använder gzip-komprimering [50] [51] utgör detta inget praktiskt problem.

4.3 Hantering av tidszoner

Ett problem som upptäcktes när webbapplikationen började provköras i driftmiljön var att tidsstämplarna på databasposterna blev felaktiga. Det visade sig att driftmiljön använde UTC-tid medan applikationen förväntade sig centraleuropeisk (sommar)tid[1].

Efter viss diskussion bestämdes att UTC-tid skulle sparas i databasen och konverteras till centraleuropeisk tid för visning. På så vis spelar det längre ingen roll vilken tidszon som används i driftmiljön.

För att underlätta detta skapades en hjälpklass i modulen Repository med funktioner för att konvertera till och från UTC. Följande kodexempel visar ett utdrag ur klassen. Den kompletta klassen hämtar den tidszon som skall användas från App.config samt har konverteringsfunktioner för DateTime-parametrar som kan vara null (`DateTime?`).

```
public static class TimeZoneConverter{
    private static readonly TimeZoneInfo tz = TimeZoneInfo
        .FindSystemTimeZoneById("W. Europe Standard Time");

    public static DateTime UtcToLocal(DateTime dateTime) {
        return TimeZoneInfo.ConvertTimeFromUtc(dateTime, tz);
    }

    public static DateTime LocalToUtc(DateTime dateTime) {
        return TimeZoneInfo.ConvertTimeToUtc(dateTime, tz);
    }
}
```

[1] Centraleuropeisk tid heter *W. Europe Standard Time* internt i Windows.

För att automatiskt visa lokala tider på webbapplikationens sidor skapades en visningsmall för datatypen `DateTime?`. Mallen används även för `DateTime` som inte kan vara null. Mallen används automatiskt när man anropar ASP.NET:s inbyggda `Html.Display()` och `Html.DisplayFor()`-funktioner med en parameter av typen `DateTime`.

```
@model DateTime?

@if (Model.HasValue)
{
    if (ViewData.Keys.Contains("ShortDate"))
    {
        @Html.Encode(TimeZoneConverter.UtcToLocal(Model.Value).ToShortDateString())
    }
    else
    {
        @Html.Encode(TimeZoneConverter.UtcToLocal(Model.Value).ToString())
    }
}
```

Exempel på användning där mallen och tidszonkonverteraren används för att skriva ut tiden då modellen senast uppdaterades:

```
<p>@Html.DisplayFor(model => model.UpdatedWhen)</p>
```

4.4 Repository

För att förenkla hanteringen av de synkroniseringsrelaterade fälten som togs upp i *4.2 Databas och synkronisering* valde vi att skapa en generisk [52] implementation av repository-mönstret [53].

De primära målen med implementationen var att:

- Automatiskt sätta ett objekts primärnyckel [54] när ett nytt objekt läggs till i ett set med manuell primärnyckelhantering av heltalstyp.
- Automatiskt sätta ett objekts skapad- och uppdateradtidsstämpel när ett nytt objekt läggs till i ett set.
- Automatiskt sätta ett objekts uppdateradtidsstämpel när ett existerande objekt modifieras.
- Ta bort objekt ur set vars modell har Deleted-flagga genom att sätta Deleted till sant och Visible till falskt.
- Endast returnera de icke-borttagna objekten ur set med Deleted-flagga.

Om begreppen objekt, set och modell känns främmande finns en kort beskrivning av dem i kapitel *2.7.5 Entity Framework*.

4.4.1 Klassdefinition

Klassen Repository definieras som:

```
public class Repository<TEntity, TKey> where TEntity : class, IIdentity<TKey>
```

Repository är således en generisk klass med två typparametrar. Den första, TEntity, specificerar den modell som skall användas. Den andra, TKey, specificerar datatypen för modellens primärnyckel.

Repository har en restriktion som kräver att TEntity är en klass som implementerar gränssnittet [55] IIdentity<TKey>. Gränssnittet garanterar att det i modellen finns en egenskap med namn Id som är av typen TKey. Egenskapen Id används som primärnyckel i databastabellerna.

En initiering av klassen Repository för att få den att arbeta mot databastabellen Fish med en primärnyckel av typen int ser alltså ut som följer:

```
var fishRepository = new Repository<Fish, int>();
```

4.4.2 Konstruktor och egenskaper

Repository har fyra egenskaper som styr hur klassen fungerar. HasManualIdentity är en flagga som indikerar om databastabellens primärnyckel måste anges manuellt eller om den tilldelas automatiskt av databasen. IncludeDeletedItems bestämmer om objekt som tagits bort genom att Deleted-flaggan satts till sant skall visas eller inte. IsSyncable och IsTraceable är flaggor som indikerar om datamodellen ärver från basklasserna Syncable respektive Traceable.

Datamodeller som ärver från Syncable kan synkroniseras med mobilapplikationerna eftersom Synacable tillhandahåller de egna egenskaperna Visible och Deleted samt de ärvda egenskaperna CreatedWhen och UpdatedWhen från Traceable. Uppdelningen i två basklasser gjordes eftersom vissa datamodeller som till exempel BlogPost kunde dra nytta av att vara spårbara (Traceable) utan att för den skull vara synkroniseringsbara (Syncable).

Av de fyra egenskaperna är IncludeDeletedItems den enda vars värde kan ändras utanför konstruktorn. Övriga tre egenskaper är skrivskyddade.

För att bestämma värdet på IsSyncable och IsTraceable undersöks vilken basklass TEntity har. För IsSyncable ser koden i konstruktorn ut som följer:

```
this.IsSyncable = typeof(TEntity).BaseType == typeof(Syncable);
```

För att undersöka hur primärnyckeln hanteras och bestämma värdet på HasManualIdentity används följande kod:

```
try {
    var attribute = typeof(TEntity).GetProperty("Id").GetCustomAttributes(
        typeof(DatabaseGeneratedAttribute), false).Single() as DatabaseGeneratedAttribute;

    this.HasManualIdentity = attribute.DatabaseGeneratedOption == DatabaseGeneratedOption.None;
}
catch (InvalidOperationException ex)
{
    throw new RepositoryException("Attribute DatabaseGenerated not set on property Id.", ex);
}
```

Funktionen undersöker det DatabaseGenerated-attribut [56] som bör ha satts på egenskapen Id för typen TEntity. De restriktioner som satts för TEntity (Se *4.4.1 Klassdefinition*) garanterar att egenskapen Id finns men det kan vara så att attributet inte blivit satt. Om så är fallet kastar Single() ett undantag som tas om hand och bakas in i ett nytt undantag av typen RepositoryException (Se *4.4.3 Felhantering*).

En primärnyckel där databasservern hanterar värdet definieras som följer:

```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Id { get; set; }
```

Om hanteringen är manuell definieras primärnyckeln enligt följande:

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
public int Id { get; set; }
```

4.4.3 Felhantering

För att förenkla felhanteringen i funktioner som anropar Repository försöker Repository fånga undantag [57] som kan förutspås, och kasta en ny RepositoryException med ursprungsundantaget som InnerException.

RepositoryException ärver från Exception [58] och lägger till två egenskaper. Egenskapen HttpStatusCode innehåller den HTTP-statuskod [59] som Repository föreslår skall returneras till användaren om ett fel inträffar. ValidationErrors innehåller en sammanfattning av de

valideringsfel som inträffat under en databasoperation och är menad som ett hjälpmedel för utvecklaren.

4.4.4 Entities

Metoden Entities används för att arbeta med alla objekt i ett set. Den filtrerar automatiskt bort objekt som ”tagits bort” genom att Deleted-flaggan satts till sant såvida inte egenskapen IncludeDeletedItems satts till sant.

I metoden konstrueras ett LINQ-uttryck [60] manuellt eftersom vi inte har direkt tillgång till egenskapen Deleted för objekten i setet. Detta eftersom TEntity bara garanteras implementera IIdentity vilket endast ger direktåtkomst till egenskapen Id. I efterhand upptäcktes tredjepartsbiblioteket System.Linq.Dynamic [61] som hade abstraherat bort konstruktionen av LINQ-uttrycket. Vi valde dock att inte använda biblioteket då det redan fanns en fungerande lösning och vi inte kunde se någon annan plats i källkoden som skulle dra nytta av dynamiska LINQ-uttryck.

```
public IQueryable<TEntity> Entities
{
    get
    {
        if (this.IsSyncable && !this.IncludeDeletedItems)
        {
            // Build an e.Deleted == false expression.
            var parameter = Expression.Parameter(typeof(TEntity), "e");
            var left = Expression.Property(parameter, typeof(TEntity).GetProperty("Deleted"));
            var right = Expression.Constant(false);
            var notDeletedExpression = Expression.Equal(left, right);

            // Build an Set<TEntity>.Where(e => e.Deleted == false) expression.
            var queryableSet = this.context.Set<TEntity>().AsQueryable<TEntity>();
            var whereExpression = Expression.Call(typeof(Queryable), "Where",
                new Type[] { queryableSet.ElementType }, queryableSet.Expression,
                Expression.Lambda<Func<TEntity, bool>>(notDeletedExpression,
                new ParameterExpression[] { parameter })
            );

            return queryableSet.Provider.CreateQuery<TEntity>(whereExpression);
        }
        else
        {
            return this.context.Set<TEntity>();
        }
    }
}
```

Eftersom metoden returnerar en IQueryable<TEntity> går det att fortsätta bygga ut returvärdet med fler LINQ-satser utan att databasen faktiskt anropas. Ett databasanrop görs först när resultatet sparas i en variabel eller när en aggregeringsfunktion körs. Tekniken kallas ”Deferred execution” [62]. Till exempel resulterar följande kod i ett databasanrop:

```
var updatedTodayCount = fishRepository.Entities.Where(e => e.UpdatedWhen >= midnight).Count();
```

I databasanropet, här återgivet något förenklat för läsbarhetens skull, går det att se att filtreringen på både Deleted och UpdatedWhen sker i samma anrop som aggregeringen med funktionen COUNT.

```
SELECT
    COUNT(1)
    FROM Fish
    WHERE (Fish.Deleted = 0) AND (Fish.UpdatedWhen >= '2015-04-21 00:00:00')
```

4.4.5 FindById

Metoden FindById används för att söka efter objekt med en specifik primärnyckel. Metoden drar, precis som efterföljande metoder, nytta av datatypen dynamic [63] som introducerades i C# 4.0 [64]. I korthet innebär dynamic att typkontroll sker under körning av programmet istället för under kompilering vilket är det vanliga för statiskt typade språk som C#.

Genom att lagra information från databasen i en variabel av datatypen dynamic istället för TEntity går det att komma åt alla objektets metoder och egenskaper med vanlig C#-syntax. I tidigare versioner av C# kunde reflection [65] användas för att lösa liknande problem men då med syntax olik den som normalt används i C# (Se exempel *4.4.8 Delete*). Förutom enklare syntax ger användandet av dynamic också bättre prestanda än reflection då objekts egenskaper cachelagras av Dynamic Language Runtime [66] vilket är den komponent i .NET som tillhandahåller dynamic.

När dynamiska variabler används blir det viktigt att först säkerställa att en metod eller egenskap finns innan den anropas. I FindById sker detta genom att undersöka om den underliggande modellen är synkroniseringsbar. Om så är fallet vet programmet att egenskapen Deleted finns och dess värde kan evalueras.

```
public TEntity FindById(object id)
{
    if (id == null)
    {
        throw new RepositoryException(HttpStatusCode.BadRequest, "ID cannot be null.");
    }

    dynamic dynamicEntity = this.context.Set<TEntity>().Find(id);

    if (dynamicEntity == null)
    {
        throw new RepositoryException(HttpStatusCode.NotFound, "No entity with the specified ID
            was found.");
    }
    else if (!this.IncludeDeletedItems && this.IsSyncable && dynamicEntity.Deleted)
    {
        throw new RepositoryException(HttpStatusCode.NotFound, "An entity with the specified ID
            was found but it has been deleted. Set IncludeDeletedItems to find it.");
    }

    return dynamicEntity;
}
```

4.4.6 Add

Metoden Add lägger till nya objekt i ett set. Om den underliggande datamodellen ärver från Traceable, det vill säga har tidsstämplar för när objektet skapades och senast uppdaterades, sätts dessa tidsstämplar automatiskt till just nu gällande UTC-tid.

Vid projektets start upptäcktes ett flertal datamodeller vars primärnycklar var av heltaltyp och krävde manuell tilldelning. Add innehåller funktionalitet för att underlätta hanteringen av dessa primärnycklar. När nya objekt läggs till i set med dessa datamodeller är det önskvärt att tilldela primärnycklar i sekventiell stigande ordning. Add undersöker därför om den aktuella datamodellen (TEntity) har manuell primärnyckelhantering och om primärnyckelns typ (TKey) är ett heltal. Om så är fallet sätter Add primärnyckeln till det just nu högsta primärnyckelvärdet plus ett.

För att förhindra att två objekt tilldelas samma nyckel används ett lås [67] runt koden som räknar ut och sparar den nya nyckeln. Låset garanterar att koden inom låset gränser exekveras till fullo innan processorn växlar till att exekvera en annan tråd.

Följande kodexempel visar den del av Add som hanterar manuella primärnycklar.

```
if (this.HasManualIdentity && typeof(TKey) == typeof(int))
{
    lock (idLock)
    {
        var entityWithHighestId = this.context.Set<TEntity>().OrderByDescending
            (e => e.Id).FirstOrDefault();

        if (entityWithHighestId == null)
        {
            entity.Id = (TKey)Convert.ChangeType(1, typeof(TKey));
        }
        else
        {
            entity.Id = (TKey)Convert.ChangeType(Convert.ToInt32(entityWithHighestId.Id) + 1,
                typeof(TKey));
        }

        this.context.Set<TEntity>().Add(entity);
        this.context.SaveChanges();
    }
}
```

Låset deklareras som en statisk variabel i klassen.

```
private static object idLock = new object();
```

4.4.7 Edit

Edit används för att förändra ett objekt i ett set. Precis som Add undersöker metoden om datamodellen ärver från Traceable och om så är fallet, sätter tidsstämpeln för när objektet senast uppdaterades.

```
public int Edit(TEntity entity)
{
    if (this.IsTraceable)
    {
        dynamic dynamicEntity = entity;
        dynamicEntity.UpdatedWhen = DateTime.UtcNow;
    }

    this.context.Entry(entity).State = EntityState.Modified;
    return this.context.SaveChanges();
}
```

4.4.8 Delete

Delete används för att ta bort objekt ur ett set. Beroende på om datamodellen är synkroniseringsbar eller inte kan Delete bete sig på två olika sätt. För synkroniseringsbara modeller sker borttagning genom att flaggan Visible sätts till falsk och flaggan Deleted sätts till sann (Se *4.2 Databas och synkronisering*). För modeller som inte är synkroniseringsbara sker borttagning genom att objekt tas bort ur set.

```
public int Delete(TEntity entity)
{
    if (this.IsSyncable)
    {
        dynamic dynamicEntity = entity;

        dynamicEntity.Visible = false;
        dynamicEntity.Deleted = true;
        dynamicEntity.UpdatedWhen = DateTime.UtcNow;

        this.context.Entry(entity).State = EntityState.Modified;
    }
    else
    {
        this.context.Set<TEntity>().Remove(entity);
    }

    return this.context.SaveChanges();
}
```

Om modellen är synkroniseringsbar används datatypen dynamic används för att enkelt komma åt de egenskaper som behöver modifieras. En tidigare version av Repository använde Reflection istället för dynamic för att modifiera egenskaperna. De fyra raderna som innehåller dynamicEntity i kodexemplet ovan bestod då istället av tre rader. Som synes i kodexemplet nedan var dock raderna betydligt längre och mer svårlästa än när dynamic används.

```
entity.GetType().GetProperty("Visible").SetValue(entity, false);
entity.GetType().GetProperty("Deleted").SetValue(entity, true);
entity.GetType().GetProperty("UpdatedWhen").SetValue(entity, DateTime.UtcNow);
```

4.5 ASP.NET Identity och modulen Common

Några veckor in i projektet visade det sig att lösenordshanteringen skiljde sig mellan Web och WebApi. Användare som skapades genom Web behövde lösenord med högre komplexitet än användare som skapades genom WebApi. Problemet spårades till den automatiskt genererade källkoden som konfigurerar den lösenordsvaliderare som finns inbyggd i ASP.NET Identity. Denna lösenordsvaliderare har till uppgift att kontrollera att nya lösenord uppfyller inställda komplexitetskrav.

I modulen WebApi konfigurerades PasswordValidator enligt följande:

```
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = false,
    RequireDigit = true,
    RequireLowercase = true,
    RequireUppercase = true,
};
```

I modulen Web såg konfigurationen ut som följer:

```
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = true,
    RequireDigit = true,
    RequireLowercase = true,
    RequireUppercase = true,
};
```

Som synes krävde Web tillskillnad från WebApi minst ett specialtecken i lösenordet.

Istället för att modifiera modulen Web till att se ut som WebApi valde vi att bryta ut inställningarna till en egen modul som sedan refererades av både Web och WebApi. På så vis undviks framtida problem med olika lösenordsregler som skulle kunna uppstå om en utvecklare ändrar reglerna i en modul men glömmer att ändra i den andra.

Vi valde att skapa en ny modul, kallat Common, istället för att använda den redan delade modulen Repository eftersom ändringen inte var databasrelaterad.

I modulen Common finns klassen PasswordValidatorSettings:

```
public static class PasswordValidatorSettings
{
    public static readonly int RequiredLength = 6;
    public static readonly bool RequireNonLetterOrDigit = false;
    public static readonly bool RequireDigit = true;
    public static readonly bool RequireLowercase = true;
    public static readonly bool RequireUppercase = true;
}
```

I både Web och WebApi ser koden som konfigurerar lösenordsvalideraren nu ut som följer:

```
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = PasswordValidatorSettings.RequiredLength,
    RequireNonLetterOrDigit = PasswordValidatorSettings.RequireNonLetterOrDigit,
    RequireDigit = PasswordValidatorSettings.RequireDigit,
    RequireLowercase = PasswordValidatorSettings.RequireLowercase,
    RequireUppercase = PasswordValidatorSettings.RequireUppercase
};
```

4.6 Zoneditering

Under projektets planering bestämdes att zonediteringsfunktionens primära gränssnitt skulle vara grafiskt i form av en karta som visade det vatten som för tillfället redigerades. Zonerna skulle visas som semitransparenta figurer ovanpå kartan samt i listform bredvid kartan. Ett sekundärt gränssnitt där administratören kunde ange de exakta koordinaterna för en zon skulle också implementeras. Målet var att skapa ett gränssnitt som kändes dynamiskt och som skulle tillåta administratören att snabbt definiera zoner genom att justera zonernas position och storlek direkt på kartan.

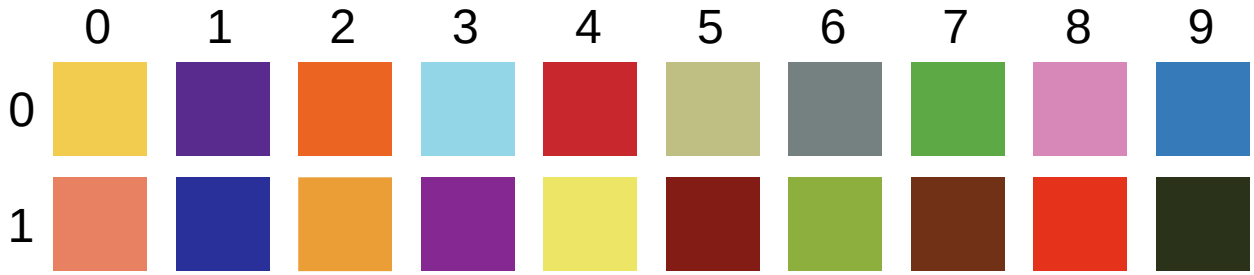
För att åstadkomma detta användes ett flertal tredjepartsbibliotek. Kartan och zonfigurerna skapades med hjälp av Google Maps Javascript API [35]. Dialogrutan för detaljredigering av zonerna skapades med hjälp av Bootstrap [24]. Delarna kopplades samman med hjälp Knockouts [28] databindningsfunktioner mot den underliggande datamodellen.

4.6.1 Färgkodning

För att kunna skilja zonerna åt i kartvyn samt snabbt kunna se vilken zon i kartvyn som hör samman med vilken zon i listvyn valde vi att färgkoda zonerna.

Då färgkodning kan innebära problem för personer med nedsatt färgseende undersökte vi om det finns färger som är bättre lämpade för ändamålet än andra. Det visade sig finnas flera publikationer inom området. Vi valde att använda ”Twenty-Two Colors of Maximum Contrast” [68] då vi upplevde att det var den mest frekvent refererade publikationen.

Då de två första färgerna (svart och vitt) var olämpliga[2] att använda lämnade det oss med en palett om 20 färger.



Figur 4.5: 20 färger med maximal kontrast

Då färgerna kommer att visas med viss transparens samt ha en färgad karta under sig är det inte säkert att den resulterande färgen som visas för användaren kommer vara lämplig för personer med nedsatt färgseende. Då zonediteringsfunktionen kommer vara begränsad till det fåtal användare som har administrationsbehörighet har vi dock valt att inte undersöka saken vidare.

4.6.2 Zonform

I databasen lagras koordinaterna för zonernas övre vänstra och nedre högra hörn enligt koordinatsystemet WGS 84 [69]. Det innebär i praktiken att en zon endast kan anta formen av en rektangel.

I skrivande stund finns inga önskemål om att kunna ange icke-rektangulära zoner. Vi har därför begränsat oss till detta för att hålla zonredigeraren så enkel som möjligt och för att slippa modifiera databasstrukturen samt mobilapplikationen. Det finns dock inga tekniska begränsningar i Google Maps som skulle göra det omöjligt att definiera zoner i andra former. En figur i Google Maps definieras i form av en polygon där koordinaterna för varje hörn anges. En rektangel är således en delmängd av alla möjliga polygoner där antalet hörn är fyra och hörnen är placerade horisontellt och vertikalt i förhållande till varandra.

[2] Text på sidan visas med svart färg vilket gör den oläsbar på svart bakgrund. En zon med vit färg skulle inte gå att urskilja mot sidans vita bakgrund.

4.6.3 Initiering av Google Maps

Initiering av Google Maps sker genom att inkludera en JavaScript-fil från Google. Som argument till filen skickas önskad version och språk av API:t. En API-nyckel skickas också med för att identifiera vilket Google Maps-konto som skall belastas med användandet. Som nämndes i *2.7.10 Google Maps* kan ett gratiskonto göra 25 000 kartvisningar per dag.

```
<script type="text/javascript" src="//maps.googleapis.com/maps/api/js?v=3&language=sv
    &key=MyApiKey"></script>
```

För att visa kartan körs följande kod på klientsidan. Dess uppgift är att göra div-taggen ”map-canvas” till en karta och sedan flytta och zooma kartvyn.

```
var map = new google.maps.Map(document.getElementById('map-canvas'));
map.fitBounds(bounds);
```

Om inga zoner finns definierade för vattnet som redigeras ställs kartan in för att visa hela Sverige. Om zoner finns definierade ställs positionen och zoomnivån in så att alla zoner syns på kartan. Beräkningen av variabeln bounds sker på serversidan genom att loopa igenom alla zoner för vattnet och hitta extremvärden för alla fyra väderstreck.

```
if (fishingArea.Zones.Where(z => z.Deleted == false).Count() == 0)
{
    // No zones are defined. Use Sweden as bounds.
    ViewBag.East  = 24.166667;
    ViewBag.West  = 10.9575;
    ViewBag.North = 69.06;
    ViewBag.South = 55.336944;
}
else
{
    // Calculate the combined bounds of all zones for this fishing area.
    foreach (var zone in fishingArea.Zones.Where(z => z.Deleted == false))
    {
        ViewBag.East  = Max(ViewBag.East,  zone.BottomRight.Longitude);
        ViewBag.West  = Min(ViewBag.West,  zone.TopLeft.Longitude);
        ViewBag.North = Max(ViewBag.North, zone.TopLeft.Latitude);
        ViewBag.South = Min(ViewBag.South, zone.BottomRight.Latitude);
    }
}
```

De beräknade värdena skrivs sedan ut på sidan och blir på så vis tillgängliga för JavaScript-koden som körs på klienten. InvariantCulture innebär här att koordinaterna skrivs ut med decimalpunkt istället för decimalkomma.

```
var bounds = new google.maps.LatLngBounds(
    new google.maps.LatLng(@ViewBag.South.ToString(CultureInfo.InvariantCulture),
        @ViewBag.West.ToString(CultureInfo.InvariantCulture)),
    new google.maps.LatLng(@ViewBag.North.ToString(CultureInfo.InvariantCulture),
        @ViewBag.East.ToString(CultureInfo.InvariantCulture)));
```

4.6.4 Initiering av Knockout

Knockout initieras precis som kartan genom att JavaScript-kod genereras på serversidan och sedan körs på klientsidan. På serversidan genereras JavaScript-kod som bildar en lista med zoner. Denna lista används sedan som datamodell av Knockout. För de tre första zonerna i Vänern ser den genererade datamodellen ut som följer.

```
function ZoneListViewModel() {
    var self = this;
    self.zones = ko.observableArray([
        zoneFactory({
            bounds: new google.maps.LatLngBounds(
                new google.maps.LatLng(58.808764, 12.474976),
                new google.maps.LatLng(59.117473, 12.821045)
            ),
            id: '1', name: ko.observable('1'), visible: true
        }),
        zoneFactory({
            bounds: new google.maps.LatLngBounds(
                new google.maps.LatLng(58.808764, 12.821045),
                new google.maps.LatLng(59.117473, 13.213806)
            ),
            id: '2', name: ko.observable('2'), visible: true
        }),
        zoneFactory({
            bounds: new google.maps.LatLngBounds(
                new google.maps.LatLng(58.573265, 12.474976),
                new google.maps.LatLng(58.808764, 12.821045)
            ),
            id: '3', name: ko.observable('3'), visible: true
        }),
        // ...
    ]);
    self.modified = ko.computed(function ...
```

Listans innehåll är som synes flera anrop till funktionen zoneFactory vars uppgift det är att skapa en ny instans av klassen Rectangle som finns i Google Maps. zoneFactory slår ihop de egenskaper som skickats som argument till funktionen med de standardegenskaper en rektangel som representerar en zon skall ha. Funktionen skapar också en hanterare för händelsen bounds_changed (Se *4.6.6 Redigera eller ta bort zoner*).

```
function zoneFactory(opts) {
    // Create a new rectangle by merging the default settings with those passed in opts.
    var zone = new google.maps.Rectangle($.extend({
        strokeColor: getColor(),
        strokeOpacity: 0.75,
        strokeWeight: 2,
        fillColor: getColor(),
        fillOpacity: 0.5,
        map: map,
        editable: true,
        draggable: true,
        visible: true,
        updated: ko.observable(false),
        id: undefined, // Id is set to the primary key from the FishingAreaZone table. New
            rectangles have undefined id until saved.
    }, opts));

    // bounds_changed

    return zone;
}
```

De första standardegenskaperna som sätts för en rektangel är färgen och genomskinligheten för dess fyllning och ram. Färgen fås genom anrop till funktionen getColor vars uppgift det är att returnera nästa tillgängliga färg i färglistan (Se *4.6.1 Färgkodning*). getColor byter returvärde vartannat anrop eftersom funktionen anropas en gång för fyllnadsfärgen och en gång för ramen och dessa skall ha samma färg.

Egenskapen map bestämmer vilken karta rektangeln skall visas på. Egenskapen map tilldelas värdet av variabeln med samma namn som skapades i *4.6.3 Initiering av Google Maps*.

Editable sätts till sann för att tillåta storleksförändring av rektangeln genom att dra i dess kanter. Draggable möjliggör förflyttning av rektangeln på kartan. Visible indikerar om zonen skall vara synlig eller ej.

Ovanstående satta egenskaper är sådana som finns definierade i klassen RectangleOptions [70] som ingår i Google Maps API. JavaScripts dynamiska natur utnyttjas genom att bygga ut rektangeln med tre extra egenskaper. Updated är en observerbar egenskap som indikerar om zonen modifierats eller inte. Observerbarheten innebär att Knockout omvärderar databindingar som gjorts mot Updated så snart någon zons Updated-egenskap förändras. Detta används för att aktivera eller inaktivera knapparna för att spara eller återställa gjorda förändringar. Id används för att lagra värdet av zonens primärnyckel från databasen och har standardvärdet undefined. Den tredje egenskapen, name, har inget standardvärde utan hämtas antingen från databasen för zoner som finns sparade eller genereras för nya zoner.

Den källkod som hittills presenterats resulterar i att kartan och zonrektanglarna i *Figur 3.9: Gränssnittet för att hantera zoner* visas för användaren. För att skapa menyn till höger måste Knockouts databindning konfigureras och aktiveras.

Knockout nyttjar det data-attribut [71] som finns i HTML. Data-attributet tillåter att implementationsspecifika attribut sätts för HTML-element utan att källkoden flaggas som ogiltig vid en validering. Knockout läser attribut med namn data-bind för att bestämma hur databindningen skall ske.

För att skapa listan med zoner används nedanstående kod.

```
<table class="table" data-bind="foreach: zones">

    <tr data-bind="style: { backgroundColor: fillColor },
      event: { mouseover: $parent.mouseOver, mouseout: $parent.mouseOut }">

        <td data-bind="text: name"></td>

        <td><input type="checkbox" data-bind="checked: visible,
          click: $parent.toggleVisible"></td>

        <td><span class="glyphicon glyphicon-edit"
          data-bind="click: $parent.editZone"></span></td>

        <td><span class="glyphicon glyphicon-trash"
          data-bind="click: $parent.removeZone"></span></td>

    </tr>

</table>
```

Den första databindningen, foreach: zones, på elementet table innebär att koden mellan table-taggarna kommer att användas som mall för att visa varje element i listan zones. För varje rad i listan binds bakgrundfärgen till samma färg som motsvarande zon har i kartvyn. Bindningar skapas också för när muspekaren förs in över och ut från raden. Dessa bindningar används för att göra zonrektanglarna på kartan mindre genomskinliga när muspekaren förs över motsvarande zon i listan. Databindningar vars värde börjar med $parent anropar metoder i datamodellen och skickar med aktuell zon som argument. Databindningar utan $parent binder till metod eller egenskap direkt i aktuell zon.

Efter att utvalda HTML-element givits databindningsattribut aktiveras Knockouts databindning.

```
listViewModel = new ZoneListViewModel();
ko.applyBindings(listViewModel, document.getElementById('zone-map'));
```

Databindningen resulterar i att knapparna och zonlistan till höger om kartan skapas vilket visas i *Figur 3.9: Gränssnittet för att hantera zoner*.

4.6.5 Skapa zoner

Zonens namn bestäms genom att undersöka de andra zonerna som tillhör vattnet. Om de andra zonerna har numeriska namn, till exempel 1 eller 12, ges den nya zonen namnet av det högsta numeriska värdet plus ett. Om namnen inte är numeriska eller om den nya zonen är den första för ett vatten ges zonen namnet av listans längd plus 1. Den nya zonens initiala storlek sätts till halva kartans bredd och höjd. Positionen sätts så att zonen centreras på kartan.

I kodexemplet nedan beräknas den nya zonens position innan zonen läggs till i zonlistan och visas på kartan.

```
var ne = map.getBounds().getNorthEast();
var sw = map.getBounds().getSouthWest();

var diffLat = ne.lat() - sw.lat();
var diffLng = ne.lng() - sw.lng();

var newBounds = new google.maps.LatLngBounds(
    new google.maps.LatLng(sw.lat() + (diffLat * 0.25), sw.lng() + (diffLng * 0.25)),
    new google.maps.LatLng(sw.lat() + (diffLat * 0.75), sw.lng() + (diffLng * 0.75))
);

self.zones.push(zoneFactory({
    bounds: newBounds, name: ko.observable(name.toString())
}));
```

Då inget värde sätts för egenskapen Id kommer zonen att antas vara ny och läggas till i databasen när sidan postas till servern (*4.6.7 Spara zoner till databas*).

4.6.6 Redigera eller ta bort zoner

Zoner kan redigeras på flera sätt. Position och storlek kan ändras genom att modifiera zonrektanglarna direkt på kartan. Synligheten kan ändras direkt i zonlistan vid sidan av kartan och genom att klicka på redigeringssymbolen kan zonens namn ändras och dess position finjusteras (Se *Figur 3.10: Gränssnitt för redigering av zondetaljer*). Gemensamt för alla sätt är att de sätter zonens updated-flagga till sann. Exemplet nedan taget ur funktionen zoneFactory lägger till en händelsehanterare som anropas när en zon flyttas eller får storleken förändrad.

```
google.maps.event.addListener(zone, 'bounds_changed', function() {
    zone.updated(true);
});
```

För att ta bort zoner anropas Knockout-funktionen destroy. Till skillnad från remove tar destroy inte bort zonen från listan utan sätter dess _destroy-flagga till sann [72]. Genom att på serversidan sedan undersöka _destroy-flaggans värde, då kallad Deleted, bestäms om zonen skall tas bort ur databasen eller inte.

4.6.7 Spara zoner till databas

För att spara zonförändringarna görs listan med zoner i Knockouts datamodell om till HTML input-element av typen hidden. De nya elementen läggs till i ett HTML-formulär och skickas sedan till servern via ett HTTP POST-anrop. Detta tillvägagångssätt gör det möjligt att använda det inbyggda skydd mot cross-site request forgery [73] som finns i ASP.NET.

```
for (var index = 0; index < self.zones().length; index++) {
    var zone = self.zones()[index];

    // Append updated, deleted and new zones to the form to be posted.
    if (zone.updated() || zone._destroy || !zone.id) {
        $('<input>').attr('type', 'hidden').attr('name', 'zone' + index).attr('value',
            prepForPost(zone)).appendTo('form');
    }
}

$('form').submit();
```

Funktionen prepForPost serialiserar zonen till JSON i ett format som kan deserialiseras till objekt av typen ZoneViewModel på serversidan.

På serversidan itereras alla postade fält igenom och fält vars namn börjar med zone deserialiseras till ZoneViewModel-objekt. Från databasen hämtas en eventuellt existerande zon med samma ID och beroende på olika jämförelser av objektens egenskaper bestäms om zonen skall läggas till, tas bort eller redigeras.

```
foreach (string key in Request.Form)
{
    if (key.StartsWith("zone", StringComparison.InvariantCultureIgnoreCase))
    {
        var viewZone = JsonConvert.DeserializeObject<ZoneViewModel>(Request.Form[key]);
        var dbZone = zoneRepository.Entities.SingleOrDefault(z => z.Id == viewZone.Id);

        if ((viewZone.Id == null && viewZone.Deleted == false) || (dbZone == null))
        {
            dbZone = Mapper.Map<FishingAreaZone>(viewZone);
            dbZone.FishingArea = fishingArea;

            zoneRepository.Add(dbZone);
        }
        else if (viewZone.Id != null && viewZone.Deleted == true && dbZone != null)
        {
            zoneRepository.Delete(dbZone);
        }
        else if (viewZone.Id != null && viewZone.Deleted == false && dbZone != null)
        {
            zoneRepository.Edit(Mapper.Map(viewZone, dbZone));
        }
    }
}
```

4.7 Statistik

Statistikdelen var ett område där kravspecifikationen innehöll ett minimikrav och frivilliga tillägg. Under möten med Sportfiskarna visade det sig också att detta är ett område där önskemålen kan komma att ändras över tid. Detta var anledningen till att D3 (se *2.7.8 Data-Driven Documents*) valdes som verktyg för att visa statistik på webbsidan. Att använda ett kraftfullt bibliotek som D3 för att visa enkla stapeldiagram kan tyckas onödigt men det ger möjligheter att vidareutveckla statistiksidan efter eventuella framtida behov.

Statistiksidan på webbplatsen har till syfte att visa statistik över den inloggade användarens eget fiske. När användaren går in på sidan skapas en lista över användarens fångster i sidans controller. Listan sparas i JSON format och skickas sedan till vyn. Vyn är skriven nästan helt i JavaScript och använder både Knockout och D3. Knockout används för att binda listans innehåll till tre dropdown-menyer som används för att filtrera statistiken efter art, vatten och fångstmetod.

För att visualisera vald data skapas ett stapeldiagram i D3.

D3 skapar först en arbetsyta (canvas) att visa grafik i. Till arbetsytan läggs sedan staplar och axlar.

För att skapa arbetsytan:

```
var svg = d3.select("#barchart")
        .append("svg")
            .attr("width",  width  + margin.left + margin.right)
            .attr("height", height + margin.top  + margin.bottom);
```

Detta väljer div-taggen med id ”barchart” på sidan och lägger till (append) en SVG-canvas. Till denna canvas läggs attributen width och height för att definiera arbetsytans storlek.

Staplarna skapas med följande kod:

```
var label = svg.selectAll(".label")
           .data(data);

        label.enter().append("g")
           .attr("class", "g")
           .attr("transform", function(d) { return "translate(" + x(d.label) + ",0)"; });

        var bar = label.selectAll("g")
              .data(function(d) { return d.taken; });

        bar.enter().append("rect")
              .attr("class",  "bar")
              .attr("width",  x.rangeBand())
              .attr("y",      function(d) { return y(d.y1); })
              .attr("height", function(d) { return y(d.y0) - y(d.y1); })
              .style("fill",  function(d) { return color(d.name); });
```

Variabeln “label” skapar en grupp ”g” för varje storleksintervall i ”data”, det vill säga varje position på x-axeln som kan innehålla en stapel. För var och en av dessa grupper binds det antal fångster som finns i data-variabeln i ”bar”. Antalet fångster binds till ett rektangel-element (.append(”rect”)) vars storlek och färg beräknas utifrån om det är en tillbakasläppt eller upptagen fisk.

4.8 Sammanfattning

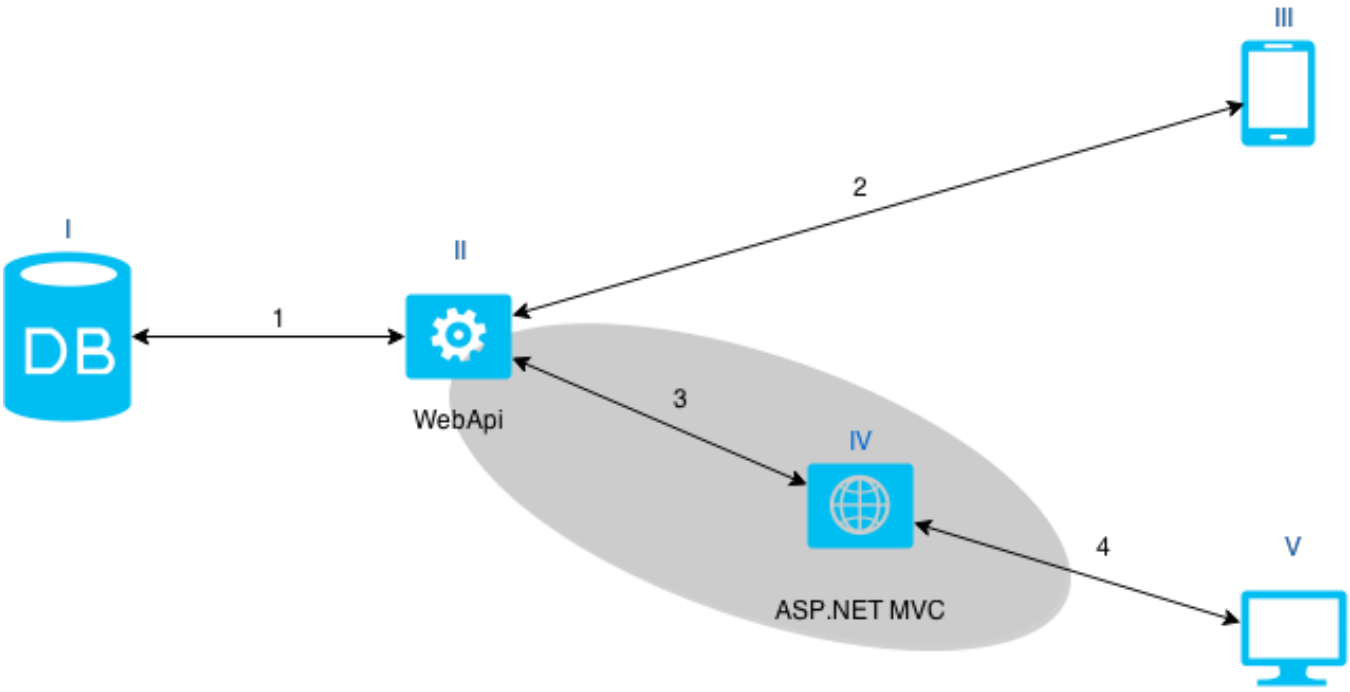
I detta kapitel har olika implementationsaspekter tagits upp. Det är naturligtvis inte möjligt att ge en total genomgång av implementationen av projektet på några få sidor men de viktigaste och mest intressanta delarna har presenterats. Några speciellt intressanta delar av projektet har fått extra uppmärksamhet med kodexempel och detaljerade förklaringar.

5 Resultat och utvärdering

Detta kapitel innehåller en utvärdering av de resultat som framkommit av projektet. Kapitlet inleds med en översikt och en diskussion om valet av .NET plattformen för genomförandet av projektet. Kapitlet fortsätter med en uppskattning av kravuppfyllnad och en jämförelse av mängden kod för olika implementationer av databasåtkomst.

5.1 Översikt

Enligt den ursprungliga planeringen av projektet skulle webbplatsen använda samma WebApi som de mobila applikationerna använder (*Figur 5.1: Ursprunglig planerad översikt av systemet*).



Figur 5.1: Ursprunglig planerad översikt av systemet

Tanken var att all åtkomst till databasen skulle skötas av API:t och att det skulle anpassas efter behov. Autentisering av användare skulle ske mot den OAuth-endpoint [74] som tillhandahölls av WebApi. Denna plan fick överges tidigt i projektet då det visade sig vara mycket besvärligt att hantera autentiseringen av användare från ASP.NET MVC.

I WebApi får en korrekt inloggad användare en ”bearer token” som bevis på lyckad inloggning. Denna token skall sedan ingå i HTTP-headern vid alla anrop till API:t för att säkerställa att användaren är autentiserad. I ASP.NET MVC finns inbyggd funktionalitet för att hantera inloggning och autentisering men dessa funktioner var inte fullt ut kompatibla med WebApi.

I detta läge fanns två val: antingen överge idén att använda WebApi från websidan och jobba direkt mot databasen eller att använda någon annan teknik för att bygga webplatsen. Ett alternativ skulle kunna vara AngularJs [75] som också kan användas med MVC mönstret men implementerar det på klientsidan med hjälp av JavaScript.

Valet föll på att fortsätta utveckla i .NET miljö. Det fanns flera anledningar till detta.

- Det fanns egentligen inte något skäl till att gå via WebApi till databasen då webbplatsen och databasen alltid kommer att köras i samma servermiljö. Det finns då ingen anledning att gå via ett externt API eftersom man kan använda databasanrop direkt via Entity Framework.

- I .NET finns många färdiga lösningar för att hantera olika användare och inloggningar. Till exempel kan man mycket enkelt hantera accesskontroll genom attribut på klasser. För att begränsa tillgången av en klass till inloggade användare kan man använda attributet [Authorize] och för att begränsa tillgängligheten ytterligare och bara tillåta administratörer dekoreras klassen med [Authorize(Roles="Admin")].

- Vår utgångspunkt var att göra ett projekt i .NET miljö då det är en av de största plattformarna som används i dag och vi ville skaffa större kunskap och erfarenhet inom området.

Vi gjorde uppskattningen att .NET-plattformen tillför tillräckligt mycket extra funktionalitet för att det skulle vara värt att fortsätta använda den för utvecklingen. Det hade dock varit intressant att göra en jämförelse mellan någon annan teknik (till exempel AngularJS) och .NET för att undersöka skillnader och likheter. Detta ligger dock utanför omfattningen för detta projekt.

5.2 Kravuppfyllelse

Redan från början av projektet fanns en tydlig kravspecifikation (*Bilaga 1 – Kravspecifikation*). Denna låg till grund för hela utvecklingsarbetet och eftersom

specifikationen dessutom innehöll prioriteringar av de olika kraven var det lätt att planera arbetet. Webbplatsens huvudsakliga uppgift kommer att vara som administrativt verktyg för Sportfiskarna och vanliga användare kommer främst att använda de mobila applikationerna. Därför gavs de administrativa delarna av webbplatsen extra hög prioritet och vid projektets slut uppfyller webbplatsen samtliga funktionalitetskrav för administration (FU1-9).

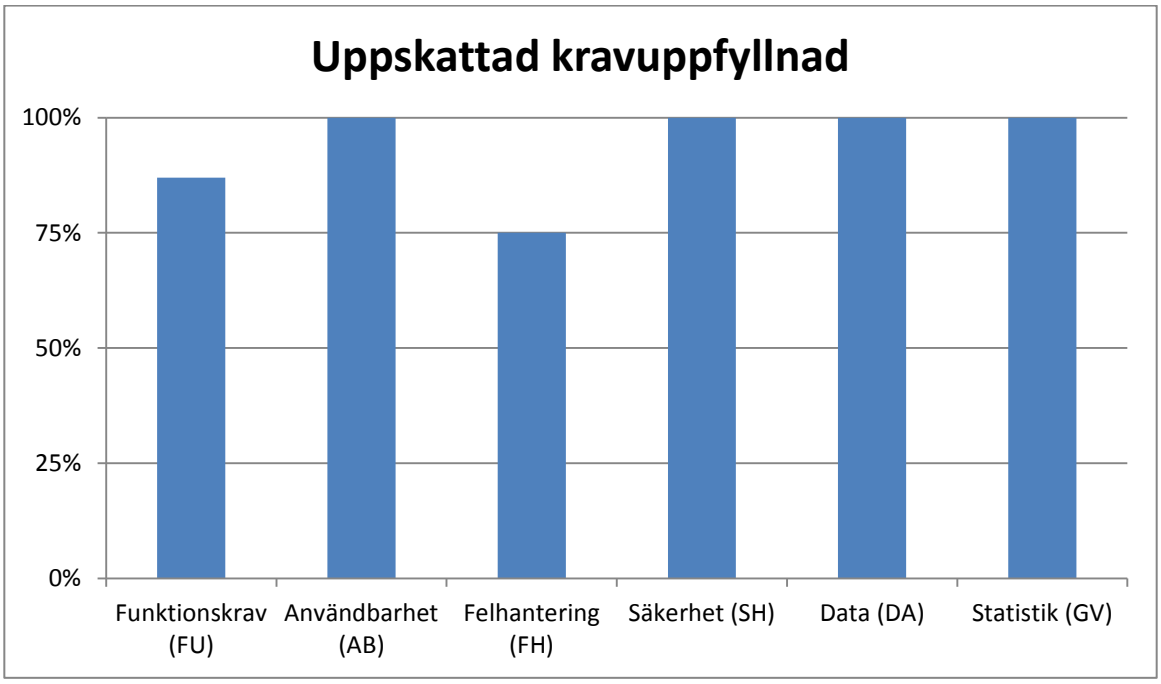
Resterande funktionalitetskrav har alla uppfyllts utom de som handlar om utökad statistik (FU12P2 och FU13P3). Dessa implementerades inte på grund av låg prioritet och tidsbrist. Statistiksidan på webbplatsen uppfyller dock de krav som satts som ”Grundversion” (GV1-4). Den statistikdel som betecknas som ”Extra uppgift” (*3.7.2 i Bilaga 1*) har inte implementerats då tiden inte räckte till.

De krav som rör användbarhet (AB1-5) kan vara svåra att bedöma objektivt. Speciellt de krav som rör tydlighet och effektivitet är svåra att mäta. Vi upplever dock att samtliga krav i denna kategori är uppfyllda och baserar detta på kommentarer och synpunkter från kunden och handledare på Sogeti.

Kravet angående felhantering (FH1) är delvis uppfyllt. I zonhanteringen blir felaktigt ifyllda fält rödmarkerade. På övriga sidor markeras felaktiga inmatningar med ett allmänt felmeddelande om att den önskade operationen inte gått att utföra.

Säkerhetskraven (SH1-6) är uppfyllda. Dock har kravet om export av databasinnehåll (SH2) förändrats under projektets gång. Kunden önskade en rådatabehörighet (*3.1.3 Inloggad rådatabehörig*) som skall ha tillgång till utvalt databasinnehåll.

Datakraven (DA) är uppfyllda. Dock är inte DA2 tillämplig då ingen mailfunktion finns på webbplatsen.



Figur 5.2: Uppskattad kravuppfyllnad

5.3 Tekniker och verktyg

De tekniker och verktyg vi använt under projektet har i stort sett fungerat som förväntat. Microsofts utvecklingsmiljö har vi upplevt som mycket komplett och användarvänlig. Versionshanteringen med TFS har fungerat bra under hela projektet. Vid något enstaka tillfälle har vi råkat ut för att en lokal version av källkoden kommit i konflikt med serverns version. Detta har orsakats av att vi båda gjort ändringar i samma fil och har gått att lösa utan några större problem.

Planeringsverktygen i TFS var nya för oss. Dessa har fungerat bra och även om vi bara använt de mest grundläggande funktionerna har de varit en bra hjälp för att organisera vårt arbete.

5.4 Från BaseController till Repository

Vår ursprungliga idé för att åstadkomma vissa[3] av de förenklingar Repository senare kom att åstadkomma var att implementera en generisk basklass som sedan ärvdes av alla controllers. Basklassen tillhandahöll metoderna GetItem och GetAllItems som kom att motsvaras av FindById och Entites i Repository. En controller som skulle arbeta med bloggposter definierades således som följer.

```
public class BlogController : BaseController<BlogPost>
```

För enkla controllers som endast arbetade mot en databastabell fungerar denna lösning tillfredställande. Problem uppstod dock när en controller behövde använda flera tabeller eftersom hjälpfunktionerna endast arbetade mot controllerns ”huvudtabell”. Detta resulterade i att vi för de övriga tabellerna fick återgå till manuell hantering av borttagna objekt och att felhanteringen blev mer komplex eftersom många olika typer av undantag kunde kastas.

Ett annat tillkortakommande med BaseController var att klassen bara tillhandahöll metoder för att läsa data. Uppdatering av tidsstämplar och tabellspecifik borttagningskod var fortfarande tvungen att skrivas manuellt.

BaseController kunde mycket väl ha byggts ut med nya metoder för att lägga till, modifiera eller ta bort objekt men på grund av begränsningen att klassen bara kunde arbeta mot en tabell valde vi att istället implementera en generisk variant av Repository-mönstret [53].

Den nya Repository-klassen tillät oss att komma åt förenklingsfunktionerna för alla tabeller en controller behövde arbeta emot vilket resulterade i att mängden databasrelaterad kod i varje controller blev betydligt mindre.

För att påvisa detta gjordes en jämförelse mellan direkt användning av Entity Framework som användes i första iterationen av databaskoden, BaseController som användes i andra iterationen och Repository som var den tredje och slutgiltiga iterationen. Uppgiften var att ta bort den första zonen för ett specifikt vatten. Alla implementationer utförde samma förändringar i databasen och resulterade i samma felmeddelanden om något gick fel. Som visas

---

[3] Filtrera bort poster som ”tagits bort” genom att Deleted satts till sant. Enhetlig felhantering där endast en typ av undantag kastas.

i *Tabell 5.1: Kodmängd för databasåtkomst* resulterade användandet av Repository i att endast en tredjedel så mycket databasrelaterad kod behövde skrivas som om Entity Framework hade använts direkt.

Metod	**Kodrader**	**Relativ storlek**
Entity Framework	33	100 %
BaseController	26	79 %
Repository	11	33 %

Tabell 5.1: Kodmängd för databasåtkomst

Källkoden som utgjorde testerna finns bifogad i *Bilaga 2 – Källkod för jämförelse av databasåtkomst*.

6 Utvärdering av projektet

I detta kapitel utväderas projektet. Aspekter som tidsåtgång och arbetsmiljö tas upp men också lärdomar och tankar inför webbplatsens framtid och möjligheter till vidareutveckling.

6.1 Tidsåtgång

Vi har jobbat på projektet tre dagar i veckan från slutet av januari till slutet av maj. Vi planerade att dela tiden lika mellan programmering och uppsatsskrivning men detta har inte fungerat fullt ut. I början av projektet och under perioder när vi närmat oss kunddemonstrationer har uppsatsskrivningen fått stå tillbaka. I lugnare perioder för projektet har dock skrivandet prioriterats.

Vi hade mycket hjälp av kravspecifikationen för planeringen av projektet. Genom att följa prioriteringarna av kraven kunde vi lägga upp en plan där de högst prioriterade kraven klarades av tidigt i projektet och de lägst prioriterade kraven fick utgöra en tidsbuffert i slutet. På så sätt behövde vi inte känna någon stor stress i slutet av projektet utan kunde börja prioritera uppsatsskrivande och förberedelser för presentation i god tid.

Under planeringen av projektet gjordes tidsestimeringar av de olika uppgifterna. Att korrekt kunna tidsestimera sina uppgifter och projekt är mycket viktigt på ett konsultföretag. Vår erfarenhet efter projektet är att tidsestimeringar kan vara mycket svåra att göra. I några fall gick uppgifterna mycket fortare än vi uppskattat men i de allra flera fall tog de betydligt längre tid än planerat. Detta beror naturligtvis till stor del på oerfarenhet och ovana vid de olika tekniker vi använt. I flera fall har vi behövt lägga mer tid än planerat på att läsa in oss på hur en teknik fungerar innan vi börjat implementera den.

6.2 Arbetsmiljö

Allt arbete på projektet har skett på Sogetis kontor i Karlstad. I början av projektet pågick utvecklingen av applikationen för iOS på kontoret. Detta var värdefullt då vi kunde få stöd och hjälp framförallt i frågor runt databasen och synkroniseringen mellan applikationen och webbplatsen. Vi har under hela projektet haft goda möjligheter att få stöd från vår handledare eller annan personal på Sogeti. Efter ungefär halva projekttiden genomfördes en

kodgenomgång där vi tillsammans med en konsult på Sogeti gick igenom vår kod och fick tips och råd på hur den kunde förbättras.

6.3 Kundkontakt

En stor och inspirerande del av projektet har varit kontakten med kunden. Under projektets gång genomfördes ett tiotal demonstrationer för kunden tillsammans med personal från Sogeti. Utöver detta fick vi möjlighet att följa med kunden till Sportfiskemässan i Jönköping för att visa upp webbplatsen för Havs- och Vattenmyndigheten och Jordbruksverket. Mottagandet av webbplatsen har hela tiden varit mycket positivt och det har varit tydligt att detta projekt leder fram till en produkt som kommer att vara till stor nytta för kunden.

6.4 Kravspecifikationer

Att ha en färdig kravspecifikation vid projektets start var mycket värdefullt. Många krav var entydiga och lätta att förstå men andra krav krävde en tolkning baserad på tidigare erfarenhet och åsikter. Vad som gör en webbplats ”lätt att förstå” eller ”effektiv” med ”så få klick som möjligt” är svårt att utvärdera objektivt och kan variera med användarens erfarenhet och datorvana.

Sammantaget har dock kravspecifikationen givit en bra bild av vilka prioriteringar som skall göras och vad som anses viktigt av kunden. Detta har också bekräftats på våra olika möten med kunden under projektets gång.

6.5 Lärdomar

Alla projekt som löper över en längre tid ger naturligtvis fördjupad kunskap och förståelse för de olika tekniker som använts, även de man kommit i kontakt med tidigare. I vårt fall har vi fått mycket mer kunskap om och erfarenhet från ASP.NET MVC och Entity Framework.

Några tekniker var nya för oss och dessa har vi fått läsa in oss på och prova i olika miljöer innan de tagits i bruk i projektet. Detta handlar främst om Knockout, D3 och Google Maps API.

En lärdom vi tar med oss är inte vänta med att skapa generella lösningar på upprepande mönster. I början av projektet såg vi att vissa kodfragment återkom men snarare än att försöka skriva en generell lösning fortsatte vi att upprepa oss. BaseController och senare Repository

blev lösningen på detta men om vi hade skrivit dem tidigare i processen hade många kodrader sparats.

6.6 Framtid

I slutet av projektet kom beskedet att Havs- och Vattenmyndigheten går in med 130 000 kr för att arbetet med FångstDataBanken skall kunna fortsätta både på IT-sidan av projektet men också när det gäller omhändertagandet och analysen av statistiken som genereras [76].

I skrivande stund planeras webbplatsen sättas i drift under sommaren (2015) och kommer då inledningsvis att fungera för att administrera den nya iOS-applikationen. Den befintliga Androidapplikationen kommer att uppdateras till att använda samma API som iOS-applikationen och kommer då också att kunna administreras av webbplatsen.

6.7 Möjligheter till vidareutveckling

På en del punkter finns intressanta möjligheter att vidareutveckla webbplatsen och systemet som helhet.

På statistiksidan skall, enligt kravspecifikationen, en användare kunna jämföra sitt fiske med andra användares. Detta gäller både fångster från samtliga användare (FU12, Bilaga 1) och fångster från specificerade användare (FU13, Bilaga 1). Här skulle man kunna utveckla en mer social del av webbplatsen där en användare har möjlighet att lägga till sina vänner för att kunna följa och jämföra sitt eget fiske med deras. Man kan också tänka sig ett system där användare har möjlighet att skicka meddelanden till varandra eller få uppdateringar när någon vän varit ute på fisketur.

Det finns utrymme att utöka informationen som lagras i databasen med flera fält om sådant som är av intresse för fiskare. Det kan röra sig om parametrar som till exempel väder och lufttryck. Även möjligheter att spara bilder av sin fångst skulle kunna vara intressant. Detta är dock inte specifikt för webbplatsen utan skulle behöva uppdateras i hela systemet, även i de mobila applikationerna.

Det har också funnits diskussioner om att införa inslag av gamification [77] i en framtida version av systemet. Här kan man tänka sig att användaren belönas med någon typ av virtuell medalj, poäng eller liknande vid vissa milstolpar i fisket, till exempel största fångsten eller 10

fiskar av en viss art i ett vatten. Detta är dock något som kunden vill vara försiktig med av oro för att det i vissa kretsar skulle kunna uppfattas som oseriöst och avskräcka från användning av systemet.

6.8 Avslutning

Vi är mycket nöjda med vårt examensarbete och stolta över resultatet. Vi har fått möjligheten att jobba med ett roligt och varierande projekt på en arbetsplats full av inspirerande medarbetare. Dessutom har vi lärt oss många nya verktyg och tekniker och fått värdefull erfarenhet av att jobba mot en kund över längre tid.

Vår uppdragsgivare är mycket nöjd med webbplatsen som uppfyller deras önskemål och ger dem möjlighet att själva administrera sitt system utan att behöva ta in externa konsulter.

Utöver de rent tekniska detaljerna har vi dessutom fått en insyn i Sportfiskarnas värld, både teoretiskt genom informationen som presenteras på webbplatsen och genom mer praktiska inslag. Vi gjorde ett besök på den årliga sportfiskemässan där vi förutom att demonstrera webbplatsen hade tid att besöka de olika montrarna och lära oss mer om sportfiske i allmänhet. Vi fick dessutom möjlighet att följa med vår kund på fisketur för att lära oss flugfiske. En upplevelse som gav mersmak, speciellt då vi fick en varsin öring på kroken.

Referenser

[1] Sogeti Sverige AB, ”Vilka är vi?,” [Online]. Tillgänglig: http://www.sogeti.se/om-oss/vilka-ar-vi/. [Hämtad: 2015-01-29].

[2] Sveriges Sportfiske- och Fiskevårdsförbund, ”Om Sportfiskarna,” [Online]. Tillgänglig: http://www.sportfiskarna.se/Omsportfiskarna/OmSportfiskarna/tabid/300/Default.aspx. [Hämtad: 2015-01-29].

[3] J. Wahman och E. Larsson, ”Sportfiskeapplikation till Android,” Karlstad Universitet, Karlstad, 2014.

[4] Wikipedia, ”Relational database,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Relational_database. [Hämtad: 2015-02-27].

[5] Wikipedia, ”SQL,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/SQL. [Hämtad: 2015-02-27].

[6] Wikipedia, ”Hypertext Transfer Protocol,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Hämtad: 2015-02-27].

[7] G. E. Krasner och S. T. Pope, ”A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System,” Mountain View, ParcPlace Systems, 1988, pp. 1-5.

[8] Wikipedia, "Separation of concerns," [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Separation_of_concerns. [Hämtad: 2015-02-11].

[9] Wikipedia, "ASP.NET," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/ASP.NET. [Hämtad: 2015-02-11].

[10] T. K. Gu Ming-xia, "Comparative analysis of WebForms MVC and MVP architecture," in *2010, vol.2, pp. 391-4. Publisher: Piscataway, NJ USA; Wuhan China: IEEE Country of Publication: USA*, 2010.

[11] Microsoft, ”Overview of the .NET Framework,” [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/zw4w595w.aspx. [Hämtad: 2015-02-11].

[12] Wikipedia, ”C# (programming language),” [Online]. Tillgänglig: http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29. [Hämtad: 2015-02-11].

[13] Wikipedia, ”Visual Basic .NET,” [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Visual_Basic_.NET. [Hämtad: 2015-02-11].

[14] Wikipedia, "ASP.NET MVC Framework," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/ASP.NET_MVC_Framework. [Hämtad: 2015-02-11].

[15] J. Miller, "Design For Convention Over Configuration," Microsoft, [Online]. Tillgänglig: http://msdn.microsoft.com/en-us/magazine/dd419655.aspx#id0080100. [Hämtad: 2015-02-11].

[16] Microsoft, ”ASP.NET Web API,” [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/hh833994%28v=vs.108%29.aspx. [Hämtad: 2015-02-11].

[17] J. J. Garrett, "Ajax: A New Approach to Web Applications," Adaptive Path, [Online]. Tillgänglig: https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php. [Hämtad: 2015-01-11].

[18] R. T. Fielding, J. Gettys, J. C. Mogul, H. Frystyk Nielsen, L. Masinter, P. J. Leach och T. Berners-Lee, ”RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1,” The Internet Society, [Online]. Tillgänglig: http://tools.ietf.org/html/rfc2616. [Hämtad: 2015-02-11].

[19] Wikipedia, "Create, read, update and delete," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Create,_read,_update_and_delete. [Hämtad: 2015-01-29].

[20] Wikipedia, "Representational state transfer," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Representational_state_transfer. [Hämtad: 2015-01-29].

[21] Wikipedia, "Entity Framework," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Entity_Framework. [Hämtad: 2015-02-27].

[22] Wikipedia, ”Object-relational mapping,” [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Object-relational_mapping. [Hämtad: 2015-02-27].

[23] Microsoft, "Entity Framework Code First to a New Database," [Online]. Tillgänglig: http://msdn.microsoft.com/en-us/data/jj193542.aspx. [Hämtad: 2015-02-27].

[24] Bootstrap, "About Bootstrap," [Online]. Tillgänglig: http://getbootstrap.com/about/. [Hämtad: 2015-02-11].

[25] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. Doyle Navara, E. O'Connor och S. Pfeiffer, ”HTML5,” W3C, [Online]. Tillgänglig: http://www.w3.org/TR/html5/. [Hämtad: 2015-02-11].

[26] B. Bos, "Cascading Style Sheets," W3C, [Online]. Tillgänglig: http://www.w3.org/Style/CSS/. [Hämtad: 2015-02-11].

[27] Wikipedia, "JavaScript," [Online]. Tillgänglig: en.wikipedia.org/wiki/JavaScript. [Hämtad: 2015-02-11].

[28] knockoutjs.com, "Knockout: Home," [Online]. Tillgänglig: http://knockoutjs.com/. [Hämtad: 2015-03-13].

[29] Wikipedia, ”Model View ViewModel,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Model_View_ViewModel. [Hämtad: 2015-03-13].

[30] ”D3.js - Data-Driven Documents,” [Online]. Tillgänglig: http://d3js.org/. [Hämtad: 2015-04-17].

[31] ”W3C Document Object Model,” [Online]. Tillgänglig: http://www.w3.org/DOM/. [Hämtad: 2015-04-17].

[32] WC3, ”Scalable Vector Graphics (SVG) 1.1 (Second Edition),” [Online]. Tillgänglig: http://www.w3.org/TR/SVG/. [Hämtad: 2015-04-22].

[33] ”CKEditor.com | The best web text editor for everyone,” [Online]. Tillgänglig: http://ckeditor.com/. [Hämtad: 2015-05-11].

[34] Wikipedia, "Google Maps," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Google_Maps. [Hämtad: 2015-02-26].

[35] Google, "Google Maps API - Google Developers," [Online]. Tillgänglig:

https://developers.google.com/maps/. [Hämtad: 2015-02-26].

[36] Google, "Markers," [Online]. Tillgänglig: https://developers.google.com/maps/documentation/javascript/markers. [Hämtad: 2015-02-26].

[37] Google, "Shapes," [Online]. Tillgänglig: https://developers.google.com/maps/documentation/javascript/shapes. [Hämtad: 2015-02-26].

[38] Google, "Heatmap Layer," [Online]. Tillgänglig: https://developers.google.com/maps/documentation/javascript/heatmaplayer. [Hämtad: 2015-02-26].

[39] Google, "Traffic, Transit and Bicycling Layer," [Online]. Tillgänglig: https://developers.google.com/maps/documentation/javascript/trafficlayer. [Hämtad: 2015-02-26].

[40] Google, ”Usage Limits and Billing,” [Online]. Tillgänglig: https://developers.google.com/maps/documentation/javascript/usage. [Hämtad: 2015-04-17].

[41] Wikipedia, "Microsoft Visual Studio," [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio. [Hämtad: 2015-02-11].

[42] Wikipedia, "Integrated development environment," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Integrated_development_environment. [Hämtad: 2015-05-20].

[43] Wikipedia, "Team Foundation Server," [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Team_Foundation_Server. [Hämtad: 2015-02-13].

[44] Wikipedia, ”Comma-separated values,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Comma-separated_values. [Hämtad: 2015-05-06].

[45] P. Rastogi, R. Anderson, T. Dykstra and J. Galloway, "Introduction to ASP.NET Identity," Microsoft, [Online]. Tillgänglig: http://www.asp.net/identity/overview/getting-

started/introduction-to-aspnet-identity. [Hämtad: 2015-02-27].

[46] Microsoft, "Capitalization Conventions," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/ms229043%28v=vs.110%29.aspx. [Hämtad: 2015-03-26].

[47] M. Fowler, "Data Transfer Object," [Online]. Tillgänglig: http://martinfowler.com/eaaCatalog/dataTransferObject.html. [Hämtad: 2015-03-18].

[48] Wikipedia, "Serialization," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Serialization. [Hämtad: 2015-03-18].

[49] ”JSON,” [Online]. Tillgänglig: http://json.org/. [Hämtad: 2015-04-20].

[50] Wikipedia, ”HTTP compression,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/HTTP_compression. [Hämtad: 2015-04-08].

[51] Wikipedia, ”gzip,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Gzip. [Hämtad: 2015-04-08].

[52] Microsoft, "Generics (C# Programming Guide)," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/512aeb7t.aspx. [Hämtad: 2015-04-20].

[53] E. Hieatt and R. Mee, "Repository," in *Patterns of Enterprise Application Architecture*, Indianapolis, Addison-Wesley Professional, 2002, p. 322.

[54] M. Chapple, "Primary Key Definition: What is a Primary Key?," [Online]. Tillgänglig: http://databases.about.com/cs/administration/g/primarykey.htm. [Hämtad: 2015-04-28].

[55] Microsoft, "Interfaces (C# Programming Guide)," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/ms173156.aspx. [Hämtad: 2015-05-11].

[56] J. Lerman, "Code First DataAnnotations," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/gg193958.aspx. [Hämtad: 2015-05-11].

[57] Wikipedia, ”Undantagshantering,” [Online]. Tillgänglig: http://sv.wikipedia.org/wiki/Undantagshantering. [Hämtad: 2015-05-11].

[58] Microsoft, "Exception Class," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/system.exception%28v=vs.110%29.aspx. [Hämtad: 2015-05-11].

[59] Wikipedia, "List of HTTP status codes," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes. [Hämtad: 2015-05-11].

[60] Wikipedia, ”Language Integrated Query,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Language_Integrated_Query. [Hämtad: 2015-05-08].

[61] N. Arnott, "NArnott/System.Linq.Dynamic," [Online]. Tillgänglig: https://github.com/NArnott/System.Linq.Dynamic. [Hämtad: 2015-04-24].

[62] C. Calvert, "LINQ and Deferred Execution," [Online]. Tillgänglig: http://blogs.msdn.com/b/charlie/archive/2007/12/09/deferred-execution.aspx. [Hämtad: 2015-05-11].

[63] Microsoft, "dynamic (C# Reference)," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/dd264741.aspx. [Hämtad: 2015-05-06].

[64] M. Torgersen and J. Korsgaard, "New Features in C# 4.0," [Online]. Tillgänglig: https://intranet.cs.aau.dk/uploads/media/JacobKorsgaard2.pdf. [Hämtad: 2015-05-06].

[65] Microsoft, "Reflection (C# and Visual Basic)," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/ms173183.aspx. [Hämtad: 2015-05-06].

[66] Microsoft, "Dynamic Language Runtime Overview," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/dd233052.aspx. [Hämtad: 2015-05-06].

[67] Microsoft, "lock Statement (C# Reference)," [Online]. Tillgänglig: https://msdn.microsoft.com/en-us/library/c5kehkcz.aspx. [Hämtad: 2015-05-11].

[68] K. L. Kelly, ”Twenty-Two Colors of Maximum Contrast,” [Online]. Tillgänglig: http://www.iscc.org/pdf/PC54_1724_001.pdf. [Hämtad: 2015-04-14].

[69] Lantmäteriet, ”WGS 84,” [Online]. Tillgänglig: https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/Referenssystem/Tredimensionella-system/WGS-84/. [Hämtad: 2015-04-10].

[70] Google, ”google.maps.RectangleOptions object specification,” [Online]. Tillgänglig: https://developers.google.com/maps/documentation/javascript/reference#RectangleOptions. [Hämtad: 2015-05-04].

[71] J. Resig, "HTML 5 data- Attributes," [Online]. Tillgänglig: http://ejohn.org/blog/html-5-data-attributes/. [Hämtad: 2015-05-08].

[72] knockoutjs.com, "Observable Arrays," [Online]. Tillgänglig: http://knockoutjs.com/documentation/observableArrays.html. [Hämtad: 2015-05-04].

[73] Wikipedia, "Cross-site request forgery," [Online]. Tillgänglig: http://en.wikipedia.org/wiki/Cross-site_request_forgery. [Hämtad: 2014-04-29].

[74] Wikipedia, ”OAuth,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/OAuth. [Hämtad: 2015-04-28].

[75] Wikipedia, ”AngularJS,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/AngularJS. [Hämtad: 2015-04-28].

[76] J. Hansson, ”Fiskeapp får penganapp,” [Online]. Tillgänglig: http://sverigesradio.se/sida/artikel.aspx?programid=93&artikel=6150507. [Hämtad: 2015-05-08].

[77] Wikipedia, ”Gamification,” [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Gamification. [Hämtad: 2015-04-29].

[78] GitHub, Inc., "Search · stars:>1," [Online]. Tillgänglig: https://github.com/search?q=stars%3a%3E1&s=stars&type=Repositories. [Hämtad: 2015-05-08].

Bilaga 1 – Kravspecifikation

Kravspecifikation av Kristina Lehtonen Nilsson

1. Projektbakgrund

1.1 Beställaren

Beställaren av detta projekt är Sportfiskarna, Sveriges Sportfiske och fiskevårdsförbund, som är en ideell organisation och utgörs idag av 35 heltidsanställda, 35 säsongsanställda och ungefär 30 000 medlemmar. Utöver ideella arbeten och olika fiskevårdsprojekt bedriver Sportfiskarna bland annat webbförsäljning och fiskekortsförsäljning. För mer info se sportfiskarna.se

Kontaktperson och representant för detta projekt är Peter Belin, platschef för Sportfiskarna region Värmland.

Definitionen på sportfiske beskrivs som fritidsfiskare som fiskar med spö och lina för nöje och rekreation. Sportfiskarna arbetar främst med fisketillsyn, restaurering av vatten och fiskerelaterade undersökningar. De arbetar även för vidarestimulering och utvecklat fiskeintresse/naturintresse hos barn och ungdomar, exempelvis genom att arrangera olika fisketävlingar och ungdomsfiske. Sportfiskarna ser som sin viktigaste uppgift att vidareutveckla sportfisket, värna och vårda vatten, skapa opinion för sportfiske samt fiskevård. Förbundet samverkar med flera organisationer med gemensam målsättning att bevara och bevaka sportfiskets intresse. Exempel på regionala samverkanspartner är Länsstyrelser, fiskevårdsområden, Karlstads Universitet, Sveriges Lantbruksuniversitet och Havs- och Vattenmyndigheten.

1.2 Produkten

Sportfiskeförbundet vill skapa en ny webblösning av ”FångstDataBanken”. Webblösningen ska kopplas till de applikationer som framtagits i ett tidigare projekt som genomfördes år 2014. ”FångstDataBanken” är ett delmoment i ett större projekt för att få in fler rapporter från sportfiskare och därigenom ge ökad kunskap om sportfisket och fångster i utvalda vatten. Det ger möjlighet till förbättrad fiskevård och förvaltning. Idén är att sportfiskare ska rapportera in sina fångster via en applikation och att sportfiskare sedan genom ”FångstDataBanken” ska kunna se statistik över sitt eget fiske. Webblösningen syftar till att Sportfiskarna ska få bättre statistik över vad (vilken art) som fiskas samt var det fiskas och i vilken omfattning. Statistiken lämnas vidare till myndigheter och

forskare för att användas för fiskevård och förvaltning. Problematiken som är orsaken till att fångstdataprojektet startades är att fiske i vissa vatten i Sverige inte kräver något fiskekort, det vill säga fisket är fritt. Detta medför att sportfiskeverksamheten i dessa vatten är svåridentifierade. Ambitionen med ”FångstDataBanken” är att bidra till ökade kunskaper angående sportfiskeverksamheten för specifika vatten och därigenom bidra till en förbättrad fiskevård och förvaltning av vatten.

En studie har genomförts för att identifiera viktiga framgångsfaktorer för upplevd tillfredsställelse med ”FångstDataBanken”. Platschefen för Sportfiskeförbundet region Värmland, samt några sportfiskare har intervjuats för att undersöka orsaker till missnöje med den tidigare existerande webblösningen och vad som kan motivera fler sportfiskare till att rapportera in sina fångster. Resultatet av studien ligger till grund för kravspecifikationen.

2. Användare

I denna version av ”FångstDataBanken” finns två typer av användare: administratörer och externa användare. Administratörerna är de som arbetar för Sportfiskeförbundet och externa användare är sportfiskare som önskar använda webblösningen. För att Sportfiskarna ska få ut vad de önskar med ”FångstDataBanken”, det vill säga statistik över sportfiskeverksamheten, krävs att sportfiskare väljer att rapportera in sina fångster. Studien visar på att det finns en grupp sportfiskare som ogärna rapporterar sina fångster eftersom det innebär att de måste delge sin lokalisering. Lokalisering anses vara känslig information eftersom att sportfiskaren inte vill att andra sportfiskare ska upptäcka och fiska på samma vatten. Att lokalisering upplevs som känslig information försvårar således inrapporteringen och skapar därmed brister i statistiken. Studien visar även att statistik över väderförhållanden, vattentryck och vattentemperaturer anses av många sportfiskare som intressant statistik. Förhoppningsvis skapar en mer detaljerad statistik ökat intresse för inrapportering.

3. Krav

3.1 Funktionalitet

Webblösningens funktioner är uppdelade i olika prioriteringar (1-3). Funktionalitet med prioritet 1 är ”ska krav” och funktionalitet prioritet 3 är endast ”vore bra” krav. Tanken med prioriteringarna är att utvecklarna ska veta vilken funktionalitet som är viktigast att implementera. Om tiden för projektet tillåter kan även med fördel prioritet 2 och 3 implementeras. De funktionella kraven är strukturerade efter typ av användare, administratör *respektive* extern användare. Samtliga funktionaliteter för administratören presenteras först. Funktionaliteterna är indelade i huvudrubriker. Bokstäverna FU

följt av en siffra är en unik referens. Bokstaven P följt av en siffra beskriver vilken prioritet funktionaliteten har.

FU1P1: En administratör ska kunna logga in på och ut från webblösningen. Endast när administratören är inloggad ska tillgång till funktionalitet 2-9 ges.

FU2P1: En administratör ska kunna exportera databasinnehållet (samtliga inrapporterade fångster) till en Excel-fil på datorn. Ett program ska installeras av Sogeti, vilket möjliggör hantering av exporterad data.

FU3P1: En administratör ska kunna lägga till, redigera samt radera vatten i databasen. Det är avgörande att en administratör kan styra vilka vatten som är valbara vid inrapporteringen för att kunna anpassa datainsamlingen efter målsättning.

FU4P1: En administratör ska kunna ange zoner för ett specifikt vatten i databasen. Det är avgörande att en administratör kan styra vilka zoner som är valbara vid inrapportering eftersom att zonindelningen kan förändras.

FU5P1: En administratör ska kunna lägga till, redigera samt radera fiskarter i databasen. Det är avgörande att en administratör kan styra vilka fiskarter som är valbara vid inrapporteringen för att kunna anpassa datainsamlingen efter målsättning.

FU6P1: En administratör ska kunna lägga till, redigera samt radera fiskemetod i databasen. Det är avgörande att en administratör kan styra vilka fiskemetoder som är valbara vid inrapporteringen för att kunna anpassa datainsamlingen efter målsättning.

FU7P2: En administratör ska kunna redigera samt radera rapporter i databasen. Det är önskvärt att en administratör kan administrera databasen eftersom att databasen kan behöva rensas från icke önskvärda rapporter. Ett exempel på sådana rapporter är rapporter som har lagts in i testningssyfte.

FU8P3: En administratör ska kunna skriva inlägg. Dessa inlägg ska dateras samt organiseras i form av ett arkiv. En extern användare (inloggad som utloggad) ska kunna ta den av inlägg.

FU9P3: En administratör ska ha möjligheten att, utöver text, kunna bifoga bilder i inläggen.

FU10P1: En extern användare ska kunna logga in på och logga ut från webblösningen. Först när en extern användare är inloggad ska tillgång till funktionalitet 11-13 ges.

FU11P1: En extern användare ska utifrån sina egna inrapporteringar kunna se statistik över fisket. En extern användare ska kunna ange vilken art, fiskemetod, tidperiod samt vatten som det ska visas statistik över. För vidare beskrivningar av statistik se statisk 3.7.

FU12P2: En extern användare ska kunna se statistik över samtliga offentliga inrapporterade fångster. En extern användare ska kunna se statistik över den totala sportfiskeverksamheten. En extern användare ska kunna ange vilken art, fiskemetod, tidperiod samt vatten som det ska visas statistik över. För vidare beskrivningar av statistik se statisk 3.7.

FU13P3: En extern användare ska kunna se statistik över en annan extern användares offentliga inrapporteringar. En extern användare ska kunna ange vilken art, fiskemetod, tidperiod samt vatten som det ska visas statistik över. För vidare beskrivningar av statistik se statisk 3.7.

FU14P1: Webblösningen ska innehålla kontaktinformation till administratören.

FU15P1: Webblösningen ska innehålla index-sida med information angående syftet med ”FångstDataBanken”.

3.2 Användargränssnitt

Samtliga texter och Samtliga bilder ska Sportfiskarna producera. Loggan för ”FångstDataBanken” (se bilaga 1) ska vara synligt placerad på webblösningen (förslagsvis på det högra översta hörnet). Även loggan för sportfiskeförbundet (se bilaga 2) ska vara synligt placerad på webblösningen (förslagsvis på nedersta högra hörnet). Sportfiskarnas grafiska formgivare har inte satt några designrestriktioner specifikt för ”FångstDataBanken” utan vill ge synpunkter på framtagna designförslag. Däremot är det mycket viktigt att gränssnittet ska följa Sportfiskarnas standard (sportfiskarna.se) samt att det ska vara enkelt och stilrent. Slutligen är det viktigt att det finns en direktlänk från webblösningen till applikationen, exempelvis i form av en SQR-kod.

3.3 Användbarhet

Nedan finns användbarhetskraven, det vill säga krav på lättheten att använda respektive lära sig webblösningen. Ordet användare innefattar både administratören och externa användare. Bokstäverna AB följt av en siffra är en unik referens.

AB1: Viktigt är att göra webblösningen lätt att förstå. Så långt det går skall användarfel undvikas genom att låta användaren välja bland fördefinierade värden.

AB2: De felmeddelanden som härrör från felaktigt inmatad data ska presenteras längst upp på sidan där inmatningen gjorts.

AB3: Det ska vara klart och tydligt för en extern användare vad statistiken visar. Tydligheten kan stärkas genom att de val en användare gjort visas någonstans på sidan.

AB4: Viktigt är att göra webblösningen effektiv. Så få klick som möjligt ska krävas av användaren vid användning av webblösningen.

AB5: Viktigt är att göra webblösningen tydlig. Det ska tydligt framgå vart på webblösningen användaren befinner sig. Det ska även tydligt framgå om en användare är inloggade eller utloggad.

3.4 Felhantering

Nedan följer förslag till eventuell felhantering för webblösningen. Bokstäverna FH följt av en siffra är en unik referens.

FH1: Om obligatoriska fält ej är korrekt ifyllda ska det fällt som är felaktigt ifyllt rödmarkeras.

3.5 Säkerhet

Nedan följer säkerhetskrav för webblösningen, det vill säga händelser eller scenarion som inte får förekomma. Bokstäverna SH följt av en siffra är en unik referens.

SH1 En extern användare ska inte kunna kommentera eller på annat sätt göra inlägg på webblösningen.

SH2 Endast en administratör ska kunna exportera hem databasinnehållet till en Excel-fil på datorn.

SH3 Inrapporteringar från externa användare som är klassificerade som privata ska inte vara tillgängliga för andra externa användare. Dessa rapporteringar ska heller inte räknas in i statistik över den totala fiskeverksamheten.

SH4 En extern användare ska inte kunna rapportera eller på annat sätt hantera sina fångster på webblösningen. Rapportering ska endast skötas via applikationen.

SH5 Vid redigering eller annan förändring i databasinnehållet måste administratören bekräfta sin handling för att undvika oönskade förändringar i databasen.

SH6 Webblösningen måste ta hänsyn till uppdateringar i databasen föra att extern användare och administratör får korrekt och uppdaterat information.

3.6 Data

Nedan följer en beskrivning av hur webblösningen ska hantera data, det vill säga krav för begränsade värden. Bokstäverna DA följt av en siffra är en unik referens.

DA1: En administratör ska inte kunna skapa tomma inlägg på webblösningen.

DA2: En extern användare ska inte kunna skicka tomma (innehållslösa) mail till administratören. En extern hanvändare ska heller inte kunna skicka ett mail utan att ange avsändarens mailadress. (DA2 ska tillämpas om webblösningen ska innehålla en mailfunktion).

3.7 Statistik

Nedan följer kraven på vilken statistik som skall vara möjliga att ta fram.

3.7.1 Grundversion

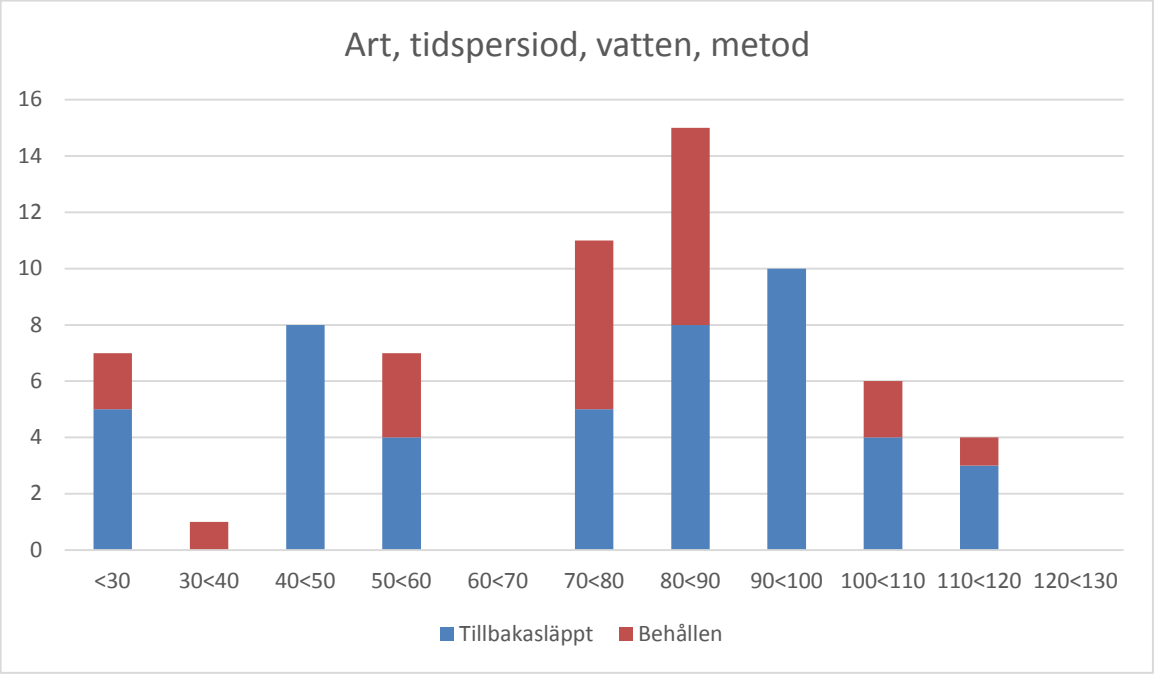
Grundversion beskriver minimumkraven för hur statistiken ska se ut och för hur en extern användare ska interagera med statistiken. Bokstäverna GV följt av en siffra är en unik referens.

GV1: En extern användare ska kunna se statistik över följande punkter: art, tid, lokalisering, metod, längd samt statistik över vad som är behållet och vad som är tillbakasläppt i vattnet. Statistiken ska vara liknande bilden nedan (se Figur 1).

GV2: En extern användare ska själv kunna anpassa vilken art, under vilken tidsperiod, med vilken metod och för vilket vatten alternativt zon som diagrammet ska visa statistik över.

GV3: En extern användare ska kunna se statistik över alla egna rapporteringar som har rapporterats in, förslagsvis genom att välja ska kunna välja ”alla arter”, för en odefinierad tid, ”alla metoder” och ”alla vatten”.

GV4: Hur en extern användare har valt att anpassa statistiken/diagrammet ska tydligt framgå (se lösningsförslag X).



Y-axeln beskriver antal och X-axeln längd i centimeter. Den andel av antalet fångade fiskar som är orange betyder att fisken är behållen. Resterande antal i färgen blå är fisk som är tillbakasläppt i vattnet igen.

3.7.2 Extra uppgift

Extra uppgiften beskriver en förbättring av grundversionen och är frivillig. Istället för att en extern användare ska ange art, tid, vatten och metod genom exempelvis listor (se 3.7.1 grundversion) ska metoden ”drag and drop” tillämpas. Metoden innebär att ett objekt kan ”dras” till en ny lokalisering. Detta innebär att en extern kund ”drar” in sina alternativ till diagrammet med musen och ”släpper” dem där. Diagrammet ska uppdateras efter respektive ”drag and drop” som görs och visa statistik över det som är indraget till diagrammet.

4. Lösningsförslag

Eventuella lösningsförslag kommer när kravspecifikationen är godkänd.

För undvika eventuella missnöjen med webblösningen föreslås ett iterativt arbetssätt. På så vis kan kunden följa produktens utveckling och bidra med åsikter och funderingar.

Bilaga 1

Loggan för ”FångstDataBanken”:

Bilaga 2

Loggan för Sportfiskeförbundet:

Bilaga 2 – Källkod för jämförelse av databasåtkomst

```
namespace Sportfiskarna.Web.Controllers
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Linq;
    using System.Net;
    using System.Web.Mvc;
    using Sportfiskarna.Repository;
    using Sportfiskarna.Repository.Models;
    using Sportfiskarna.Web.Extensions;

    public class TestController : BaseController<FishingArea>
    {
        private SportfiskarnaContext context;

        public TestController(SportfiskarnaContext context)
        {
            this.context = context;
        }

        public ActionResult UsingEfDirectly(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }

            var area = this.context.FishingArea.Find(id);

            if (area == null || area.Deleted == true)
            {
                return HttpNotFound();
            }
            else
            {
                var zone = this.context.FishingAreaZone.Where(z => z.FishingAreaId == area.Id &&
                    z.Deleted == false).FirstOrDefault();

                if (zone == null)
                {
                    return HttpNotFound();
                }
                else
                {
                    zone.Deleted = true;
                    zone.Visible = false;
                    zone.UpdatedWhen = DateTime.UtcNow;

                    this.context.Entry(zone).State = EntityState.Modified;

                    try
                    {
                        this.context.SaveChanges();
                    }
                    catch (DbUpdateConcurrencyException)
                    {
                        return new HttpStatusCodeResult(HttpStatusCode.InternalServerError);
                    }
```

```
                }
            }

            return new HttpStatusCodeResult(HttpStatusCode.OK);
        }

        public ActionResult UsingBaseController(int? id)
        {
            try
            {
                var area = GetItem(id);
                var zone = this.context.FishingAreaZone.Where(z => z.FishingAreaId == area.Id &&
                    z.Deleted == false).FirstOrDefault();

                if (zone == null)
                {
                    return HttpNotFound();
                }
                else
                {
                    zone.Deleted = true;
                    zone.Visible = false;
                    zone.UpdatedWhen = DateTime.UtcNow;

                    this.context.Entry(zone).State = EntityState.Modified;
                    this.context.SaveChanges();
                }
            }
            catch (InvalidIdException ex)
            {
                return ex.ErrorResult;
            }
            catch (DbUpdateConcurrencyException)
            {
                return new HttpStatusCodeResult(HttpStatusCode.InternalServerError);
            }

            return new HttpStatusCodeResult(HttpStatusCode.OK);
        }

        public ActionResult UsingRepository(int? id)
        {
            try
            {
                var area = new Repository<FishingArea, int>(this.context).FindById(id);
                var zoneRepository = new Repository<FishingAreaZone, int>(this.context);

                zoneRepository.Delete(zoneRepository.Entities.Where(z => z.FishingAreaId ==
                    area.Id).FirstOrDefault());
            }
            catch (RepositoryException ex)
            {
                return ex.AsHttpStatusCodeResult();
            }

            return new HttpStatusCodeResult(HttpStatusCode.OK);
        }
    }
}
```