Husmusen

En manual för skapandet av databassystem

Första upplagan, november 2022

Inledning

Den här manualen riktar sig till museum och liknande institutioner — stora som små — som vill skapa sina egna offentligt sökbara inventariesystem, men som kanske inte riktigt vet vad som behöver göras. Den här manualen är tänkt att vara en hjälp på vägen till ert inventariesystem.

Manualen är riktad dels till de som styr och ställer på museum, men också i delar till de som skulle programmera systemet. Dock är den ämnad att ge stöd och exempel, inte vara en hel programmeringskurs. En förhoppning med manualen är också slippa en situation där alla museum måste återuppfinna hjulet genom att skapa sina egna protokoll och standarder gång på gång. Det kan undvikas genom att försöka strömlinjeforma en modell eller ett system som alla kan följa för att få liknande, men behovsanpassade system. Det kommer medfölja exempel med manualen som är skrivna i *Javascript* i *Node JS*-miljön och som använder *MariaDB* som databasmjukvara. Dock går det givetvis att skriva ett databassystem i många andra språk och med en annan databasmjukvara också.

Denna manual är skriven och skapad på uppdrag av Bergdala glastekniska museum — ett arbetslivsmuseum i *Glasriket*. Mer information om dem hittas på deras hemsida: https://bergdala-glastekniska-museum.se/

Innehåll

Sammanfattning

Digitalt inventariesystem

Om ni läser denna manual, så vill ni antagligen skapa ett digitalt inventariesystem som låter er dela med er av ert museums inventarium till offentligheten. Det är inget litet projekt, men förhoppningen är att denna manual ska hjälpa er en bit på vägen och se till att vi alla slipper en situation då varenda museum behöver ta reda på all information själva — istället kan denna manual ge en strategi som kan användas av flera museum.

Resurser, kunskap, och kompetens

Det kommer krävas resurser, främst tid och människokraft. Det krävs också olika kunskaper och kompetenser för att skapa ett bra system. De mest relevanta i just en sådan här situation är kunskap inom: kulturarvsvärlden, programmering, samt grafisk- och gränssnittsdesign. Det medföljer också lite användbara resurser i manualens appendix.

Steg för steg

Här är en kort sammanfattning av ungefär vilka steg manualen går igenom:

1. Hitta/skapa ett team.
2. Skapa en planering för projektet.
3. Skriv en kravspecifikation.
4. Skapa en *API-definition*.
5. Implementera/skapa systemet.
6. Lansera systemet.

Det här är ett mycket förenklat tillvägagångssätt och det finns givetvis massvis med delmål. Dock ger det förhoppningsvis en liten inblick i ungefär vad som krävs. Hela processen beskrivs i mer detalj i resten av manualen.

Lycka till!

Kapitel 1: Digitalt inventariesystem

Ett stort projekt

Att skapa digitalt inventariesystem för ert museum är inte ett litet åtagande. Dock är det viktigt i ett alltmer digitaliserat samhälle att information finns både fysiskt och digitalt. Att helt från grunden skapa ett digitalt inventariesystem kan verka svårt och skrämmande. Det är svårt, men det ska inte hindra museum från att göra det. Om tillräckligt med resurser och kompetens finns blir det enormt mycket lättare.

Resurser, kunskap, och kompetens

Som sagt kommer det krävas resurser — främst i form av tid och mänsklokraft — för att skapa ett digitalt inventariesystem. Det krävs också olika kompetenser och kunskaper. Det finns många delar i att skapa ett digitalt inventariesystem. Det krävs planering, kodning och programmering, dokumentation, grafisk- och gränssnittsdesign, och säkert andra mindre bitar. Eftersom det är ett så brett område så ska denna manual försöka skapa en grund att stå på och utgå ifrån. Manualen kommer gå in lite på alla de tidigare bitarna, men dessvärre finns det ingen exakt lösning som kommer fungera för alla. Därför kommer manualen vara relativt generell.

Som extra resurser till manualen finns:

1. *Husmusen — protkolldefinition*. En protokolldefinition — om än ganska grundläggande — att utgå ifrån när ni väl skapar er egen.
2. *Husmusen*. En exempelimplementering av ovan nämnda protokolldefinition som också går att utgå ifrån vid skapandendet av ert digitala inventariesystem.
3. *Husmusen — loggbok*. En loggbok över utvecklandet av en plan och utförandet av den för skapandet av exempelimplementeringen.

Var ni kan finna resurserna står i appendix längst bak i manualen.

Kapitel 2: Hitta kunskap & kompetens

Som nämndes i föregående kapitel så krävs en bredd av kompetens och kunskap. Det kan verka skrämmande när det behövs så mycket olika kompetenser och kunskaper. Dock så kommer denna mångfald att leda till ett bättre system, eftersom alla delar är väsentliga! Det blir inget bra system utan en programmerare som kan skriva det, men det blir heller inget bra system utan en person inom kulturarvsvärlden som kan ge programmeraren en insyn som hjälper den att veta vad som behövs eller är bäst för systemet. Vidare behövs det också kunskaper inom gränssnittsdesign för att skapa ett användargränssnitt som offentligheten kan använda för att söka i systemet. Alla dessa kunskaper har även svårt att komma någonstans utan en plan och en samordnare/projektledare som kan leda projektet i rätt riktning.

Det här kapitlet kommer att gå in på vilka kompetenser och kunskaper som behövs och vad det är bra om de tänker på. Det är inte säkert att varje kompetens- och kunskapsområde kräver en person vardera. Det kan mycket väl vara så att en person kan täcka två eller fler områden eller att ett område kräver två eller fler personer för att få ett bra resultat.

Kulturarvsvärlden

Förhoppningen är att en av er — som läser den här manualen — jobbar inom kulturarvsvärlden och därmed vet vilka delar och komponenter i ett system som skulle vara viktiga för just er. Det är inte säkert att alla system kommer behöva exakt samma saker. Ett system riktat mot glaskonst och pappersindustri har säkert sina likheter, men också olikheter. Det främsta som du eller ni som har kunskaper och kompetens inom kulturarvsvärlden behöver fokusera på är vad som behövs för just ert museum eller er institution. Dock kan det vara bra att ha en liten aning om vad som kan behövas för andra institutioner och vad som skulle krävas för att era system skulle kunna samverka.

Projektledning

Förhoppningen är också att det finns någon som kan vara en god lagledare som för laget framåt och ger stöd och stöttning där det behövs. Det kan vara bra om projektledaren har lite insyn i alla de andra bitarna, men behöver absolut inte vara en expert. Främsta fokuset för dig som agerar projektledare är att se till att det finns en bra plan och att den följs eller att det görs ändringar i planen där det behövs allteftersom projektet utförs.

Programmering

När det väl kommer till skapandet av systemet krävs en eller flera programmerare. Det är bra om någon av dem känner sig bekväm i mjukvaruarkitektur, så att en struktur kan skapas för programmet. Struktur innebär bland annat *filstruktur*. Vilka typer av filer ska ligga var i projektet? Vilka modeller/klasser/strukturer kommer behövas för ert system? Vad står de för och vad gör dem? Struktur innebär också saker som: Vilka teknologier, ramverk. och kodbibliotek ska användas? Vilket språk ska användas[1]? Det är också bra om någon har insyn i säkerhetsvärlden. Det är viktigt att undvika sårbarheter i så stor utsträckning som möjligt. Om ekonomin finns så går det att hyra in firmor som testar säkerheten i ert system.

Gränssnittsdesign

När det väl finns ett fungerande system kan det vara bra om det faktiskt kan användas av allmänheten. För att allmänheten lätt ska kunna använda systemet är det bra om någon med kunskap inom gränssnittsdesign kan ta fram ett grafisk användargränssnitt. Fokus här bör ligga på att göra det så lättanvänt som möjligt, utan att ta bort viktiga funktioner. Ett annat viktigt fokus är tillgänglighet. Det innebär att se till att så många som möjligt kan använda gränssnittet, oavsett eventuella svårigheter eller hinder, såsom färgblindhet eller annan synskada.

Administration

1 Viktigt att komma ihåg är att det kan finnas krav på språk. Det kan vara så att museets webbhotell bara har stöd för t.ex. *PHP*. Då ska ni inte använda ett annat språk, eftersom museets webbhotell i så fall inte skulle ha stöd för det.

När systemet väl är klart så behöver det finnas någon eller några som kan administrera det. Administration innebär två saker: se till att systemet är online och kan nås av allmänheten och att lägga in inventariet i systemet. Det förstnämnda är mer åt *serveradministratörshållet* och det andra mer åt *museiarbeterhållet*. Antagligen kommer samma personer lägga in inventariet som är de som tillhör *kulturarvsvärlden*.

Hitta ett team

När ni nu känner till kunskaps- och kompetensområdena behöver ni hitta ett lagom stort team som kan skapa ett system. *Lagom stort* innebär att det finns tillräckligt många personer så att alla kompetensområden täcks, men inte så många att det blir för många kockar som rör om i grytan. Här är ett exempel på hur ett team omfattande fyra personer skulle kunna se ut:

1x Projektledare som är kunnig inom kulturarvsvärlden.

2x Programmerare. En med inriktning på säkerhet och bakdelen av systemet. Den andra med inriktning systemets framsida, d.v.s. användargränssnittet.

2x Administratörer som kan lägga in föremål i systemet. En av är också projektledaren.

Det här är såklart bara ett exempel på hur ett team skulle kunna se ut. Teamets storlek kan bero på flera saker, tidsomfång, tillgång till människokraft, ekonomi, och mer. Det kan också vara lite klurigt att hitta de olika kompetenserna. Dock, efter lite letande bör ni kunna finna ett team.

När det väl finns ett team så behöver det skapas en projektplan.

Kapitel 3: Projektplan

När det väl finns ett team som kan skapa ett inventariesystem, så behöver det skapas en projektplan där projektets omfång beskrivs. En projektplan ska aldrig vara helt huggen i sten, för ibland kan den behöva ändras lite i efterhand. Dock är det bra att ha en liten aning om hur ni önskar att projektet ska gå tillväga.

Delar i en projektplan

I en projektplan finns det lite olika delar. Delarna kan variera lite beroende på projekt. De vanligaste delarna är en lista med ansvar/roller, en tidsplan, och en ekonomisk plan eller budget.

Ansvar och roller

Det är viktigt med samarbete i ett projekt, men det betyder inte att alla ska behöva fokusera på allt. Det kan vara bra att dela upp ansvarsområdena. Då kan uppdragen delegeras på så sätt att alla får uppdrag de är bekväma med och kunniga om. I föregående kapitel presenterades olika kunskaps- och kompetensområden som kan vara relevanta under projektet. Roller och ansvar bör delas ut till de med kunskap och kompetens. Det innebär, som exempel, att programmeraren får som roll att vara programmerare, inte t.ex. en gränssnittsdesigner. Det gör så att alla i teamet kan ta hand om det som den kan bäst. Den här delen av projektplanen handlar helt enkelt om att få ned de olika kompetenserna och kunskaperna på papper.

För att skapa ett exempel: ett team består av fyra personer. Eva, Tobias, Ali, och Lisa. Eva jobbar på ett arbetslivsmuseum tillsammans med Tobias. Tobias har dessutom kunskap inom projektledning då han tidigare jobbat med det. Ali och Lisa är programmerare. Ali har inriktat sig mer på gränssnitt, tillgänglighet, och webbteknologier, medan Lisa har riktat sig mer mot serverteknologier, algoritmer, och säkerhet. Då väljer de att lägga upp rollerna på följande vis: Tobias är projektledare. Lisa har jobbat längst inom museivärden och vet därför vad som rent konkret behövs för systemet. Ali tar hand om användargränssnittet och Lisa kommer ta hand om att programmera *baksidan* av systemet.

Det är viktigt att komma ihåg att rollerna inte är helt huggna i sten. Ibland kan de olika personerna behöva hjälpa varandra. Det kan till exempel komma en situation då Ali inte har något att göra, då kan han hjälpa Lisa att programmera *baksidan* av systemet så att projektet går framåt snabbare.

Tidsplan

En tidsplan är en grovplanering på ungefär när något ska göras och i vilken ordning. Den behöver inte följas helt och hållet till pricka, men det är bra att vara något i fas med den för att bli klar inom den tänkta tidsramen.

En enkel tidsplan kan se ut såhär:

> **Fas 1: Planering**
>
> **Vecka 1:** Få klart en grovplanering.
>
> **Vecka 2:** Få fram en kravspecifikation.
>
> **Vecka 3-4:** Skissa fram en protokolldefinition för systemet. Skissa dessutom fram en idé på hur ett grafisk användargränssnitt för systemet kan se ut.
>
> **Fas 2: Implementera protokollet**
>
> **Vecka 5-8:** Arbeta med att implementera protokollet. Gör lite tester där det behövs. Ändra protokolldefinition vid behov — ta ifrån, lägg till, ändra saker.
>
> **Vecka 7-9:** Skapa det grafiska användargränssnittet.
>
> **Fas 3: Integrera**
>
> **Vecka 10:** Lansera systemet på internet.

Detta är en mycket förenklad tidsplan, och det kan mycket väl vara så att den har allokerat för lite tid. Det är ofta smart att planerna att saker tar lite längre tid än vad ni tror. Det kan t.ex. komma saker i vägen eller så dyker något nytt upp som ni inte tänkt på innan. Då är det skönt med lite mer tid. Observera att olika moment i tidsplanen också kan överlappa, såsom den gör i *fas 2*. Ibland går det att göra fler saker samtidigt för att spara tid. Detta gäller främst när moment X inte är beroende av moment Y.

Budget — ekonomisk plan

Det kan också vara en bra idé att skapa en budget för projektet. Det kan vara så att budgeten endast består av löner till de som jobbar på projektet, men i vissa fall kan det vara så att vissa inköp behöver göras, beroende på hur systemet utvecklas och vilka resurser, tjänster, och verktyg som behövs.

Ett levande dokument

Som sagt så kan saker ändras, därför kan det vara bra att se projektplanen lite som ett *levande dokument* — alltså att det kan förändras med tidens gång. Huvudsyftet med projektplanen är att ge ett ramverk att arbeta utifrån så att projektet har någon form och struktur.

I de kommande kapitlen kommer olika delar som är viktiga för projektet att presenteras: kravspecifikation, protokolldefinition, vanliga säkerhetsbrister.

Kapitel 4: Kravspecifikation

En kravspecifikation är ett dokument som beskriver vad en mjukvara ska kunna göra eller hur den ska se ut. En väldigt enkel kravspecifikation skulle kunna se ut ungefär såhär:

- Användare ska kunna söka efter föremål i databasen.
- Användare ska kunna sortera resultaten efter alfabetisk ordning, senast tillagda/uppdaterade, relevans, osv.
- Varje objekt i databasen ska ha ett eget unikt ID och en *permalänk*.

Den här kravspecifikationen är definitivt ofullständig, men den kan ge en liten blick över hur grunden till en kravspecifikation ser ut. Ju mer djupgående och detaljrik kravspecifikationen är, desto lättare kommer programmerare ha att följa den. En kravspecifikation kan skrivas av i princip vem som helst, men den eller de som skriver den behöver ha en idé över hur de vill att systemet som ska skapas ska se ut eller fungera.

Skapa en kravspecifikation

Det första steget blir att skapa er egen kravspecifikation alternativt använda eller anpassa exemplet som medföljer manualen. Exemplet är en del av *Husmusen — protokolldefinition* som ni hittar i appendix. Om exemplet följs till punkt och pricka så skulle ni skapa ett *Husmusenföljsamt* system. Det skulle innebära att exempelkoden är mer överensstämmande med ert system, och är därmed mer relevant för er. Dock är det viktigt att komma ihåg att ni kan behöva göra egna anpassningar också.

Det är bra om kravspecifikationen är så heltäckande som möjligt redan från början, men ibland kan ändringar/tillägg/borttagningar behöva göras under skapandet av systemet. Det skulle t.ex. kunna komma fram ett nytt krav på en funktion som behöver finnas med i systemet. Det kan vara bra att kolla upp med programmerarna hur de vill att en situation ska gå till om kravspecifikationen behöver ändras.

Kapitel 5: Skapa en API-definition

En *API-definition* beskriver en mjukvaras struktur. Den beskriver bland annat hur en kommer åt eller ändrar viss data. Den bör skrivas av någon som har koll på hur mjukvaran kommer att fungera, till skillnad från kravspecifikationen som kan skrivas av i princip vem som helst. Detta innebär att det är en programmerare som bör skriva den.

API-definitionen bör täcka allt som specificeras i kravspecifikationen. Det betyder att varje funktion som nämns i kravspecifikationen behöver vara dokumenterad i *API-definitionen*. *API-definitionen* kan också ses som ett *levande dokument*. Det är bra att skriva den till en början. Då är det lättare att veta vad som behöver implementeras. Dock kan tillägg/ändringar/borttagningar behöva göras under projektets gång.

Ett API kan syfta på två lite olika saker: hur en använder ett kodbibliotek, men också hur en kan få information av en webbserver. Båda bör vara dokumenterade i er *API-definition*.

Utöver att dokumentation kan vara bra som planering och för att komma ihåg saker, så kan det också hjälpa andra personer komma in i systemet och kunna utveckla det. Det kan vara bra om ni i framtiden vill t.ex. utöka eller förbättra systemet.

Enkelt exempel

Såhär skulle en väldigt enkel *API-definition* av en boks utseende i databassystemet kunna se ut:

```
Book {
  Title:  String,
  Author: String,
  Year:   Integer,
  ISBN:   String
}
```

String är en textsnutt och *Integer* är ett heltal. Den här definitionen säger helt enkelt att en bok består av fyra fält: *titel*, *författare*, *år*, och *ISBN*. I appendix finns *Husmusen — protkolldefinition*. Den har med mer utvecklade exempel på hur en *API-definition* kan se ut.

Kapitel 6: Filsystemsstruktur

När det väl finns en *API-definition*, så är det dags att börja programmera. Då krävs det först en struktur för koden i programmet. *Struktur* innebär dels kodningsstil i alla filer, men också att det finns en *filsystemsstruktur*. Det kan bli svårt att läsa kod från ett projekt om det är skrivet med olika stil överallt. Därför är det bra att komma överens om en stil som alla följer. *Stil* kan handla om när det ska finnas radbrytningar och dylikt. *Filsystemsstruktur* handlar om vilka typer av filer som ska i vilken mapp. I appendix finns *Husmusen — filsystemsstruktur* där finns ett exempel på hur ett projekt kan strukturera. Nedan kommer jag beskriva några av mapparna och filerna.

index.js

index.js är ingången till projektet. Den läser in all kod och startar den.

api/

I mappen *api* ligger all kod som serverar information. Där ligger bland annat sökfunktionen och funktionen att få information om museet.

app/

I *app*-mappen ligger det grafiska användargränssnittet. Det är indelat i två andra mappar *static* och *views*. I *static* ligger alla statiska filer, så som logotyper och skript. I *views*-mappen ligger olika layouter och vyer som finns i gränssnittet. Det finns en mapp för mallar, *templates*, en för komponenter, *components*, och en för sidor, *pages*.

lib/ och models/

I *lib*-mappen ligger olika filer som hjälper API:t. Där ligger bland annat *database.js* som används för att koppla systemet till en databasmjukvara. I mappen *models* ligger alla modeller för systemet, däribland *Item* och *Keyword*.

Kapitel 7: Säkerhet

Det finns lite olika typer av attacker och säkerhetsbrister. Nedan beskrivs några vanliga som kan påverka ert system och sätt att åtgärda eller förhindra dem. Beroende på programmeringsspråk och kodbibliotek kan de olika sårbarheterna vara olika förekommande eller allvarliga.

Attack: överbelastningsattack

En typ av attack är *överbelastningsattack*. Det kan innebära lite olika saker, men kort och gott så innebär det att någon skickar väldigt många förfrågningar till en server på en gång i hopp om att överbelasta och crasha den. Det går att försöka förhindra dem genom en bra brandvägg och/eller genom så kallade *rate limits*. *Rate limits* begränsar hur många förfrågningar en server kan ta på en gång. Det kan vara per IP eller globalt.

Sårbarhet: SQL-injektion

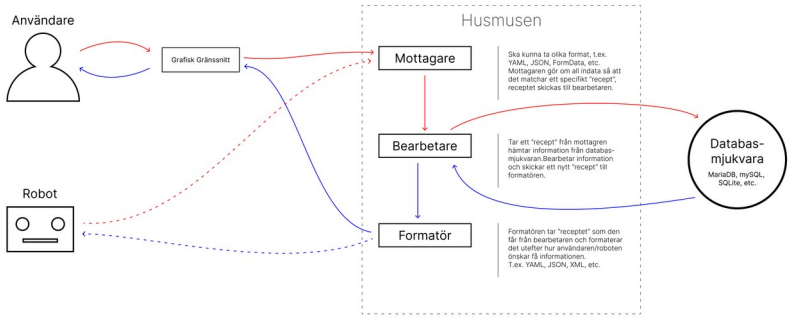
En *SQL-injektion* innebär att någon, med hjälp av någon funktion — t.ex. sökfunktionen — i servern försöker göra saker med databasen som den inte ska kunna, t.ex. ta bort föremål. Vissa kodbibliotek har funktioner som förhindrar detta. Läs era kodbiblioteks dokumentation och se hur de kan användas för att förhindra *SQL-injektioner*. Ofta handlar det om att *sanitisera* förfrågningar så att de inte gör något de inte borde.

Sårbarhet: fel typer

Om en förfrågan innehåller två *parametrar/queries* med samma namn hanterar olika kodbibliotek det olika. Vissa lägger ihop alla de olika värdena till en *array*, medan andra tar det först-eller sistnämnda. Ett exempel är om någon har parametrarna *?val=1&val1=2&val1=3*. Det kan alltså tolkas på tre olika sätt: som en *array* med värdena 1, 2, och 3; som värdet 1; eller som värdet 3. Om din kod får en typ som den ”inte är beredd på” kan det orsaka att programmet crashar då det inte vet vad det ska göra med den oväntade typen. Det är därför smart att läsa era kodbiblioteks dokumentation för att ta reda på hur de hanterar att få flera *parametrar/queries* med samma namn.

Kapitel 8: Hur fungerar Husmusen?

Detta kapitel kommer handla om hur exempelimplementering av *Husmusen* fungerar och lite olika upptäckter som gjorts under utvecklingen. Nedan finns ett väldigt simplifierat diagram av hur *Husmusen* fungerar:



Husmusen består av fem komponenter: *grafiskt gränssnitt*, *mottagare*, *bearbetare*, *formatör*, och *databas*. Dock är dessa delar abstrakta. Det är inte så att det finns en fil per komponent, till exempel. Faktum är att *mottagare* och *formatör* ligger i samma fil. Den ena tar *JSON* eller *YAML* och gör om det till till data som kan användas i koden och den andra gör tvärtom: tar data från koden och gör om den till *JSON* eller *YAML* och skickar iväg det. Interaktionerna med databasen är spridda över de olika modellerna som finns, däribland *Item* och *Keyword*.

Dock, från början var det tänkt att alla dessa komponenter skulle vara lösa från varandra och därför vara *modulära* så att de, i teorin, skulle kunna vara utbytbara med en komponent skrivet i annat språk. Dock, detta skulle öka komplexiteten kortsiktigt — och långsiktigt kanske inte ge så mycket? Därför blev det aldrig av. I ert system kan det vara värt att tänka på modulär kod.

Något som också kan vara viktigt är att ha ett offentligt API som tillåter att andra maskiner kan läsa er data. I diagrammet ovan står *grafisk gränssnitt* separat från *Husmusen*. Det är för att ”robotar” (datorer) ska kunna nå data i *JSON*- och *YAML*-format så att de lätt kan bearbeta dem. Dock, för människor är inte *JSON* eller *YAML* så önskvärda format — de går att förstå av en människa, men är mer tänkt för *programmerare-till-dator-kommunikation*. Den genomsnittliga människan skulle föredra t.ex. en tabell som visar informationen. Därför hamnar ett gränssnitt mellan användaren och *Husmusen*.

Kapitel 9: Tips

I det här kapitlet kommer jag snabbt gå igenom lite tips som kan hjälpa till när det kommer till att skapa ert system.

Använd en linter

En *linter* är ett program som söker igenom koden och granskar kodstilen för att se att den är likadan överallt. Om det är flera personer som jobbar på projektet så kan de hjälpa dem att skriva kod som ser liknande ut och som därmed är lättare att gå igenom och utveckla. *Linters* kan också ge information om olika error som finns i koden, t.ex. felstavade variabler.

Exempel på linters:

Här är några exempel på olika *linters*:

- *Eslint* Javascript, Typescript, JSX, TSX
- *Stylelint* CSS, SASS, SCSS
- *Mypy* Python
- *Cpplint* C, C++
- *Checkstyle* Java

Listan bör betraktas som ofullständig. Om det språk ni använder inte finns med i listan så kan ni söka efter ”LANG linter” på internet, där *LANG* givetvis är det språk ni tänker använda.

Kodskanning: säkerhetsbrister

Det finns olika program och tjänster som kan skanna er kod efter vanliga säkerhetsbrister. Om ni använder *GitHub* så är det relativt lätt att integrera med *CodeQL*. Det går att göra med hjälp av en *GitHub action*. Det gör att ni lättare kan upptäcka, och därmed ”bekämpa” säkerhetsbrister i ert system. För mer information se *Kodskanning* i kapitlet *Vidare läsning*.

Kapitel 10: Vidare läsning

Kodskanning

1. https://github.com/github/codeql-action
2. https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning-with-codeql

Kulturarvets digitalisering

Boken *Kulturarvets digitalisering* av Henrik Summanen handlar om varför stora webbtjänster kan lyckas tillgängliggöra så mycket information till allmänheten, men de flesta museum inte gör det. Summanen diskuterar olika problem och lösningar som kan möjliggöra digitalisering av kulturarv.

ISBN: 978-9-18-934516-4

RAÄ:s dokumentation — K-samsök

Riksantikvarieämbetet (RAÄ) har ett kulturarvsarkiv med objekt som de skördar från olika databaser. Om ni vill göra era objekt mer synliga kan ni koppla er till dem. Det finns mycket information på deras hemsida som ni hittar här: https://www.raa.se/hitta-information/k-samsok/

Appendix

Husmusen — protkolldefinition

Husmusen — prokolldefinition är ett exempel på hur en protokolldefinition kan se ut. Den är relativt grundläggande, men förhoppningen är att den kan bidra till en lite mer stabil grund att utgå ifrån.

Den ligger uppe på *GitHub* här: https://github.com/mushuset/docs

För att hitta den, navigera in i mappen ”protokolldefinition” och leta efter den senaste upplagan och revisionen.

Husmusen — kodexempel

Husmusen är en exempelimplementering av ovanstående protkolldefinition. Den har i princip bara de funktioner som krävs för att vara kompatibelt med protokolldefinitionen. Dock har den också ett grundläggande grafisk användargränssnitt. Koden är ämnad att ge stöd till hur liknande projekt skulle kunna implementeras. Det finns också med en guide i *README*-filen om hur en går tillväga för att starta upp servern.

Den ligger också uppe på *GitHub*: https://github.com/mushuset/husmusen

Husmusen — loggbok

Husmusen — loggbok är ett dokument som innehåller utvecklingsloggboken från skapandet av *Husmusen — kodexempel*. Även den finns på *GitHub*. Den finns i mappen ”loggbok” här: https://github.com/mushuset/docs

Husmusen — filsystemsstruktur

```
Husmusen/
├── api/
│   ├── 1.0.0
│   │   ├── 1.0.0.js
│   │   ├── file.js
│   │   ├── item.js
│   │   ├── keywords.js
│   │   └── log.js
│   ├── api.js
│   ├── auth.js
│   └── db_info.js
├── app/
│   ├── static/
│   │   ├── scripts
│   │   ├── fonts.css
│   │   ├── logo.svg
│   │   ├── main.css
│   │   ├── OpenSans-Italic.ttf
│   │   └── OpenSans.ttf
│   ├── views/
│   │   ├── components/
│   │   ├── pages/
│   │   └── templates/
│   └── routes.js
├── lib/
│   ├── authHandler.js
│   ├── database.js
│   ├── graphicalApp.js
│   ├── log.js
│   └── requestHandler.js
├── models/
│   ├── DBInfo.js
│   ├── Error.js
│   ├── File.js
│   ├── Item.js
│   └── Keyword.js
├── index.js
├── LICENSE
├── package.json
├── pnpm-lock.yaml
├── README.md
└── setup.js
```