**Teknisk specifikation av JSON-editor för kravlistor**

**Version:** 1.1 (baserad på senaste diskuterade kod och UI-justeringar)  
**Datum:** 2024-05-15

**1. Översikt**

Projektet är en webbaserad klientapplikation (single-page application) för att visa, redigera och hantera JSON-filer som följer strukturen för Diggs (Myndigheten för digital förvaltning) tillsynsmanuals kravlista. Applikationen är byggd med HTML, CSS och modulär JavaScript (ES-moduler) och körs helt i användarens webbläsare. Ingen backend-server krävs förutom för att servera de statiska filerna.

**2. Teknisk Stack**

* **Frontend:**
  + HTML5
  + CSS3 (med CSS Custom Properties för theming, Flexbox/Grid för layout)
  + JavaScript (ECMAScript 2015+ moduler)
* **Externt Bibliotek:**
  + FileSaver.js (för att trigga filnedladdning i webbläsaren)

**3. Projektstruktur (Filer)**

.

├── index.html // Huvudsaklig HTML-fil

├── style.css // All CSS-styling

├── js/

│ ├── main.js // Applikationens startpunkt, binder händelselyssnare

│ ├── \_-----\_constants.js // Konstanter (ikoner, standardnycklar, etc.)

│ ├── \_-----\_dom\_element\_references.js // Referenser till DOM-element

│ ├── \_-----\_file\_handling.js // Filuppladdning, parsning, nedladdning

│ ├── \_-----\_global\_state.js // Globala tillståndsvariabler

│ ├── \_-----\_metadata\_functions.js // Funktioner för att visa/redigera metadata

│ ├── \_---\_requirement\_functions.js // Funktioner för att visa/redigera krav

│ ├── \_-----\_ui\_functions.js // Generella UI-uppdateringsfunktioner

│ └── \_-----\_utils\_\_helpers.js // Hjälpfunktioner (escape, parseMarkdown, etc.)

└── README.md // (Rekommenderas)

**4. Datamodell (JSON-struktur)**

Applikationen förväntar sig och producerar JSON-filer med följande huvudstruktur:

{

"metadata": {

"title": "String",

"description": "String",

"version": "String (format: YYYY.M.rSEQ)", // Ex: "2025.5.r1"

"language": "String (ISO 639-1 code, e.g., 'sv')",

"publisher": {

"name": "String",

"contactPoint": "String (email)"

},

"source": {

"url": "String (URL)",

"title": "String",

"retrievedDate": "String (YYYY-MM-DD)",

"format": "String"

},

"dateCreated": "String (YYYY-MM-DD)",

"dateModified": "String (YYYY-MM-DD)",

"license": "String",

"keywords": ["String", "String", ...],

"pageTypes": "String (komma-separerad lista)",

"contentTypes": [

{ "id": "String (auto-genererad slug)", "text": "String" },

...

],

"monitoringType": { // Obligatorisk, default till 'web' om saknas

"type": "String ('web', 'app', 'product', etc.)",

"text": "String (användarvänlig beskrivning)"

}

},

"requirements": {

"unique-requirement-key-1": { // Nyckeln är oftast en UUID eller genererad från titel+UUID

"metadata": { // Metadata specifik för detta krav

"mainCategory": { "text": "String", "key": "String (slug)" } | "String (legacy, konverteras)",

"subCategory": { "text": "String", "key": "String (slug)" } | "String (legacy, konverteras)" | "", // Tom sträng om ingen

"impact": { // Defaultvärden appliceras om saknas

"isCritical": false, // boolean

"primaryScore": 0, // number

"secondaryScore": 0, // number

"assumedCompliance": false // boolean, default till false vid skapande/redigering

}

},

"id": "String (oftast en UUID, eller en mer läsbar ID)",

"title": "String (obligatorisk)",

"standardReference": { "text": "String", "url": "String (URL)" } | "String (legacy, konverteras)", // Tom sträng om ingen

"instructions": [ // Array av instruktionsobjekt

{ "id": "String (auto-genererad)", "text": "String (stödjer enkel Markdown)" }

],

"checks": [ // Array av kontrollobjekt

{

"id": "String (auto-genererad)",

"condition": "String (stödjer enkel Markdown, obligatorisk för att spara check)",

"logic": "String ('AND' eller 'OR', default 'AND')",

"passCriteria": [

{ "id": "String (auto-genererad)", "requirement": "String (stödjer enkel Markdown)" }

],

"ifNo": [ // Alternativa kriterier

{ "id": "String (auto-genererad)", "requirement": "String (stödjer enkel Markdown)" }

]

}

],

"exceptions": "String (stödjer enkel Markdown)",

"examples": "String (stödjer enkel Markdown)",

"tips": "String (stödjer enkel Markdown)",

"commonErrors": "String (stödjer enkel Markdown)",

"expectedObservation": "String (stödjer enkel Markdown)", // Nytt fält

"contentType": ["String (ID från metadata.contentTypes)", ...], // Array av contentType IDs

"key": "String (samma som nyckeln i requirements-objektet, auto-genererad)"

},

"unique-requirement-key-2": { ... }

}

}

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Json

IGNORE\_WHEN\_COPYING\_END

**Viktiga punkter gällande datamodellen:**

* **Standardnycklar:** Vid sparande säkerställer applikationen att varje kravobjekt innehåller en uppsättning standardnycklar (STANDARD\_REQUIREMENT\_KEYS i \_-----\_constants.js). Saknade nycklar läggs till med defaultvärden (REQUIREMENT\_KEY\_DEFAULTS).
* **Nyckelgenerering:**
  + metadata.contentTypes[].id: Genereras från text med generateKeyFromName.
  + requirements[reqKey].key: Sätts till att vara samma som reqKey.
  + requirements[reqKey].id: Genereras som UUID om nytt, annars behålls befintlig.
  + requirements[reqKey] (själva nyckeln reqKey): Genereras från titel och ID med generateRequirementKey.
* **Markdown:** Fält markerade med "(stödjer enkel Markdown)" parsas med parseSimpleMarkdown vid visning.

**5. Applikationsflöde och UI-komponenter**

**5.1. Initialvy (initializeUI)**

* **HTML:** #uploadSection visas inuti #contentDisplay.
* **Funktionalitet:** En filuppladdningsknapp (#fileInput) och instruktioner. Alla andra kontroller och innehållsområden är dolda.

**5.2. Filuppladdning (handleFileUpload i \_-----\_file\_handling.js)**

1. Användaren väljer en JSON-fil.
2. Filen valideras (typ application/json).
3. Filens innehåll läses som text och parsas som JSON.
4. JSON-strukturen valideras (måste ha metadata och requirements).
5. metadata.monitoringType defaultas om den saknas.
6. Data lagras i state.jsonData.
7. UI uppdateras:
   * #uploadSection och #contentDisplay döljs.
   * #postUploadControlsContainer och #controlsDivider visas.
   * #dynamicContentArea visas med ett bekräftelsemeddelande.
   * #saveChangesButton är initialt dold.
   * #filterSortRow är initialt dold.

**5.3. Huvudkontroller (#postUploadControlsContainer)**

* **"Visa metadata" (#showMetadataButton):** Anropar displayMetadata.
* **"Visa kontroller" (#showRequirementsButton):** Anropar displayRequirements (återställer filter/sortering).
* **Sök- och filterrad (#filterSortRow):**
  + Visas endast när kravlistan (displayRequirements) är aktiv.
  + **Sortering (#sortOrderSelect):** Dropdown för att välja sorteringsordning. Vid ändring uppdateras state.currentSortOrder och displayRequirements anropas.
  + **Sökfält (#searchInput):** Fritextsökning. Vid input uppdateras state.currentSearchTerm och displayRequirements anropas (med debounce).
  + **"Lägg till krav" (#addRequirementButton):** Anropar renderRequirementForm(null).
* **"Spara ändringar (JSON)" (#saveChangesButton):**
  + Visas endast om state.isDataModified är true.
  + Anropar downloadJsonFile.
  + En inline-bekräftelse visas bredvid knappen efter lyckad nedladdning.

**5.4. Dynamiskt Innehåll (#dynamicContentArea)**

Detta område renderar olika vyer beroende på användarens val. Funktionen setupContentArea förbereder området och hanterar synlighet för #filterSortRow.

* Hämtar data från state.jsonData.metadata.
* Renderar varje metadatafält.
  + monitoringType visas först.
  + publisher och source (objekt) renderas med sina undernycklar.
  + language visas med fullständigt namn.
  + pageTypes (komma-separerad sträng) visas som en punktlista.
  + contentTypes (array av objekt) visas som en punktlista (endast text-egenskapen).
  + keywords (array) visas som en punktlista sist.
* En "Redigera metadata"-knapp visas som anropar renderMetadataForm.
* **Formulär (renderMetadataForm):**
  + Ett formulär skapas med fält för varje redigerbar metadataegenskap.
  + monitoringType: Dropdown.
  + publisher, source: Grupperade i <fieldset>.
  + language: Textfält med <datalist>.
  + pageTypes, keywords, contentTypes: Renderas som <textarea> (en post per rad).
  + title, description, license, etc.: Vanliga textfält eller textareor.
  + Read-only fält: version, dateCreated, dateModified.
  + Knappar: "Spara ändringar", "Avbryt".
* **Spara (saveMetadata):**
  + Hämtar data från formuläret.
  + pageTypes konverteras från rader i textarea till en komma-separerad sträng.
  + keywords och contentTypes konverteras från rader i textarea till arrayer (för contentTypes genereras id-fält). Ordningen från textarean bevaras för dessa.
  + Jämför med ursprunglig data. Om ändringar finns:
    - Uppdaterar state.jsonData.metadata.
    - Sätter state.isDataModified = true och visar #saveChangesButton.
    - **Synkroniserar contentTypes:** Om contentTypes i metadata har ändrats (texter ändrade, tillagda, borttagna), uppdateras contentType-arrayerna i alla requirements för att reflektera dessa ändringar (tar bort referenser till borttagna ID:n, uppdaterar ID:n om texten ändrats så att ett nytt ID genererats).
    - Anropar displayMetadata för att visa uppdaterad metadata.
    - Visar bekräftelsemeddelande.
* Hämtar state.jsonData.requirements.
* Filtrerar kraven baserat på state.currentSearchTerm (söker i titel, ref, instruktioner, kontroller, expectedObservation, etc.).
* Sorterar de filtrerade kraven baserat på state.currentSortOrder med hjälp av getSortFunction.
  + **Kategorisortering:** Renderar kraven grupperade per mainCategory och sedan subCategory (funktion renderGroupedRequirements). Inom varje grupp sorteras kraven på titel.
  + **Annan sortering (referens, påverkan):** Renderar en platt lista (funktion renderFlatRequirements).
* Varje krav i listan renderas med renderRequirementListItem och inkluderar:
  + Referens och titel.
  + Knappar: "Visa", "Redigera", "Radera".
* Hanterar fokus på senast redigerat/sparat krav (state.lastFocusedReqKey).
* Hämtar specifikt krav från state.jsonData.requirements baserat på reqKey.
* Renderar all information för kravet:
  + Titel, standardreferens (med länk om URL finns).
  + Instruktioner (som numrerad lista, Markdown-parsas).
  + expectedObservation (Markdown-parsas).
  + Exempel, Undantag, Vanliga Fel, Tips (alla Markdown-parsas).
  + Kontroller (checks):
    - Varje check visas med sitt condition.
    - logic ("Alla" eller "Minst ett") visas om det finns flera passCriteria.
    - passCriteria och ifNo visas som punktlistor (Markdown-parsas).
  + contentType (visar text från metadata.contentTypes baserat på ID:n i kravet).
  + Kategori (huvud/under) och Påverkan (kritisk, poäng, antagen efterlevnad).
* Knappar: "Redigera" (anropar renderRequirementForm(reqKey)), "Radera" (anropar confirmDeleteRequirement(reqKey)), "Tillbaka till listan".
* **Formulär (renderRequirementForm):**
  + Liknande renderMetadataForm men för kravspecifika fält.
  + title: Obligatoriskt.
  + standardReference (text, url): Grupperade.
  + instructions: Dynamisk lista av textareor (skapas med createInstructionListItem).
  + examples, exceptions, tips, commonErrors, expectedObservation: Textareor.
  + checks: Dynamisk lista av fieldsets (skapas med createCheckFieldset).
    - Varje check-fieldset innehåller:
      * condition (textarea, obligatorisk).
      * logic (dropdown AND/OR).
      * passCriteria (dynamisk lista av textareor, skapas med createCriterionListItem).
      * ifNo (dynamisk lista av textareor, skapas med createCriterionListItem).
    - **Layout för check-fieldset är nu vertikal:** "Villkor" och "Logik" först, sedan blocket för "Godkänd-kriterier", sedan blocket för ""Om Nej"-kriterier".
  + contentType: Checkboxar baserade på metadata.contentTypes.
  + metadata (för kravet):
    - mainCategory (obligatorisk), subCategory: Dropdowns (om kategorier finns) eller textfält (skapas med createCategorySelect / createFormField).
    - impact (isCritical, primaryScore, secondaryScore, assumedCompliance): Checkbox/nummerfält.
  + Knappar: "Spara", "Avbryt".
* **Spara (saveRequirement):**
  + Validerar obligatoriska fält (titel, huvudkategori).
  + Genererar id och key om det är ett nytt krav, eller om titeln ändrats för ett befintligt.
  + Samlar in data från formuläret.
  + Konstruerar kravobjektet. Tomma subCategory och standardReference (om både text/url är tomma) sparas som "".
  + Jämför med ursprungligt krav (om redigering). Om ändringar finns:
    - Tar bort gammal nyckel om nyckeln har ändrats (p.g.a. titeländring).
    - Uppdaterar/lägger till kravet i state.jsonData.requirements.
    - Sätter state.isDataModified = true och visar #saveChangesButton.
    - Sätter state.lastFocusedReqKey till det sparade kravets nyckel.
    - Anropar displayRequirementDetail för det sparade kravet.
    - Visar bekräftelsemeddelande.
* **Bekräftelse (confirmDeleteRequirement):** Visar en bekräftelsedialog med kravets titel.
* **Radering (deleteRequirement):**
  + Tar bort kravet från state.jsonData.requirements.
  + Sätter state.isDataModified = true och visar #saveChangesButton.
  + Återställer currentRequirementKey och lastFocusedReqKey.
  + Anropar displayRequirements för att visa den uppdaterade listan.
  + Visar bekräftelsemeddelande.

**5.5. Hjälpfunktioner**

* **\_-----\_utils\_\_helpers.js:**
  + isValidEmail: Validerar e-postformat.
  + escapeHtml: Escapar HTML-specialtecken.
  + parseSimpleMarkdown: Tolkar enkel Markdown (**fet**, *kursiv*, behåller <a>, <pre>, <br>, <hr>).
  + generateKeyFromName: Skapar en URL-säker slug från en sträng.
  + generateRequirementKey: Skapar en unik nyckel för ett krav.
  + getVal: Hämtar nästlade värden från objekt säkert.
* **\_---\_requirement\_functions.js (formulärhjälpredor):**
  + createFormField: Generell funktion för att skapa label + input/textarea/select.
  + createInstructionListItem, createCheckFieldset, createCriterionListItem, createDynamicListButton: Specifika för att bygga upp de dynamiska delarna av kravformuläret.
  + extractCategories, createCategorySelect, getCategoryValue: För kategorihanering i formulär.

**5.6. Sparande av Fil (downloadJsonFile i \_-----\_file\_handling.js)**

1. Anropas när #saveChangesButton klickas.
2. Anropar updateVersion och updateDateModified på state.jsonData.metadata.
   * updateVersion: Ökar sekvensnumret (rSEQ) om samma år/månad, annars återställs sekvensnumret till 1 för nytt år/månad. Format: YYYY.M.rSEQ.
   * updateDateModified: Sätter till dagens datum.
3. **Dataintegritet:** Går igenom alla requirements och säkerställer att STANDARD\_REQUIREMENT\_KEYS finns. Defaultvärden från REQUIREMENT\_KEY\_DEFAULTS appliceras för saknade nycklar. Speciell hantering för nästlade objekt som metadata och standardReference för att säkerställa korrekt struktur.
4. Genererar ett filnamn baserat på metadata.title och metadata.version.
5. Konverterar det (potentiellt modifierade) state.jsonData till en JSON-sträng (pretty-printed).
6. Skapar en Blob och använder FileSaver.js (eller fallback-metod) för att initiera nedladdning.
7. Återställer state.isDataModified = false och döljer #saveChangesButton.
8. Visar en inline-bekräftelse.

**5.7. Varning vid Osparade Ändringar**

* En beforeunload-lyssnare kontrollerar state.isDataModified. Om true, visas en standarddialog från webbläsaren som varnar användaren innan sidan stängs/navigeras bort.

**6. CSS (style.css)**

* Använder CSS Custom Properties (:root) för färger, vilket underlättar theming (ljust/mörkt tema via @media (prefers-color-scheme: dark)).
* Layout med Flexbox och/eller CSS Grid för responsivitet.
* Specifika klasser för olika vyer (.form-view, .requirement-detail, .grouped-list-view, etc.) för att anpassa utseendet i #dynamicContentArea.
* Styling för knappar, formulärfält, listor, meddelanden etc.
* Den senaste layouten för "Kontrollpunkter" i formulär använder display: flex; flex-direction: column; på .check-fieldset för att stapla dess barn vertikalt.

**7. Utvecklingsanteckningar och Potentiella Förbättringar**

* **Validering:** För närvarande sker grundläggande validering (t.ex. obligatoriska fält). Mer omfattande validering av datatyper och format kan implementeras.
* **Felhantering:** Förbättrad felhantering och mer användarvänliga felmeddelanden.
* **Debounce för sökning:** Sökfunktionen i kravlistan anropar displayRequirements vid varje input. För stora dataset kan debounce vara nödvändigt för prestanda.
* **Tillgänglighet (a11y):** Grundläggande ARIA-attribut används, men en mer grundlig a11y-granskning rekommenderas.
* **State Management:** För en större applikation kan ett mer robust state management-bibliotek övervägas, men för nuvarande omfattning fungerar den enkla modulbaserade state-hanteringen.
* **Testning:** Ingen automatiserad testning nämns, men enhetstester och integrationstester skulle vara värdefulla.
* **Prestanda:** För mycket stora JSON-filer kan DOM-manipulationer bli långsamma. Optimeringar som virtuell scrollning eller mer selektiva DOM-uppdateringar kan övervägas vid behov.

Detta bör ge en utvecklare en solid grund för att förstå och kunna återskapa projektet.