Generera användargränssnitt baserade på JSON Scheman

JULIUS RECEP COLLIANDER CELIK

Civilingenjör Informationsteknik
Datum: 24 april 2018
Handledare: Patric Dahlqvist
Examinator: Anders Västberg
Uppdragsgivare: LS Elektronik AB
Engelsk titel: Generating user interfaces based on JSON Schemas
Skolan för elektroteknik och datavetenskap

Innehåll

Kapitel 1

Introduktion

Examensarbetet handlade om att undersöka möjligheten att skapa en modell för att beskriva och annotera redigerbar data, och sedan automatiskt generera ett användargränssnitt. Det här kapitlet introducerar hela arbetet. Kapitel 1.1 ger en bakgrund till arbetet, vilket innefattar både en enkel teoretisk bakgrund, samt en bakgrund till företaget arbetet utfördes hos, samt systemet som arbetet utvecklades mot. Kapitel 1.2 diskuterar systemet arbetet utvecklades mot i större detalj. Kapitel 1.3 presenterar problemet med en frågeställning. Kapitel 1.4 föreslår en hypotes som arbetet grundar sig på. Kapitel 1.5 och 1.6 diskuterar syftet och målet med arbetet. Riskerna diskuteras i Kapitel 1.7. Kapitel 1.8 presenterar metodvalet. Kapitel 1.9 beskriver arbetets omfattning. Resten av rapportens disposition presenteras i kapitel 1.10.

1.1 Bakgrund

Det här kapitlet beskriver bakgrunden till varför och i vilka ämnesområden arbetet utfördes, samt för att ge en förståelse för resten av Introduktionen. Kapitel 1.1.1 beskriver JSON och JSON Schema vilket är en stor del av teknikerna som arbetet undersökte. Kapitel 1.1.2 beskriver företaget arbetet utfördes hos, samt systemet som arbetet utvecklades mot.

1.1.1 JSON och JSON Schema

JavaScript Object Notation *(JSON)* är ett textbaserat dataformat för att utbyta data mellan webbtjänster. Till skillnad mot andra alterna-

tiv, som exempelvis XML, är det både läsbart för människor och datorer, samtidigt som det är väldigt kompakt, vilket är en anledning till att det är ett av de mest populäraste dataformaten för datautbyte mellan webbtjänster. [1] JSON utvecklades med inspiration till språket ECMAScript *(JavaScript)* men samtidigt programmeringsspråksoberoende, vilket lett till att implementationer för att generera och parsa JSON finns tillgängliga i många olika programmeringsspråk [2]. ECMAScript är ett språk som stöds av alla moderna webbläsare, och har därför blivit en kärnteknik för webben.

Trots att JSON är det populäraste dataformatet för datautbyte mellan webbtjänster saknas det ett väletablerat standardiserat ramverk för metadata-definition [1]. En väldigt lovande formell standard är JSON Schema, vilket är ett ramverk som fortfarande utvecklas av Internet Engineering Task Force *(IETF)*. JSON Schema är ett ramverk för att beskriva och annotera JSON-data [3]. Kapitel 2 beskriver JSON och JSON Schema i mer detalj.

1.1.2 Mimer SoftRadio

Arbetet utfördes hos LS Elektronik AB *(LSE)*, som är ett tekniskt företag, som utvecklar och tillverkar elektroniska produkter [4]. LSE erbjuder bland annat ett radiosystem som heter Mimer SoftRadio vilket kan användas för att ansluta ett flertal annars inkompatibla radioenheter i ett och samma system, samt fjärrstyra radioenheterna från en persondator med ett klientprogram. I resten av rapporten kan datorn med klientprogrammet kallas operatörsdator, där användaren kan kallas operatör.

Mimer SoftRadio är ett program med väldigt många möjliga inställningar. I många fall är dessa inställningar för komplexa för de vanliga operatörerna att redigera själva, så därför brukar vissa kunder låsa redigeringsmöjligheterna, och bara tillåta vissa administratörer att ställa in alla inställningar på rätt sätt. Det finns också kundfall där flera operatörer använder samma dator, vid olika tidpunkter. Ett förekommande kundfall är att en operatör jobbade dagtid med att leda och organisera dagsarbete, medan en annan operatör tar över nattskiftet för att övervaka många fler radioenheter.

För att förenkla dessa två kundfall påbörjade LSE utvecklingen av funktionalitet som skulle erbjuda användare att spara uppsättningar av inställningar i olika *profiler*. Det skulle gå att enkelt byta mellan flera

förinställda konfigurationer av Mimer SoftRadio. För att konfigurera dessa profiler skapades ett administratörsprogram, som skulle kunna fjärrkonfigurera profilerna hos operatörsdatorerna. Fjärrstyrningen skulle underlätta administratörer att ställa in profiler på flera datorer samtidigt, som sannolikt skulle innehålla liknande inställningar.

1.2 Problemområde

Systemet för att konfigurera profiler, som beskrevs i kapitel 1.1.2, bygger på att alla operatörsdatorer exponerar ett API mot en server, över en TCP-port. Administratörsprogrammet skulle erbjuda ett användargränssnitt för att konfigurera profilinställningar, för att sedan kommunicera ändringarna till operatörsdatorerna. I resten av rapporten kan operatörsdatorerna och administratörsprogrammet kallas server respektive klient. Administrationsprogrammet skrevs som en skrivbordsapplikation till Windows, med språket Delphi.

Kommunikationsprotokollet var ett eget skapat protokoll som byggde på att skicka JSON-Objekt via TCP. För en beskrivning av JSON, se kapitel 2.1. För en mer utförlig beskrivning av kommunikationsprotokollet se Appendix A. Problemet som LSE hade inför utvecklandet av användarprofilerna var skapandet av ett användargränssnitt på administratörsprogrammet. Olika operatörsdatorer, hos olika kunder, kunde ha olika versioner av Mimer, med olika funktionalitet tillgänglig, och därmed olika uppsättningar konfigurerbara inställningar.

Det vore orimligt kostsamt för LSE att skapa ett administratörsprogram för varje version av Mimer, då både Mimer ändrades med tiden, samt att olika kunder köpte till extra funktionalitet. Samtidigt behövde användargränssnittet på administratörsprogrammet anpassas så att det skulle vara tydligt vad en administratör kunde konfigurera. Helst skulle ett administratörsprogram fungera bra med framtida versioner av Mimer, utan några eller utan stora justeringar av programmet. LSE ville helt enkelt att servern kommunicerade tillgängliga inställningar, till klienten så att klienten sen skulle kunna anpassa sitt användargränssnitt, och det skulle ske på ett tillräckligt generellt sätt att administratörsprogrammet var framtidssäkert för framtida version av Mimer.

Problemet kan förenklat delas up i två problem. Ena problemet är att olika operatörsdatorer har olika uppsättningar konfigurationer att

konfigurera, vilket är anledningen till att användargränsnittet måste anpassas beroende på operatörsdatorn den kopplar upp sig mot. För att lösa det problemet skulle möjligtvis användargränsnittet kunna extrapolera information från konfigurationsfilen, som sparar inställningar på operatörsdatorn, och därmed lista ut hur olika inställningar kan redigeras. Det diskuteras mer i kapitel **??**. Anledningen till att en sådan lösning inte är lämplig är delvis för att den inte skulle kunna vara felfri, vilket diskuteras i kapitel **??** men också för att konfigurationsfilen innehåller inställningar som administratörsklienten inte ska kunna konfigurera. Det andra problemet är därför att LSE vill kunna bestämma vilka inställningar administratörsklienten möjliggör konfiguration av. Därför måste det finnas en fil, antingen hårdkodad eller genererad, på serverdatorn som bestämmer hur användargränsnittet på klientdatorn ska se ut, och hur klienten får manipulera datan på serverdatorn.

1.3 Problem

Vilka svårigheter finns det med att använda JSON Schema för att automatisk generera ett användargränssnitt, är det möjligt, samt hur generella JSON Scheman går det att använda sig av?

1.4 Hypotes

En förutsatt hypotes sattes innan arbetet påbörjades. Hypotesen var att det borde vara möjligt att automatiskt skapa ett användargränssnitt från ett JSON Schema, men för att skapa ett användarvänligt användargränssnitt var det nödvändigt att klienten utvecklades för ett JSON Schema som delvis följde en förutbestämd struktur.

1.5 Syfte

Syftet med tesen är att systematiskt analysera problemen med att försöka skapa automatiskt genererade användargränssnitt utifrån olika JSON Scheman. Målet är att föreslå både en strukturell modell samt en metod för att lösa detta problem.

Syftet med arbetet är att med hjälp av JSON Scheman skapa ett dynamiskt användargränssnitt som anpassade sig efter syfte. Utan att

uppdatera administratörsklienten ska samma administratörsklient kunna konfigurera inställningar hos olika datorer med olika versioner av Mimer SoftRadio, och därmed olika uppsättningar konfigurerbara inställningar. Det skulle inte bara innebära stark kompatibilitet utan också framtidssäkerhet hos LSEs produkter.

1.6 Mål

Målet med arbetet är att kunna skapa en grund för användare av Mimer SoftRadio att enkelt kunna konfigurera inställningar, oavsett version eller uppsättning extra funktionalitet. Det skulle kunna innebära att Mimer SoftRadio blir ett bättre verktyg för många potentiella kunder. Samtidigt finns det ett mål med att utforska samt att metodiskt utvärdera och beskriva hur användargränssnitt för att redigera data, automatiskt kan genereras.

1.6.1 Samhällsnytta, Etik och Hållbarhet

Ur ett samhällsnyttigt och etiskt perspektiv kan en implementation av fjärrstyrda användarprofiler i Mimer SoftRadio innebära effektivisering av samhällsnyttiga funktioner. Mimer SoftRadio används bland annat av polis, ambulans, brandkår, kollektivtrafik och internationella flygplatser. Fjärrstyrda användarprofiler skulle hos befintliga kunder i många fall innebära effektivare arbete. Som följd av detta går det att argumentera för att det leder till effektivare kommunikation för organisationer som använder Mimer SoftRadio. Då dessa organisationer arbetar med säkerhet i samhället, räddandet av liv, och upprätthållandet av ett effektivt samhälle, lider hela samhället när dessa organisationer inte kan kommunicera ordentligt. Det är därför väldigt etiskt försvarbart att arbeta med att effektivisera arbetet hos dessa organisationer.

1.7 Risker

En ekonomisk risk är risken som alltid finns vid all hantering av data. Om någon datahantering skulle bli fel, och data skulle försvinna, skrivas över eller bli korrupt, så måste kunder tillägna tid åt att återskapa datan. Därför är det viktigt att implementera ett robust system

som är delvis feltolerant. Ingen data som kommer hanteras kommer vara kritisk, och kommer vara relativt enkel att återskapa. Problemet blir att det skulle innebära en kostnad att behöva skapa profiler igen och ställa in inställningar igen, och utöver det så skulle korrupta filer till och med kunna innebära att Mimer SoftRadio inte går att använda alls.

1.8 Metodval

Arbetet som ska utföras är till viss del en fallstudie, men samtidigt ska den utforska något nytt och med det föreslå en ny modell. Designorienterad forskning *(Design science research)* är den metod som passar bäst för den här sortens arbete och därför har den metoden valts. Arbetet följde de följande stegen:

1. **Medvetenhet** En beskrivning av problemet som ska lösas med modellen.
2. **Förslag** Förslag på lösning presenteras.
3. **Utveckling** Modellen utvecklas.
4. **Utvärdering** Modellen utvärderas. Lyckades modellen lösa problemet beskrivet i *Medvetenhet*?
5. **Sammanfattning** Dra slutsatser

1.9 Avgränsningar

En viktig avgränsning är att rapporten endast väldigt ytligt kommer undersöka olika användargränssnitt, och användbarheten hos dem. Arbetet handlar inte primärt om användbarhet, utan arbetet handlar i större grad om hur JSON Schema kan automatisera skapandet av användbara gränssnitt. Med hjälp av kunskapen som arbetet presenterar kan användbara användargränssnitt enklare skapas.

Ett annat ämne som också är viktigt är säkerhet av systemet som skapas. Säkerheten hos applikationen omfattas inte av arbetet, men det ignoreras samtidigt inte. Systemet som utvecklas för att utbyta JSON

Scheman och JSON-data sker över en ssl-krypterad säker uppkoppling. Det här arbetet utvärderar inte säkerheten hos den uppkopplingen.

Att skapa ett användarvänligt användargränssnitt utifrån alla möjliga sorters JSON Scheman med samma verktyg omfattas inte av arbetet. Arbetet kommer utforska olika strategier och metoder för att arbeta med förutbestämda JSON Scheman.

Validering av data är något som webbtjänster ofta måste ta hänsyn till. En klient kan annars skicka otillåten data till en webbserver och därför måste webbservern alltid validera data när den tar emot data, innan data används eller lagras. JSON Scheman fungerar utmärkt för validering av data, men då datan valideras hos klienten, både klienten och servern omfattas av arbetet, samt att användarna inte anses ha uppsåt att förstöra eller falsifiera data, kommer JSON Scheman inte användas för att validera data hos servern.

1.10 Disposition

Kapitel 2 presenterar den teoretiska bakgrunden.

Kapitel 2

JSON och JSON Scheman

Det här kapitlet beskriver vad JSON och JSON Scheman är, samt hur de används. Kapitel 2.1 beskriver vad JSON är. Kapitel 2.2 beskriver hur JSON används för kommunikation mellan webbtjänster. Kapitel 2.3 beskriver JSON Scheman, vad de är och hur de beskrivs. Kapitel 2.4 diskuterar användningsområden av JSON Schema samt listar kända implementationer.

2.1 JSON

JSON erbjuder stöd för några enkla datastrukturer: textsträngar *(string)* (`"hej värld"`), tal *(number)* (`4`), ett tomt värde *(null)* (**`null`**), samt booleska värden *(booleans)* (**`false`**). JSON erbjuder dessutom stöd för två komplexa datatyper vilket är vektorer *(array)*, en ordnad lista av JSON-värden vilket visas i Figur 2.1, samt objekt *(object)*, vilket är en oordnad mängd av namn-värde-par *(properties)*, som visas i Figur 2.2. Resten av rapporten kommer utbytbart använda JSON-värde, JSON-data, JSON-fil och JSON-dokument för att förklara en av de sex datastrukturerna som kan representeras med JSON.
Med hjälp av att rekursivt använda *array* eller *object* går det att representera komplexa datastrukturer med hjälp av JSON. Det finns inga begränsningar i hur komplext datastrukturer kan representeras.

```
["hej värld", 4, null, false]
```

Figur 2.1: Exempel på JSON-array

```
{
  "firstName": "Erik",
  "lastName": "Andersson",
  "age": 30
}
```

Figur 2.2: Exempel på JSON-object

```
{
  "country": "Sweden",
  "city": "Stockholm"
}
```

Figur 2.3: Exempel på förfrågan till webbserver

2.2 JSON i webbkommunikation

På grund av att JSON är kompakt, enkelt läsbart och har brett stöd hos många språk och implementationer, har JSON blivit väldigt utbrett bland webbtjänster. En hypotetisk förfrågan till en webbtjänst skulle kunna se ut som i Figur 2.3, där en klient förfrågar om de nuvarande väderförhållandena i Stockholm i Sverige. Svaret från webbservern skulle kunna se ut som i Figur 2.4 där webbservern svarar att temperaturen är minus tre grader Celsius och att det snöar. Exemplet visar hur simpelt JSON som dataformat är att förstå, vilket skulle kunna delvis vara en förklaring för populariteten.

```
{
  "timestamp": "06/01/2018 10:45:08",
  "country": "Sweden",
  "city": "Stockholm",
  "weather": "Snowing",
  "temperature": -3
}
```

Figur 2.4: Exempel på svar på förfrågan från webbserver

```
{
  "type": "object",
  "required": ["firstName", "age"],
  "properties": {
    "firstName": { "type": "string" },
    "lastName": { "type": "string" },
    "age": { "type": "integer" }
  }
}
```

Figur 2.5: Exempel på simpelt JSON Schema

2.3 JSON Schema

JSON Schema är ett ramverk för att förklara hur JSON-värden kan se ut. JSON Schema specificerar regler som kan användas för att antingen bestämma om befintliga JSON värden är giltiga, eller för att i förväg beskriva hur giltiga värden får se ut. Objektet i Figur 2.2 skulle kunna valideras enligt JSON Schemat som visas i Figur 2.5. Den senaste fastslagna versionen *(Draft 7)* av ramverket bygger på tre dokument: *Core*, *Validation* samt *Hyper-Schema*. [3], [5], [6]

2.3.1 JSON Schema Core

JSON Schema Core täcker grunderna för JSON Schema. Dokumentet fastställer exempelvis mediatypen som borde användas för att skicka JSON Scheman över HTTP, förhållandet mellan flera JSON Scheman, samt hur heltal borde behandlas. Att JSON Scheman själva är JSON-dokument bestäms också. Dokumentet fastställer också att validering och annotering av JSON-värden ska ske enligt dokumentet draft-handrews-json-schema-validation-01 *(Validation)*, samt att draft-handrews-json-schema-hyperschema-01 *(Hyper-Schema)* behandlar reglerna kring att beskriva hypertextstrukturen hos JSON-dokument. [3]

2.3.2 JSON Schema Validation

JSON Schema Validation beskriver tre saker: hur man beskriver ett JSON-dokument, hur man ger tips åt användargränsnitt för att jobba med JSON-dokument samt hur man kan beskriva påståenden om ett

dokuments validitet. Förenklat beskriver det här dokumentet strukturen hos ett JSON Schema, med beskrivningar av nästan alla nyckelorden. Utöver att beskriva hur JSON-dokument ska valideras, presenteras nyckelord som `"title"` och `"description"`, där `"title"` är en kort förklaring för JSON värdet den validerar, och `"description"` är en längre förklaring. [5]

2.3.3 JSON Schema Hyper-Schema

JSON Schema skapas till stor del för användandet hos webbtjänster. Därför beskriver det tredje dokumentet, JSON Schema Hyper-Schema, hur resurser kan manipuleras och interageras med över hypermediamiljöer som HTTP. JSON Schema Validation skulle kunna beskriva hur ett API anrop ska hanteras och vad som förväntas från förfrågningar och svar på dem. JSON Schema Hyper-Schema kan då användas för att beskriva ett helt API och hur de olika anropen och resurserna är relaterade till varandra. [6]

2.4 Användningsområden för JSON Schema

Användningsområden för JSON Scheman är bland annat:

1. Validering av data.
2. Annotering av data.
3. Beskrivning av REST APIer.
4. Automatisk generering av kompatibel kod, för att hantera JSON värden beskrivna med JSON Schema.
5. Automatisk generering av API-dokumentation.
6. Automatisk generering av användargränsnitt.

Att använda JSON Schema för användningsområdena 1-3 är trivialt. Det går att utveckla program som kan hantera alla oändligt möjliga permutationer av JSON Schema. Det som däremot inte är trivialt är hur användningsområdena 4-6 skulle kunna generaliseras så pass

mycket att ett program eller algoritm skulle kunna hantera vilket giltigt JSON Schema som helst. Användningsområde fyra och fem omfattas inte av den här rapporten, och varför användningsområde sex inte är trivialt diskuteras mer i resultatet.

The Json Schema organisation listar kända implementationer på sin hemsida, och har delat upp dem i följande kategorier [7]:

- Validators
- Hyper-Schema
- Schema generation
- Data parsing
- UI generation
- Editors
- Compatibility
- Documentation generation

Arbetet kommer behöva implementera tre av de listade implementationerna: Schema generation, Data parsing samt UI generation, vilket diskuteras i kapitel 2.4.1, 2.4.2 samt 2.4.3.

2.4.1 Generering av scheman

Schemagenerering som kategori består av tolv implementationer där det går att ytterligare dela upp implementationerna i två kategorier. Det finns implementationer som utgår från JSON data, och genererar ett JSON Schema för att beskriva datan. Det kan användas om det går att anta att all användning av JSON Schemat kommer att användas på data med exakt likadan struktur. Kapitel xXxTODOxXx kommer att diskutera prestationen av de implementationerna, samt visa hur ingen av dem fungerar perfekt. Den andra kategorin av implementationer är implementationer som genererar JSON Scheman utifrån kända datatyper i ett typat språk, eller verktyg för att beskriva scheman. De borde enligt förväntan prestera bättre men inte alltid perfekt. [7]

Implementationerna som genererar JSON Scheman från JSON data:

- Schema Guru *(Scala)* [8]
- JSON Schema Generator *(Visual Studio)* [9]

Implementationerna som genererar JSON Scheman från kända typer:

- Json.NET Schema *.NET* [10]
- NJsonSchema for .NET *.NET* [11]
- Liform *(PHP)* [12]
- JSL *(Python)* [13]
- json-schema-generator *(JavaScript / JSON)* [13]
- typescript-json-schema *(TypeScript)* [14]
- Typson *(TypeScript)* [15]
- JSONSchema.net *(Online webbverktyg)* [15]
- Schema Guru Web UI **Obs:** Verktyget hittades ej och kommer därför exkluderas från resten av rapporten.
- APIAddIn *(Sparx Enterprise Architect)* [16]

2.4.2 Parsning av JSON Scheman

En parser tolkar JSON Scheman och representerar schemat med någon annan datastruktur. Ofta är parsning viktigt för att schemat ska kunna representeras med en datastruktur som programmeringsspråket är kompatibelt med. Vissa implementationer använder ett färdigt JSON Schema och genererar kod som är kompatibelt med att hantera JSON som är formaterad utifrån schemats struktur. Andra implementationer kan dynamiskt hantera vilket schema som helst under exekvering, och dynamiskt skapa parsers för JSON formaterad utifrån schemat. De parsers som listas på The Json Schema organisations hemsida är följande:

- DJsonSchema *Delphi* [17]

- jsonCodeGen *Groovy* [18]
- aeson-schema *Haskell* [19]
- AutoParse *Ruby* [20]
- json-schema-codegen *Scala* [21]
- Argus *Scala* [22]
- Bric-à-brac *Swift* [23]
- gojsonschema *Golang* **Obs:** Verktyget saknade information på engelska eller svenska och kommer därför exkluderas från resten av rapporten. [24]
- jsonschema *Golang* [25]

2.4.3 Tidigare försök av generering av användargränsnitt baserade på JSON Scheman

Det finns olika implementationer av att generera ett användargränsnitt utifrån ett JSON Schema. Samtliga kända implementationer är skrivna i språket JavaScript och bemöter därför ingen av svårigheterna med att använda JSON eller JSON Schema med andra språk. Samtliga implementationer är implementationer för att generera hemsidor eller komponenter till hemsidor, vilket skiljer sig mycket mot att generera användargränsnitt åt Windows med Delphi, vilket arbetet gjorde.

Vissa av implementationerna används för att generera ett användargränsnitt för att förklara ett API beskrivet med JSON Schema och andra implementationer används för att generera ett formulär för att manipulera data beskrivet av JSON Schema. Att generera ett formulär för att manipulera data beskrivet av JSON Schema är exakt vad den här rapporten utvärderar. Användargränsnittsgenererarna som listas på The Json Schema organisations hemsida är följande:

- Alpaca Forms [26]
- Angular Schema Form [27]
- Angular2 Schema Form [28]
- JSON Editor [29]

- JSON Form [30]
- json-forms [31]
- JSONForms [32]
- Jsonary **OBS!**
- liform-react [33]
- Metawidget [34]
- pure-form webcomponent **Obs not found**
- React JSON Schema Form [35]
- React Schema Form [35]

Litteratur

[1] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte och D. Vrgoč, "Foundations of JSON Schema", i *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, New York, New York, USA: ACM Press, 2016, s. 263–273, ISBN: 9781450341431. DOI: 10.1145/2872427.2883029. URL: http://dl.acm.org/citation.cfm?doid=2872427.2883029.

[2] I. ECMA, "The JSON Data Interchange Format", *ECMA International*, årg. 1st Editio, nr October, s. 8, 2013, ISSN: 2070-1721. DOI: 10.17487/rfc7158. arXiv: arXiv:1011.1669v3. URL: http://www.ecma-international.org/publications/standards/Ecma-404.htmhttp://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[3] H. Andrews, A. Wright och Internet Engineering Task Force, "Draft-handrews-json-schema-01", tekn. rapport, 2018. URL: https://tools.ietf.org/html/draft-handrews-json-schema-01.

[4] R. Ehne, *LS Elektronik About - LS Elektronik*. URL: http://www.lse.se/about/ (hämtad 2018-04-06).

[5] H. Andrews, A. Wright och Internet Engineering Task Force, "Draft-handrews-json-schema-validation-01", tekn. rapport, 2018. URL: https://tools.ietf.org/html/draft-handrews-json-schema-validation-01.

[6] ——, "Draft-handrews-json-schema-hyperschema-01", tekn. rapport, 2018. URL: https://tools.ietf.org/html/draft-handrews-json-schema-hyperschema-01.

[7] The JSON Schema organisation, *Implementations | JSON Schema*. URL: http://json-schema.org/implementations (hämtad 2018-04-18).

[8] Snowplow, *Schema Guru*. URL: https://github.com/snowplow/schema-guru (hämtad 2018-04-20).

[9] Mads Kristensen, *JSON Schema Generator - Visual Studio Marketplace*. URL: https://marketplace.visualstudio.com/items?itemName=MadsKristensen.JSONSchemaGenerator (hämtad 2018-04-20).

[10] Newtonsoft, *Json.NET Schema - Newtonsoft*. URL: https://www.newtonsoft.com/jsonschema (hämtad 2018-04-20).

[11] R. Suter, *NJsonSchema for .NET*. URL: https://github.com/RSuter/NJsonSchema (hämtad 2018-04-20).

[12] Limenius, *Liform*. URL: https://github.com/Limenius/liform (hämtad 2018-04-20).

[13] A. Romanovich, *JSL*. URL: https://github.com/aromanovich/jsl (hämtad 2018-04-20).

[14] Y. El-Dardiry, *Typescript-json-schema*. URL: https://github.com/YousefED/typescript-json-schema (hämtad 2018-04-20).

[15] L. Bovet och Swisspush, *Typson*. URL: https://github.com/lbovet/typson (hämtad 2018-04-20).

[16] P. Tomlinson, *APIAddIn*. URL: https://github.com/bayeslife/api-add-in (hämtad 2018-04-20).

[17] Schlothauer & Wauer GmbH, *DJsonSchema*. URL: https://github.com/schlothauer-wauer/DJsonSchema (hämtad 2018-04-20).

[18] ——, *jsonCodeGen*. URL: https://github.com/schlothauer-wauer/jsoncodegen (hämtad 2018-04-20).

[19] M. Kowalczyk och T. Baumann, *Aeson-schema*. URL: https://github.com/Fuuzetsu/aeson-schema (hämtad 2018-04-20).

[20] Google, *AutoParse*. URL: https://github.com/google/autoparse (hämtad 2018-04-20).

[21] Tundra, *Json-schema-codegen*. URL: https://github.com/VoxSupplyChain/json-schema-codegen (hämtad 2018-04-20).

[22] A. Fenton, *Argus*. URL: https://github.com/aishfenton/argus (hämtad 2018-04-20).

[23] Glimpse I/O Inc., *Bric-à-brac*. URL: https://github.com/glimpseio/BricBrac (hämtad 2018-04-20).

[24] andy Zhangtao, *Gojsonschema*. URL: https://github.com/andy-zhangtao/gojsonschema (hämtad 2018-04-20).

[25] Q. inc., *Jsonschema*. URL: https://github.com/qri-io/jsonschema (hämtad 2018-04-20).

[26] Gitana Software Inc., *Alpaca Forms - Easy Forms for jQuery*. URL: http://www.alpacajs.org/ (hämtad 2018-04-24).

[27] Textalk, *Angular Schema Form*. URL: http://schemaform.io/ (hämtad 2018-04-24).

[28] Makina Corpus, *Angular2 Schema Form*. URL: https://github.com/makinacorpus/angular2-schema-form (hämtad 2018-04-24).

[29] Jeremy Dorn, *JSON Editor*. URL: https://github.com/json-editor/json-editor (hämtad 2018-04-24).

[30] Joshfire, *JSON Form*. URL: https://github.com/joshfire/jsonform (hämtad 2018-04-24).

[31] Brutusin.org, *Json Forms*. URL: https://github.com/brutusin/json-forms (hämtad 2018-04-24).

[32] EclipseSource, *JSON Forms*. URL: https://jsonforms.io/ (hämtad 2018-04-24).

[33] Nacho Martín, *Liform-react*. URL: https://github.com/Limenius/liform-react (hämtad 2018-04-24).

[34] Metawidget, *Metawidget*. URL: http://metawidget.org/ (hämtad 2018-04-24).

[35] Mozilla Services, *React-jsonschema-form*. URL: https://github.com/mozilla-services/react-jsonschema-form (hämtad 2018-04-24).

Bilaga A

API-beskrivning

TODO!