Lo Standard publiccode.yml

italia

10 dic 2020

Indice

`publiccode.yml` è uno standard di metadati ideato per essere inserito in repository contenenti software della Pubblica Amministrazione con lo scopo di renderli facilmente individuabili e, di conseguenza, riutilizzabili da altri enti.

Inserendo nella root di un repository un file chiamato `publiccode.yml` che descrive le caratteristiche del software se ne agevola la comprensione ai tecnici e ai decisori pubblici interessati a valutarlo; al tempo stesso si permette di costruire strumenti automatici di indicizzazione, poiché il formato è facilmente leggibile sia da esseri umani sia da macchine.

`publiccode.yml` è obbligatorio per tutto il software pubblico sviluppato in Italia, come da linee guida[1]: questo consente al crawler automatico di Developers Italia di costituire il catalogo del software a riuso[2]. Lo standard è tuttavia pensato in ottica internazionale, per cui tutte le specificità nazionali sono separate dal core e definite in apposite sezioni estensibili autonomamente dai governi nazionali.

Tra le informazioni contenute in un `publiccode.yml` vi sono:

- il titolo e la descrizione del progetto o prodotto (in una o più lingue);
- lo stato dello sviluppo ad es., `concept`, `development`, `beta`, `stable`, `obsolete`;
- i riferimenti dell'organizzazione che ha sviluppato il progetto;
- chi si sta occupando della sua manutenzione e quando il rapporto finirà;
- per quale quadro giuridico è stato pensato questo progetto o prodotto;
- quali dipendenze software esistono.

e molte altre informazioni rilevanti.

[1] https://docs.italia.it/AgID/linee-guida-riuso-software/lg-acquisizione-e-riuso-software-per-pa-docs/
[2] https://developers.italia.it/

CAPITOLO 1

Link utili

- Maggiori informazioni sul riuso del software[3]
- Editor web per publiccode.yml[4]

[3] https://developers.italia.it/it/riuso
[4] https://publiccode-editor.developers.italia.it/

CAPITOLO 2

Indice dei contenuti

2.1 Lo standard (core)

La struttura di un file `publiccode.yml` prevede l'esistenza di chiavi top-level e sezioni che possono contenere al proprio interno altre chiavi. Lo standard ha rilevanza internazionale ma è possibile dichiarare una sezione dedicata per le chiavi relative ad un Paese specifico (si veda *Estensioni nazionali* (pagina 19) per maggiori dettagli).

2.1.1 Chiavi e Sezioni Top-Level

Chiave `publiccodeYmlVersion`

- Tipo: stringa
- Presenza: obbligatoria
- Esempio: `"0.1"`

Questa chiave specifica la versione alla quale il presente `publiccode.yml` aderisce, per una questione di compatibilità diretta.

Chiave `name`

- Tipo: stringa
- Presenza: obbligatoria
- Esempio: "Medusa"

Questa chiave contiene il nome del software. Contiene il nome (abbreviato) pubblico del prodotto, che può essere tradotto nella sezione specifica chiamata `localisation`. Dovrebbe essere il nome con il quale la maggior parte delle persone si riferisce al suddetto software. Nel caso in cui il software abbia sia un nome in "codice" interno che uno commerciale, è preferibile usare il nome commerciale.

Chiave `applicationSuite`

- Tipo: stringa
- Presenza: opzionale
- Esempio: "MegaProductivitySuite"

Questa chiave contiene il nome della "suite" alla quale il software appartiene.

Chiave `url`

- Tipo: stringa (URL)
- Presenza: obbligatoria
- Esempio: `"https://example.com/italia/medusa.git"`

Un identificatore unico per questo software. Questa stringa deve essere una URL che punta al repository di codice sorgente (git, svn, ...) nel quale il software è pubblicato. Se il repository è disponibile sotto diversi protocolli, preferire URL HTTP/HTTPS che non richiedono l'autenticazione.

I fork creati con lo scopo di contribuire *upstream* non devono modificare questo file; questo aiuta i software che fanno il parsing di `publiccode.yml` a saltare immediatamente i fork tecnici. Al contrario, un fork completo che sarà manutenuto in modo separato rispetto al software originale, dovrebbe modificare questa linea per essere trattato come una variante distinta.

Vedi *Fork e varianti* (pagina 21) per una descrizione completa del significato di variante software e di come gestire i fork sia lato parser che lato autore.

Chiave `landingURL`

- Tipo: stringa (URL)
- Presenza: opzionale
- Esempio: `"https://example.com/italia/medusa"`

Se la chiave `url` non porta ad una pagina leggibile da un umano, ma serve solo ad indirizzare un client di source control verso il codice sorgente, con questa chiave viene introdotta l'opzione di specificare la *landing page*. Questa pagina, idealmente, è il punto di arrivo dell'utente quando seleziona un pulsante chiamato, ad esempio, "Vai al codice sorgente dell'applicazione". Nel caso in cui il prodotto preveda un installer grafico automatico, questa URL può puntare alla pagina contenente un riferimento al codice sorgente ma che offra anche la possibilità di scaricare tale installer.

Chiave `isBasedOn`

- Tipo: stringa o array di stringhe
- Presenza: opzionale
- Esempio: `"https://github.com/italia/otello.git"`

Nel caso in cui questo software sia una variante o un fork di un altro software, che opzionalmente può contenere un file `publiccode.yml`, questa chiave conterrà la `url` di uno o più progetti originali.

L'esistenza di questa chiave identifica il fork come una variante (vedi *Fork e varianti* (pagina 21)), discendente dal repository specificato.

Chiave `softwareVersion`

- Tipo: stringa
- Presenza: opzionale
- Esempio: `"1.0"`, `"dev"`

Questa chiave contiene il numero dell'ultima versione stabile del software. Il numero di versione è una stringa che non è pensata per essere interpretata dal parser ma solamente visualizzata; i parser non devono assumere l'utilizzo del semantic versioning o altri specifici formati di versionamento.

Questa chiave può essere omessa nel caso in cui il software sia in una fase iniziale di sviluppo e non sia stato ancora rilasciato.

Chiave `releaseDate`

- Tipo: stringa (data)
- Presenza: obbligatoria
- Esempio: `"2017-04-15"`

Questa chiave contiene la data di ultimo rilascio del software. Questa data è obbligatoria se il software è stato rilasciato almeno una volta e dunque esiste un numero di versione.

Chiave `logo`

- Tipo: stringa (percorso verso il file)
- Presenza: opzionale
- Formati accettabili: SVG, SVGZ, PNG
- Esempio: `"img/logo.svg"`

Questa chiave indica il logo del software. Il valore può essere il percorso relativo al file a partire dalla root del repository, oppure una URL assoluta che punta al logo in versione raw. In entrambi i casi, il file deve risiedere all'interno del medesimo repository che contiene il `publiccode.yml`. Il logo dovrebbe essere in formato vettoriale; i formati raster sono solo accettabili come fallback. In questo caso, dovrebbero essere PNG trasparenti, con una larghezza minima di 1000px.

Chiave `monochromeLogo`

- Tipo: stringa (percorso verso il file)
- Presenza: opzionale
- Formati accettabili: SVG, SVGZ, PNG
- Esempio: `"img/logo-mono.svg"`

Questa chiave indica il logo monocromatico (nero) del software. Il valore può essere il percorso relativo al file a partire dalla root del repository, oppure una URL assoluta che punta al logo in versione raw. In entrambi i casi, il file deve risiedere all'interno del medesimo repository che contiene il `publiccode.yml`. Il logo dovrebbe essere in formato vettoriale; i formati raster sono solo accettabili come fallback. In questo caso, dovrebbero essere PNG trasparenti, con una larghezza minima di 1000px.

Chiave `inputTypes`

- Tipo: array di stringhe
- Presenza: opzionale
- Valori: vedi RFC 6838
- Esempio: `"text/plain"`

Una lista di Media Types (MIME Types), come specificato dal RFC 6838[5], che possono essere gestiti in input dall'applicazione.

Nel caso in cui il software non supporti alcun input, è possibile saltare questo campo o usare `application/x.empty`.

Chiave `outputTypes`

- Tipo: array di stringhe
- Presenza: opzionale
- Valori: vedi RFC 6838
- Esempio: `"text/plain"`

Una lista di Media Types (MIME Types), come specificato dal RFC 6838[6], che possono essere gestiti in output dall'applicazione.

Nel caso in cui il software non supporti alcun output, è possibile saltare questo campo o usare `application/x.empty`.

Chiave `platforms`

- Tipo: stringhe o array di stringhe
- Presenza: obbligatoria
- Valori: `web`, `windows`, `mac`, `linux`, `ios`, `android`. Valori leggibili da un umano al di fuori di questa lista sono permessi.
- Esempio: `web`

Questa chiave specifica su quale piattaforma funziona il software. È pensata per descrivere le piattaforme che l'utente userà per accedere ed utilizzare il software, piuttosto che la piattaforma sul quale il software gira.

Se possibile, usare i valori predefiniti. Se il software gira su una piattaforma per la quale un valore predefinito non è disponibile, un diverso valore può essere usato.

Chiave `categories`

- Tipo: array di stringhe
- Presenza: obbligatoria
- Valori accettabili: vedi *Lista delle categorie di software* (pagina 22)

[5] https://tools.ietf.org/html/rfc6838
[6] https://tools.ietf.org/html/rfc6838

Una lista di parole che possono essere usate per descrivere il software e possono aiutare a costruire il catalogo di software open.

Il vocabolario controllato *Lista delle categorie di software* (pagina 22) presenta la lista dei valori accettabili.

Chiave `usedBy`

- Tipo: array di stringhe
- Presenza: opzionale

Una lista di nome di prominenti Pubbliche Amministrazioni (che serviranno come "testimonial") che il maintainer riconosce come utilizzatori attuali di questo software.

I parser sono incoraggiati ad accrescere questa lista anche con altre informazioni che riescono ad ottenere in modo indipendente; ad esempio, il fork di un software, di proprietà di un'amministrazione, può essere usato come un segnale di uso del software.

Chiave `roadmap`

- Tipo: stringa
- Presenza: opzionale

Un link ad una *roadmap* pubblica del software.

Chiave `developmentStatus`

- Tipo: stringa
- Presenza: obbligatoria
- Valori permessi: `concept`, `development`, `beta`, `stable`, `obsolete`

Le chiavi sono:

- `concept` - Il software è solo un "concept". Non è stato sviluppato codice e il repository potrebbe semplicemente essere un placeholder.
- `development` - Qualche sforzo è stato fatto in direzione dello sviluppo del software ma il codice non è pronto per l'utenza finale, nemmeno in una versione preliminare (beta o alpha) per essere testato dall'utenza.
- `beta` - Il software è in fase di testing (alpha o beta). In questo stage, il software potrebbe aver o non aver ancora avuto una release pubblica preliminare.
- `stable` - Il software ha già avuto una prima release pubblica ed è pronto per essere usato in un contesto di produzione.
- `obsolete` - Il software non è più manutenuto o aggiornato. Tutto il codice sorgente è archiviato e tenuto per ragioni di storico.

Chiave `softwareType`

- Tipo: stringa
- Presenza: obbligatoria

- Valori permessi: `"standalone/mobile"`, `"standalone/iot"`, `"standalone/desktop"`, `"standalone/web"`, `"standalone/backend"`, `"standalone/other"`, `"addon"`, `"library"`, `"configurationFiles"`

Le chiavi sono:

- `standalone/mobile` - Il software è un pacchetto *self-contained*, *standalone*. Il software è un'applicazione nativa per dispositivi mobile.
- `standalone/iot` - Il software è adatto ad essere utilizzato nel contesto *Internet of Things*.
- `standalone/desktop` - Il software è tipicamente installato e utilizzato su un sistema operativo desktop.
- `standalone/web` - Il software rappresenta un applicativo fruibile attraverso il web.
- `standalone/backend` - Il software è un applicativo backend.
- `standalone/other` - Il software ha una natura diversa rispetto a quanto specificato alle chiavi precedenti.
- `softwareAddon` - Il software è un *addon*, come ad esempio un plugin o un tema, per un software più complesso (e.g., un CMS o una suite per ufficio).
- `library` - Il software contiene una libreria o una SDK che facilita la creazione di nuovi prodotti a sviluppatori di terze parti.
- `configurationFiles` - Il software non contiene script eseguibili ma una serie di file di configurazione. Questi potrebbero documentare come ottenere un certo tipo di *deployment*. I suddetti file potrebbero avere la forma di semplici file di configurazione, script bash, playbook ansible, Dockerfile, o altri set di istruzioni.

Sezione `intendedAudience`

Chiave `intendedAudience/countries`

- Tipo: array di stringhe
- Presenza: opzionale

Questa chiave include in modo esplicito alcuni Paesi tra il pubblico previsto, i.e., il software rivendica esplicitamente la conformità con processi specifici, tecnologie o leggi. Tutti i Paesi sono specificati usando *country code* a due lettere seguendo lo standard ISO 3166-1 alpha-2.

Chiave `intendedAudience/unsupportedCountries`

- Tipo: array di stringhe
- Presenza: opzionale

Questa chiave contrassegna esplicitamente i Paesi **NON** supportati. Questa situazione potrebbe verificarsi nel momento in cui esista un conflitto tra la modalità di funzionamento del software ed una legge specifica, un processo o una tecnologia. Tutti i Paesi sono specificati usando *country code* a due lettere seguendo lo standard ISO 3166-1 alpha-2.

Chiave `intendedAudience/scope`

- Tipo: array di stringhe
- Presenza: opzionale
- Valori accettabili: vedi *Lista dei campi di applicazione* (pagina 25)

Questa chiave contiene una lista di tag che rappresentano il campo di applicazione del software.

I tag consentiti sono elencati nella *Lista dei campi di applicazione* (pagina 25).

Sezione `description`

Questa sezione contiene una descrizione generale del software. I parser possono usare questa sezione ad esempio per creare una pagina web che descriva il software.

Nota bene: siccome tutte le stringhe contenute in questa sezione sono visibili all'utente e scritte in un linguaggio specifico, è **necessario** specificare il linguaggio con il quale si sta modificando il testo. Per farlo è necessario creare una sezione dedicata alla lingua seguendo le specifiche IETF BCP 47[7]. Si ricorda che il *primary language subtag* non può essere omesso, come specificato nel BCP 47.

Un esempio per l'italiano:

```
description:
  it:
    shortDescription: ...
    longDescription: ...
```

Nelle parti successive del documento, tutte le chiavi sono assunte essere all'interno di una sezione con il nome della lingua (annoteremo questo con `[lang]`).

Nota bene: è obbligatorio avere *almeno* una lingua in questa sezione. Tutte le altre lingue sono opzionali.

Chiave `description/[lang]/localisedName`

- Tipo: stringa
- Presenza: opzionale
- Esempio: `"Medusa"`

Questa chiave rappresenta un'opportunità di tradurre il nome in una lingua specifica. Contiene il nome pubblico (corto) del prodotto. Dovrebbe essere il nome con il quale la maggioranza delle persone normalmente si riferisce al software. Nel caso in cui il software abbia sia un nome "interno" che uno commerciale, è preferibile utilizzare quello commerciale.

Chiave `description/[lang]/genericName`

- Tipo: stringa (max 35 caratteri)
- Presenza: obbligatoria
- Esempio: `"Text Editor"`

Questa chiave rappresenta il "Nome generico", riferito alla categoria specifica alla quale il software appartiene. Normalmente è possibile trovare il nome generico nella presentazione del software, quando si scrive una frase del tipo: "Il software xxx è un yyy". Esempi degni di nota includono "Editor di Testi", "Word Processor", "Web Browser", "Chat" e così via. Il nome generico può avere una lunghezza fino a 35 caratteri.

[7] https://tools.ietf.org/html/bcp47

Chiave `description/[lang]/shortDescription`

- Tipo: stringa (max 150 caratteri)
- Presenza: obbligatoria
- Esempio: `"Sistema avanzato di prenotazione per ospedali"`

Questa chiave contiene una breve descrizione del software. Dovrebbe essere una singola linea contenente una singola frase. L'estensione massima consentita è di 150 caratteri.

Chiave `description/[lang]/longDescription`

- Tipo: stringa (min 500 caratteri, max 10000 caratteri)
- Presenza: obbligatoria (almeno per una lingua)

Questa chiave contiene una descrizione più lunga del software, con una lunghezza che può variare da 500 a 1000 caratteri. Questa chiave è pensata per fornire una panoramica delle caratteristiche del software per un potenziale utente. Il destinatario di questo testo dovrebbe essere l'utente finale, non nello sviluppatore. E' possibile pensare a questo testo come alla descrizione del software che potrebbe stare nel sito web (nel caso in cui il software ne possieda uno).

Questa descrizione può contenere del Markdown base: `*italic*`, `**bold**`, elenchi puntati e `[link](#)`.

Chiave `description/[lang]/documentation`

- Tipo: URL
- Presenza: opzionale

Questa chiave contiene un riferimento alla documentazione lato utente (non lato sviluppatore) Questo valore deve essere una URL che punta ad una versione ospitata della documentazione.

È suggerito che questa URL punti ad una versione ospitata della documentazione che sia direttamente leggibile utilizzando un comune web browser sia in formato desktop che mobile. La documentazione dovrebbe essere renderizzata in HTML e navigabile come un sito web (con un indice, una barra di ricerca, etc.).

Se la documentazione dovesse invece essere disponibile esclusivamente sotto forma di documento, è possibile inserire il link diretto per vedere/scaricare tale documento, sotto forma di URL, in questa chiave. E' consigliabile trattare la documentazione come parte del codice sorgente e dunque gestirla tramite commit sul repository del codice sorgente. In questo modo, sarà possibile fornire una URL diretta alla piattaforma di hosting del codice (ad es., GitHub URL al file). E' preferibile utilizzare formati aperti quali PDF o ODT per avere la massima interoperabilità. Qualunque sia il formato della documentazione, è importante ricordare di rilasciarne i sorgenti coperti da licenza aperta, possibilmente effettuandone un commit all'interno del repository stesso.

Chiave `description/[lang]/apiDocumentation`

- Tipo: URL
- Presenza: opzionale

Questa chiave contiene un riferimento alla documentazione delle API del software. Il valore deve essere una URL verso una versione ospitata della documentazione.

E' suggerito che questa URL punti ad una versione ospitata della documentazione che sia direttamente leggibile utilizzando un comune web browser. La documentazione dovrebbe essere renderizzata in HTML e navigabile come

un sito web (con un indice, una barra di ricerca, etc.), e se c'è un riferimento ad un deployment di prova, questo dovrebbe offrire un'interfaccia navigabile (e.g. Swagger).

Se la documentazione dovesse invece essere disponibile esclusivamente sotto forma di documento, è possibile inserire il link diretto per vedere/scaricare tale documento, sotto forma di URL, in questa chiave. E' consigliabile trattare la documentazione come parte del codice sorgente e dunque gestirla tramite commit sul repository del codice sorgente. In questo modo, sarà possibile fornire una URL diretta alla piattaforma di hosting del codice (ad es., GitHub URL al file). E' preferibile utilizzare formati aperti quali PDF o ODT per avere la

Qualunque sia il formato della documentazione, è importante ricordare di rilasciarne i sorgenti coperti da licenza aperta, possibilmente effettuandone un commit all'interno del repository stesso.

Chiave `description/[lang]/features`

- Tipo: array di stringhe
- Presenza: obbligatoria (almeno per una lingua)

Questa chiave contiene una lista di *feature* del software, che descriva le possibilità offerte dallo stesso. Il target di questo testo sono i decisori pubblici che potranno decidere di adottarlo o modificarlo. Per questo motivo, queste feature *non* devono riferirsi agli sviluppatori: invece di elencare le caratteristiche tecniche riferite ai dettagli implementativi, è preferibile elencare le funzionalità lato utente.

Anche se questa chiave è obbligatoria, non c'è un limite minimo o massimo sul numero di feature da elencare in questa chiave. Ogni feature deve però avere un massimo di 100 caratteri.

Il numero di feature suggerito da elencare è tra 5 e 20, a seconda della dimensione del software e della sua complessità. Non c'è bisogno di fare una lista esaustiva, dal momento che gli utenti hanno sempre a disposizione la documentazione per reperire ulteriori informazioni.

Chiave `description/[lang]/screenshots`

- Tipo: array di stringhe (percorsi)
- Presenza: opzionale
- Formati: PNG, JPG
- Esempio: `"data/screenshots/configuration.png"`

Questa chiave indica una o più immagini del software (screenshot). Queste hanno lo scopo di dare una panoramica dell'aspetto del software e del suo funzionamento. Il valore può essere il percorso relativo al file a partire dalla root del repository, oppure una URL assoluta che punta all'immagine in versione raw. In entrambi i casi, il file deve risiedere all'interno del medesimo repository che contiene il `publiccode.yml`.

Queste immagini possono essere di qualsiasi formato e dimensione; i formati suggeriti sono:

- Desktop: 1280x800 @1x
- Tablet: 1024x768 @2x
- Mobile: 375x667 @2x

Chiave `description/[lang]/videos`

- Tipo: array di stringhe (URL)
- Presenza: opzionale

- Esempio: `"https://youtube.com/xxxxxxxx"`

Questa chiave contiene una o più URL di video che mostrano il funzionamento del software. Così come gli screenshot, i video dovrebbero essere usati per dare una rapida panoramica sull'aspetto e le funzionalità del software. I video devono essere ospitati su una piattaforma di video sharing che supporti lo standard oEmbed[8]; le opzioni più popolari sono YouTube e Vimeo.

Nota bene: dal momento che costituisce parte integrante della documentazione, è opportuno che il video sia pubblicato con una licenza aperta.

Chiave `description/[lang]/awards`

- Tipo: array di stringhe
- Presenza: opzionale

Una lista di premi assegnati al software.

Sezione `legal`

Chiave `legal/license`

- Tipo: stringa
- Presenza: obbligatoria
- Esempio: `"AGPL-3.0-or-later"`

Questa stringa descrive la licenza con cui il software è distribuito. La stringa deve contenere un'espressione SPDX valida che si riferisca ad una (o più) licenze open-source. Per avere ulteriori informazioni a riguardo è possibile visitare la documentazione SPDX[9].

Chiave `legal/mainCopyrightOwner`

- Tipo: stringa
- Presenza: opzionale
- Esempio: `"Città di Roma"`

Questa stringa descrive l'entità che possiede il copyright sulla maggior parte del codice presente nel repository. Normalmente, questa è la linea che viene riportata con il simbolo di copyright all'inizio della maggior parte dei file nel repository.

E' possibile elencare diversi proprietari se necessario, usando una frase in inglese. E' anche possibile riferirsi ad una community o ad un gruppo di persone come ad esempio "Linus Torvalds and all Linux contributors".

Nel caso in cui non sia possibile individuare il maggior proprietario di copyright, è possibile omettere questa chiave; in questi casi, se il repository ha un file contenente il nome degli autori, è possibile puntare a quel file attraverso `legal/authorsFile` (vedi più sotto).

[8] https://oembed.com
[9] https://spdx.org/licenses/

Chiave `legal/repoOwner`

- Tipo: stringa
- Presenza: opzionale
- Esempio: `"Città di Roma"`

Questa stringa descrive l'entità che possiede il repository; questa può essere o non essere la stessa che possiede il copyright del codice stesso. Ad esempio, nel caso di un fork del software originale, il `repoOwner` è probabilmente diverso dal `mainCopyrightOwner`.

Chiave `legal/authorsFile`

- Tipo: stringa (percorso al file)
- Presenza: opzionale
- Esempio: `"doc/AUTHORS.txt"`

Qualche software open-source adotta una convenzione che identifica il detentore del copyright attraverso un file elencante tutte le entità che possiedono il copyright. Questo è comune nei progetti fortemente sostenuti dalla community ove esistono diversi contributori esterni e non c'è un chiaro singolo detentore del copyright. In questi casi, questa chiave può essere usata per riferirsi al suddetto file degli autori, usando un percorso relativo alla radice (root) del repository.

Sezione `maintenance`

Questa sezione fornisce informazioni sullo stato di manutenzione del software, utile per valutare se il software è attivamente sviluppato o meno.

Chiave `maintenance/type`

- Tipo: enumerate
- Presenza: obbligatoria
- Valori: `"internal"`, `"contract"`, `"community"`, `"none"`

Questa chiave descrive come il software è attualmente manutenuto. Le chiavi sono:

- `internal` - significa che il software è manutenuto internamente dal proprietario del repository;
- `contract` - significa che c'è un contratto commerciale che lega un'entità alla manutenzione del software;
- `community` - significa che il software è attualmente manutenuto da una o più persone che offrono il loro tempo al progetto;
- `none` - significa che il software non è al momento manutenuto.

Chiave `maintenance/contractors`

- Tipo: array di Contractor (vedi sotto)
- Presenza: obbligatoria (se `maintenance/type` **è** `contract`)

Questa chiave descrive l'entità o le entità, se ce ne sono, che attualmente hanno un contratto di manutenzione del software. Queste possono essere aziende, organizzazioni o altri nomi collettivi.

Chiave `maintenance/contacts`

- Tipo: Lista di Contatti (vedi sotto)
- Presenza: obbligatoria (se `maintenance/type` **è** `internal` oppure `community`)

Uno o più contatti di chi sta mantenendo il software.

Questa chiave descrive le persone tecniche che attualmente sono responsabili della manutenzione del software. Tutti i contatti devono essere di una persona fisica, non un'azienda o un'organizzazione. Se un contatto funge da rappresentante di un'istituzione, questo rapporto deve essere esplicitato attraverso la chiave `affiliation`.

Nel caso di un accordo commerciale (o una catena di tali accordi), specificare le entità finali che sono effettivamente contrattate per fornire la manutenzione. Non specificare il proprietario del software a meno che sia tecnicamente coinvolto anche nella manutenzione del prodotto.

Sezione `localisation`

Questa sezione fornisce una panoramica sulle funzionalità di localizzazione del software.

Chiave `localisation/localisationReady`

- Tipo: booleano
- Presenza: obbligatoria

Se `yes`, il software ha l'infrastruttura o è stato progettato per essere multi-lingua. Ad ogni modo, questo campo non pregiudica l'esistenza di una traduzione in altre lingue ma si riferisce esclusivamente all'aspetto tecnologico. Per l'elenco delle lingue disponibili si veda la chiave `localisation/availableLanguages`.

Chiave `localisation/availableLanguages`

- Tipo: lista di *language tag* secondo le specifiche IETF BCP 47
- Presenza: obbligatoria
- Esempio: `"it"`, `"en"`, `"sl-IT-nedis"`

Se presente, questa è la lista di lingue in cui è disponibile il software. Ovviamente, questa lista dovrà contenere almeno una lingua. Si ricorda che il *primary language subtag* non può essere omesso, come specificato dal BCP 47[10].

Sezione `dependsOn`

Questa sezione fornisce una panoramica delle dipendenze a livello di sistema necessarie per installare ed utilizzare il software.

Nota bene: non elencare le dipendenze a livello di codice sorgente (ad es., librerie software usate), e focalizza solo su dipendenze di sistema e/o a runtime che devono essere installate e manutenute separatamente. Ad esempio, un database è un buon esempio di questo tipo di dipendenza.

[10] https://tools.ietf.org/html/bcp47

Chiave `dependsOn/open`

- Tipo: array di `dependency` (vedi sotto)
- Presenza: opzionale

Questa chiave contiene una lista di dipendenze a runtime che sono distribuite con una licenza di tipo open-source.

Chiave `dependsOn/proprietary`

- Tipo: array di `dependency` (vedi sotto)
- Presenza: opzionale

Questa chiave contiene una lista di dipendenze a runtime che sono distribuite con una licenza proprietaria.

Chiave `dependsOn/hardware`

- Tipo: array di `dependency` (vedi sotto)

This key contains a list of hardware dependencies that must be owned to use the software.

2.1.2 Formati di dato speciali

Dependency

Una `dependency` è un oggetto complesso. Le proprietà sono le seguenti:

- `name` - **obbligatoria** - Il nome della dipendenza (e.g. MySQL, NFC Reader);
- `versionMin` - la prima versione compatibile;
- `versionMax` - l'ultima versione compatibile;
- `version` - l'unica versione major con la quale il software è compatibile. Si assume la compatibilità con tutte le *patch* e i *bugfix* che saranno applicati successivamente a questa versione;
- `opzionale` - se la dipendenza è opzionale o obbligatoria.

Versioning complesso

E' ovviamente possibile utilizzare le varie chiavi per specificare una matrice di compatibilità complessa.

Ex. 1

```
- name: PostgreSQL
  version: "3.2"
  opzionale: yes
```

Questo snippet segnala una dipendenza opzionale verso PostgreSQL, nell'esattezza la sua versione 3.2.

Ex. 2

```
- name: MySQL
  versionMin: "1.1"
  versionMax: "1.3"
```

Questo snippet segnala una dipendenza obbligatoria verso MySQL, permettendo ogni versione tra la 1.1 e la 1.3.

Contatto

Un Contatto è un oggetto con le seguenti proprietà:

- `name` - **obbligatoria** - Questa chiave contiene il nome completo di uno dei contatti tecnici. Deve essere una persona reale; NON popolare questa chiave con informazioni di contatto generiche, dipartimenti dell'azienda, associazioni, etc.
- `email` - Questa chiave contiene l'indirizzo email del contatto tecnico. Deve essere un indirizzo email per il contatto diretto con il tecnico; NON popolare questa chiave con mailing-list o punti di contatto generico tipo "info@acme.inc". Questo indirizzo email non deve essere offuscato. Per migliorare la resistenza contro la raccolta di indirizzi email, usare `\x64` per sostituire `@`, siccome questo è permesso dalle specifiche YAML.
- `phone` - Numero telefonico (con prefisso internazionale). Questa chiave deve essere una stringa.
- `affiliation` - Questa chiave contiene informazioni esplicite sui contatti tecnici. Nel caso esistano diversi maintainer, questa chiave può essere usata per creare relazioni tra diversi contatti tecnici e entità di manutenzione. Ad esempio, può contenere il nome di un'azienda, il nome di un'associazione, etc.

Contractor

Un Contractor è un oggetto con le seguenti proprietà:

- `name` - **obbligatoria** - Il nome del contractor, sia esso un'azienda o una persona fisica.
- `until` - **obbligatoria** - Questa è una data (YYYY-MM-DD). Questa chiave deve contenere una data alla quale la manutenzione finirà. Nel caso di manutenzione gestita dalla community, questo valore non deve essere maggiore di 2 anni nel futuro, e quindi deve essere regolarmente aggiornata man mano che la community continua a lavorare al progetto.
- `email` - Questa chiave contiene l'indirizzo email del contatto tecnico. Deve essere un indirizzo email per il contatto diretto con il tecnico; NON popolare questa chiave con mailing-list o punti di contatto generico tipo "info@acme.inc". Questo indirizzo email non deve essere offuscato. Per migliorare la resistenza contro la raccolta di indirizzi email, usare `\x64` per sostituire `@`, siccome questo è permesso dalle specifiche YAML.
- `website` - Questa chiave punta al sito del maintainer. Può puntare al principale sito istituzionale, o ad una pagina o sito più specifica.

Data

Tutte le date in `publiccode.yml` devono aderire al formato "YYYY-MM-DD" che è uno dei formati permessi da ISO8601. **Nota bene:** questo è l'unico formato permesso, quindi non sono consentiti gli altri formati previsti da ISO8601.

Codifica

La codifica di `publiccode.yml` **DEVE** essere UTF-8.

2.2 Estensioni nazionali

Mentre il core è strutturato per essere significativo a livello internazionale, vi sono informazioni addizionali che possono essere aggiunte a livello nazionale, come ad esempio una dichiarazione di compatibilità con una legge locale. Il meccanismo di estensione fornito prevede l'utilizzo di sezioni specifiche per ogni Paese (*country-specific*).

Tutte le sezioni specifiche per ogni Paese sono contenute in una sezione denominata con l'ISO 3166-1 alpha-2 country code[11]. Ad esempio, `spid` è una proprietà definita per i software italiani per la dichiarazione dell'eventuale compatibilità con il Sistema Pubblico di Identità Digitale.

Dunque, se un software è compatibile, troveremo:

```
it:
  countryExtensionVersion: "0.2"
  piattaforme:
     - spid: yes
```

Nota bene che le chiavi *country-specific* **non** sono valide all'interno delle sezioni internazionali. I Paesi che vogliano estendere il formato devono aggiungere una sezione dedicata.

2.2.1 Italia

Tutte le estensioni elencate qui di seguito sono specifiche per l'Italia e, di conseguenza, devono essere inserite in una sezione denominata con il codice `it`. Tutti i Paesi sono specificati usando *country code* a due lettere seguendo lo standard ISO 3166-1 alpha-2.

Chiave `countryExtensionVersion`

- Tipo: stringa
- Presenza: obbligatoria
- Esempio: `"1.0"`

Questa chiave specifica la versione alla quale il presente schema di estensioni aderisce.

Nota Bene: il valore di questa chiave è indipendente da quello contenuto nella chiave top-level `publiccodeYmlVersion` (vedi *Lo standard (core)* (pagina 5)). In questo modo, il versioning di ogni schema di estensioni è indipendente sia dalla versione core dello schema che da ogni altra estensione per Paese.

Sezione `conforme`

Questa sezione contiene delle chiavi per auto dichiarare la conformità con la normativa vigente, rispetto ad alcune sezioni. Se queste chiavi non vengono incluse, si intende che la conformità non è nota o non viene dichiarata.

Chiave `conforme/lineeGuidaDesign`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software è conforme alle leggi in materia di accessibilità (L. 4/2004), come descritto ulteriormente nelle linee guida di design[12].

[11] https://it.wikipedia.org/wiki/ISO_3166-1_alpha-2
[12] https://docs.italia.it/italia/designers-italia/design-linee-guida-docs

Chiave `conforme/modelloInteroperabilita`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software è conforme alle linee guida sull'interoperabilità[13].

Riferimento normativo: Art. 73 del CAD[14].

Chiave `conforme/misureMinimeSicurezza`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software è conforme alle Misure minime di sicurezza ICT per le Pubbliche amministrazioni[15].

Chiave `conforme/gdpr`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software rispetta il GDPR.

Sezione `piattaforme`

Chiave `piattaforme/spid`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software si interfaccia con SPID - il Sistema Pubblico di Identità Digitale[16].

Chiave `piattaforme/cie`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software si interfaccia con la Carta di Identità Elettronica.

Chiave `piattaforme/anpr`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software si interfaccia con ANPR.

[13] https://docs.italia.it/italia/piano-triennale-ict/lg-modellointeroperabilita-docs
[14] https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/_rst/capo8_art73.html
[15] https://www.agid.gov.it/it/sicurezza/misure-minime-sicurezza-ict
[16] https://developers.italia.it/it/spid

Chiave `piattaforme/pagopa`

- Tipo: booleano
- Presenza: opzionale

Se presente e impostato a `yes`, il software si interfaccia con pagoPA.

Sezione `riuso`

Questa sezione contiene una serie di chiavi legate alla pubblicazione del software sul Catalogo del Riuso[17].

Chiave `riuso/codiceIPA`

- Tipo: stringa (codice iPA)
- Presenza: obbligatoria se `repoOwner` è una Pubblica Amministrazione
- Esempio: `c_h501`

Questa chiave rappresenta il codice dell'amministrazione all'interno dell'Indice delle Pubbliche Amministrazioni (codice IPA).

2.3 Fork e varianti

Come già accennato precedentemente, il fork di un progetto software può essere inteso in modo diverso a seconda dello scopo finale di tale diramazione. Per questioni di chiarezza, qui di seguito distinguiamo le due possibilità: fork tecnici e varianti software.

2.3.1 Fork tecnici (i.e. pubblicare patch)

Un fork tecnico è un fork eseguito da uno sviluppatore con lo scopo di lavorare sulla *code base* originale o per inviare miglioramenti agli autori del software originale, senza la finalità esplicita di creare e mantenere una variante alternativa del software originale.

Nel contesto di un sistema di controllo distribuito e piattaforme di hosting di codice collaborative quali GitHub, lo strumento del *fork* è quasi sempre usato dagli sviluppatori come il primo passo per lavorare ad un contributo su una *code base* già esistente inviando *pull request*.

Visto il modo in cui il sistema di *forking* funziona su GitHub ed altre piattaforme simili, gli sviluppatori pubblicano i propri fork sotto forma di copie perfette del software originale, quindi includendo anche il file `publiccode.yml` originale. I parser devono essere in grado di distinguere questi fork tecnici dalla *code base* originale.

Parser

I *parser* **DOVREBBERO** identificare un fork tecnico notando che la chiave top-level `url` non punta al repository dove si trova il file `publiccode.yml`.

I *parser* **POTREBBERO** identificare un fork tecnico anche attraverso i metadati che potrebbero essere esposti dalla piattaforma di code hosting (e.g., GitHub contrassegna i fork esplicitamente come fork).

[17] https://developers.italia.it

Autori

Gli autori di un fork tecnico **NON DOVREBBERO** modificare il file `publiccode.yml` in alcun modo. Più nello specifico, **NON DEVONO** modificare la chiave top-level `url` in quanto questa **DEVE** continuare a puntare al repository originale.

Non c'è una chiave specifica per contrassegnare un fork come tecnico. Questa è una scelta consapevole di design perché non vogliamo che gli autori di un fork tecnico debbano necessariamente essere consapevoli del file `publiccode.yml` e di come doverlo modificare. Il design corrente non richiede che questi autori facciano alcunché.

2.3.2 Varianti software

Una variante software è un fork che rappresenta una valida alternativa al software originale upstream.

Questa contiene modifiche che non sono ancora parte della versione upstream, come ad esempio più funzionalità, dipendenze diverse, ottimizzazioni, etc.

Contrassegnando un fork come una variante, l'autore indica che questa variante include una serie di modifiche complete e funzionanti che potrebbero giovare ad altre persone.

Contrassegnare un fork come una variante **non** pregiudica la volontà di contribuire upstream; l'autore potrebbe comunque voler contribuire upstream o essere in attesa di farlo. Perciò, anche se il fork alla fine verrà inclusa (merge) upstream, potrebbe aver senso contrassegnarla come una variante durante queste fasi di lavoro intermedie, in modo tale che anche altri possano trovarla e beneficiarne.

Parser

I parser **DOVREBBERO** identificare una variante notando che la chiave top-level `url` è uguale al repository nel quale il file `publiccode.yml` risiede, **E** una chiave top-level `isBasedOn` esiste e punta ad un altro repository.

I parser dovrebbero aspettarsi e analizzare altre differenze nelle due varianti dei file `publiccode.yml`. In particolare, la chiave `description/features` è pensata per essere comparata tra diverse varianti in modo da identificare e visualizzare le differenze lato utente.

Autori

Gli autori che vogliano pubblicare un fork come una variante **DEVONO** almeno:

- Aggiungere una chiave `isBasedOn` che punti a uno o più repository upstream dai quali questa variante deriva.
- Cambiare il valore di `url` per farla puntare al repository che ospita la variante.
- Cambiare il valore di `legal/repoOwner` per identificarsi come autori della variante.
- Rivisitare la sezione denominata `maintenance` per aggiornare lo stato di manutenzione della variante.

Inoltre, gli autori **DOVREBBERO** valutare di apportare anche i seguenti cambiamenti:

- Aggiungere le funzionalità che differenziano le varianti alla chiave `description/features`. Le funzionalità esistenti **NON DOVREBBERO** essere modificate o rimosse da questa lista a meno che esse siano state rimosse dalla variante, per permettere ai parser di comparare facilmente la lista delle funzionalità.

2.4 Lista delle categorie di software

Qui di seguito è presentato un vocabolario controllato di tag utilizzabili per categorizzare il software.

2.4.1 Tag Validi

- accounting
- agile-project-management
- applicant-tracking
- application-development
- appointment-scheduling
- backup
- billing-and-invoicing
- blog
- budgeting
- business-intelligence
- business-process-management
- cad
- call-center-management
- cloud-management
- collaboration
- communications
- compliance-management
- contact-management
- content-management
- crm
- customer-service-and-support
- data-analytics
- data-collection
- data-visualization
- digital-asset-management
- digital-citizenship
- document-management
- donor-management
- e-commerce
- e-signature
- email-management
- email-marketing
- employee-management
- enterprise-project-management
- enterprise-social-networking

- erp
- event-management
- facility-management
- feedback-and-reviews-management
- financial-reporting
- fleet-management
- fundraising
- gamification
- geographic-information-systems
- grant-management
- graphic-design
- help-desk
- hr
- ide
- identity-management
- instant-messaging
- inventory-management
- it-asset-management
- it-development
- it-management
- it-security
- it-service-management
- knowledge-management
- learning-management-system
- marketing
- mind-mapping
- mobile-marketing
- mobile-payment
- network-management
- office
- online-booking
- online-community
- payment-gateway
- payroll
- predictive-analysis
- procurement

- productivity-suite
- project-collaboration
- project-management
- property-management
- real-estate-management
- remote-support
- resource-management
- sales-management
- seo
- service-desk
- social-media-management
- survey
- talent-management
- task-management
- taxes-management
- test-management
- time-management
- time-tracking
- translation
- video-conferencing
- video-editing
- visitor-management
- voip
- warehouse-management
- web-collaboration
- web-conferencing
- website-builder
- workflow-management

2.5 Lista dei campi di applicazione

Qui di seguito è presentato un vocabolario controllato di tag utilizzabili per i campi di applicazione del software.

2.5.1 Tag Validi

- agriculture
- culture

- defence
- education
- emergency-services
- employment
- energy
- environment
- finance-and-economic-development
- foreign-affairs
- government
- healthcare
- infrastructures
- justice
- local-authorities
- manufacturing
- research
- science-and-technology
- security
- society
- sport
- tourism
- transportation
- welfare

2.6 Esempi

Qui di seguito vi sono due esempi di file *publiccode.yml*, il primo rappresenta la configurazione minima, ovvero contiene i campi obbligatori, mentre il secondo è più esteso.

2.6.1 Versione Minima

```yaml
publiccodeYmlVersion: "0.2"

name: Medusa
url: "https://example.com/italia/medusa.git"
softwareVersion: "dev"    # Optional
releaseDate: "2017-04-15"
platforms:
  - web

categories:
  - financial-reporting
```

(continues on next page)

(continua dalla pagina precedente)

```yaml

developmentStatus: development

softwareType: "standalone/desktop"

description:
  en:
    localisedName: medusa    # Optional
    genericName: Text Editor
    shortDescription: >
          A rather short description which
          is probably useless

    longDescription: >
          Very long description of this software, also split
          on multiple rows. You should note what the software
          is and why one should need it. We can potentially
          have many pages of text here.

    features:
       - Just one feature

legal:
  license: AGPL-3.0-or-later

maintenance:
  type: "community"

  contacts:
    - name: Francesco Rossi

localisation:
  localisationReady: yes
  availableLanguages:
    - en
```

2.6.2 Versione Estesa

```yaml
publiccodeYmlVersion: "0.2"

name: Medusa
applicationSuite: MegaProductivitySuite
url: "https://example.com/italia/medusa.git"
landingURL: "https://example.com/italia/medusa"
isBasedOn: "https://github.com/italia/otello.git"
softwareVersion: "1.0"
releaseDate: "2017-04-15"
logo: img/logo.svg
monochromeLogo: img/logo-mono.svg

inputTypes:
  - text/plain
outputTypes:
  - text/plain

```

(continues on next page)

(continua dalla pagina precedente)

```yaml
platforms:
  - android
  - ios

categories:
  - content-management
  - office

usedBy:
  - Comune di Firenze
  - Comune di Roma

roadmap: "https://example.com/italia/medusa/roadmap"

developmentStatus: development

softwareType: "standalone/desktop"

intendedAudience:
  scope:
    - science-and-technology
  countries:
    - it
    - de
  unsupportedCountries:
    - us

description:
  en:
    localisedName: Medusa
    genericName: Text Editor
    shortDescription: >
          This description can have a maximum 150
          characters long. We should not fill the
          remaining space with "Lorem Ipsum". End

    longDescription: >
          Very long description of this software, also split
          on multiple rows. You should note what the software
          is and why one should need it.

    documentation: "https://read.the.documentation/medusa/v1.0"
    apiDocumentation: "https://read.the.api.doc/medusa/v1.0"

    features:
       - Very important feature
       - Will run without a problem
       - Has zero bugs
       - Solves all the problems of the world
    screenshots:
       - img/sshot1.jpg
       - img/sshot2.jpg
       - img/sshot3.jpg
    videos:
       - https://youtube.com/xxxxxxxx
    awards:
       - 1st Price Software of the year
```

(continues on next page)

(continua dalla pagina precedente)

```yaml

legal:
  license: AGPL-3.0-or-later
  mainCopyrightOwner: City of Chicago
  repoOwner: City of Chicago
  authorsFile: AUTHORS

maintenance:
  type: "contract"

  contractors:
    - name: "Fornitore Privato SPA"
      email: "dario.bianchi@fornitore.it"
      website: "https://privatecompany.com"
      until: "2019-01-01"

  contacts:
    - name: Francesco Rossi
      email: "francesco.rossi@comune.reggioemilia.it"
      affiliation: Comune di Reggio Emilia
      phone: "+3923113215112"

localisation:
  localisationReady: yes
  availableLanguages:
    - en
    - it
    - fr
    - de

dependsOn:
  open:
    - name: MySQL
      versionMin: "1.1"
      versionMax: "1.3"
      optional: yes
    - name: PostgreSQL
      version: "3.2"
      optional: yes
  proprietary:
    - name: Oracle
      versionMin: "11.4"
    - name: IBM SoftLayer
  hardware:
    - name: NFC Reader
      optional: yes

it:
  countryExtensionVersion: "0.2"

  conforme:
    lineeGuidaDesign: yes
    modelloInteroperabilita: yes
    misureMinimeSicurezza: yes
    gdpr: yes

  piattaforme:
```

(continues on next page)

(continua dalla pagina precedente)

```
    spid: yes
    cie: yes
    anpr: yes
    pagopa: yes

  riuso:
    codiceIPA: c_h501
```