

تحديث المكتبة الرقمية دون توقف: ثماني استراتيجيات للانتقال

جاستن ليتمان، مايك جيارلو، بيتر مانجيافيكو، لورا وروبل، نعومي دوشي، آرون كولير، أركاديا فالكوني



المخلص:

يعرض المقال تجربة مستودع ستانفورد الرقمي في الانتقال من مخزن قديم للبيانات الوصفية إلى نموذج (Cocina) ومخزن أحدث، مع استمرار الخدمات. ويشرح دوافع التغيير المرتبطة بالاعتمادية والقيود التقنية وعدم اتساق البيانات، ثم يقدم ثماني استراتيجيات شملت التحقق بالمخططات، وفصل المخزن خلف واجهة برمجية، وتوزيع المسؤوليات، واختبار التحويل بيانات حقيقية، وإعداد التقارير، وقوائم الاختبارات، والانتقال المتدرج، ودمج العمل في المشروعات الجارية. ويبرز أهمية المعرفة بالبيانات والتطبيع والقياس وإمكان التراجع، إلى جانب التفاهم مع القيادة ومالكي المنتجات. ويعرض نتائج الانتقال وتحسن الأداء والاستقرار والدروس التنظيمية والتقنية. ويقدم التجربة أساسًا تستفيد منه المكتبات عند تحديث بنية المستودعات دون إهمال احتياجات المستخدمين واستمرارية الإتاحة.

الكلمات المفتاحية: المكتبات الرقمية، ترحيل البيانات، المستودعات الرقمية، البيانات الوصفية، هندسة البرمجيات

مقدمة:

عمل فريق من مكتبات جامعة ستانفورد، خلال الأعوام الأربعة السابقة لكتابة المقال وبصورة متقطعة، على الانتقال إلى مخزن بيانات ونموذج جديدين للبيانات الوصفية للكائنات الرقمية في مستودع ستانفورد الرقمي (SDR). ونشرح دوافع العمل واستراتيجيات تنفيذه، التي قد تفيد مشروعات انتقال مستودعات أو تطبيقات مكتبات رقمية أخرى.

عن مستودع ستانفورد الرقمي:

المستودع منظومة من التطبيقات والنظم والخدمات التي تضم مجموعات مكتبات ستانفورد الرقمية دعمًا للبحث والتعليم والتعلم. وتشمل كتبًا رقميًا (Google)، ورسائل الجامعة وأطروحاتها، وأرشيف الجامعة، وأوراق ألن غينسبرغ وبكمنستر فولر، ومكتبة باركر، وأرشيف ويب الوكالات الأمريكية التي توقفت عن العمل.

وقت كتابة الأصل، ضم (SDR) قرابة خمسة ملايين كائن رقمي تتكون من أكثر من 530 مليون ملف محتوى. ويتسم بتنوع شديد في أنواع المحتوى، كالكتب والصور وأرشيفات الويب وبيانات نظم المعلومات الجغرافية، وصيغ الملفات مثل (XML) و (TIFF) و (WARC) و (MP4).

منظومة المستودع وتدفعات البيانات:

توجد مسارات متعددة لإيداع الكائنات الرقمية، ببياناتها الوصفية وملفاتها؛ واجهة ويب للإيداع الذاتي لمادة واحدة يستخدمها الباحثون من أعضاء هيئة التدريس والطلاب والموظفين؛ وواجهة لإيداع الرسائل والأطروحات لطلاب الدراسات العليا والبيكالوريوس؛ وواجهة لموظفي المكتبة تسمح بالإيداع الدفعي؛ وإيداع عبر (API) كان مقصورًا على الاستخدام الداخلي؛ وبرمجيات خارجية مخصصة لإدارة رقمنة المواد المادية على يد موظفي المكتبة.

ويتفاعل (SDR) مع تطبيقات رقمية أخرى؛ فيودع الكائنات في نظام الحفظ، وينشرها إلى نظام الإتاحة، ويجلب البيانات الوصفية من نظام المكتبة المتكامل (Symphony). ولا نتناول نظامي الحفظ والإتاحة تفصيلًا هنا. لكن الأول يوفر نسًا احتياطية متكررة داخل الموقع وخارجه، مع فحوص سلامة وتدقيق؛ والثاني يتيح البحث والاكتشاف والعرض عبر الويب، بما فيه (IIIF)، والتنزيل. وتُضبط مستويات متنوعة من قيود الاكتشاف والإتاحة والحظر المؤقت وتُطبق عبر المنظومة.

مخزن إدارة البيانات الوصفية:

بدأ (SDR) العمل سنة 2006، واعتمد على (Fedora 3) لتخزين البيانات الوصفية للكائنات مدة أحد عشر عامًا على الأقل. وتشمل البيانات الوصف والتعريف والإتاحة والحقوق والبنية وبيانات إدارية أخرى، بينما لا تُخزن ملفات المحتوى نفسها في (Fedora).

ورغم أن (Fedora 3) حمل عبء العمل سنوات، فقد واجه أوجه قصور جوهرية: توقف دعمه، وكان آخر إصداراته سنة 2015؛ واعتمد على (Java 8) الذي يصف الأصل انتهاء عمره سنة 2019. ووحدات التخزين الأساسية لملفات (XML) على القرص، المسماة «تدفعات بيانات»، وهي غير خاضعة للتحقق ولا لمخطط بحكم التصميم. كما توقفت مكتبتا (Ruby) الداعمتان، (ActiveFedora 8.x) و (Rubydora)، عن التطوير النشط سنة 2018، مما قيد تحديث (Ruby) وإطار (Rails) المستخدمين في المنظومة.

ولا يدعم (Fedora 3) المعاملات، فاضطراب الشبكة مثلًا قد ينتج كائنًا ببيانات ناقصة أو معطوبة. ولا يدعم القيود، مثل التفرد، فيسمح بإدراج بيانات إشكالية. ولا يوفر بذاته استعلامات متقدمة عن البيانات؛ لذلك وفر (SDR) خدمة (Solr) للبحث النصي، مع ضرورة مزامنتها مع مخزن البيانات الأصلي. ولم يُصمم (Fedora 3) لموازنة الحمل، فأصبح نقطة فشل وحيدة تحت الضغط، حتى مع توزيع الحمل في بقية النظام. وقد لاحظنا انهياره مرارًا عند زيادة الحمل.

وعانت منظومة التطبيقات والبيانات أيضًا: ارتبطت التطبيقات مباشرة بـ (Fedora) لتعديل تدفعات (XML)، واستغلت مرونتها لتخزين بيانات خاصة بكل تطبيق. ونتج عن غياب المخطط والتحقق عدم اتساق (XML)، بسبب تعديل كل تطبيق له مباشرة، وتمكين المستخدمين من تحريره يدويًا في تطبيق الإدارة لإحداث تغييرات وظيفية بوصف ذلك مسارًا معتمدًا، وبقاء طرائق وصف وتطبيقات قديمة مهجورة دون إصلاح، وضعف أولوية أدوات تدقيق مجمل بيانات (XML) وفحص جودتها.

دفعت هذه المشكلات مجتمعة إلى الانتقال إلى مخزن إدارة جديد ونموذج غني قابل للتحقق. ونركز فيما يلي على دوافع الاستراتيجيات ومنطقها، أكثر من الشيفرة أو تفاصيل التنفيذ.

الاستراتيجية الأولى: اعتماد نموذج بيانات قابل للتحقق:

من أهم الدروس ضرورة إمكان فحص بيانات كل كائن والتأكد من مطابقتها للنموذج. وكانت الوسيلة الأساسية تحديد مخطط يؤدي أيضًا وظيفة توثيق النموذج.

طورنا نموذجًا محليًا اسمه (Cocina). وتُسلّس بياناته بصيغة (JSON)، بينما يُمثل مخطّطه بمواصفة (OpenAPI). واختارناها لشيوعها وقوة أدواتها في لغات متعددة، وإمكان استخدام المواصفة نفسها لتعريف واجهات التطبيقات المتخاطبة بلغة (Cocina)، ولتوازنها بين سهولة الاستخدام وقوة التحقق.

وكانت مكتبة توفر أصناف (Ruby) لـ (Cocina) عنصرًا حاسمًا في دمجها؛ إذ تعالج الشيفرة البيانات في صورة كائنات (Ruby)، لا (JSON) مباشرة. وتُنشأ الأصناف بمولد شيفرة كتبناه لتحويل جزء من (OpenAPI) إلى (Ruby). وعند تعديل النموذج نعدل المواصفة ثم نولد الأصناف مجددًا.

وجوهر الطريقة أن إنشاء كائن (Cocina) جديد يطلق التحقق تلقائيًا وفق المخطط. وهكذا يجري التحقق على مستوى الشيفرة ومستوى الواجهة البرمجية بالآلية نفسها.

وثمة قواعد قليلة يصعب تمثيلها في (OpenAPI)، مثل اتساق حقوق الإتاحة بين مستوى الكائن ومستوى ملفاته. ولهذه نستخدم تحققًا برمجيًا ينفذ أيضًا عند إنشاء كل كائن. ويمنحنا الجمع بينه وبين المخطط ثقة كبيرة في الاتساق.

لكن قسم الوصف أكثر مرونة: فقد صمّمته أركاديا فالكوني، اختصاصية بيانات المستودع، ليدعم التحويل إلى صيغ متعددة مثل (MODS) و (DataCite). وتتحقق (OpenAPI) من بنيته الأساسية، بينما تُعرّف قواعد دلالية إضافية في ملف (YAML) داخل مكتبة (Ruby). ويسرد الملف مصطلحات، مثل أنواع الملاحظات المقبولة، تُفحص قيم خصائص مختارة على أساسها عند إنشاء الكائن. وتتغير هذه القوائم أكثر من النموذج العام؛ لذلك يجعل فصل هذه القوائم المواصفة أكثر استقرارًا. كما يُستخدم الملف أساسًا لتوثيق موجه إلى منشئي البيانات الذين لا يحتاجون إلى تفاصيل النموذج كلها.

الاستراتيجية الثانية: وضع مخزن البيانات خلف واجهة برمجية:

لفك الارتباط الوثيق بين التطبيقات و (Fedora 3)، وضعنا المخزن خلف (API) تسمى (DOR Services Application) أو (DSA). وتتيح إنشاء كائنات (Cocina) واسترجاعها وتحديثها، إلى جانب وظائف مثل بدء الإدخال وإدارة الإصدارات وتسجيل الأحداث. وربما كان أكبر جهد في الانتقال إعادة كتابة التطبيقات أو هيكلتها لاستخدام (Cocina) داخليًا بدل (Fedora)، والتعامل مع (DSA). وتم ذلك تدريجيًا بإضافة أجزاء النموذج ووظائف الواجهة بحسب حاجة كل تطبيق، كما نشرح في الاستراتيجية الثامنة.

قبل الانتقال، نفذت (DSA) تحويلًا آنيًا ثنائي الاتجاه بين (Cocina) و (Fedora). فعندما يطلب تطبيق كائن (Cocina)، تسترجع (DSA) كائن (Fedora) وتحوله وتعيده. وعند طلب إنشاء كائن، تنشئ كائن (Fedora) جديدًا وتحديثه من (Cocina) وتحفظه. وعند التحديث، تسترجع الكائن القائم وتحفظه وتحفظه.

ونظرًا لتعقيد النموذجين والتنوع الكبير للكائنات الموجودة، كانت شيفرة التحويل معقدة للغاية؛ وسنعرض وسائل إدارتها لاحقًا.

بعد فك ارتباط جميع التطبيقات، أصبحت (DSA) التطبيق الوحيد الذي يتعامل مباشرة مع (Fedora)؛ أي إن الارتباط الضيق انتقل إليها. وفي مرحلة أخيرة جمعنا كل شيفرة التعامل مع (Fedora) داخل صنف خدمة متخصص اسمه (Cocina Object Store). أما خدمات (DSA) الأخرى، فتعاملت مع (Cocina) فقط؛ والتحويل يجري داخل هذا المخزن. وأتاح ذلك الانتقال المتدرج الموصوف في الاستراتيجية السابعة.

الاستراتيجية الثالثة: فصل المسؤوليات:

لتقليل تعقيد النموذج وتيسير الانتقال، فصلنا الخدمات ذات المسؤوليات المختلفة في تطبيقات مستقلة، وخصصنا مخازن لبيانات خدمات كل تطبيق بدل جمع كل شيء في تدفقات (Fedora) المركزية.

مثلًا، كانت البيانات الوصفية التقنية تُولد بواسطة (JHOVE) وتُخزن بصيغة (XML) في تدفق مستقل داخل (Fedora). وكانت لهذه الطريقة نقائص: الارتباط الضيق بـ (Fedora)، وتزايد عدم الاتساق مع تغير صياغة مخرجات (JHOVE)، وغياب وسيلة فعالة للاستعلام.

استبدلنا ذلك بخدمة تقنية مستقلة، هي تطبيق (Rails) ذو (API) ومخزن علائقي لتخزين البيانات والاستعلام عنها. وأغنى هذا أيضًا عن تضمين البيانات التقنية، الطويلة والمعقدة أصلًا، في نموذج (Cocina).

وطبقنا الفصل على خدمات أخرى مثل الأحداث وسير العمل، كانت مرتبطة بـ (Fedora) داخل تطبيق أحادي كبير. ومن فوائده إمكان تغيير خدمة أو تقنياتها بصورة مستقلة، ما دامت واجهتها البرمجية محفوظة.

الاستراتيجية الرابعة: اختبار التحويل بيانات كائنات حقيقية مخزنة مؤقتًا:

تطور التحويل ثنائي الاتجاه تدريجيًا. واعتمدنا أولًا على اختبارات الوحدات لضمان الدقة، لكن سرعان ما ظهرت مشكلات. فرغم الاختبارات وبيئة التجهيز، جلب كل إصدار جديد موجة عيوب، بعضها يعطل تطبيقًا كاملًا وبعضها يؤثر في كائن منفرد؛ وقد تكون أخطاء برمجية غير معالجة أو تحويلات غير صحيحة.

كان التعقيد سببًا جزئيًا، لكن السبب الأساسي تنوع البيانات؛ فلم نكن نعرف كل الحالات التي يجب أن يغطيها التحويل أو الاختبار. لذلك اخترنا كل تغيير على مجموعة كبيرة من البيانات الحقيقية، تصل عادة إلى بيانات 100 ألف كائن لكل تغيير برمجي.

واجهتنا عقبة: استرجاع هذا الحجم مرارًا من (Fedora) بطيء، واستخدام بيانات حقيقية يضغط نسخة الإنتاج. فأنشأنا ذاكرة تخزين مؤقتة للقراءة فقط، تحفظ تدفقات كل كائن داخل (ZIP). وتحافظ عملية اسمها (cache-o-matic) على تحديثها، باستعلام (Solr) عن الكائنات المتغيرة حديثًا وإعادة توليد ملفاتها. وكانت أسرع بكثير، جزئيًا لاستخدام تخزين سريع، وأقل أثرًا على الإنتاج.

للتحقق، يشغل المطور اختباري «قبل» و«بعد» ليتأكد من أن التغيير يحسن التحويل ذهابًا وإيابًا، أو لا يجعله أسوأ، ولا يُدخل خطأ غير معالج. ولكل كائن تُنفذ ست خطوات: إنشاء كائن (Fedora) من النسخة المؤقتة؛ تحويله إلى (Cocina)؛ تمرير (Cocina) عبر عملية إنشاء الكائن لتوليد (Fedora) جديد؛ تطبيع تدفقات الأصل ومقارنتها بالجديد؛ تحويل (Fedora) الجديد إلى (Cocina) جديد؛ ثم مقارنة كائني (Cocina).

ويستحق التطبيق شرحًا. فهو يعالج التنوع؛ إذ كُتبت التسميات في بيانات المحتوى القديمة بهذه الصورة:

```
<attr type="label">My label</attr>
```

ثم تغيرت إلى:

```
<label>My label</label>
```

يفهم التحويل من (Fedora) الصيغتين، لكن التحويل العكسي يولد دائمًا `</label>`. لذا وجب تطبيع الصيغة الأولى إلى الثانية قبل مقارنة تدفقي المحتوى.

طورنا أدوات موسعة للاختبار ذهابًا وإيابًا، تشمل التنفيذ المتوازي لتقليل الزمن، وإحصاءات ملخصة، وتسجيل النتائج لكل كائن مع فروق التدفقات و (JSON) الخاص بـ (Cocina).

ومع تقدم العمل استخدمنا الاختبار لاكتشاف أجزاء نموذج (Fedora) غير المحولة، والجوانب التي تحتاج إلى تحسين التحويل أو التطبيع، والبيانات المحتاجة للإصلاح. ووسعنا العينات تدريجيًا حتى شملت جميع الكائنات، وحللنا النتائج وتتبعنا الإحصاءات مع الزمن لقياس التقدم.

نجحت الطريقة في تقليل أثر تطوير التحويل على المستخدمين، وقياس مدى اكتماله. وكانت الإحصاءات مفيدة خاصة في إبلاغ أصحاب المصلحة بتقدم المشروع.

الاستراتيجية الخامسة: استخدام التقارير لفهم البيانات:

أتاحت النسخة المؤقتة تطوير أدوات تقارير تجيب بسهولة عن أسئلة حاسمة للتحويل والانتقال: أي الكائنات تفتقد نوع (MIME) لملفاتها؟ وأيها يفتقد أبعاد الصور؟ وما القيم المستخدمة لأنواع الملخصات في البيانات الوصفية؟ وأيها يستخدم تواريخ (EDTF) وما قيمها؟ وأيها يفتقد بيان استخدام ضمن حقوق الإتاحة؟

الاستراتيجية السادسة: قوالب لاختبارات الوحدات:

اقتضى تعقيد التحويل عددًا هائلًا من الاختبارات. وباستخدام ميزة «الأمثلة المشتركة» في (RSpec)، أصبح تعريف الاختبار يتكون من مقطع (Cocina JSON) و (XML) المقابل في تدفق (Fedora). ثم تنفذ آليًا اختبارات عديدة على كل زوج: التحويل في الاتجاهين، والتحويل من (Cocina) الجديد إلى (Fedora) مع التطبيع، ومقارنة (Cocina) في البداية والنهاية، ومقارنة (Fedora) كذلك.

قلل ذلك الشيفرة النمطية المتكررة مع زيادة شمول الاختبارات. وكان فعالًا خاصة للبيانات الوصفية، إذ مكن أركاديا من كتابة اختبارات مباشرة، تبقى معطلة بالتعليق حتى ينفذ المطور التحويل. وكانت هذه أفضل كثيرًا من الاحتفاظ بأمثلة منفصلة تتطلب المزامنة المستمرة مع اختبارات المطورين.

الاستراتيجية السابعة: الانتقال المتدرج:

ظهر مبكرًا شرط عمل مهم: لا «يوم انتقال حاسم» يُوقف النظام؛ بل يجب أن يبقى (SDR) متاحًا دائمًا.

لتحقيق ذلك استخدمنا انتقالًا متدرجًا. كانت جميع عمليات المخزن خلف (Cocina Object Store) داخل (DSA)، ومخزنها أولًا (Fedora). ثم أضفنا (Postgres) ليعمل الاثنان معًا. وتُخزن كائنات (Cocina) في (Postgres) بصيغة (JSON) وفق نموذجها، دون حاجة للتحويل بين نموذجين.

أثناء الاسترجاع، إذا وُجد الكائن محفوظًا في (Postgres) استُرجع منه، وإلا استُرجع (Fedora) وحُول إلى (Cocina)؛ أي أولوية (Postgres) مع الرجوع إلى (Fedora) عند الحاجة.

وعند إنشاء كائن، يُحفظ في (Postgres)، ثم يُحول إلى (Fedora) ويُحفظ فيه أيضًا؛ أي الكتابة إلى المخزين.

وعند تحديثه، يُستبدل الكائن في (Postgres)، ثم يُسترجع كائن (Fedora) القائم ويُعدل وفق (Cocina) ويُحفظ؛ أي تحديث المخزين معًا.

وشملت العملية سكرتيرًا يمر على كل كائن في (Fedora)، ويحوّل بياناته ويحفظها في (Postgres) إن لم تكن موجودة هناك، وضمن ذلك انتقال الكائنات قليلة الاستخدام أيضًا. واستُخدم التنفيذ المتوازي لرفع الأداء.

بعد اكتمال حفظ كل كائن في صورة (Cocina)، أزلنا (Fedora) وبقية شيفرته وشيفرة التحويل وإطار التخزين المؤقت والتقارير والاختبار كله. وأصبح كل الاسترجاع والحفظ من (Postgres) وإليه.

وتسمح الطريقة بانتقال ممتد دون أثر على المستخدم أو توقف. وإذا ظهرت مشكلة في أي وقت، يمكن التراجع بحذف بيانات (Cocina) من (Postgres)، فيرجع المخزن تلقائيًا إلى (Fedora).

الاستراتيجية الثامنة: دمج الانتقال في عمل المشروعات:

احتاجت الاستراتيجيات السابقة إلى وقت وعمل كبيرين، معظم أثرهما غير مرئي للمستخدم. وفي الوقت نفسه، لم تختف مشروعاتنا القائمة؛ بل احتاجت إلى متابعة مستمرة، ولا سيما الممولة خارجيًا ذات المواعيد الصارمة. لذلك احتجنا، إلى جانب التقنية، إلى خطة لتمويل عمل الانتقال وتنظيمه داخل العمل الجاري وبالتوازي معه.

لا يحدد الفريق أولوياته وخطته منفردًا أو بمعزل عن غيره. وكان أساس الاستراتيجية فهم مشترك وواضح مع قيادة القسم، وأفضت مناقشات إلى مبادئ لخطة سمينها «تطور (SDR)»، تسمح بتقدم تدريجي على سنوات؛ تقديم قيمة ظاهرة للمستخدم خلال التطور وربطها بمتطلباته؛ وتطوير المكونات بدل استبدالها، دون أيام إيقاف حاسمة، وفصل مكوناتها تدريجيًا لتصبح أكثر اعتمادًا على الواجهات والخدمات، والتركيز على تحسينات محددة بدل تغيير المنظومة كلها دفعة واحدة.

وعلى أساس هذه المبادئ والفهم المشترك، ربطنا الفجوات المعروفة في محفظة العمل، الأرجح أن تنال الأولوية في الخطة المقبلة، باحتياجات الانتقال.

مثلًا، تأخر استبدال تطبيق الإيداع الذاتي طويلًا وكان مطلبًا ملحًا للمستخدمين. ورأينا فيه فرصة لدفع الانتقال، بوضع (API) وسيطة بين التطبيق و (Fedora)، وفق الاستراتيجية الثانية، أو نقل تعريفات سير العمل المرتبطة به إلى خدمة مستقلة، وفق الثالثة.

وكان على الفريق ومديره التفاوض مع مالكي المنتجات بشأن مقدار هذه الاحتياجات الممكن إدراجه في دورات العمل. ولأن القيادة وأصحاب المصلحة اقتصروا مسبقًا، فهم المالكون قيمة الجمع بين الخصائص التي يريدونها وأعمال الانتقال. ويصف رئيس القسم توم كرامر الهدف بأنه تطوير (SDR) بتحديث مكوناته الرئيسية منهجيًا، وصولًا على المدى الطويل إلى بيئة جديدة أحدث وأكثر متانة وغنى بالوظائف.

ولولا استثمار دورات العمل المرتبطة بـ (SDR) أو المجاورة له لتمويل الانتقال، لتعطل تغيير البنية الخلفية أو تأخر بشدة. بهذه الطريقة تقدم الانتقال بينما استمرت أولوية المشروعات والخصائص الجديدة.

خاتمة:

بدأ الانتقال في مايو 2022. وكانت المشكلة الوحيدة أثناء التنفيذ امتلاء مساحة سجلات خادم (Fedora) أسرع من تدويرها. فوسعنا المساحة، لكن ذلك تطلب إيقاف (Fedora) مؤقتًا وإعادة تشغيله. وفي تحدٍ أخير رفض العودة! واستعان الفريق بخبرته وسجلات (Slack) القديمة والمعرفة المتراكمة على (Google) لحل المشكلة. وفي النهاية انتقل أكثر من أربعة ملايين كائن في أقل من أسبوع، وأصلحت الكائنات الإشكالية، ودقق الانتقال للتأكد من وجود كل كائن (Fedora) في (Postgres)، ثم أُزيل (Fedora).

جلب الانتقال تحسينات في الأداء والاستقرار، وإعادة كتابة أو تحديث بعض التطبيقات المهمة طويلًا، لكنه تطلب صبرًا كبيرًا من المستخدمين. ويتطلع الفريق إلى مستقبل لا تقيد فيه حدود (Fedora) إمكانيات (SDR)، وإلى العمل مع المستخدمين لخدمة رسالة الجامعة العلمية.

المراجع:

مواصفة الإتاحة الدولية للصور: <https://iiif.io>

نظام الحفظ في ستانفورد: <https://journal.code4lib.org/articles/8482>; <https://2018.code4lib.org/talks/airing-our-dirty-laundry-digital-preservation-gaps-and-how-were-fixing-them>

السياق التاريخي لمستودع ستانفورد الرقمي: <https://www.dlib.org/dlib/september10/cramer09cramer.html>

مخطط نموذج (Cocina) بصيغة (YAML): <https://github.com/sul-dlss/cocina-models/blob/main/openapi.yml>; وعرضه: <https://sul-dlss.github.io/cocina-models>

مكتبة نموذج (Cocina): <https://github.com/sul-dlss/cocina-models>

قواعد الوصف في النموذج: https://github.com/sul-dlss/cocina-models/blob/main/description_types.yml