SciELO Balaio Documentation

Release v0.01

SciELO Team

September 08, 2014

Contents

Balaio é uma ferramenta automatizada para apoiar o processo de ingresso de artigos em coleções SciELO.

Esta aplicação foi projetada para operar no contexto da nova plataforma tecnológica de gestão de dados, e suas funcionalidades serão expostas aos usuários por meio do SciELO Manager.

CHAPTER 1

Funcionalidades

- Monitoramento do sistema de arquivos para novos depósitos.
- **Validação automática de diversos aspectos do pacote.**
  - Metadados.
- Criação de tickets com pendências associadas ao artigo em processo de submissão.

CHAPTER 2

Funcionalidades pendentes

- Validação automática de arquivos complementares (imagens, pdf etc).
- Integração do sistema de notificações com o SciELO Manager (painel de controle do usuário).

CHAPTER 3

Guias de desenvolvimento

Technical guides and artifacts explaining software components and design decisions.

3.1 Balaio Gateway HTTP API

Attention: this document is currently under development

RESTful API which provides services related to the XML packages and their attempts.

Current version: API v1

3.1.1 Available endpoints

Packages API

List all packages

Request:

```
GET /api/v1/packages/
```

Parameters:

–

Optional Parameters:

callback

String of the callback identifier to be returned when using JSONP.

journal_pissn

String of the **journal_pissn** to be used as a filter param.

journal_eissn

String of the **journal_eissn** to be used as a filter param.

issue_volume

String of the **issue volume** to be used as a filter param.

issue_number

String of the **issue number** to be used as a filter param.

issue_suppl_volume

String of the **issue volume supplement** to be used as a filter param.

issue_suppl_number

String of the **issue number supplement** to be used as a filter param.

issue_year

String of the **issue year** to be used as a filter param.

article_title

String of the **artile title** to be used as a filter param.

journal_title

String of the **journal title** to be used as a filter param.

id

Integer of the **id** to be used as a filter param.

aid

String of the **article-id** to be used as a filter param.

Response:


```
{
  "meta": {
    "limit": 20,
    "next": "/api/v1/packages/?limit=20&offset=40",
    "offset": 20,
    "previous": "/api/v1/packages/?limit=20&offset=0",
    "total": 100
  },
  objects: [
      {
        article_title: "Article Title",
        issue_year: 2010,
        journal_title: "Journal Title",
        journal_pissn: "1234-1234",
        journal_eissn: "1234-1234",
        issue_suppl_number: "1",
        attempts: [
          "/api/v1/attempts/1/"
        ],
        issue_suppl_volume: "1",
        issue_volume: "1",
        aid: "yp4529bp8g",
        resource_uri: "/api/v1/packages/3/",
        id: 3,
        issue_number: "1"
      },
      {
        article_title: "Article Title",
        issue_year: 1998,
        journal_title: "Journal Title",
        journal_pissn: "2349-2309",
```

```
        journal_eissn: "9832-1987",
        issue_suppl_number: "8",
        attempts: [
          "/api/v1/attempts/1/"
        ],
        issue_suppl_volume: "7",
        issue_volume: "11",
        aid: "jp4599bq8g",
        resource_uri: "/api/v1/packages/4/",
        id: 4,
        issue_number: "3"
      }
    ]
  }
```

Get a single package

Request:

```
GET /api/v1/packages/:id/
```

Parameters:

–

Optional Parameters:

callback

String of the callback identifier to be returned when using JSONP.

Response:

```
{
  article_title: "Article Title",
  issue_year: 2013,
  journal_title: "Journal Title",
  journal_pissn: "1234-1234",
  journal_eissn: "1234-1234",
  issue_suppl_number: "1",
  attempts: [
      "/api/v1/attempts/1/"
  ],
  issue_suppl_volume: "1",
  issue_volume: "1",
  aid: "yp4529bp8g",
  resource_uri: "/api/v1/packages/3/",
  id: 3,
  issue_number: "1"
}
```

Attempts API

List all attempts

Request:

```
GET /api/v1/attempts/
```

Parameters:

–

Optional Parameters:

callback

String of the callback identifier to be returned when using JSONP.

articlepkg_id

Integer of the **article package ID** to be used as a filter param.

started_at

DateTime of the **started_at** to be used as a filter param.

finished_at

DateTime of the **finished_at** to be used as a filter param.

is_valid

Boolean of the **is valid** to be used as a filter param.

package_checksum

String of the **package_checksum** to be used as a filter param.

id

integer of the **id** to be used as a filter param.

articlepkg_id

articlepkg_id of the **articlepkg_id** to be used as a filter param.

Response:


```
{
  "meta": {
    "limit": 20,
    "next": "/api/v1/attempts/?limit=20&offset=40",
    "offset": 20,
    "previous": "/api/v1/attempts/?limit=20&offset=0",
    "total": 100
  },
  "objects": [
    {
      "articlepkg_id": 1,
      "id": 1,
      "collection_uri": "/api/v1/collections/1/",
      "finished_at": "2012-07-24T21:59:23.909404",
      "filepath": "/files/ajfajfajfa",
      "is_valid": true,
      "package_checksum": "12345678901234567890123456789012",
      "resource_uri": "/api/v1/attempts/1/",
      "started_at": "2012-07-24T21:53:23.909404",
      "checkin": {
        "finished_at": "2012-07-24T21:53:23.909404",
        "started_at": "2012-07-24T21:53:23.909404",
        "notices": [
```

```
            {
              "label": "Checkin",
              "message": "",
              "status": "ok",
              "date": "2012-07-24T21:53:23.909404",
            },
          ],
        },
        "validations": {
          "finished_at": "2012-07-24T21:53:23.909404",
          "started_at": "2012-07-24T21:53:23.909404",
          "notices": [
            {
              "label": "journal",
              "message": "",
              "status": "ok",
              "date": "2012-07-24T21:53:23.909404",
            },
            {
              "label": "journal",
              "message": "",
              "status": "error",
              "date": "2012-07-24T21:53:23.909404",
            },
            {
              "label": "front",
              "message": "",
              "status": "warning",
              "date": "2012-07-24T21:53:23.909404",
            },
            {
              "label": "front",
              "message": "",
              "status": "ok",
              "date": "2012-07-24T21:53:23.909404",
            },
            {
              "label": "references",
              "message": "",
              "status": "ok",
              "date": "2012-07-24T21:53:23.909404",
            },
            {
              "label": "references",
              "message": "",
              "status": "ok",
              "date": "2012-07-24T21:53:23.909404",
            },
          ],
        }
      }
    ]
}
```


Get a single attempt

Request:

```
GET /api/v1/attempts/:id/
```

Parameters:

–

Optional Parameters:

callback

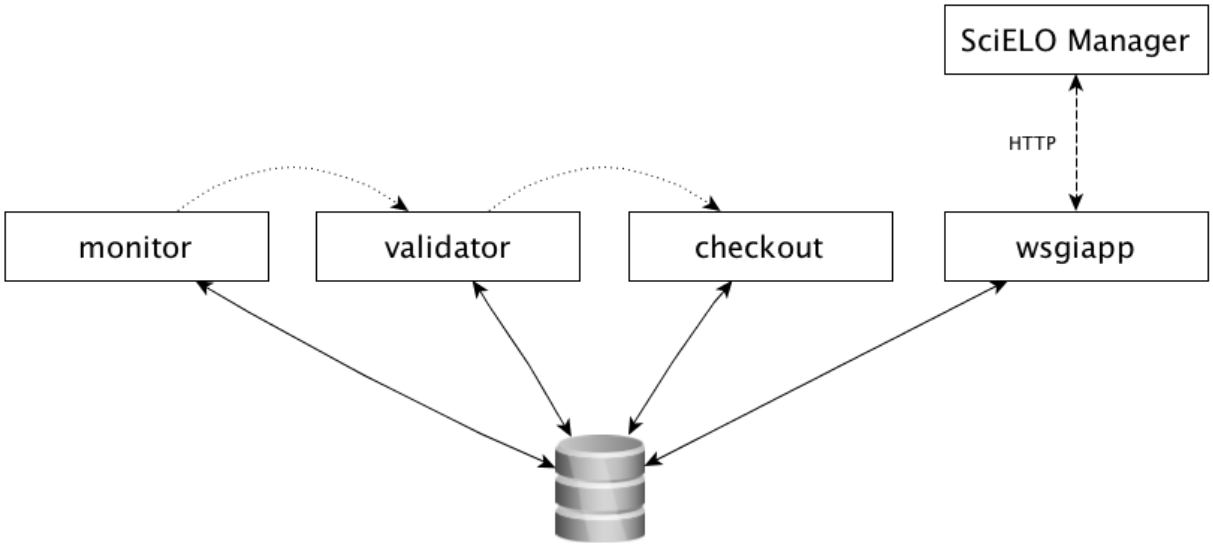
String of the callback identifier to be returned when using JSONP.

Response:


```
{
  "articlepkg_id": 1,
  "id": 1,
  "collection_uri": "/api/v1/collections/1/",
  "finished_at": "2012-07-24T21:59:23.909404",
  "filepath": "/files/ajfajfajfa",
  "is_valid": true,
  "package_checksum": "12345678901234567890123456789012",
  "resource_uri": "/api/v1/attempts/1/",
  "started_at": "2012-07-24T21:53:23.909404",
  "checkin": {
    "finished_at": "2012-07-24T21:53:23.909404",
    "started_at": "2012-07-24T21:53:23.909404",
    "notices": [
      {
        "label": "Checkin",
        "message": "",
        "status": "ok",
        "date": "2012-07-24T21:53:23.909404",
      },
    ],
  },
  "validations": {
    "finished_at": "2012-07-24T21:53:23.909404",
    "started_at": "2012-07-24T21:53:23.909404",
    "notices": [
      {
        "label": "journal",
        "message": "",
        "status": "ok",
        "date": "2012-07-24T21:53:23.909404",
      },
      {
        "label": "journal",
        "message": "",
        "status": "error",
        "date": "2012-07-24T21:53:23.909404",
      },
      {
        "label": "front",
        "message": "",
        "status": "warning",
        "date": "2012-07-24T21:53:23.909404",
      },
      {
        "label": "front",
```

```
        "message": "",
        "status": "ok",
        "date": "2012-07-24T21:53:23.909404",
      },
      {
        "label": "references",
        "message": "",
        "status": "ok",
        "date": "2012-07-24T21:53:23.909404",
      },
      {
        "label": "references",
        "message": "",
        "status": "ok",
        "date": "2012-07-24T21:53:23.909404",
      },
    ],
  }
}
```

3.2 Design



3.2.1 balaio

Utilitário de linha de comando para apoiar a instalação e operação da aplicação.

Exemplo:

```
$ python manage.py
usage: manage.py [-h] --config CONFIGFILE
                 [--alembic-config ALEMBIC_CONFIGFILE]
                 {syncdb,shell}
```

Os comandos disponíveis até o momento são:

syncdb

Cria a estrutura de banco de dados da aplicação. Esse comando só deve ser executado no momento da instalação. Para a aplicação das migrações de schema de dados, o *alembic* deve ser utilizado diretamente.

shell

Acessa o console interativo do python com o contexto pré-configurado para facilitar o acesso ao banco de dados, por exemplo:

```
>>> from lib import models
>>> session = Session()
>>> session.query(models.Attempt).all()
```

3.2.2 monitor

Monitora eventos no sistema de arquivos em busca de pacotes SPS e rSPS que deverão ser incorporados, em 1 ou N coleções, e é dividido em:

- **monitor:** Monitora eventos no sistema de arquivos por meio da lib *inotify* (linux), produz uma instância de `models.Attempt` representando a tentativa de ingresso de um pacote no sistema, e encaminha essa instância para o módulo `validator`, onde será realizada sua validação.
- **checkin** Inspeciona o conteúdo do pacote em busca de extrair sua identidade. A identidade de um pacote é composta pelo seu *checksum*, e metadados do artigo presentes no arquivo *xml* que deve estar presente.

3.2.3 validator

Submete a instância de `models.Attempt` a um *pipeline* de validação, onde cada *pipe* (segmento do *pipeline*) é responsável pela validação de um aspecto do pacote.

3.2.4 loggingserver

Recebe e consolida os *logs* dos módulos `monitor`, `validator` e `balaio`.

3.2.5 wsgiapp

Disponibiliza uma interface HTTP para acesso e manipulação dos dados.

3.2.6 uploader

Abstrai os backends de persistência de arquivos estáticos, por meio da superclasse `uploader.BlobBackend`.

Classes concretas de `uploader.BlobBackend` devem implementar os métodos *connect(self)* e *cleanup(self)*. Ambos os métodos não devem receber argumentos, que devem ser passados na inicialização da instância quando necessário.

uploader.StaticScieloBackend

Classe concreta para a persistência de arquivos no servidores de ativos estáticos do SciELO. A comunicação é realizada via protocolo SFTP, e depende das credenciais de um usuário apto a realizar a ação.

Exemplo:

```
>>> from uploader import StaticScieloBackend
>>> with StaticScieloBackend(u'some.user', u'some.pass', u'/base/path/') as backend:
...     backend.send(open(u'article.pdf', 'rb'), u'/abc/article.pdf')
...
u'http://static.scielo.org/abc/article.pdf'
```

Note: É importante que diferentes aplicações, que manipulam *base paths* distintos, possuam usuários com as devidas restrições de acesso, para evitar perda de dados acidental.

3.3 Objetos de domínio do negócio

3.3.1 lib.models.ArticlePkg

Representa um artigo em processo de ingresso no sistema. Uma instância de `lib.models.ArticlePkg` pode conter *n* `lib.models.Attempt` associadas, dependendo de quantas iterações foram necessárias até que um pacote fosse validado durante o processo. Instâncias de `lib.models.ArticlePkg` contém apenas metadados do artigo objeto de análise.

3.3.2 lib.models.Attempt

Representa a submissão de um pacote para validação. É marcada como válida ou inválida (*is_valid*) para sinalizar tentativas que possuem condições de serem submetidas ao processo de validação.

Uma instância de `lib.models.Attempt` deve ser criada para cada pacote de artigos depositado, exceto quando se tratar de um pacote duplicado.

3.4 Status de notificação

Durante a operação, o *balaio* dispara uma série de notificações para o SciELO Manager, para noticiar a chegada de um novo pacote, erros ou resultados de validação automática.

Uma notificação deve sempre acompanhar um instância enumerável de `lib.models.Status`, que são divididas em:

3.4.1 Status de validação

Qualificam as notificações sobre validações automáticas, que podem ser:

- *lib.models.Status.ok*: A tag referida está presente e/ou seu conteúdo está correto.
- *lib.models.Status.warning*: Existem pequenos problemas que é bom saber, mas podemos seguir em frente.
- *lib.models.Status.error*: Alguma tag/dado essencial está faltando, ou o dado está expresso de maneira inválida.

Status de validação podem ocupar valores de 1 a 49 (inclusive), que uma vez atribuídos não podem ser alterados

3.4.2 Status de serviço

Forma limitada de comunicação inter-aplicações. Atualmente é usado apenas para delimitar conjuntos de mensagens correlatas.

- *lib.models.Status.SERV_BEGIN*: Sinaliza o início de um bloco de mensagens correlatas.
- *lib.models.Status.SERV_END*: Sinaliza o término de um bloco de mensagens correlatas.

Status de serviço podem ocupar valores de 50 a 99 (inclusive), que uma vez atribuídos não podem ser alterados

3.5 Guia de Instalação

Guia de instalação do balaio e suas dependencias.

3.5.1 Requisitos

- python2.7
- git
- circus
- chaussette
- postgres
- Linux
- virtualenv (opcional)

3.5.2 Instalação

Clonar o repositório: https://github.com/scieloorg/balaio

Instalar bibliotecas python:

```
pip install -r requirements.txt
```

Note: É possível que durante a instalação do **psycopg2** ocorra um erro relacionado ao **pg_config**. Nesse caso certifique-se de ter instalado os headers do **postgresql**.

Note: É possível que durante a instalação do **psycopg2** ocorra um erro relacionado ao **python.h**. Nesse caso certifique-se de ter instalado os headers do **python**.

Note: É possível que durante a instalação do **lxml** ocorra um erro relacionado ao **libxml2**. Nesse caso certifique-se de ter instalado os headers do **libxml**.

Note: É possível que durante a instalação do **lxml** ocorra um erro relacionado ao **libxslt**. Nesse caso certifique-se de ter instalado os headers do **libxslt**.

Instalar aplicação:

```
python setup.py [develop|install]
```

3.5.3 Configurar aplicação

alembic.ini:

```
Configurar regras de conexão com banco de dados

sqlalchemy.url = driver://user:pass@localhost/dbname
```

config.ini:

```
#. Configurar regras de conexão com banco de dados

    sqlalchemy.url = driver://user:pass@localhost/dbname

#. Configurar conta de usuário da api do SciELO Manager
```

circus.ini:

```
#. Configurar variáveis de ambiente

    [env]
    ;---- absolute paths only
    virtualenv_path =
    BALAIO_SETTINGS_FILE =
    BALAIO_ALEMBIC_SETTINGS_FILE =
    app_working_dir =
```

Sincronizar banco de dados:

```
python balaio/balaio.py -c conf/config.ini --syncdb
```

3.5.4 Testar aplicação

Criar banco de dados de testes no postgresql. O banco deve se chamar **app_balaio_tests**

Executar testes:

```
python setup.py nosetests
```