



C3SL - Centro de Computação Científica e Software Livre

Entre as linhas do SimCAQ

Documentação sobre o projeto SimCAQ

Curitiba

29 de janeiro de 2021

Sumário

1	Membros	1
2	Introdução	2
2.1	Motivações	2
3	Licença	3
4	Banco de Dados	4
4.1	Biblioteca	5
4.1.1	Instalação e Configuração	6
4.1.2	Árvore de Arquivos	7
4.1.3	Funcionalidades	8
4.1.4	O script do script	12
4.2	Atualização de Bases	12
5	Indicadores e suas tabelas	14
5.1	LDE	14
5.2	SimCAQ	15
6	APIs	16
6.1	API SimCAQ	16
6.1.1	Instalação	16
6.1.2	Arquitetura	16
6.2	API CDN	20
6.2.1	Instalação	20
6.2.2	Arquitetura	20
6.3	API LDE	21
6.3.1	Instalação	21
7	LDE - Dados Educacionais	22
7.1	Instalação	22
7.2	Painel do Administrador	22

7.2.1	Funcionalidades	22
7.2.2	Instalação	23
7.2.3	Principais Problemas	23
8	SimCAQ - Simulador Custo Aluno Qualidade	24
9	Ambientes	25
9.1	Introdução	25
9.2	Breve Explicação dos comandos	25
9.3	Atualização	27
9.3.1	API SimCAQ (simcaq-node)	27
9.3.2	Front End SimCAQ (simcaq-ui)	27
9.3.3	API LDE (LDE API)	29
9.3.4	Front End LDE (labdados-ui)	30
9.3.5	Documentação (api-doc)	30
10	Outras ferramentas	31
10.0.1	Monitoramento	31

1 **Membros**

Aqui jaz o nome dos membros do C3SL que fizeram/fazem parte do projeto Sim-CAQ: Fernando Claudécir Erd, Gabriel Ruschel Castanhel, Glenda Train, Gustavo Soviersovski, Hamer Iboshi, Lucas Gabriel Lima, Marina Hoshiba Pimentel, Rudolf Copi Eckelberg, Victor Picussa, Vytor Calixto. O coordenador do projeto é o professor Dr. Marcos Didonet Del Fabro.

2 Introdução

O projeto SimCAQ[2] - Simulador de Custo-Aluno-Qualidade - visa a disponibilização gratuita e via internet de um simulador para dar suporte ao processo de elaboração/adequação e monitoramento/avaliação dos Planos Estaduais e Municipais de Educação visando a articulação das metas educacionais locais com as metas do Plano Nacional de Educação (PNE) - Lei 13.005 de 25/06/2014 - e a previsão do montante de recursos financeiros necessário para a oferta da educação básica em condição de qualidade no período dos planos.

É uma ação da Secretaria de Articulação com os Sistemas de Ensino do Ministério da Educação (SASE/MEC) executado em parceria firmada com a Universidade Federal do Paraná (UFPR), por meio do Departamento de Planejamento e Administração Escolar (DEPLAE) do Setor de Educação formalizado pelo Convênio SICONV nº 820692/2015.

2.1 Motivações

A criação desse arquivo deu-se pela necessidade de manter-se documentado todo o processo de instalação das ferramentas, utilização e atualização/manutenção. Com a rotatividade média de 2 anos de desenvolvedores, manter os estudos e informações sobre o projeto em um documento pode facilitar o trabalho de futuros desenvolvedores do mesmo. Dessa forma, o objetivo é apresentar o que foi dito anteriormente de forma detalhada para não deixar dúvidas sobre os processos feitos.

3 Licença

Os repositórios do projeto SimCAQ utilizam a licença GNU GPLv3. [4], em termos gerais, a GPL baseia-se em 4 liberdades:

- A liberdade de executar o programa, para qualquer propósito (liberdade nº 0).
- A liberdade de estudar como o programa funciona e adaptá-lo às suas necessidades (liberdade nº 1). O acesso ao código-fonte é um pré-requisito para esta liberdade.
- A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo (liberdade nº 2).
- A liberdade de aperfeiçoar o programa e liberar os seus aperfeiçoamentos, de modo que toda a comunidade beneficie deles (liberdade nº 3). O acesso ao código-fonte é um pré-requisito para esta liberdade.

Com a garantia destas liberdades, a GPL permite que os programas sejam distribuídos e reaproveitados, mantendo, porém, os direitos do autor por forma a não permitir que essa informação seja usada de uma maneira que limite as liberdades originais. A licença não permite, por exemplo, que o código seja apoderado por outra pessoa, ou que sejam impostos sobre ele restrições que impeçam que seja distribuído da mesma maneira que foi adquirido.

A GPL está redigida em inglês e atualmente nenhuma tradução é aceita como válida pela Free Software Foundation, com o argumento de que há o risco de introdução de erros de tradução que poderiam deturpar o sentido da licença. Deste modo, qualquer tradução da GPL é não-oficial e meramente informativa, mantendo-se a obrigatoriedade de distribuir o texto oficial em inglês com os programas.

4 Banco de Dados

O sistema de banco de dados do SimCAQ é composto por uma divisão de dois bancos. Um banco para a produção, presente na máquina chamada *c3sldatabase* - *dados_educacionais* -, e um banco para homologação, presente na máquina *sim-caqdb3* - *simcaq_dev4*. O sistema utiliza-se das ferramentas MongoDB e MonetDB para a gerencia do banco de dados.

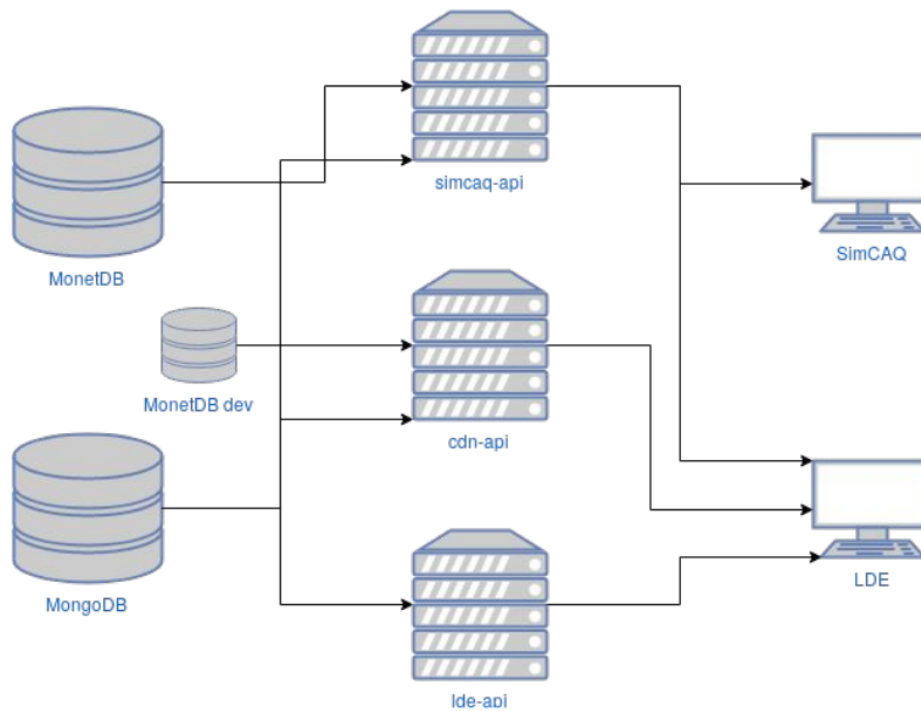


Figura 1: Diagrama da arquitetura geral.

A escolha do MonetDB[6] foi consequência de testes e pesquisas sobre outras ferramentas de gerenciamento de banco de dados. A imagem [2] apresenta as ferramentas testadas e a tabela [3] de comparação entre as duas melhores opções.



Figura 2: Ferramentas de banco de dados.

4.1 Biblioteca

F A biblioteca é um programa feito na linguagem Python com o objetivo de facilitar a criação de tabelas e inserção de dados com o MonetDB e o SQLAlchemy[9]. Para utilizar, é necessário a instalação do MonetDB acessando a página da aplicação e procurando pela seção de download.

Após a instalação do MonetDB, é necessário fazer a criação da *farm* e do banco de dados[5]. Para isso, executa-se os comandos ***monetdbd create [nome farm]*** - para possibilitar a criação do servidor de banco de dados do MonetDB - e ***monetdbd start [nome farm]*** - para inicializar a *farm* criada. O comando ***monetdb create [nome banco de dados]*** - para criar o banco de dados - e, por fim, ***monetdb release [nome banco de dados]*** - para retirar o banco de dados do estado de manutenção. Utilizando o comando ***mclient -d [nome bando de dados]*** é possível verificar se tudo ocorreu corretamente.

- **Requisitos:** pymonetdb, SQLAlchemy, sqlalchemy-monetdb, numpy[7], pandas[8], xlrd, XlsxWriter.

Query	Tempo de execução em ms		Desempenho do MonetDB em relação ao PostgreSQL
	PostgreSQL	MonetDB	
1	14583	741	1968.02%
3	2056	2067	99.47%
4	942	234	402.56%
5	1406	1278	110.02%
6	1665	179	930,17%
10	3446	908	379.52%
12	2567	330	777.88%
14	1677	86	1950.00%
16	2047	204	1003.43%
19	2143	136	1575.74%

Figura 3: Comparação TCP-H.

4.1.1 Instalação e Configuração

De início, é preciso fazer uma clonagem do projeto *database*[1] presente no repositório do GitLab do C3SL com o comando ***git clone [url]***.

Após clonar o repositório, a pasta *database* conterà os arquivos da biblioteca e, para utilizar a versão mais atualizada deve-se mudar o repositório local para a *branch development*. Para tanto, acessa utilizando o seguinte comando ***git checkout development***. Finalmente, é preciso atualizar o submódulo contido na pasta *mapping_protocols*[3] terminando a organização das pastas. Isso é feito entrando na pasta e digitando os comandos ***git submodule init*** e ***git submodule update***.

Com a pasta *database* atualizada, faz-se então a instalação de um ambiente virtual para utilizar a biblioteca sem afetar os pacotes do sistema operacional. Para que o comando em sequência funcione corretamente, é preciso instalar o módulo *virtualenv* com *pip3*. Executando o comando ***python3 -m virtualenv -p python3 env*** fará com que a pasta *env* contenha todos os arquivos necessários para o funcionamento do ambiente virtual com python3. Em seguida, com o comando ***source env/bin/activate*** o ambiente virtual estará ativo. Instala-se então os requeri-

mentos para a biblioteca presentes no arquivo *requirements.txt* com o comando **pip install -r requirements.txt**.

Listing 1: Ativando o ambiente virtual e instalando os requisitos.

```
usuario:~/((caminho))/database $ source env/bin/activate
(env) usuario:~/((caminho))/database $ pip install -r requirements.txt
```

Em uma das versões do SQLAlchemy, um de seus arquivos possui uma linha duplicada que provoca erros de *foreign key*, à vista disso, o arquivo *dialect.py* presente no caminho *env/lib/python4.5/site-packages/sqlalchemy_monetdb/dialect.py* deve ser editado, comentando a linha 207.

É de grande importância que o arquivo de configuração *settings.py* deva ser verificado para estar de acordo com o nome do banco que será utilizado, além do nome do usuário e senha do MonetDB. O nome do banco presente nesse arquivo será utilizado pela biblioteca para criar as tabelas, fazer inserções e outras ações.

4.1.2 Árvore de Arquivos

O diretório *database* é constituída pelas pastas: *sql*, *database*, *mapping_protocols*, *table_definitions*, *tests*. Além dessas pastas, também há os arquivos: *auto.sh*, *generate_schema.py*, *manage.py*, *protocols_comparisson.py*, *pnad_protocol_from_dic.py*, *pytest.ini*, *README.md*, *requirements.txt*, *settings.py*.

A pasta *sql* contém os arquivos com tabelas base e agregadas para o banco de dados. Nesse caminho são adicionado as tabelas criadas a partir de um script sql. Por exemplo, o arquivo *cub.sql* contém a tabela CUB que possui algumas colunas e dados que serão adicionados na criação. Alguns dos arquivos possuem os dados separados e são nomeados como *seed*.

A pasta *database* possui os códigos da biblioteca. Toda a lógica e funcionamento está disposta nos arquivos presentes nessa pasta. O arquivo *manage.py* é o CLI (Command Line Interface) da biblioteca, contendo os comandos de execução.

A pasta *mapping_protocols* dispõe das definições abrangentes das tabelas que são mais utilizadas pelo SimCAQ e LDE. Essas tabelas estão em arquivos de extensão CSV. Esses arquivos são divididos em colunas, onde cada coluna define a variável, o tipo de dado, o nome no banco de dados, o rótulo e o valor que recebe para cada ano em específico. A partir dessas definições, e dos arquivos presentes na pasta

table_definitions, a biblioteca faz a criação final da tabela representante no banco de dados.

O arquivo *settings.py* compõe as configurações da biblioteca. O banco no qual a biblioteca irá utilizar-se para manipular tabelas é definido nesse arquivo, assim como o nome de usuário e senha do banco de dados e outras configurações importantes. Já o arquivo *requirements.txt* possui todos os requisitos necessários para o funcionamento da biblioteca.

4.1.3 Funcionalidades

A biblioteca é composta por diversas funcionalidades, sendo elas: *create*, *insert*, *remap*, *drop*, *generate_pairing_report*, *generate_backup*, *update_from_file*, *run_aggregations*, *run_sql_group*, *drop_group*, *rebuild_group*, *run_script*. Essas funcionalidades serão detalhadas abaixo uma a uma, de forma a compreender melhor o funcionamento da biblioteca.

Como padrão da biblioteca, a execução sem parâmetros retornará uma breve ajuda dos comandos disponíveis.

Listing 2: Invocação da CLI.

```
$python manage.py [comando] [argumentos posicionais] [argumentos opcionais com valor]
```

- **create:** Utilizando os protocolos de pareamento, o comando *create* irá criar a tabela passada como parâmetro na base de dados descrita no arquivo de configurações *settings.py*.

Listing 3: Criação de tabela.

```
$ python manage.py create <nome da tabela>
```

- **insert:** O comando *insert* é utilizado para inserir dados em uma tabela específica do banco de dados. O comando é composto pelo caminho absoluto de um arquivo no formato CSV, o qual terá seus dados adicionados na tabela passada como parâmetro seguinte já existente no banco de dados.

Listing 4: Inserção de dados na tabela.

```
$ python manage.py insert <caminho para o arquivo> <nome da tabela> <ano> \
[--sep separador] [--null valor_nulo]
```

A tabela utilizada deve existir e estar sincronizada com o protocolo de mapeamento correspondente. O separador padrão utilizado é *ponto e vírgula* (","); caso outros separadores sejam utilizados pelo arquivo fonte, devem ser especificados com a opção `--sep` (por exemplo `--sep \|` para pipe). A opção de valor nulo não é obrigatória, contendo como padrão uma string vazia. Caso os valores nulos devam receber um valor específico, este deve ser estabelecido com `--null`.

- **drop:** Para retirar uma tabela específica do banco de dados utiliza-se o comando `drop` seguido pelo nome da tabela. A tabela só irá ser retirada se não houver dependência sobre ela.

Listing 5: Derrubando uma tabela.

```
$ python manage.py drop <nome da tabela>
```

- **remap:** O comando `remap` sincroniza uma tabela com o protocolo de mapeamento equivalente.

Listing 6: Remapeando uma tabela.

```
$ python manage.py remap <nome da tabela>
```

O remapeamento permite a criação de novas colunas, derrubada de colunas existentes, renomeação de colunas e mudança de tipo. Dependendo do tamanho da tabela, o uso de memória primária pode ser intenso. Essa função pode ocasionar problemas no banco de dados, dessa forma, a utilização dela deve ser somente em casos específicos.

- **generate_pairing_report:** Para gerar relatórios de pareamento para comparação de dados ano a ano utiliza-se o comando `generate_pairing_report`.

Listing 7: Relatórios de pareamento.

```
$ python manage.py generate_pairing_report [--output xlsx|csv]
```

Os relatórios resultados da execução do comando são criados na pasta nomeada *pairing*. Caso o formato do arquivo não seja especificado, o formato padrão CSV será utilizado. Outro formato de arquivo que pode ser gerado é o XLSX, utilizando o parâmetro `--output`. O CSV irá gerar um arquivo para cada tabela, já o XLSX gerará um arquivo com todas as tabelas separadas em diferentes planilhas.

- **generate_backup**: O comando *generate_backup* cria/atualiza o arquivo monitorado para o backup.

Listing 8: Backup de dados.

```
$ python manage.py generate_backup
```

O arquivo produzido é criado ou atualizado na máquina onde o banco de dados da produção está, o procedimento de backup da equipe de infraestrutura o monitora para realizar o procedimento.

- **update_from_file**: Com *update_from_file* é possível atualizar colunas específicas de uma tabela específica.

Listing 9: Atualização de dados de uma coluna.

```
$ python manage.py update_from_file <caminho para o arquivo> <nome da tabela> \
<ano> [--columns colunas] [--sep separador]
```

Um exemplo de utilização é o comando abaixo. Esse comando foi utilizado para atualizar a tabela *docentes*, com dados para a coluna *localidade_area_rural* do banco de dados da *produção*. Para que a atualização ocorresse corretamente, foi necessário remover a coluna da tabela, aplicar o comando *remap* e, em sequência, utilizar o comando *update_from_file* para aplicar os dados.

Listing 10: Exemplo de atualização de dados de uma coluna.

```
$ ./manage.py update_from_file ~/banco/datafiles/2018_DOCENTES_CO.CSV docente 2018 \
--columns localidade_area_rural --sep=|
```

- **run_aggregations**: As tabelas dos protocolos de mapeamentos possuem variáveis que são formadas por certas agregações. Para atualizar as variáveis de agregações é preciso utilizar o comando *run_aggregations*.

Listing 11: Atualização de colunas com agregações.

```
$ ./manage.py run_aggregations <nome da tabela> <ano>
```

Essas agregações são comandos como: *AVG*, *SUM*, *DIV*, outros. Por exemplo, a variável *educacao_profissional* da tabela *escola* possui uma agregação de soma sob a variável *profissionalizante* da tabela *turma*.

- **execute_sql_group**: Para a criação de tabelas base ou agregadas é necessário executar vários scripts sql, podemos criar todas estas tabelas de uma vez utilizamos o comando *execute_sql_group*.

Listing 12: Executando um grupo de sql

```
$ ./manage.py execute_sql_group <nome_do_grupo>
```

Os grupos são definidos no arquivo *./database/groups.py*, ou podem ser passados utilizando a opção *-files* e passados na forma *"sql1,sql2,sql3"*. Por padrão é utilizado os arquivos da pasta *sql*, mas isso pode ser alterado nas configurações ou utilizando o comando *-sqlpath=<diretório>*.

Listing 13: Exemplo especificando os arquivos em outra pasta

```
$ ./manage.py execute_sql_group "idm.sql,projecao.sql,cub.sql" --files\
--sql-path=~/dados/agregadas/
```

- **drop_group**: Utilizado para retirar várias tabelas do banco de uma vez, utiliza-se o comando *drop_group*. A definição do grupo é feita da mesma forma do comando *execute_sql_group*, e assim como *drop* a tabela só será retirada se não houver dependência sobre elas.

Listing 14: Derrubando um grupo de tabelas.

```
$ ./manage.py drop_group <nome_do_grupo>
```

- **rebuild_group**: O comando *rebuild_group* executa o comando *drop_group* seguido de *execute_sql_group*. É bastante útil caso haja a necessidade de recriar as tabelas agregadas quando existe alterações ou novos dados nas suas tabelas base.

Listing 15: Recriando um grupo de tabelas.

```
$ ./manage.py rebuild_group <nome_do_grupo>
```

- **run_script**: O comando *run_script* serve para executar scripts externos utilizando as configurações da database. Atualmente suporta sql, python e bash.

Listing 16: Rodando um script externo.

```
$ ./manage.py run_script <nome_do_script com extensao> [--args="arg1,arg2,..."] \
[--folder=<diretorio>]
```

O script pode ser pego da pasta padrão ou pode ser passada a pasta usando `-folder=<diretório>`, além disso argumentos necessários para execução do script externo deve ser passados na forma `-args=< "arg1,arg2,..." >`

Listing 17: Exemplo de execução de um script externo

```
$ ./manage.py run_script auto.sh --folder=./ --args="all,~/datafiles,2017,2018"
```

4.1.4 O script do script

A pasta *database* possui um arquivo chamado *auto.sh*. Esse arquivo é uma script em bash para rodar comandos da biblioteca sob os bancos e arquivos do SimCAQ. O script se torna útil quando há a necessidade de criar todas as tabelas base, as tabelas do mapeamento de protocolo e adicionar dados de múltiplos arquivos e múltiplos anos. Os possíveis comandos são: *all* - executa todos os comandos para a criação das tabelas e a adição dos dados conforme a limitação dos anos; *base* - cria todas as tabelas base; *create* - faz a criação das tabelas do mapeamento de protocolo; *insert* - utiliza-se para a inserção de dados limitado pelos anos.

Listing 18: Exemplo de utilização do script

```
$ ./auto.sh all /home/victhor/Documents/c3sl/datafiles/ 2013 2017
```

4.2 Atualização de Bases

As tabelas que o SimCAQ possui são as seguintes:

Tabelas	Tipo	Situação
matricula	mapeamento	biblioteca
escola	mapeamento	biblioteca
turma	mapeamento	biblioteca
pnad	mapeamento	biblioteca
populacao_fora_da_escola	mapeamento	script
docente	mapeamento	biblioteca
docente_por_escola	mapeamento	
ibge_pib	mapeamento	
diagnostico_matricula	mapeamento	
indicadores_financeiros	mapeamento	script
indice_distribuicao_matriculas	mapeamento	script
matricula_dependencia_adm	mapeamento	
matricula_localizacao	mapeamento	
projecao_matricula	mapeamento	
cub	base	script
estado	base	
formacao_superior	base	
instituicao_superior	base	
municipio	base	
regiao	base	
siope_mun	base	script
siope_uf	base	script
ibge_pib	base	

5 Indicadores e suas tabelas

5.1 LDE

Abaixo está listado os indicadores e suas respectivas tabelas que o Laboratório de Dados Educacionais consulta.

Indicador	Tabelas Usadas
Número de Matrículas	matricula
Escolas	escola
Turmas	turma
Taxa de Atendimento	pnad
Taxa de Matrícula Bruta	pnad e matricula
Taxa de Matrícula Líquida	pnad e matricula
População Fora da Escola	populacao_fora_da_escola
Número de Professores	docente e turma
Número de Auxiliares Docentes	docente
Número de Funcionários	docente_por_escola
Número de Salas de Aula	
Número de Alunos Por Turma	turma
Carga Horária Diária	turma
Transporte Escolar	matricula
População	ibge_populacao
PIB per capita	ibge_pib
IDHM	adh_idh e adh_idh_uf
IDHME	adh_idh e adh_idh_uf
IDHME	adh_idh e adh_idh_uf
IDHML	adh_idh e adh_idh_uf
IDHMR	adh_idh e adh_idh_uf

5.2 SimCAQ

Abaixo está listado os indicadores e suas respectivas tabelas que o Laboratório de Dados Educacionais consulta.

Indicador	Tabelas Usadas
CUB	cub e estado
Indicadores Financeiros	indicadores_financeiros

6 APIs

6.1 API SimCAQ

É a principal API do projeto, responsável por gerar as consultas. Inicialmente, é necessário ter instalada uma versão igual ou superior a v6.8.1 do NodeJS, mas é recomendado utilizar a versão mais recente LTS (atualmente a v10). Após isso, é preciso clonar o repositório *simcaq-node* no GibLab do C3SL com o comando ***git clone [url]***. Esse repositório pode ser encontrado [aqui](#)

6.1.1 Instalação

Após clonar o repositório, instale as dependências com o comando ***npm i***, que irá criar a pasta *node_modules* com todas as bibliotecas necessárias para executar o projeto.

Para execução local você precisa ter o MongoDB instalado na sua máquina. A forma mais simples é utilizando o gerenciador de pacotes da sua distribuição, que deve instalar e configurar para uso local. Você pode testar se a sua versão do mongo está funcionando com o comando ***mongo***; se tudo estiver certo, você terá acesso ao *shell*.

Antes de executar a API, copie o arquivo *config.json.example* para *config.json* e mude as variáveis de desenvolvimento caso necessário. Se for executar na sua máquina pessoal no C3SL, isso não deve ser necessário. Por padrão, a API executa no *localhost* na porta 3000 no ambiente de desenvolvimento. Você pode agora testar a execução da API com o comando ***gulp build && gulp***.

6.1.2 Arquitetura

A API do SimCAQ é baseada no padrão REST e utiliza NodeJS com o framework Express. Isso significa que ela é *stateless*, ou seja, não guarda informações entre as requisições e por isso, cada rota funciona em isolamento das outras.

Por padrão, as rotas da API são para a realização de consultas às métricas e disponibilização de dados abertos. Bons exemplos são as rotas */classroom* e */teacher*. O fluxo das rotas segue, de forma simplificada, os seguintes passos:

- Identificação das dimensões e filtros
- Consulta base da métrica/processamento base
- Construção da *query*
- Consulta a base de dados
- Inserção de campos descritivos humanos (conversão de ids para strings)
- Resposta ao cliente

Identificação das dimensões e filtros

A identificação das dimensões e filtros é feita pelo *middleware* ***reqQueryFields***, popularmente conhecido como RQF. O RQF é uma classe instanciada como objeto em cada rota que define os campos da URL que serão aceitos e quais seus valores. Por exemplo, uma rota que aceite diferentes filtros possui o campo *filter* em sua URL e é definido através do RQF quais valores são aceitos, neste caso específico, quais os filtros válidos da rota. Quaisquer outros campos ou valores não registrados no RQF serão ignorados.

A identificação então é feita pelo método *parse()* do RQF, um *middleware* do Express. Este método verifica os campos da URL, seu valores aceitos e, em alguns casos, os atributos destes campos. Voltando para o exemplo do filtro, podemos filtrar uma rota por sua região. Para isto utilizaríamos o campo *filter* com o valor *region*. Mas, para identificar a região que desejamos filtrar, temos de passar um terceiro argumento, um atributo para o valor *region*. Neste caso, teríamos a seguinte configuração: *filter=region:5*. Os atributos de um valor são sempre separados por ":"(dois pontos) de seu valor. É possível passar mais de um valor para o mesmo campo. Neste caso, os valores são separados por vírgulas. A função *parse()* então analisa os campos, valores e atributos e os adiciona no objeto *req* do Express utilizando como chave o nome do campo.

A definição dos campos é feita através de um objeto com os seguintes atributos:

- *name*: o nome do campo que será utilizado na URL
- *field*: campo booleano que determina se os valores deste campo são utilizados para trazer novas colunas no resultado, ou seja, se será incluso na cláusula SELECT

- *where*: campo booleano que determina se os valores deste campo vão filtrar a consulta, ou seja, se será incluso na cláusula WHERE

Os valores dos campos podem ser determinados para um campo específico através do método *addValueToField()* ou podem ser determinados para todos os campos através do método *addValue()*. Sua definição também é feita por um objeto, com os seguintes atributos:

- *name*: nome do valor, utilizado na URL
- *table*: a tabela no banco de dados onde o valor pode ser encontrado. Você pode usar apenas o caracter @ se quiser que o RQF utilize a mesma tabela utilizada no FROM da consulta, incluída através do método *from()* do *squel.js*
- *tableField*: Nome ou vetor dos nomes das colunas originais no banco de dados
- *resultField*: Nome ou vetor de nomes que serão retornados no resultado. Caso seja um vetor, deve ter o mesmo tamanho de *tableField*
- *where*: Utilizado apenas se o valor for utilizado em um campo que tenha o atributo *where* verdadeiro
 - *relation*: A relação usada para comparar o atributo do valor da URL com a base de dados. Pode ser qualquer relação suportada pelo SQL
 - *type*: O tipo do valor. Tipos suportados são string, integer, boolean, float
 - *field*: Nome do campo utilizado na cláusula WHERE. Se não for providenciado, será utilizado o mesmo determinado em *tableField*
 - *table*: Nome da tabela utilizada na cláusula WHERE. Se não for providenciado, será utilizado a mesma previamente determinada. Você pode usar apenas o caracter @ se quiser que o RQF utilize a mesma tabela utilizada no FROM da consulta, incluída através do método *from()* do *squel.js*
- *join*: Campo utilizado se necessário realizar um JOIN para obter o valor
 - *primary*: Nome da coluna que é a chave primária da relação

- *foreign*: Nome da coluna que é a chave estrangeira da relação
- *foreignTable*: Nome da tabela que contém a chave estrangeira. Este modelo assume que os dados sempre são retirados da tabela primária. Você pode usar apenas o caracter @ se quiser que o RQF utilize a mesma tabela utilizada no FROM da consulta, incluída através do método *from()* do *squel.js*

Consulta base da métrica/processamento base

Feita a identificação das dimensões e filtros, é possível montar a consulta base e realizar quaisquer processamentos que sejam necessários para a métrica. A API utiliza o *squel.js* para montar as consultas SQL e toda requisição tem injetada a função para realizar consultas em *req.sql*. A partir dele, é possível usar todas as funções do *squel.js* e montar consultas para base de dados.

Construção da *query*

Através do método *build* do RQF, os campos identificados na primeira etapa são incluídos na consulta SQL. Note que este passo pode vir antes da consulta base, mas é uma boa prática realizar a construção o mais tarde possível. Este passo prepara a consulta final que será executada e não deve ser executado mais de uma vez.

Consulta a base de dados

A consulta é feita através do *middleware query*, que pega a consulta gerada pelo RQF e a executa no banco de dados. Há o caso de ser necessário executar mais de uma query simultaneamente. Neste caso, é necessário construir a query antes da consulta base e utilizar o *middleware multiQuery*. Para um exemplo padrão utilizando o *multiQuery*, consulte a rota *infrastructure*.

Por padrão, o resultado da consulta é retornado em *req.result*. Para o caso de múltiplas consultas, o resultado é um array que tem os resultados na mesma ordem do array de consultas.

Inserção dos campos descritivos humanos (conversão de ids para strings)

O *middleware id2str* é o responsável por converter os ids das colunas das tabelas em strings, que serão utilizadas pelos clientes da API. Este *middleware* expõe todas as funções individuais de transformação, mas no fluxo normal será utilizada a função *id2str.transform()* e *id2str.multitransform()*, sendo o último reservado para uso em conjunto com o *multiQuery*.

Resposta ao cliente

Por último temos o *middleware* ***response*** responsável por enviar a repostas para o cliente. Ele permite que o resultado seja enviado em JSON, XML e CSV e utiliza sempre como resposta o objeto em *req.result*.

6.2 API CDN

A API de CDN tem como objetivo disponibilizar e gerenciar o download das bases de dados do MonetDB utilizadas pelo projeto. CDN significa Content Delivery Network e embora esta API não possua todas as características e a arquitetura completa de um, tem o objetivo de gerenciar arquivos de forma a entregá-los aos usuários.

6.2.1 Instalação

A instalação é realizada da mesma forma que a API do SimCAQ. A única diferença é que ela precisa estar na mesma máquina que o MonetDB que será utilizado. Isso ocorre porque a query está no formato COPY INTO e o banco de dado escreve o CSV de saída num caminho local. Portanto, a API precisa de acesso ao diretório onde estão os arquivos gerados pelo banco.

6.2.2 Arquitetura

A API do CDN utiliza Node.js e Express e tem como única rota */file*, que possui todas as entradas para gerenciar os arquivos. Um novo arquivo é gerado fazendo uma requisição POST passando a consulta SQL, a tabela, o cabeçalho que será utilizado no arquivo CSV e as informações do usuário que a solicitou.

Para fazer o download dos arquivos, a API se utiliza da biblioteca [Agenda](#), uma fila de trabalhos baseada em eventos. Dessa forma, o processamento e criação dos arquivos é feita de forma ordenada e caso o trabalho falhe novas tentativas de execução serão feitas. O trabalho para criar o arquivo executa a query recebida, cria um arquivo com o cabeçalho do CSV, concatena os dois arquivos e depois compacta o arquivo final. A atual implementação deste trabalho está bem próxima do *callback hell*, portanto seria interessante utilizar *async/await* no lugar. Criado o arquivo, um novo trabalho é colocado na fila para enviar o email.

Por último há um trabalho definido para rodar a cada 4 horas para limpar os arquivos que existem há muito tempo e não foram baixados mais.

É possível utilizar uma hash para identificar o download de uma base e não ter que regravá-la se o arquivo já existir. Isso também permitiria que a URL para download fosse a mesma para uma mesma base. Implementar isso seria interessante.

6.3 API LDE

A API do Laboratório de Dados Educacionais é a mais simples de todas, e tem o objetivo de servir para o painel do administrador. Essa API possui duas rotas, uma para controlar o cadastro e acesso dos usuários, e outra para tomar conta das mensagens enviadas pelos usuários na página de contato.

6.3.1 Instalação

Inicialmente, é necessário ter instalada uma versão igual ou superior a v6.8.1 do NodeJS. Após isso, é preciso clonar o repositório *LDE API* no GibLab do C3SL com o comando ***git clone [url]***. Esse repositório pode ser encontrado [aqui](#).

Após acessar o diretório *lde-api*, as dependências globais devem ser instaladas com ***npm install -global gulp gulp-cli babel babel-cli babel-core babel-register mocha gulp-mocha gulp-eslint istanbul***.

O próximo passo é copiar o arquivo *config.json.example* para *config.json* e instalar as dependências do projeto com ***npm install***.

Para abrir no navegador, basta executar ***gulp build && gulp*** e acessar: <http://localhost:3000/api/v1/>.

7 LDE - Dados Educacionais

O LDE (Laboratório de Dados Educacionais) é uma plataforma que visa facilitar o acesso e uso de dados relacionados à educação, estimular e potencializar pesquisas focadas na elaboração de indicadores educacionais para possibilitar a implementação, monitoramento e avaliação das políticas educacionais.

7.1 Instalação

As tecnologias utilizadas para o desenvolvimento da plataforma são:

- ReactJS
- ReduxJS
- SASS

Inicialmente, é necessário clonar o repositório *LabDados UI* no GitLab do C3SL, com o comando ***git clone [url]***. Esse repositório pode ser encontrado [aqui](#). Após acessar o diretório *labdados-ui*, é necessário usar o comando ***npm install*** para instalar todas as dependências do projeto. Para executar o código no navegador, basta digitar ***npm start*** e acessar <http://localhost:7770>.

7.2 Painel do Administrador

O painel do administrador foi desenvolvido com o framework ***Admin On Rest***, que faz uso das tecnologias **JSES6**, **React** e **Material Design**. A motivação que levou à escolha deste framework é devido ao fato de que a interface do LDE também é feita em ReactJS, o que estabelece um padrão, e também por ser a ferramenta mais utilizada atualmente, pois possui um fórum ativo mantido pelos desenvolvedores.

Atualmente, o painel não está em nenhum dos ambientes do projeto, foi testado apenas localmente.

7.2.1 Funcionalidades

As funcionalidades do painel permitem listar, criar, editar e filtrar os usuários pela origem do cadastro. Todos os dados são obtidos pela API do LDE, sem conside-

rar as autenticações, pois quando ocorrer sua integração com o LDE, será necessário que o administrador tenha feito login e, portanto, como as autenticações são controladas pela API do SimCAQ, será permitido o acesso ou não.

Futuramente, o painel deverá permitir a troca de imagens da página inicial do LDE e a inclusão de notícias.

7.2.2 Instalação

Inicialmente, é necessário clonar o repositório *lde-admin* no GitLab do C3SL, com o comando ***git clone [url]***. Esse repositório pode ser encontrado [aqui](#).

A instalação é a mesma do LDE, com os comandos ***npm install*** e ***npm start***. Logo após a execução dos comandos, a aplicação será iniciada no Browser padrão, com a url: <http://localhost:3000>.

7.2.3 Principais Problemas

Um dos problemas é o fato de que o framework ***Admin On Rest*** ainda está sendo desenvolvido, o que implica que há alguns bugs que possuem mensagens de erro muito genéricas, o que dificulta quando é necessário debugar o código.

Como consequência desse problema, quando o administrador deseja criar ou editar algum usuário na interface, há um warning constante da biblioteca PolyglotJS, que é usada pelo framework.

8 SimCAQ - Simulador Custo Aluno Qualidade

O Simulador de Custo-Aluno-Qualidade (SimCAQ) web 1.0 é um sistema gratuito e disponível na internet desenvolvido para calcular o custo da oferta educacional em condições de qualidade e, desse modo, para subsidiar o planejamento orçamentário-financeiro das redes públicas de educação básica. O ambiente não é produzido pela equipe da C3SL, e sim por uma equipe terceirizada, o código é feito em angular e manda requisições para a nossa API(simcaq-node).

9 Ambientes

9.1 Introdução

Atualmente existe 3 máquinas responsáveis pelos clientes web e APIs. Abaixo será explicado o intuito de cada uma existir e local onde se encontra cada projeto.

- **simcaqdev:** Responsável pelo ambiente de desenvolvimento. Todos os repositórios estão em */home/simcaq/simcaq-dev/* e todos estão na branch *development*, com exceção da documentação que está em */var/www/html/api-doc* na branch *dev* e o frontend do SimCAQ que está na branch *RS*.

A base de usuários se encontra no mongo em *dev_users* e o banco que a API manda requisições se encontra na máquina *simcaqdb3*.

- **simcaqhomologa:** O ambiente de homologação é responsável por uma validação, sendo que essa validação estiver correta, as mudanças estão autorizadas para ser mandadas ao ambiente de produção. Todos os repositórios estão em */home/simcaq/* na branch *development* com exceção da documentação (branch *hom*) e frontend do SimCAQ (branch *RS*). Relacionado ao banco de dados, a API acessa o banco da máquina *simcaqdb3 - simcaq_dev4* - e a base de usuários cadastrados se encontram em *dev_users*.

- **simcaqprod:** Responsável pelo ambiente de produção, sendo visível para todos os usuários e sendo a versão mais estável. Todos os repositórios estão em */home/simcaq/* na branch *master*, com exceção do frontend do SimCAQ (branch *RS*). O banco de dados que a API se conecta está na máquina *c3sldatabase* e a base de usuários se encontra em *users*.

9.2 Breve Explicação dos comandos

A seguir iremos listar e explicar o que cada comando individualmente faz ao atualizar as máquinas.

- **git stash:** Fazer Stash é tirar o estado sujo do seu diretório de trabalho — isto é, seus arquivos modificados que estão sendo rastreados e mudanças na

área de seleção — e o salva em uma pilha de modificações inacabadas que você pode voltar a qualquer momento.

- **git stash pop**: Tira as mudanças da pilha de modificações inacabadas.
- **git pull origin <branch>**: Baixa na sua máquina local, todas as alterações feitas na <branch> do repositório remoto do Git.
- **npm i**: Instala todas as dependências de pacote relacionadas ao node e react, por exemplo ao adicionar uma nova biblioteca é necessário executar esse comando para instalar-lá nas máquinas.
- **npm run build**: "Compila"o código do React e o deixa executando em segundo plano.
- **gulp build**: Compila o código do NodeJS.
- **systemctl restart <processo>**: Reinicia o <processo>
- **ng build**: Todo artefato criado no Angular é escrito em TypeScript, contudo, os navegadores ainda não interpretam essa linguagem nativamente. Assim, faz-se necessário, primeiro, compilar nosso projeto para JavaScript e realizar algumas tarefas que vão deixá-lo mais apto a rodar em um servidor de produção. Para essa tarefa, existe o comando ng build, que fará todo processamento e gerará uma pasta dist, na qual estarão os arquivos prontos para serem abertos em um navegador

9.3 Atualização

Abaixo será explicado como atualizar cada repositório nas máquinas citadas acima.

9.3.1 API SimCAQ (simcaq-node)

Para atualizar a API nos ambientes de **desenvolvimento e homologação** basta seguir os seguintes passos:

Listing 19: Atualizando simcaq-node na máquina simcaqdev e simcaqhomologa.

```
$ git stash
$ git pull origin development
$ npm i
$ git stash pop
$ gulp build
## No ambiente de desenvolvimento
$ systemctl restart simcaq-dev-1
## No ambiente de homologacao
$ systemctl restart simcaq-api-1
```

Para atualizar a API na **produção** basta seguir os seguintes passos:

Listing 20: Atualizando simcaq-node na máquina simcaqprod

```
$ git pull origin master
$ npm i
$ gulp build
$ systemctl restart simcaq-api-1
```

9.3.2 Front End SimCAQ (simcaq-ui)

Listing 21: Atualizando simcaq-ui na na máquina simcaqdev

```
$ git pull origin RS
$ npm i
$ ng build
```

Listing 22: Atualizando simcaq-ui na na máquina simcaqhom e simcaqprod

```
$ git pull origin RS  
$ npm i  
$ ng build —prod
```

9.3.3 API LDE (LDE API)

Listing 23: Atualizando lde-api na na máquina simcaqhom e simcaqdev

```
$ git pull origin development
$ npm i
$ gulp build
## No ambiente de desenvolvimento
$ systemctl restart lde-dev-1
## No ambiente de homologacao
$ systemctl restart lde-api-1
```

Listing 24: Atualizando lde-api na na máquina simcaqprod

```
$ git pull origin master
$ npm i
$ gulp build
$ systemctl restart lde-api-1
```


9.3.4 Front End LDE (labdados-ui)

Listing 25: Atualizando labdados-ui na na máquina simcaqhom e simcaqdev

```
$ git stash
$ git pull origin development
$ git stash pop
$ npm i
## No ambiente de desenvolvimento
# Build
$ npm run buildDev
# Start
$ npm run startDev
## No ambiente de homologacao
# Build
$ npm run buildHom
# Start
$ npm run startHom
```

Listing 26: Atualizando labdados-ui na na máquina simcaqprod

```
$ git stash
$ git pull origin master
$ git stash pop
$ npm i
$ npm run buildProd
```

9.3.5 Documentação (api-doc)

Listing 27: Atualizando api-doc na na máquina simcaqhom, simcaqdev. simcaqprod

```
# No ambiente de desenvolvimento
$ git pull origin dev
# No ambiente de homologacao
$ git pull origin hom
# No ambiente de producao
$ git pull origin master
```

```
$ gulp build
```

10 Outras ferramentas

10.0.1 Monitoramento

Existe um monitoramento de visitas para o site do LDE piwik.c3sl.ufpr.br

Referências

- [1] C3SL. *Biblioteca para administrar base de dados*. 2018. URL: <https://gitlab.c3sl.ufpr.br/simcaq/database>.
- [2] C3SL. *Simulador de Custo-Aluno-Qualidade*. 2018. URL: <https://simcaq.c3sl.ufpr.br/>.
- [3] C3SL. *Submódulo protocolo de mapeamentos*. 2018. URL: https://gitlab.c3sl.ufpr.br/simcaq/mapping_protocols.
- [4] GNU. *Licença GNU GPLv3*. 2018. URL: <https://www.gnu.org/licenses/gpl-3.0.pt-br.html>.
- [5] MonetDB. *MonetDB man-page*. 2019. URL: <https://www.monetdb.org/Documentation/monetdbd-man-page>.
- [6] MonetDB. *The column-store pioneer*. 2018. URL: <https://www.monetdb.org/Home>.
- [7] NumPy. *The fundamental package for scientific computing with Python*. 2018. URL: <http://www.numpy.org/>.
- [8] Pandas. *Python Data Analysis Library*. 2018. URL: <https://pandas.pydata.org/>.
- [9] SQLAlchemy. *The Python SQL Toolkit and Object Relational Mapper*. 2018. URL: <https://www.sqlalchemy.org/>.