Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Documentação de Software
Joker - An Open Source Card Sorting Application.

SUMÁRIO

Sumário **1**

1 Introdução **2**

1.1 O que é o *Joker?* 2

1.2 Sobre o *Card Sorting* 2

1.3 Contribuições da ferramenta 2

2 Guia do Usuário **3**

2.1 Criação de estudos 3

2.2 Participação nos estudos 3

2.3 Obtenção de resultados 3

2.4 Consumindo a API - (Usuários avançados) 4

3 Arquitetura de Software **4**

3.1 Linguagens e *Frameworks* 5

3.2 Justificativas de Decisões de Projeto 5

3.3 Sobre os Módulos do Sistema e Suas Relações 6

4 Módulos e Componentes **7**

4.1 Estrutura de Arquivos 7

4.2 Estrutura de Arquivos em `'/src'` 7

4.3 Persistência de Estados e Dados da Aplicação 7

4.3.1 Redux 8

4.3.2 MongoDB 8

4.4 Modelagem do Banco De Dados 8

4.5 Módulo estatístico 10

4.5.1 Acoplando o módulo à API 11

4.6 Virtualização e *Containers* 11

5 Referência da API **13**

5.1 Direcionamento da seção 13

5.2 Rotas e tipos de dados 13

6 Agradecimentos **15**

REFERÊNCIAS **16**

1 INTRODUÇÃO

1.1 O QUE É O *JOKER?*

Trata-se de uma ferramenta gratuita e de código aberto voltada para a realização de estudos de usabilidade baseados em Card Sorting, desenvolvida na Universidade de São Paulo no campus de São Carlos por **Anderson Canale Garcia, André de Lima Salgado, Felipe Silva Dias, João Pedro R. Mattos e Renata P. M. Fortes**. O desenvolvimento da ferramenta foi financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (**FAPESP**) sob a vigência dos processos **nº 2018/19323-8, nº 2017/15239 - 0 e nº 2015/24525 - 0**.

1.2 SOBRE O *CARD SORTING*

Card Sorting é uma das técnicas mais populares para a avaliação de Arquiteturas de Informação (AI), uma vez que possibilita o agrupamento de informações através do modelo mental dos participantes dos experimentos. Através de *Joker*, o *Card Sorting* pode ser realizado à distância, assim a ferramenta se qualifica como uma alternativa gratuita e disponível em código aberto que visa a contribuir com pesquisas de usabilidade a serem conduzidas futuramente por outros pesquisadores.

1.3 CONTRIBUIÇÕES DA FERRAMENTA

A construção do *Joker* motivou a redação de um artigo[1] intitulado ***"Smart toys and Children's Privacy: usable privacy policy insights from a Card Sorting experiment"***. Esse artigo é um resultado que representa um esforço de todos os envolvidos nos projetos relacionados à utilização da ferramenta de apoio à técnica de *Card Sorting Joker*. O artigo foi submetido e aceito ao evento científico **SIGDOC'2019**[2] (Grupo de Interesse Especial em Design de Comunicação da *Association for Computing Machinery* - ACM) em Outubro de 2019.

Além disso, o benefício oferecido pela ferramenta *Joker* tornou-se de interesse internacional, visto a participação do pesquisador Prof. *Patrick Hung*[3], da *Ontario Tech University* (Canadá), que participou como co-autor no artigo que foi redigido.

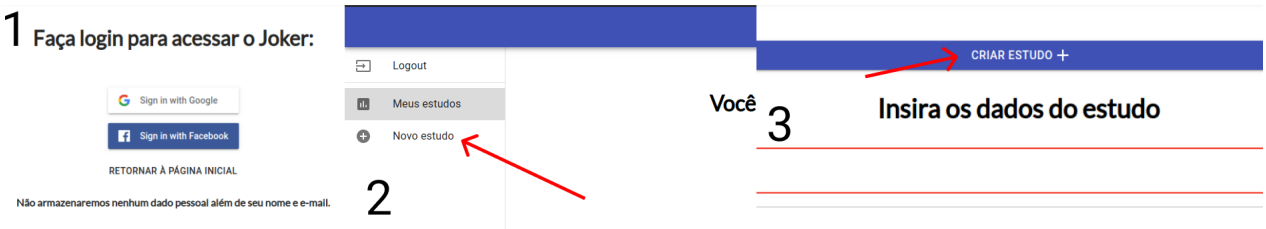
[1] <https://doi.org/10.1145/3328020.3353951>
[2] <https://sigdoc.acm.org/conference/2019/>
[3] <https://businessandit.ontariotechu.ca/people/faculty/networking-and-it-security/patrick-hung-phd.php>

2 GUIA DO USUÁRIO

2.1 CRIAÇÃO DE ESTUDOS

A criação de um novo estudo é realizada, primeiramente, clicando no botão de "Área do Pesquisador"na página inicial. Depois de efetuado o *login* na ferramenta *Joker*, basta acessar a aba "Novo estudo"na lateral esquerda do painel do pesquisador e inserir os dados do estudo. Finalmente, resta clicar em "Criar estudo"para finalizar a criação de um estudo, que já pode ser conferida na página "Meus Estudos".

Figura 1 – Processo de criação de um estudo. As flechas vermelhas indicam os botões citados na subseção 2.1.



Fonte: Elaborada pelo autor.

2.2 PARTICIPAÇÃO NOS ESTUDOS

Para disponibilizar o estudo para participações, basta clicar no símbolo ao lado do estudo desejado dentro da aba "Meus Estudos"no painel do pesquisador e, em seguida, dentro do menu de contexto aberto, clicar em "Copiar Link Para Estudo". Dessa forma, um link para acesso ao estudo será copiado para a área de transferência, de forma a possibilitar seu envio aos participantes, como demonstrado na figura 3.

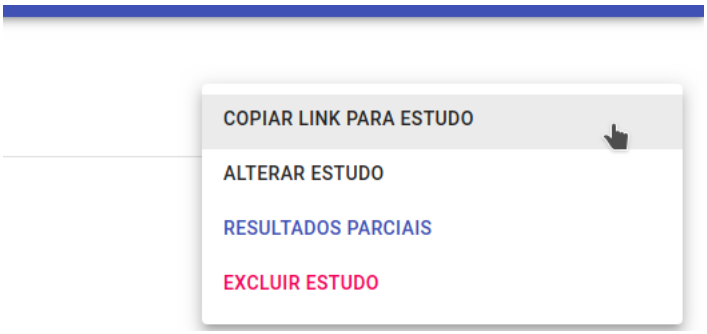
Do ponto de vista do participante, basta clicar no link recebido ou colá-lo no campo presente na página "Área do Participante".

É importante notar que só serão considerados para a "Obtenção de Resultados"as participações que forem devidamente concluídas. Logo, caso um participante feche a tela de participação sem clicar no botão de "Concluir"no canto superior direito, a participação não será considerada para a apuração final de resultados.

2.3 OBTENÇÃO DE RESULTADOS

Os resultados podem ser obtidos a partir do botão "Resultados Parciais"dentro do menu de contexto que se abre quando se clica no símbolo ao lado do estudo desejado, dentro do painel do pesquisador.

Figura 2 – Captura de tela mostrando a cópia para a área de transferência do link que dá acesso ao estudo.



Fonte: Elaborada pelo autor.

Dentro da tela de resultados do *Joker*, é possível realizar o *download* do dendrograma e dos dados de todas as participações do estudo.

Figura 3 – Captura de tela que exibe os botões mencionados na seção 2.3.

Fonte: Elaborada pelo autor.

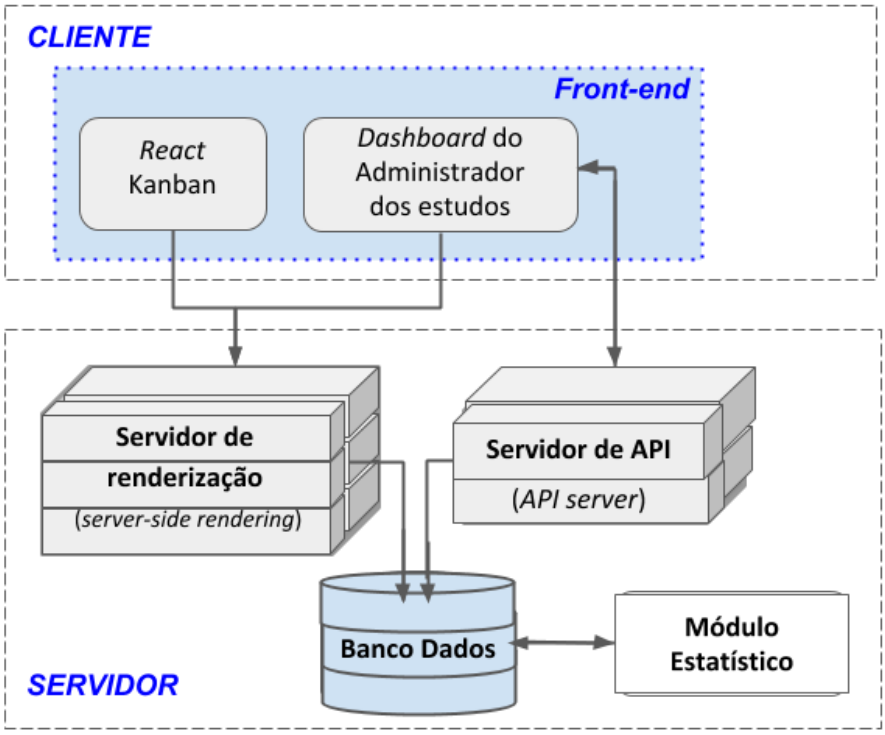
2.4 CONSUMINDO A API - (USUÁRIOS AVANÇADOS)

Através da *Joker API* é possível obter detalhes sobre o sistema e realizar *queries* no banco de dados de Joker. Trata-se de uma API RESTful em HTTP que pode ser facilmente consumida através de um sistema de requisições de qualquer linguagem de programação. Para informações sobre as rotas e tipos de resposta, ver seção 5.2.

3 ARQUITETURA DE SOFTWARE

O objetivo desta seção é fornecer ao leitor uma visão abstraída da funcionalidade de cada módulo da ferramenta *Joker*, além de suas interdependências. A leitura dessa seção é altamente recomendada a futuros contribuidores do projeto, a fim de facilitar a compreensão do funcionamento em alto nível de cada módulo antes de uma abordagem diretamente debruçada sobre a implementação direta dos mesmos em código-fonte.

Figura 4 – Arquitetura da ferramenta *Joker* - prioriza compatibilidade, simplificando assim, a integração entre os módulos do projeto.



Fonte: Elaborada pelo autor.

3.1 LINGUAGENS E *FRAMEWORKS*

O desenvolvimento da ferramenta *Joker* utiliza a biblioteca *open-source React Kanban*[4] como base de componentes, devido a sua semelhança (visual) com o processo de participação de uma sessão de *card sorting*. A partir dos estudos sobre a biblioteca *React Kanban*, foi constatado que ela possuía implementadas apenas as telas de manipulação de cartões, o suporte ao banco de dados *MongoDB*[5] e o *back-end* para *Server-Side Rendering (SSR)* construído usando *NodeJS*[6]. Nesse sentido, optamos por utilizar *ReactJS* para a construção do painel do pesquisador e *NodeJS* para a implementação do servidor de API. Para o módulo estatístico foi utilizada a linguagem *Python 3.7* em conjunto com as bibliotecas *Numpy* e *Matplotlib*. Todos os requisitos para a execução da ferramenta *Joker* podem ser visualizados na página do projeto no *Github*[7];

3.2 JUSTIFICATIVAS DE DECISÕES DE PROJETO

Com base nos expostos da seção 3.1, decidimos que o desenvolvimento da ferramenta *Joker* deveria ser voltado para a integração das estruturas da *React Kanban* com uma

[4] <https://github.com/markusenglund/react-kanban>
[5] <https://www.mongodb.com/>
[6] <https://nodejs.org/en/>
[7] <https://github.com/joaopedromattos/joker#setting-up-joker>

interface de administração e criação de estudos, junto com um Banco de Dados para armazenar e disponibilizar o conteúdo cadastrado por pesquisadores para participantes / usuários do *Card sorting*.

A modelagem da arquitetura da ferramenta *Joker* e a modelagem do Banco de Dados foram definidas com o objetivo de priorizar o aproveitamento de código e escalabilidade.

3.3 SOBRE OS MÓDULOS DO SISTEMA E SUAS RELAÇÕES

Os módulos ilustrados como mostra a Figura 4 foram implementados seguindo as características de sistemas iterativos Web (KUHR; HAMILTON, 2008) (FERNANDEZ; NAVóN, 2010), e têm suas respectivas funcionalidades descritas a seguir em alto nível.

(1.) **módulo *React Kanban*** - O objetivo principal deste módulo é permitir a execução da técnica *Card Sorting* pelos usuários, isto é, consiste nas telas de organização de cartões e de acesso ao estudo. Utiliza rotas do servidor de renderização para realizar a persistência de cada alteração do usuário no banco de dados em tempo real.

(2.) **módulo *Dashboard do Administrador dos estudos*** - consiste nas telas de planejamento e personalização da técnica *card sorting* pelo pesquisador. Neste módulo, o pesquisador poderá verificar os dados coletados e acompanhar os resultados gerados pelo Módulo Estatístico. Comunica-se com o *Servidor de API* para autenticação e armazenamento dos estudos e seus resultados.

(3.) **módulo *Servidor de Renderização*** - trata-se do *back-end* da arquitetura *Server-Side Rendering*, encarregado de tratar todas as requisições do usuário no navegador entregando as páginas web construídas em *React* e definidas nos módulos *React Kanban* e *Dashboard do Administrador de Estudos*. Comunica-se com o banco de dados para realizar a persistência das alterações no quadro de cartões a cada iteração.

(4.) **módulo *Módulo Estatístico*** - representa a execução das análises apropriadas para o agrupamento dos cartões. Este módulo é responsável por gerar a visualização gráfica da Arquitetura de Informação resultante (e.g. dendrogramas). Comunica-se com o banco de dados para recuperar todas as participações de determinado estudo.

(5.) **módulo *Servidor de API*** - encarregado de receber todas as *queries* do banco de dados que foram realizadas pela parte *front-end*. Mais especificamente, trata-se de uma API RESTful que usa o protocolo HTTP. Comunica-se com o banco de dados para realizar a autenticação do pesquisador e para salvar a criação/alteração dos dados de cada estudo.

4 MÓDULOS E COMPONENTES

Esta seção é voltada àqueles que procuram uma compreensão avançada acerca da implementação do código-fonte dos módulos do *Joker*. Nesse sentido, é recomendada a leitura da seção anterior a esta, de forma a possibilitar um primeiro contato com a arquitetura do sistema e seus aspectos técnicos em alto nível antes de iniciar uma abordagem mais detalhada.

4.1 ESTRUTURA DE ARQUIVOS

Podemos dividir a estrutura de arquivos de *Joker* em três grandes localizações iniciais:

Diretório Raiz	Comporta arquivos de configuração do projeto e de suas dependências, além armazenar os diretórios '/src' e '/apiResearcher'.
Diretório `'/src'`	Trata-se do diretório onde estão localizados todos os componentes visuais escritos em *React*, além de seus respectivos reducers (mecanismos de gerenciamento de estado) e servidor. Dessa forma, os arquivos referentes ao **React Kanban**, ao **Dashboard do administrador de estudos** e ao **Servidor de Renderização** são armazenados dentro desse diretório.
Diretório `'/apiResearcher'`	Diretório responsável por armazenar todos os arquivos referentes ao funcionamento do módulo "Servidor de API". A API é escrita segundo o padrão de projeto MVC (Model - View - Controller), logo as pastas subsequentes a esta seguirão esse padrão.

4.2 ESTRUTURA DE ARQUIVOS EM `'/SRC'`

Devido ao fato de o diretório `'/src'` conter mais de um módulo daremos destaque a sua estrutura de arquivos interna, de forma a delinear as fronteiras entre cada uma das partes de *Joker* através da tabela 1.

4.3 PERSISTÊNCIA DE ESTADOS E DADOS DA APLICAÇÃO

Dentro de *Joker*, realizamos dois tipos de gerenciamento de estados e de dados; o primeiro é realizado a partir do *Redux*, uma biblioteca de gerenciamento de estados para React, e o segundo ocorre através das chamadas à API e da atualização dos dados diretamente no MongoDB. Nas duas próximas subsubseções, trataremos em detalhes dos casos de uso relativos a cada mecanismo de controle de dados.

Diretório `'/app'`	Diretório responsável por armazenar todos os componentes dos módulos "Dashboard do Administrador"e "React Kanban".
Diretório `'/assets'`	Conjunto de imagens, ícones e configurações visuais utilizadas para a construção da identidade visual de *Joker*
Diretório `'/server'`	Diretório responsável pelo "Servidor de Renderização", composto por suas rotas e demais configurações.
Arquivo `'client.jsx'`	Arquivo raiz do Virtual DOM do React. O nó inicial da execução do React encontra-se dentro desse arquivo a partir do uso do componente inicial "App".

Tabela 1 – Definição em alto nível dos padrões de projeto utilizados em cada diretório consitituinte da pasta `'/src'`.

4.3.1 REDUX

Durante a execução de *Joker*, utilizamos a biblioteca *Redux*, através de suas instâncias chamadas de *reducers*, para armazenar dados de importância momentânea, isto é, *tokens* de autenticação, estudos de cada usuário, o estado da tela de organização de cartoes, etc.

Não cabe detalharmos os conceitos que abrangem a utilização das funções de gerenciamento de estados do *Redux*, pois podem ser encontrados na documentação oficial da biblioteca.

Ainda assim, é importante detalhar que todos os *reducers* de *Joker* podem ser encontrados sob o diretório `'/src/app/reducers/'`. Lembrando que, apesar de estarem todos em arquivos separados, eles são unidos dentro do arquivo `'/src/app/reducers/index.js'` através da função `'combineReducers'`, também disponível através do *Redux*.

4.3.2 MONGODB

O banco de dados *MongoDB* é utilizado para a persistência dos dados de maior importância para o funcionamento da aplicação, os dados de cada estudo, de suas respectivas participações e do *login* do pesquisador responsável.

Mais detalhes sobre a modelagem do banco de dados, como *Collections* e tipos de dados, estão expostos na subseção seguinte.

4.4 MODELAGEM DO BANCO DE DADOS

Para realizarmos o armazenamento dos dados dos pesquisadores e suas respectivas pesquisas, modelamos o banco de dados *MongoDB* conforme três *models* que, para o banco de dados, serão interpretados como *Collections* (o equivalente às tabelas do modelo modelo relacional).

Os *models* podem ser encontrados sob o diretório `"/apiResearcher/models"`. Dentro da pasta em questão, temos os três modelos utilizados dentro do banco de dados de *Joker*:

(1.) **`apiResearcher/models/board.js`** - Trata-se da *Collection* que modela o tipo de dado responsável por armazenar o estado final (em JSON) de uma participação de uma determinada pesquisa. **Dessa forma, caso exista interesse em realizar algum tipo de processamento de dados com base nos resultados de determinado estudo, será nessa *Collection* que o desenvolvedor deve buscar seus dados.**

Figura 5 – Captura de tela do código-fonte do arquivo `Board.js`.

```
var mongoose = require("mongoose");

var boardSchema = new mongoose.Schema({
    studyId: { type: mongoose.Schema.Types.ObjectId, ref: 'Study' },
    title: String,
    color: String,
    valid: { type: Boolean, default: false },
    lists: [{
        title: String,
        cards: [
            {
                color: String,
                text: String
            }
        ]
    }]
}, { timestamps: true})

module.exports = mongoose.model('Board', boardSchema);
```

Fonte: Elaborada pelo autor.

(2.) **`apiResearcher/models/researcher.js`** - Responsável por armazenar os *ids* dos estudos o `authId` de um pesquisador. O campo *authId* é um dos dados retornados da API do *Firebase*[8], serviço externo que utilizamos para autenticação de usuários. Dessa forma, simplificamos o sistema terceirizando toda a preocupação com gerenciamento de senhas e outros dados de acesso à ferramenta.

(3.) **`apiResearcher/models/study.js`** - Modelo responsável por armazenar os dados de um estudo, como os cartões e o nome do estudo. São aos *ids* desta *Collection* que o campo `studies` da *Collection* `Researcher` faz referência.

[8] <https://firebase.google.com/>

Figura 6 – Captura de tela do código-fonte do arquivo `Study.js`.

```
var mongoose = require('mongoose');

var studySchema = new mongoose.Schema({

    name: String,
    objective: String,
    cards: {
        type: [{ name: String, description: String }]
    },
}, { timestamps: true })

module.exports = mongoose.model('Study', studySchema);
```

Fonte: Elaborada pelo autor.

Figura 7 – Captura de tela do código-fonte do arquivo `Researcher.js`.

```
var mongoose = require('mongoose')

var researcherSchema = new mongoose.Schema({
    authId: String,
    email: String,
    name: String,
    studies: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Study' }]

});

module.exports = mongoose.model("Researcher", researcherSchema);
```

Fonte: Elaborada pelo autor.

4.5 MÓDULO ESTATÍSTICO

Durante a fase de obtenção de resultados da ferramenta *Joker*, o módulo estatístico escrito em *Python 3* é utilizado para gerar o dendrograma e a limpeza dos dados em JSON. Nesse sentido, o objetivo desta seção é explicar como um script em *Python* foi acoplado à API escrita em *Node.js* e conceder uma visão geral do funcionamento do código-fonte dentro do arquivo `/apiResearcher/statisticalModule/cardClustering.py`, responsável por toda a funcionalidade do módulo em questão.

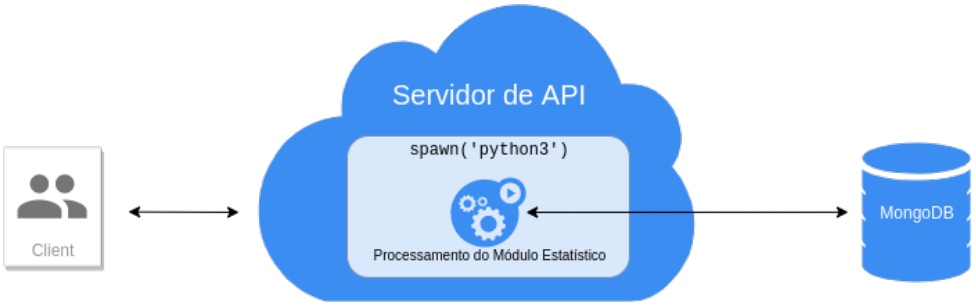
4.5.1 ACOPLANDO O MÓDULO À API

O acoplamento do módulo estatístico foi realizada através da biblioteca `child_process`[9] de gerenciamento e criação de processos do *Node.js*, mais especificamente da função `spawn`, responsável por instanciar um novo processo para a execução de código escrito em *Python*.

Assim que o Servidor de API recebe uma requisição na rota `/getResults/`, ele executa a função `getResults` do *controller* `/apiResearcher/controller/getResults.js`.

Já dentro da `getResults`, a função `spawn` é chamada, de forma a instanciar o script `/apiResearcher/statisticalModule/cardClustering.py`, que realiza o processamento dos dados do estudo desejado, e responde à requisição com um dendrograma e um arquivo JSON, que serão hospedados pelo Servidor de API. Ver figura 8 para uma referência gráfica acerca desse processo de obtenção de resultados.

Figura 8 – Ilustração em alto nível do processo de obtenção de resultados.



Fonte: Elaborada pelo autor.

A figura 9 ilustra em alto nível a ordem dos procedimentos realizados dentro do módulo estatístico, de forma a esclarecer a complexidade das etapas de manipulação de dados que acontecem durante uma requisição para obtenção de resultados.

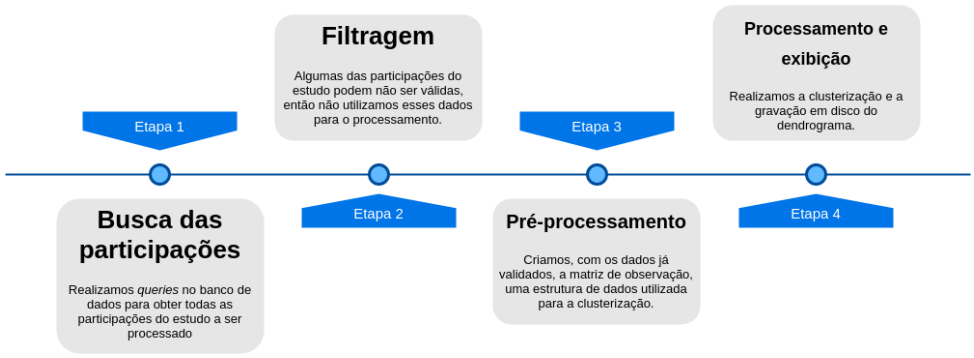
Para obter mais detalhes sobre como o procedimento de processamento de resultados é realizado, incluindo a limpeza dos dados e a utilização da biblioteca *Matplotlib* para a geração dos dendrogramas, consulte o código-fonte original no arquivo `/apiResearcher/statisticalModule/cardClustering.py`.

4.6 VIRTUALIZAÇÃO E *CONTAINERS*

Utilizamos a ferramenta de virtualização *Docker*[10] para modularizar a instalação e a compatibilidade do código fonte da aplicação para futuros *deploys*. Nesse sentido,

[9] <https://nodejs.org/api/child_process.html>
[10] <https://www.docker.com/>

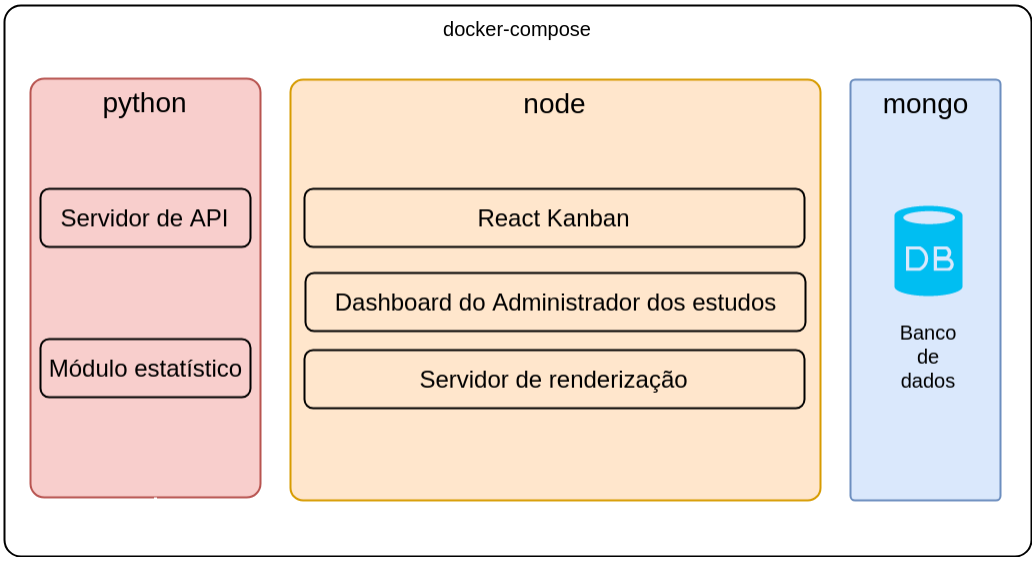
Figura 9 – Ordem dos procedimentos realizados dentro do módulo estatístico.



Fonte: Elaborada pelo autor.

utilizamos três imagens *Docker*: *"mongo"* (MongoDB), *"node"* (NodeJS) e *"python"* (Python).

Figura 10 – Ilustração que relaciona a rede virtual de *contêineres* Docker à arquitetura da ferramenta *Joker*.



Fonte: Elaborada pelo autor.

Além disso, foi utilizada a ferramenta de orquestração *Docker Compose* para criar uma rede virtual que integra os serviços dos três contêineres. A relação dessa rede virtual com os componentes da arquitetura é apresentada na Figura 10.

Todos os `Dockerfiles` e arquivos de configuração do *docker-compose* podem ser encontrados sob os diretórios `'/'` e `'/apiResearcher'`.

5 REFERÊNCIA DA API

5.1 DIRECIONAMENTO DA SEÇÃO

Esta seção é voltada àqueles que desejam realizar uma coleta de dados disponíveis na ferramenta Joker através do acesso à API. Nesse sentido, é recomendada a leitura das seções 3 e 4, além de conhecimentos no consumo de APIs RESTful na linguagem de programação à sua escolha.

5.2 ROTAS E TIPOS DE DADOS

Para uma consulta mais eficiente à esta documentação, recomenda-se ter em mente o *Controller* (disponíveis em `/apiResearcher/controllers`) responsável pela consulta desejada, uma vez que todas as rotas estão dispostas pelas funções dentro de cada um desses arquivos.

- `"/researchers"`
  - POST
    `createResearcher()` - Cria um novo perfil de pesquisador conforme dados passados pelo *body* da requisição.

- `"/researchers/email=:email"`
  - GET
    `getResearcher()` - Retorna o perfil de um pesquisador de acordo com o parâmetro da rota.
  - PUT
    `updateResearcher()` - Atualiza o perfil do pesquisador de acordo com os parâmetros passados pelo *body* da requisição.
  - DELETE
    `deleteResearcher()` - Remove um pesquisador do banco de dados através do *id*.

- `"/studies"`
  - POST
    `createStudy()` - Cria um novo estudo com os dados presentes no *body* da requisição.

- `"/studies/_id=:_id"`

  - GET
    `getStudy()` - Retorna um estudo através do *id* passado nos parâmetros da rota.
  - PUT
    `updateStudy()` - Atualiza um estudo com base nos dados passados pelo *body* da requisição.
  - DELETE
    `deleteStudy()` - Remove um estudo do banco de dados através do *id* passado nos parâmetros da rota.

- `"/boards"`
  - GET
    `listBoard()` - Retorna todas as participações em todos os estudos da plataforma *Joker*. **(Cuidado, esta requisição pode demorar muito!)**

- `"/boards/studyId=:studyId"`
  - GET
    `getBoard()` - Retorna todas as participações de determinado estudo.
  - POST
    `createBoard()` - Cria uma nova participação em um estudo.

- `"/boards/_id=:_id"`
  - PUT
    `updateBoard()` - Atualiza uma determinada participação num estudo de acordo com os dados passados pelo *body* da requisição.
  - DELETE
    `deleteBoard()` - Remove uma participação do banco de dados.

- `"/fetchBoard/_id=:_id"`
  - GET
    `createWelcomeBoard()` - Retorna o estado inicial do quadro de participação no experimento de *Card Sorting*.

- `"/getResults/studyId=:studyId"`

  - GET
    `getResults()` - Dispara a atuação do módulo estatístico para o processamento dos dados do estudo passado por parâmetro. **Retorna o endereço do gráfico do dendrograma, hospedado no servidor de API**.

- `"/getResults/dendrogram/studyId=:studyId"`
  - GET
    `downloadDendrogram()` - Realiza o *download* do *plot* do dendrograma do estudo.

- `"/getResults/json/studyId=:studyId"`
  - GET
    `downloadJson()` - *Download* do JSON que contém todos os dados processados do estudo em questão.

- `"/countResults/studyId=:studyId"`
  - GET
    countBoards() - Retorna o número de participações em determinado estudo, cujo *id* é passado por parâmetro.

6 AGRADECIMENTOS

Para a realização do processo de pesquisa e desenvolvimento do projeto *Joker* foi necessária a colaboração de diversas instituições, dentre as quais destacamos e agradecemos:

- Ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP) contribuiu, durante toda a execução do projeto, com as atividades de pesquisa desenvolvidas, a partir do oferecimento de laboratórios equipados com computadores e componentes de desenvolvimento de software necessários para o projeto;

- À Superintendência de Tecnologia da Informação da Universidade de São Paulo através de um de seus centros, o CeTI-SC[11], que mostrou-se muito solícito durante o processo de hospedagem da aplicação, a partir da abertura de portas nos roteadores da rede da universidade;

[11] <http://cetisc.sti.usp.br/>

- À infraestrutura do Laboratório *Intermídia* que foi absolutamente importante em diversas etapas do projeto, inclusive na hospedagem da ferramenta *Joker* em um de seus servidores;
- À **Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP)** pelos processos nº 2018/19323-8, nº 2017/15239 - 0 e nº 2015/24525 - 0, que financiaram esta pesquisa e o desenvolvimento deste projeto.

REFERÊNCIAS

FERNANDEZ, F.; NAVóN, J. Towards a practical model to facilitate reasoning about rest extensions and reuse. In: **Proceedings of the First International Workshop on RESTful Design**. New York, NY, USA: ACM, 2010. (WS-REST '10), p. 31–38. ISBN 978-1-60558-959-6. Disponível em: <http://doi.acm.org/10.1145/1798354.1798383>.

KUHR, M.; HAMILTON, D. Building executable service-oriented architectures with the ws-management specification. In: **Proceedings of the 2008 Spring Simulation Multiconference**. San Diego, CA, USA: Society for Computer Simulation International, 2008. (SpringSim '08), p. 315–324. ISBN 1-56555-319-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1400549.1400594>.