PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Desenvolvimento de Interfaces de Sistemas de Workflow para Estudos Relacionados ao Tratamento do Câncer

LORRAM NASCIMENTO DE OLIVEIRA

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Engenharia da Computação

Rio de Janeiro, novembro de 2021.

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

LORRAM NASCIMENTO DE OLIVEIRA

Desenvolvimento de Interfaces de Sistemas de Workflow para Estudos Relacionados ao Tratamento do Câncer

Projeto final apresentado ao **Curso de Engenharia da Computação** como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Sérgio Lifschitz

Rio de Janeiro

Novembro de 2021.

AGRADECIMENTOS

Agradeço a PUC-Rio e seu corpo docente, em especial, ao Departamento de Informática, que sempre buscam a excelência em ensino, a qual certamente será fundamental para minha carreira. Agradeço também ao PROUNI pela concessão da bolsa integral, a qual me permitiu a realização de um sonho.

Agradeço ao meu orientador, Ségio Lifschitz, por ter me dado a oportunidade de participar de projetos de grande relevância junto ao time do BioBD da PUC-Rio e, também, por sempre mostrar-se disponível para conversas e orientações, além de ser um excelente professor, o qual admiro desde que cursei a disciplina de Banco de Dados durante a graduação.

Agradeço a coordenadora do curso de Engenharia da Computação e professora Noemi Rodriguez pela sua atenção e dedicação, sempre pronta a retirar todas as dúvidas rapidamente.

Agradeço aos meus coorientadores desse projeto: a pós-doc Elvismary Molina e ao doutorando Diogo Vieira que sempre se mostraram disponíveis e solícitos para tirarem a dúvidas e sempre preocupados com o desenvolver dos projetos, sempre trazendo um aprendizado novo. Além disso, por serem sempre solidários e muito amigáveis o que tornou as conversas bem leves e produtivas. Também agradeço ao Alexandre Heine, aluno de graduação de Engenharia da Computação da PUC-Rio e atualmente formando, que sempre buscou me ajudar no desenvolvimento dos projetos, além de sempre dar boas dicas e compartilhar conhecimentos.

Agradeço a minha família que ao longo da minha vida sempre me incentivaram e apoiaram sendo fundamentais para realização desse sonho de me formar no que amo e em uma grande instituição de ensino do Brasil.

Agradeço também a doutora Inayara B. Araujo Martins, namorada e amiga que sempre me incentivou e sempre buscou me ajudar de todas as formas, inclusive na revisão e orientações da escrita desse documento.

Além disso, agradeço a PUC-Rio como universidade, por compartilhar até este momento 6 anos da minha vida, dentre eles 4 anos morando ao lado da universidade, o que me permitiu viver essa experiência por completo, conhecendo pessoas de diferentes cursos, de diferentes culturas, com diferentes opiniões e perspectivas, o que me proporcionou uma nova visão de mundo. Certamente jamais esquecerei!

A todos o meu sincero Obrigado!!!

RESUMO

Oliveira, Lorram Nascimento de; Lifschitz, Sérgio. **Desenvolvimento de Interfaces de Sistemas de Workflow para Estudos Relacionados ao Tratamento do Câncer.** Rio de Janeiro, 2021. 53p. Relatório de Projeto Final – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

O Projeto Final de Graduação, consiste na participação em dois projetos reais em desenvolvimento pelo BioBD (Laboratório de Bioinformática de Banco de Dados da PUC-Rio) em parceria com o INCA (Instituto Nacional do Câncer). O projeto envolveu o estudo conceitual de Sistemas Gerenciamento de *Workflow* Científicos (SGWfCs) e suas aplicações. O principal objetivo foi o desenvolvimento de interfaces para os *workflows*: PIPE-MB e do *workflow* SGWfC-Gene. Tais interfaces tem como finalidade facilitar a interação dos usuários do INCA sobre as ferramentas desenvolvidas, a fim de trazer resultados relevantes a estudos relacionados ao Câncer.

Palavras-chave: Workflow. SGWfC. Bioinformática. INCA. Grafos. IHC.

ABSTRACT

Oliveira, Lorram Nascimento de; Lifschitz, Sergio. **Development of Workflow System Interfaces for Studies Related to Cancer Treatment**. Rio de Janeiro, 2021. 53p. Final Project Report – Department of Informatics. Pontifical Catholic University of Rio de Janeiro.

The Final Graduation Project consists of participating in two real projects being developed by BioBD (PUC-Rio's Database Bioinformatics Laboratory) in partnership with INCA (National Cancer Institute). The project involved the conceptual study of Scientific Workflow Management Systems (SGWfCs) and their applications. The main objective was the development of interfaces for the workflows: PIPE-MB and the SGWfC-Gene workflow. Such interfaces are intended to facilitate the interaction of INCA users on the developed tools, in order to bring relevant results to studies related to Cancer.

Keywords: Workflow. SGWfC. Bioinformatics. INCA. Graphs. IHC.

SUMÁRIO

1. Introdução

Todos os anos milhares de pessoas são diagnosticadas com câncer e praticamente a metade são vítimas fatais dessa doença. Hoje, o câncer é a segunda causa de mortes no Brasil e no mundo, superado apenas por doenças cardiovasculares (WHO, 2020). Por essa razão, pesquisas sobre o câncer possuem estratégias vastas e complexas, as quais envolvem vários pesquisadores de diferentes áreas e disciplinas. Nesse sentido, o Instituto Nacional do Câncer (INCA) apresenta, dentre os seus objetivos estratégicos, a promoção de pesquisa e parcerias interinstitucionais para o controle do câncer no cenário nacional e internacional (INCA, 2020).

Os estudos do câncer no INCA abrangem desde a pesquisa básica e experimental, trabalhando no sentido de solidificar as bases da biologia tumoral e para desvendar novos mecanismos celulares e moleculares envolvidos com a tumorigênese; perpassando a pesquisa epidemiológica até a prática clínica, tendo como objetivo final desenvolver novas estratégias para o diagnóstico e tratamento do câncer (INCA, 2020).

O uso da Bioinformática tem se tornado cada vez mais importante para estudos relacionados ao câncer. Especialistas em processamento e análise de dados usam ferramentas e equipamentos de ponta e conseguem produzir a informação correta, pontual e adequada a partir de um grande volume de dados que auxiliam os profissionais da biologia e clínica no diagnóstico clínico de possíveis doenças expressadas em uma de nossas sequências de DNA (GENOMIKA, 2014).

1.1 *Workflows* Científicos

Dentre as ferramentas utilizadas para estudos relacionados as descobertas de variantes cancerígenas, temos os sistemas de *workflow*, os quais têm recebido muita atenção da comunidade de pesquisa desde os anos 80. Estes são fortemente voltados à manipulação de grandes volumes de dados complexos, sendo, atualmente, amplamente encontrados tanto nos domínios científicos quanto no industrial (BRAGHETTO e CORDEIRO, 2014). Os *workflows* científicos são bastante utilizados para automação de experimentos científicos com o uso de plataformas de computação de alto desempenho acessíveis a pesquisadores

de diversas áreas de conhecimento, a exemplo médicos, biólogos e pesquisadores de doenças como o câncer (SILVA, 2007).

Segundo Ludäscher et al., (2006) os *workflows* científicos são definidos como "*Redes de processos tipicamente utilizadas como 'pipelines de análises de dados' ou ainda para comparar dados observados ou previstos, e que podem incluir uma vasta gama de componentes, como por exemplo, para consultar bancos de dados, para transformar ou minerar dados, para executar simulações em computadores de alto desempenho, entre outros*" (*apud* BRAGHETTO e CORDEIRO, 2014, p.2).

Com relação a flexibilidade esperada de *workflows* científicos, temos que sistemas de gerência de *workflow* geralmente interpretam rigidamente a definição de um *workflow*, não permitindo qualquer tipo de desvio durante a execução. No entanto, existem situações reais em que usuários devem ter a possibilidade de desviar do fluxo pré-definido por diferentes razões, tais como, a falta de informação do valor de um parâmetro ou a indisponibilidade de recursos necessários à execução. Para alcançar uma execução flexível, pode ser usado um mecanismo de tratamento de exceções, voltado para flexibilização, que permite a continuação da execução de uma instância que deveria ser momentaneamente interrompida (VIEIRA, 2005).

Segundo Cuevas-Vicenttin et al., (2012) um *workflow* científico é uma automação de um experimento ou de um processo científico, expressa em termos das atividades a serem executadas e, principalmente, das dependências dos dados manipulados. Além disso, um *workflow científico também pode ser definido* como um encadeado de processos que permite a automação de um trabalho científico de maneira mais simples utilizando os recursos computacionais necessários para tal (VIEIRA et al., 2021).

Um *workflow* de bioinformática é uma forma especializada de sistema de gerenciamento de *workflow* projetado especificamente para compor e executar uma série de etapas computacionais ou de manipulação de dados relacionadas à bioinformática. Tais sistemas mostram uma representação abstrata da computação e como ela procede na forma de um grafo direcionado. Cada nó do grafo representa uma tarefa a ser executada e cada aresta representa o fluxo de dados ou a dependência de execução entre diferentes tarefas (Figura 1).

Alguns sistemas de *workflow* científico fornecem uma interface visual, que permite ao usuário construir e modificar diferentes *workflows* com pouco ou nenhum conhecimento de programação. Alguns dos mais famosos *workflows* de bioinformática incluem: Apache Taverna, Galaxy, Unipro UGENE.

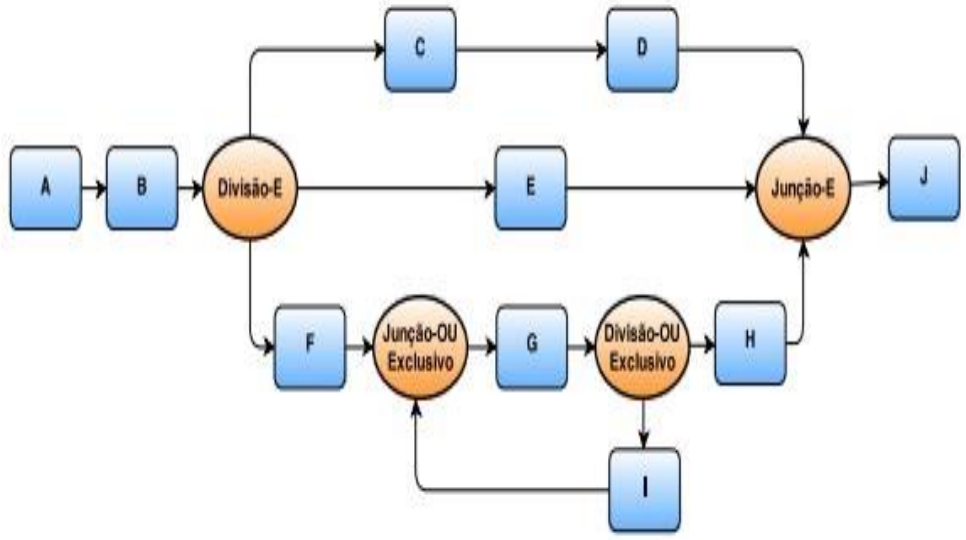


Figura 1. Exemplo de *workflow* modelado usando as construções básicas de fluxo de controle (BRAGHETTO e CORDEIRO, 2014)

2. Situação Atual

2.1 Situação atual do projeto 1 – *Workflow* PIPE-MB

Atualmente, a pesquisa está sendo feita para entender melhor a arquitetura interna do *workflow* PIPE-MB, e com isso, buscar ter um controle maior durante a sua execução, a fim de ter a possibilidade de uma melhor interação dos usuários e outras funcionalidades importantes que são encontradas em outros *workflows* científicos. Nesse sentido, minha participação na pesquisa é colaborar para torná-lo mais acessível para o uso de bioinformatas, que hoje precisam recorrer a *scripts* complexos para conseguir utilizá-lo, além de buscar melhorar a saída do *workflow* para usuários comuns, para atender diretamente médicos envolvidos em testes clínicos com pacientes.

Durante a disciplina de Projeto Final I, fui inserido no projeto pelo meu orientador Sérgio Lifschitz, com o auxílio da Elvismary Molina (coorientadora), para o desenvolvimento de uma interface para o *workflow* PIPE-MB. O primeiro passo foi o desenvolvimento de uma página *web* com um formulário para o preenchimento de diversos parâmetros do *JSON* de entrada do *workflow* (Figura 2).

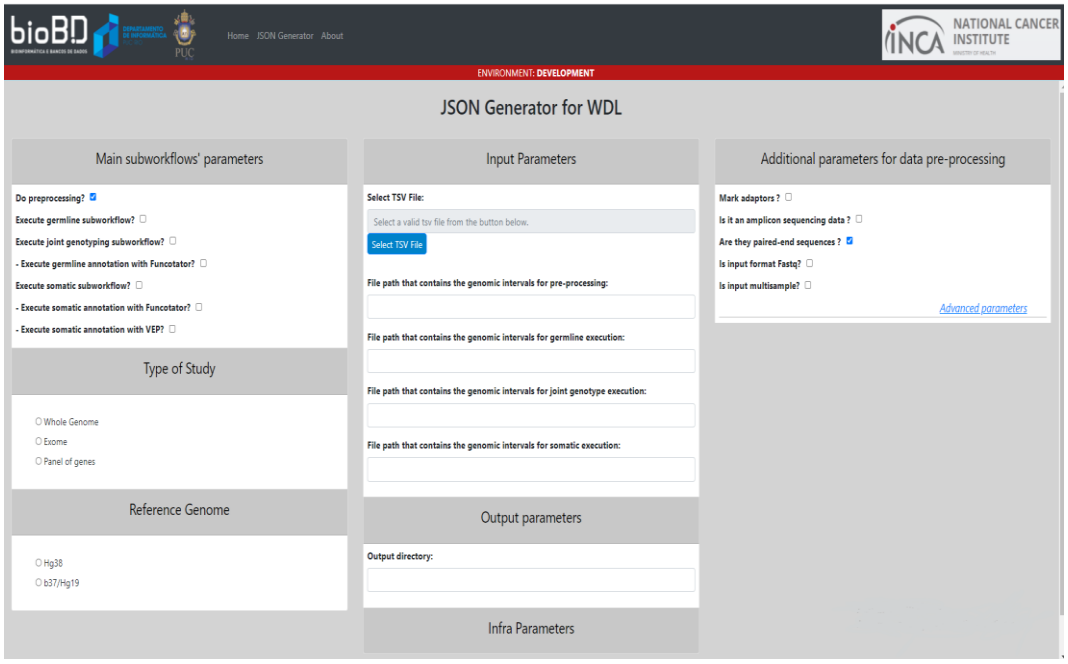


Figura 2. Interface web de entrada do *workflow* PIPE-MB.

No Projeto Final II, foi dado sequência na dinâmica dessa interface de entrada, além de criar um Gerenciador de Arquivo TSV (*Tab-Separated Values*) (Figura 3). O TSV é o principal formato de arquivo utilizado para armazenar informações de amostras genéticas dos pacientes. Esse arquivo é fornecido como entrada da execução do *workflow,* no entanto, sua edição e manuseio atualmente apresenta certa dificuldade para os usuários do INCA, uma vez que estes devem editar diretamente um arquivo tabular. Portanto, a criação de uma ferramenta para sua edição e gerenciamento tem como objetivo facilitar a edição do arquivo com as informações genéticas pelos usuários INCA.

Atualmente, a interface de entrada está na fase final de desenvolvimento em termos de sua funcionalidade e cumprindo o objetivo de facilitar o uso do *worklow* PIPE-MB para usuários do INCA, gerando um arquivo JSON que será usado como entrada para o *workflow*.

2.2 Situação atual do projeto 2 – SGWfc-Gene

O projeto do *workflow* está sendo desenvolvido em cooperação com o coorientador Diogo Vieira (doutorando da PUC-Rio) com o Cristóvão Antunes de Lanna (doutorando do INCA). Como Projeto Final, o Projeto 2 teve como objetivo

desenvolver uma interface de interação com *workflow* para que os usuários tenham maior facilidade de utilizar a ferramenta.

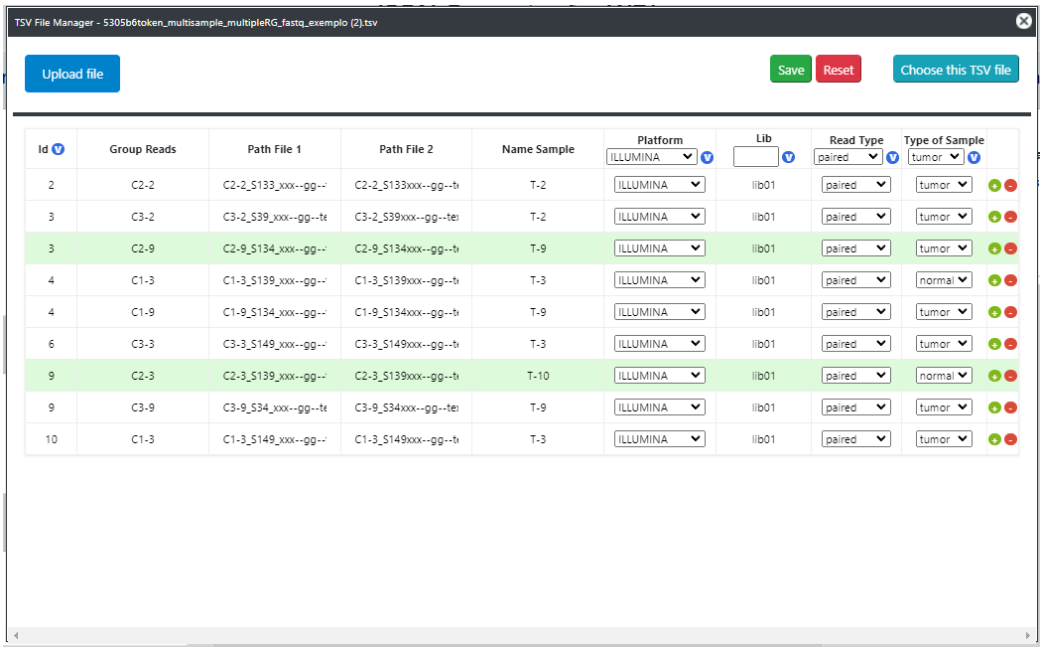


Figura 3. Interface de gerenciamento de arquivo TSV.

Ao longo do Projeto Final II, foi desenvolvida a primeira versão de uma *página web,* que permite que o usuário faça o *upload* de um arquivo no formato *CSV* (*Comma-separated Values*) esperado como entrada do *workflow* e exiba uma tabela HTML do histórico de arquivos carregados. Por meio do botão '*Run'* da tabela de arquivos carregados, o *workflow* é então executado utilizando como entrada o arquivo *CSV* referenciado pela linha da tabela (Figura 4).

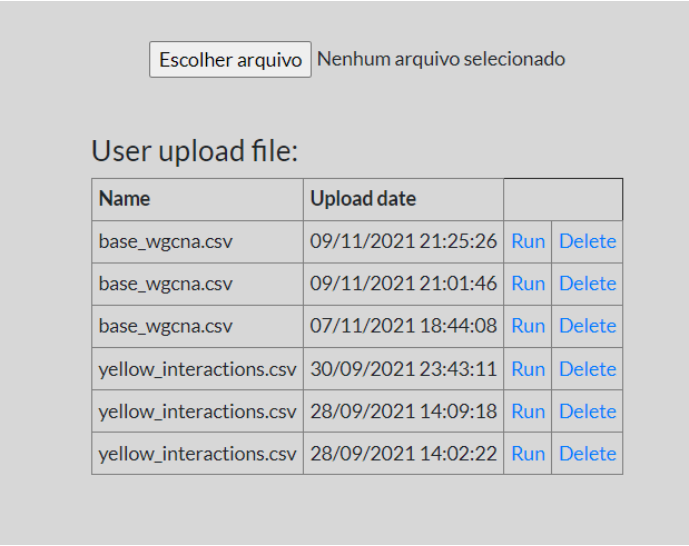


Figura 4. Componentes de *upload* e tabela de arquivos carregados pelo usuário.

Ao fim da execução do *workflow*, o resultado é um grafo que será renderizado na mesma página *web* ao usuário por meio do componente *Cytoscape-react* do *React* como é mostrado na Figura 5.

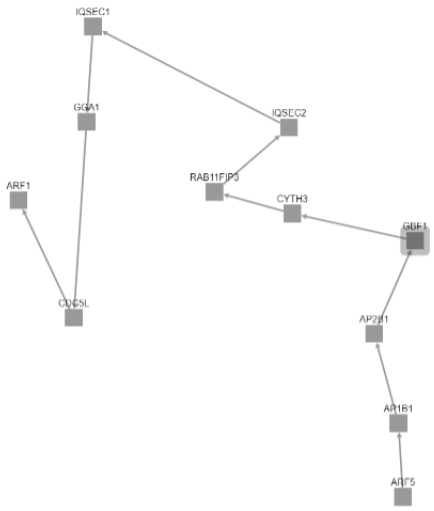


Figura 5. Exemplo da exibição de um grafo utilizando a ferramenta *Cytoscape*.

3. **Objetivos**

A escolha dos problemas abordados no Projeto Final, foram feitas com objetivo de explorar conhecimentos adquiridos ao longo da graduação em Engenharia da Computação, e obter novos conhecimentos aplicado em projetos de pesquisas reais de grande importância em colaboração com o Bio-BD ao INCA. Os projetos foram voltados a trazer interfaces aos *workflows* abordados, para facilitar o uso dos usuários finais, além de adquirir conhecimento na arquitetura de *workflows* e de conhecimentos gerais na área de bioinformática.

O projeto 1, *workflow* PIPE-MB, teve como objetivo o desenvolvimento de interfaces web para os parâmetros de entrada do *workflow*, além de facilitar a gerenciamento do arquivo de entrada de amostras genéticas de pacientes no formato de arquivo TSV, com a possibilidade de edição e pareamento de amostras genéticas de pacientes.

O projeto 2 (SGWfC-Gene) teve como objetivo desenvolvimento de interfaces *web* que permite uma maior interação do usuário final com *workflow*. A interface desenvolvida permite que os usuários façam *upload* dos arquivos CSV

no formato esperado como entrada do *workflow.* Em seguida, o *path* (caminho) de onde foi salvo esses arquivos no servidor *web* é associado ao *id* do usuário que fez o *upload* em uma tabela do banco de dados relacional (SQLite). Após o *upload* dos arquivos, as informações do mesmo são mostradas de maneira dinâmica em uma tabela HTML, a qual exibe o histórico de arquivos carregados para o usuário. Para cada arquivo carregado é exibido o botão '*Run*' na tabela HTML, permitindo ao usuário executar o *workflow* usando o arquivo referenciado como entrada do *workflow*. Com a execução do *workflow* e a resposta gerada, o Componente '*Cytoscape*' do *React* no *front-end* é utilizado para renderizar o grafo resultante na página web, permitindo uma iteração e seleção dos nós do grafo pelo usuário.

Para a completude do Projeto Final, as seguintes tarefas foram realizadas:

1. Estudo geral sobre *workflow* científicos, e da linguagem WDL para especificação de *workflows*.
2. Estudo mais aprofundado das tecnologias para desenvolvimento *web*. Sendo, no *front-end*: *HTML*, *CSS*, *Bootstrap*, *Javascript*, *JQuery; e no back-end:* framework *Python* chamado *Django. Além* das ferramentas de controle de versão de código-fonte *Git* e *GitHub*.
3. Desenvolvimento de um formulário dinâmico para geração de arquivo *JSON* de entrada do *workflow* PIPE-MB.
4. Desenvolvimento de um Gerenciador de Arquivo TSV de forma interativo em uma tabela HTML editável, que permite a edição de arquivos de amostras genéticas de pacientes nesse formato de arquivo.
5. Estudo sobre o *framework React* para desenvolvimento de *front-end*, buscando o entendimento de sua arquitetura de componentes e o uso de Gerenciadores de Estado como o *Redux*.
6. Estudo sobre o software *Cytoscape* como plataforma de *software* de bioinformática muito útil para visualização em grafos de interações moleculares.
7. Estudo do *NetworkX* como uma biblioteca na linguagem *Python* para estudos de grafos e redes.
8. Estudo sobre o *workflow Prefect* e sua arquitetura.

4. Atividades realizadas

4.1 Atividades preliminares

Antes da realização do Projeto Final, o aluno tinha experiência com linguagem de programação *Python*, *Javascript*, *JQuery* e desenvolvimento de aplicações web em outras tecnologias como .NET MVC, NodeJS e VueJS. Também tinha o conhecimento em *HTML*, *CSS* e *Booststrap* o que foram fundamentais para o desenvolvimento das interfaces webs desenvolvidas ao longo do Projeto Final.

4.2 Estudos Conceituais e de Tecnologia

Para realização do presente projeto, foi necessário o estudo sobre o funcionamento de *workflows* em Sistemas de Gerenciamento de *Workflows* Científicos (SGWfCs) em geral, bem como, nos *workflows* específicos envolvidos no projeto. Também foi necessário o estudo do *framework* Django e de sua arquitetura de desenvolvimento MTV (*Model*, *Template*, *View*), assim, como, o estudo sobre o *framework React* para o *front-end* e dos componentes *Redux* (Gerenciador de Estado) e do *Cytoscape* (Visualização de Grafos). Por fim, o estudo sobre o *NetworkX, o qual* é um pacote *Python* para manipulação de grafos que permite realizar alterações no grafo para se adequar como entrada do *Cytoscape,* para renderizar o grafo adequado para o usuário.

4.3 Método

A participação do aluno em ambos os projetos se deu por encontros remotos semanais com o grupo do BioBD da PUC-Rio, responsáveis pelos projetos dos *workflows* abordados no Projeto Final. Esses encontros foram importantes para o levantamento e esclarecimento de dúvidas sobre os projetos e para especificações das tarefas, mediante ao acompanhamento do professor e orientador *Sérgio Lifschitz*.

Para especificações das tarefas a serem realizadas semanalmente utilizou a plataforma do *Trello*. Já para o contato para dúvidas pontuais ou imediatas, utilizou a plataforma *Discord*. Para versionamento do projeto foram utilizadas a plataforma *GitHub* como versionamento remoto e o *Git* para versionamento local.

Em ambos os projetos, a mesma metodologia foi utilizada: especificação, estudo conceitual sobre as tecnologias e implementação das tarefas acordadas durante as reuniões semanais.

No projeto 1 (*workflow* PIPE-MB), as especificações e demandas provenientes de reuniões com os representantes do INCA no projeto, eram repassadas ao aluno pela pós-doc Elvismary Molina (coorientadora). Já no projeto 2 (SGWfC-Gene), as especificações e demandas das etapas de desenvolvimento foram repassadas ao aluno pelo doutorando Diogo Vieira (coorientador). Além disso, dúvidas mais específicas da regra de negócio, próximos passos de desenvolvimento para solução final, bem como, o *feedback* das etapas concluídas foi realizado por Cristóvão Antunes de Lanna, doutorando do Laboratório de Bioinformática e Biologia Computacional e representante do projeto pelo INCA.

4.4 Cronograma

O cronograma a seguir é apresentado o desenvolvimento do trabalho ao longo do ano:

– Janeiro e Fevereiro:
Estudos iniciais sobre *workflows*. Participação em apresentação de palestra do BioBD sobre o *workflow Apache Spark*.

– Março:
Integração ao projeto 1 (*workflow* PIPE-MB). O aluno teve acesso ao código-fonte do projeto em um repositório remoto do GitHub, possibilitando um primeiro contato.

– Abril:
Estudos iniciais sobre as tecnologias envolvidas no projeto 1, tais como o *framework Django* para desenvolvimento *web*. Também ocorreu a implementação da primeira versão da interface de entrada para o *workflow* PIPE-MB.

– Maio:
Iniciando a mudança da interface de entrada para o *workflow* PIPE-MB para uma segunda versão. Escrita do Relatório para o Projeto Final 1.

– Junho, Julho e Agosto:

Seguiu-se o desenvolvimento do projeto 1. O aluno teve a tarefa de modularizar todo código *Javascript* do projeto e iniciar o desenvolvimento da tela de Gerenciamento de Arquivo TSV.

– Setembro:

Integração ao projeto 2 (SGWfC-Gene), no qual o aluno teve o primeiro contato com o código-fonte do projeto. Além disso, foi continuado em paralelo o desenvolvimento do projeto 1, com o desafio de tratar o pareamento de amostras na tela de edição de arquivo TSV em desenvolvimento.

– Outubro:

Estudos iniciais sobre as tecnologias envolvidas no projeto 2, tais como o *framework React* e o *workflow* Prefect, assim como outras ferramentas envolvidas no projeto como *Cytoscape* e *NetworkX*. Foi dado sequência também no desenvolvimento do projeto 1, a fim de deixar a tabela de Gerenciamento de Arquivo TSV com mais flexibilidade e interativa para o usuário final. Além disso, foi implementado toda a lógica de controle de visibilidade dos campos do formulário de entrada do *workflow* PIPE-MB.

– Novembro:

Foi implementado uma página *web* para o projeto 2 (SGWfC-Gene) composta por componentes *React*: um componente para possibilitar o *upload* de arquivo CSV, uma tabela HTML com o histórico de arquivos carregados e outro para exibir o grafo resultante da execução do *workflow* por meio do componente *Cytoscape-react.* Também foi um momento de dar a sequência na escrita do Relatório para Projeto Final 2.

– Dezembro:

Apresentação do Projeto Final para a banca.

5. Projeto e especificação do sistema

5.1 **Projeto 1 –** Desenvolvimento de Interface de Sistema de Workflow para Descoberta de Variantes Genéticas para Estudo do Câncer

5.1.1 *Workflow* PIPE-MB

O projeto 1 vai abordar o desenvolvimento de interface para w*orkflow* PIPE-MB de descobertas de variantes genéticas do câncer. Esse *workflow* tem como objetivo auxiliar e gerar respostas à uma pesquisa já em desenvolvimento, que envolve uma parceria da PUC-Rio com o INCA, a qual é coordenada pelo professor do Departamento de Informática da PUC-Rio, Sérgio Lifschitz em colaboração da Elvismary Molina, pós-doc do laboratório BioBD da PUC-Rio.

O *workflow* PIPE-MB tem como objetivo a descoberta de variantes (SNPs e INDELs) usando GATK4 e ferramentas relacionadas. Foi implementado para cumprir as melhores práticas GATK (BROAD INSTITUTE, 2021a), e desenvolvido por pesquisadores do INCA e da PUC-Rio. Esse *workflow* integra a etapa de pré-processamento, chamada de variantes somáticas e germinativas e etapas de filtragem e anotação. Ele foi implementado usando a tecnologia WDL (©Open WDL, 2021) em sua versão 1.0 e testado no motor *Cromwell* (FRED HUTCH, 2020).

5.1.2 Linguagem WDL para descrição de *Workflows*

Existem várias linguagens para especificação de *workflow*, em particular o PIPE-MB se baseia na linguagem WDL (*Workflow Description Language*). WDL é uma padronização, ele não é executável por si só, mas requer um mecanismo de execução para funcionar. Os mecanismos de execução compatíveis devem oferecer suporte aos recursos de uma versão específica do WDL. O WDL tem como objetivo a possibilidade especificar *workflows* de processos de uma forma fácil e com uma sintaxe simples, entretanto, na prática o seu uso não é tão simples quanto mencionado em sua documentação. O WDL tem como objetivo definir tarefas de análises complexas, encadeá-las em *workflows* e paralelizar sua execução (©Open WDL, 2021).

*Broad Institute é uma organização de pesquisa dedicada à bioinformática, com sede em Cambridge no estado de Massachusetts dos Estados Unidos.

A Figura 6 apresenta um exemplo de "*Hello World*" na linguagem WDL. Como podemos observar, a figura descreve uma *task* chamada "*hello*" que possui dois parâmetros de entrada: *String pattern* e *File in*. As *tasks* têm entradas e saídas. Os parâmetros de entrada são declarados na sessão de *input* da *task*, enquanto os de saídas são definidos na sessão *output*.

A definição de *task* é uma forma de encapsular um comando em ambiente UNIX e apresentá-los como funções. Para que essa *task* possa ser executada, o usuário deve fornecer valores para esses dois parâmetros de entrada. As implementações de WDL devem aceitar suas entradas como formato *JSON (JavaScript Object Notation*).

WDL é um projeto *open-source* desenvolvido e mantido pela comunidade ©Open WDL disponibilizado no GitHub através do link abaixo:

<https://github.com/openwdl/wdl/blob/main/versions/1.0/SPEC.md>

```
task hello {
  input {
    String pattern
    File in
  }

  command {
    egrep '${pattern}' '${in}'
  }

  runtime {
    docker: "broadinstitute/my_image"
  }

  output {
    Array[String] matches = read_lines(stdout())
  }
}

workflow wf {
  call hello
}
```

Figura 6. Exemplo da definição de uma task "*hello*" na linguagem WDL (OpenWDL, 2021).

5.1.3 Ferramentas de linha de comando GATK

Segundo o Broad Institute (2021), GATK (*Genome Analysis Tool Kit*) é uma coleção de ferramentas de linha de comando para analisar dados de sequenciamento de alto rendimento com foco principal na descoberta de variantes genéticas. As ferramentas podem ser usadas individualmente ou encadeadas em *workflows*.

No projeto do *workflow* PIPE-MB está sendo utilizada a versão GATK 4. Esta versão visa reunir ferramentas bem estabelecidas das bases de código GATK e *Picard* (BROAD INSTITUTE, 2021b) em uma estrutura simplificada e permitir que ferramentas selecionadas sejam executadas de maneira massivamente paralela em *clusters* locais ou na nuvem usando o *Apache Spark*. *Picard* também é uma tecnologia do Broad Institute, ele é um conjunto de ferramentas de linha de comando para manipular dados e formatos de sequenciamento de alto rendimento (HTS), como SAM / BAM / CRAM e VCF.

GATK é um projeto *open-source* desenvolvido e mantido pelo Broad Institute disponibilizado no *GitHub* através do link abaixo:

<https://github.com/broadinstitute/gatk>

5.1.4 Sistema de Gerenciamento de workflows Cromwell

Cromwell é um sistema de gerenciamento de fluxo de trabalho voltado para fluxos de trabalho científicos. O *Cromwell* permite que pesquisadores, cientistas, desenvolvedores e analistas de genômica executem seus experimentos de maneira eficiente, sem a necessidade de profundo conhecimento dos recursos de computação de *back-end*.

Segundo o Broad Institute (2021a) e Fred Hutch (2020), *Cromwell* é uma ferramenta de *software* que pode ser usada para coordenar e agilizar a execução real de *workflows* bioinformáticos (ou outros analíticos). É o *software* que sustenta a plataforma Terra do Broad Institute. É uma das várias ferramentas e plataformas capazes de executar *workflows* escritos na linguagem WDL e outras ferramentas emergentes.

Cromwell é um projeto *open-source* desenvolvido e mantido pelo Broad Institute, disponibilizado no *GitHub* através do link abaixo:

<https://github.com/broadinstitute/cromwell>

5.1.5 Descrição dos parâmetros do *workflow* PIPE-MB

Um arquivo *JSON* (*JavaScript Object Notation*) é um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto Javascript (JSON ORG, 2021). É uma formatação leve de troca de dados. Está

baseado em um subconjunto da linguagem de programação *Javascript* (*Standard,* ECMA-262 3ª Edição, dezembro-1999). Estas são estruturas de dados universais, a qual praticamente todas as linguagens de programação modernas as suportam.

Um *JSON* é uma estrutura literal do tipo chave/valor. A Figura 7 apresenta um exemplo de um arquivo *JSON*. Nela está representado as informações de pessoas com os parâmetros ‘Nome’, ‘Idade’ e ‘Sexo’, os quais são chaves preenchidas com seus respectivos valores. Ressalta-se que em um arquivo *JSON* uma chave não pode se repetir, por isso, no referido exemplo são representados também as chaves “1” e “2” para diferenciar as informações das pessoas como um identificador único.

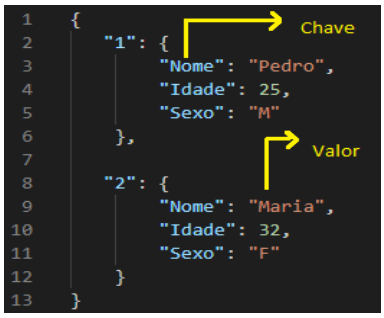


Figura 7. Exemplo básico do formato *JSON*.

O *workflow* PIPE-MB, espera receber um arquivo no formato *JSON*, no qual são preenchidos os seus parâmetros de entrada. Existem vários pontos de entrada para o *workflow* PIPE-MB, ele pode ser executado começando pelo primeiro processo (pré-processamento) ou diretamente executando a chamada de variantes, ou até fazendo somente a última etapa que é a anotação.

O arquivo *JSON* fornecido como entrada do workflow PIPE-MB é composto por diversos parâmetros que são divididos em grupos:

- *Type of Study*;
- Reference Genone;
- *Input Parameters*;
- *Output Parameters*;
- *Infra Parameters*;
- *Additional Parameters to execute data pre-processing;*
- *Additional Parameters to execute Germline study;*

- *Additional Parameters to execute Joint Genotyping for germline study;*
- *Additional Parameters to execute Somatic study;*
- *Additional Parameters for Somatic Panel of Normals (PON) Parameters;*
- *Germline annotation with Funcotator Parameters;*
- *Somatic annotation with Funcotator Parameters*;

O preenchimento do arquivo *JSON* de entrada do *workflow* PIPE-MB é composto por um genoma de referência em que é comparado com um ou mais genomas de interesse. Com objetivo de se estudar as variações do câncer, o processamento é feito, gerando resultados para que médicos e cientistas do INCA consigam fazer conclusões e análises a respeito de variações dos genomas de interesse.

No entanto, o preenchimento desse *JSON* para execução do *workflow* se torna inviável para usuários comuns do *workflow* PIPE-MB. Devido a grande quantidade de parâmetros, sua especialização em tarefas específicas, além da relação e dependência entre eles, torna-se muito difícil para usuários comuns conseguirem fazer uma adequada instanciação dos parâmetros do *workflow*.

Além disso, a especificação dos parâmetros, que é feita em formato *JSON*, não permite uma validação deles antes da execução, possibilitando inúmeros erros de especificação que somente serão vistos na hora da execução. Outro elemento que dificulta o uso dos parâmetros sem tratamento, é que muitos deles definem arquivos de entrada a partir de endereços absolutos no *storage* utilizado. Contudo, uma interface visual que ajude a especificar os parâmetros, melhoraria o seu gerenciamento, validaria as dependências e permitiria até o gerenciamento de *workspaces* dos usuários, tornando a seleção dos arquivos mais intuitiva do que ter que preencher os endereços (*path*) dos arquivos referenciados pelos parâmetros.

Então, o objetivo do projeto é fornecer uma interface adequada para que qualquer usuário tenha não só, um formulário para preenchimento dos parâmetros do *JSON*, mas também, tenha um login para acesso de um *workspace* próprio para o controle de estudos já realizados. Além disso, dar ao usuário a possibilidade de ter uma apresentação adequada para os resultados obtidos nos estudos gerados. Por fim, também é desejável monitorar as execuções do *workflow* através de uma interface mais usual, isso porque atualmente esse monitoramento é feito pelo console.

5.1.6 Representação gráfica do modelo do *workflow* PIPE-MB

Na Figura 8 é apresentada o Modelo Conceitual do *workflow* PIPE-MB. Basicamente o workflow é dividido em três etapas principais:

- Pré-processamento.
- Chamada de variantes Germinativas ou Somáticas.
- Validação das variantes encontradas (onde se aplica um filtro e se anota).

O pré-processamento é comum para chamada Germinativa e Somática. A etapa de filtro é específica para cada tipo de estudo. No caso de chamada de variantes germinativas é possível, se tiver números suficientes de amostras, realizar uma análise de coorte.

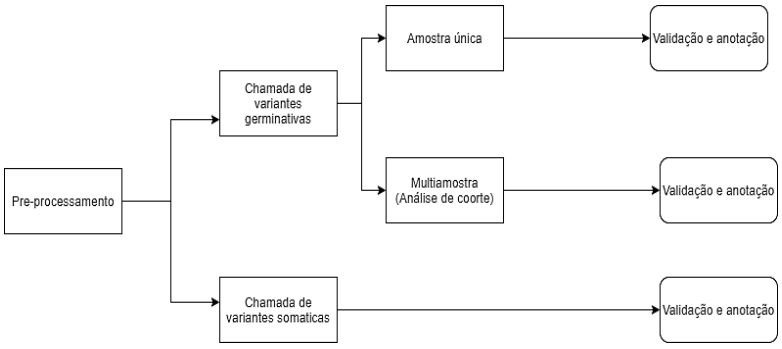


Figura 8. Modelo conceitual do *workflow*.

A Figura 9 é apresentando uma versão em desenvolvimento do modelo conceitual apresentado acima. Nela temos pequenas caixas que representam arquivos WDL que implementam cada parte do *workflow* PIPE-MB. Foram reaproveitadas algumas implementações disponíveis do GATK, readaptadas, estendidas e integradas. Um WDL principal gerencia a integração das outras partes do código, recebe os parâmetros e os distribuem para o código contido em outros arquivos, mediante a especificação dos parâmetros.

Na Figura 10 temos uma divisão em caixas da cor vermelha que correspondem a cada etapa conceitual do *workflow*. Os parâmetros foram nomeados de forma sugestiva para se corresponder a cada etapa, por exemplo, temos parâmetros que começam germline_x e somatic_x que se correspondem na mesma etapa, mas para cada estudo independente.

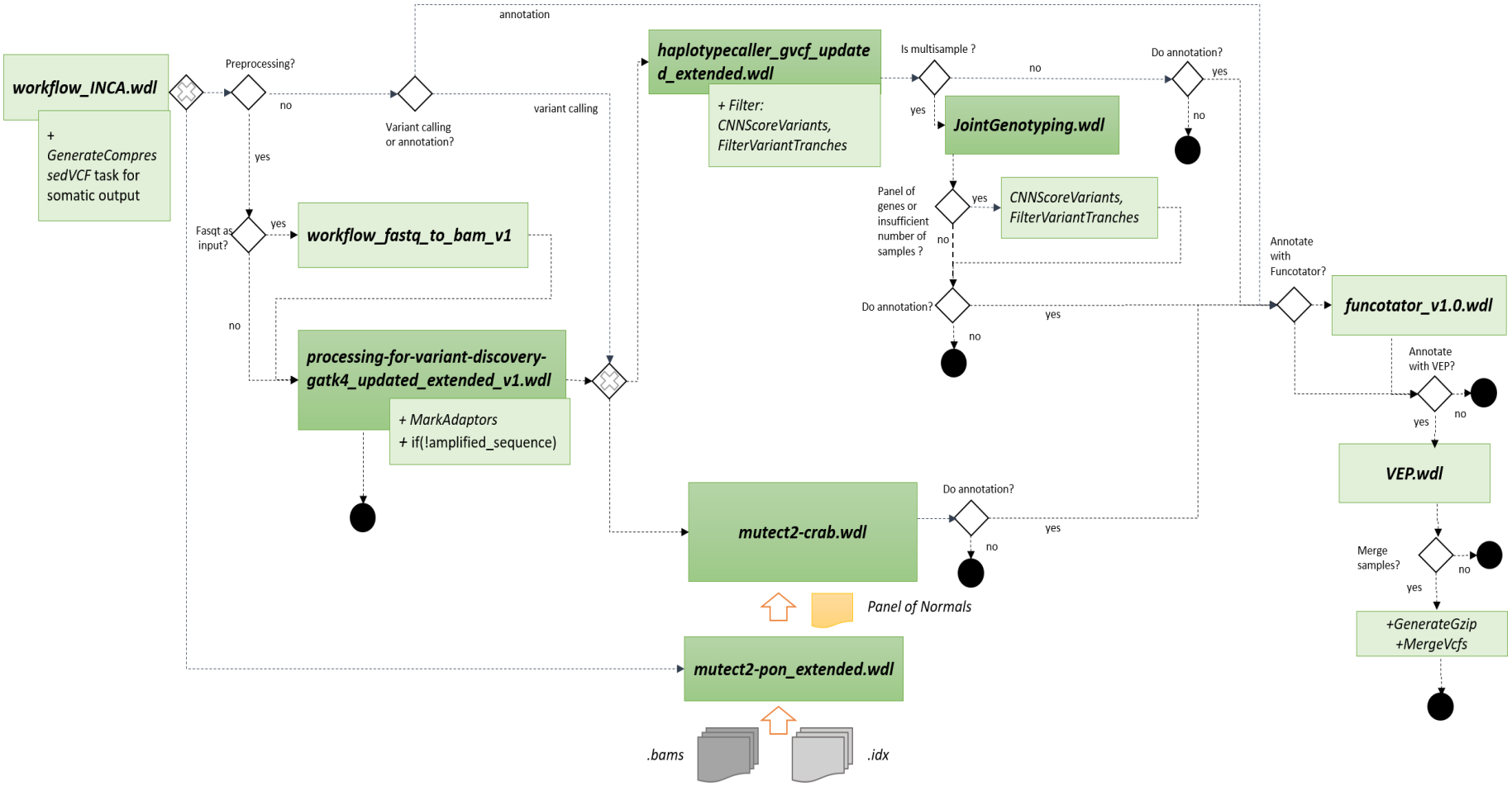


Figura 9. Representação gráfica do *workflow* PIPE-MB.

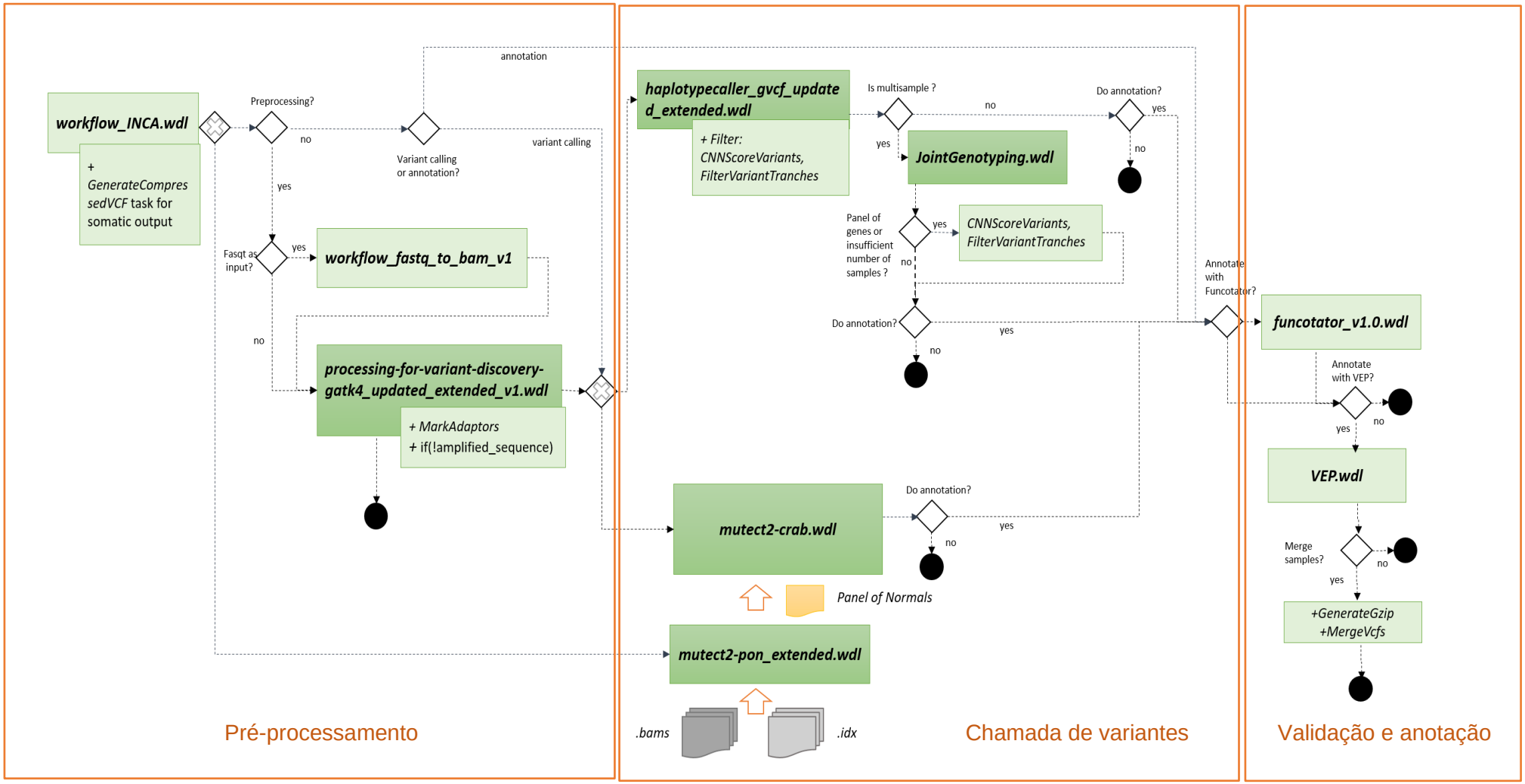


Figura 10. Divisão do *workflow PIPE-MB* correspondente a cada etapa conceitual.

5.2 **Projeto 2** - Desenvolvimento de Interface o SGWfC-Gene para Identificação de Potenciais Terapêuticos em Câncer Colorretal.

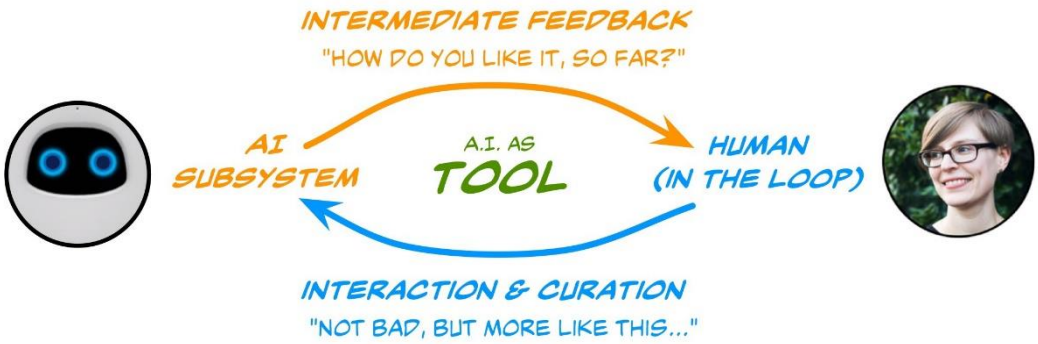
O SGWfC-Gene tem como objetivo a identificação de potenciais terapêuticos para tratamento de câncer colorretal. Câncer colorretal é um tumor maligno que se instala no reto do intestino grosso, sendo o segundo tipo de câncer de ocorrência mais frequente entre os homens (INCA, 2020), sendo assim, de grande importância a identificação de potenciais terapêuticos para o seu tratamento.

O projeto 2 aborda o desenvolvimento de interface para w*orkflow* SGWfC-Gene. Esse *workflow* tem como objetivo auxiliar e gerar respostas à uma pesquisa já em desenvolvimento, que envolve uma parceria da PUC-Rio com o INCA, a qual é coordenada pelo professor do Departamento de Informática da PUC-Rio, Sérgio Lifschitz. O projeto do *Workflow* SGWfC-Gene é um trabalho de pesquisa iniciado pelo Cristóvão Antunes de Lanna do INCA, no qual sua versão inicial foi implementada desenvolvendo módulos utilizando a linguagem R. Em seguida, houve uma parceria com Diogo Munaro Vieira (doutorando da PUC-Rio) que continuou o projeto e fez diversas alterações para se adequar a um *Workflow* convencional e reescreveu diversas rotinas inicialmente implementadas em R para a linguagem Python trazendo soluções inovadoras para o uso de *workflows* mais interativos para o usuários finais.

5.2.1 O SGWfC-Gene proposto

Sistemas gerenciadores de *workflows científicos* (SGWfC) tem sido cada vez mais comuns para a pesquisa em Bioinformática. No entanto, uma das desvantagens é que esses sistemas não conseguem dar um grau de iteratividade para o usuário interagir com o resultado gerado, a fim de dar continuidade no *workflow.* Diante desse problema, o desenvolvimento de *workflows* científicos mais interativos é de grande importância, uma vez que, permite que a execuções de tarefas científicas sejam realizadas por pessoas sem um conhecimento avançado em computação. Além de tornar o desenvolvimento científico mais acessível, também permite que pesquisadores de diferentes ramos da ciência reproduzam estudos publicados ou até façam novos estudos reaproveitando outros *workflows* previamente estabelecidos (VIEIRA et al., 2021). Nesse sentido,

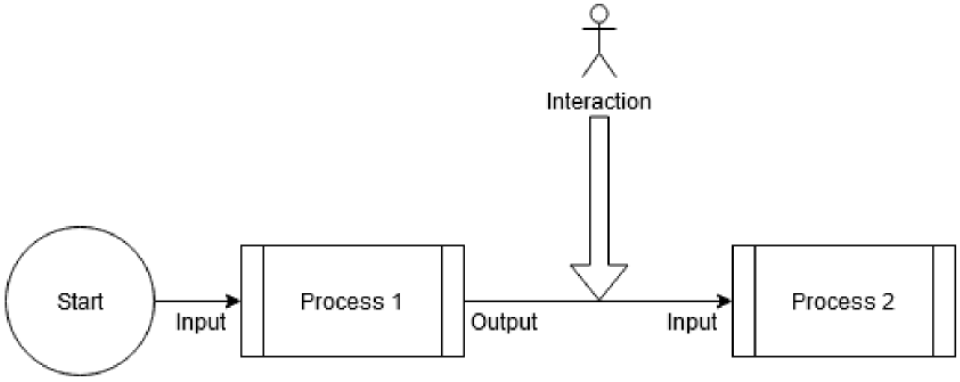
o presente projeto (Projeto 2) visa desenvolver uma arquitetura capaz de atender as demandas interativas do sistema, e assim, permitir o desenvolvimento de uma camada para usuários finais, dando maior flexibilidade para usuários interagirem diretamente com o SGWfC. Para atender essas demandas interativas do sistema foi utulizado os conceitos do modelo *Human-in-the-loop* (HITL) que permite que o usuário altere o resultado de um evento ou processo, na Figura 11 é demostrado uma ilustração desse conceito. Aprendizagem HITL descreve uma configuração pela qual uma máquina de aprendizagem ou sistema de computador pode incorporar entradas humanas selecionadas ou rótulos em suas inferências ou processo de aprendizagem e, assim, cria um ciclo virtuoso ou '*loop*' em que a máquina constantemente aprende a melhorar suas capacidades. O HITL está se tornando um método cada vez mais usado, embora ainda bastante esquecido, no avanço dos sistemas de IA de hoje - talvez não ajudado pela dicotomia popular entre as tarefas serem totalmente manuais ou totalmente automatizadas. No entanto, os designers de sistemas estão cada vez mais percebendo que a automação total não deve ser necessariamente o objetivo: que às vezes os melhores resultados são alcançados através de humanos e máquinas cooperando - com humanos (pelo menos aqueles com os tipos certos de conhecimento) interagindo com os (tipos certos de) IA (STACEY, 2021).



Fonte: (STANFORD UNIVERSITY, 2019).

Figura 11. Representação do conceito *Human-in-the-loop*.

Para uma maior interatividade, o SGWfC precisa de uma interface mais intuitiva. A Figura 12 apresenta um exemplo de funcionamento entre a interação do usuário e dois processos do *workflow.*



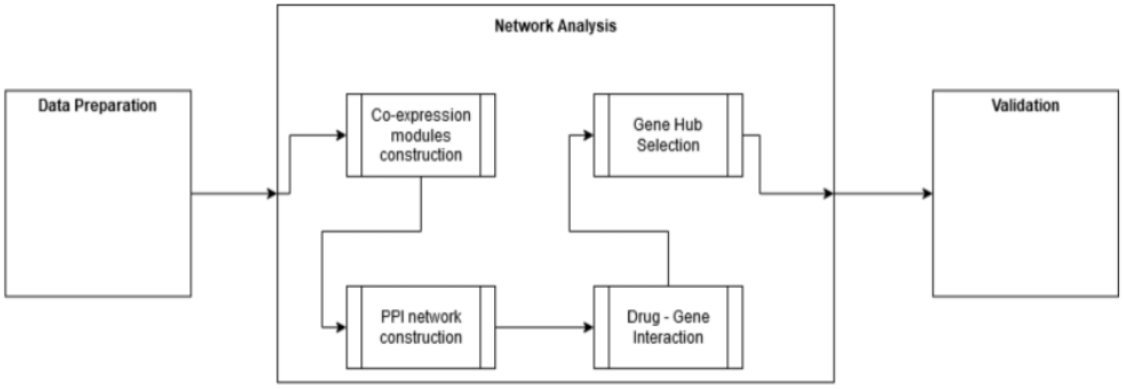
Fonte: VIEIRA et al., (2021).

Figura 12. Funcionamento entre a interação do usuário e dois processos do *workflow.*

No desenvolvimento de um *workflow*, primeiramente é necessário organizar estrategicamente como serão as etapas a serem executadas por ele, bem como, a interação entre cada uma dessas etapas. Dessa forma, além da organização do *workflow* abstrato, a padronização de cada etapa, irá favorecer sua utilização para escalar cada processo de forma mais simples.

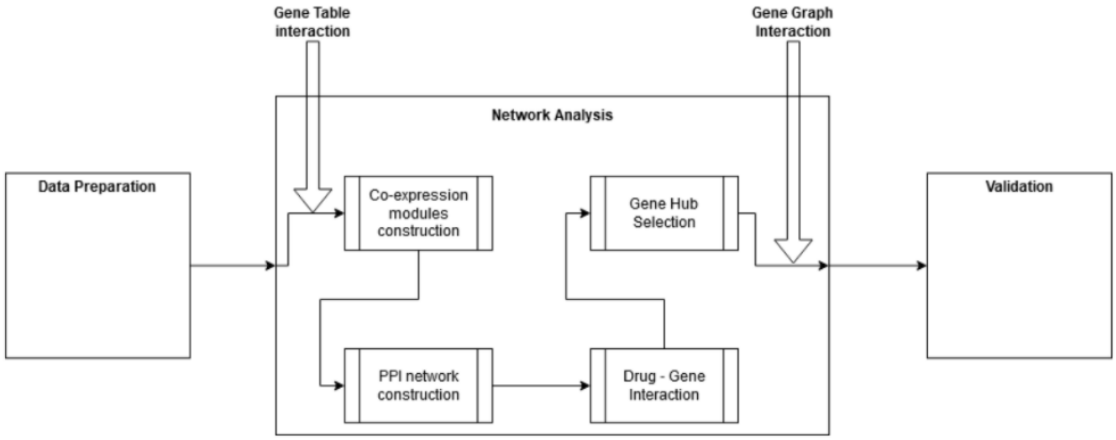
Inicialmente, somente uma componente *workflow* abstrato do Projeto 2 foi desenvolvido de forma completa. Conforme ilustrado na Figura 13 o componente de N*etwork Analysis* possui vários passos que podem ser exatamente um ou vários processos. Assim, a partir do desenvolvimento desse *workflow* abstrato é possível explorar alguns pontos de interação em cada uma das etapas e pensar em como essas interações serão feitas do ponto de vista do usuário do *workflow* (VIEIRA et al., 2021).

Conforme é mostrado na Figura 14 dois pontos de interações principais foram desenvolvidos no projeto. O primeiro ponto, quando os genes surgem do *Data Preparation*, o usuário deverá ter a possibilidade de remover genes de acordo com alguma regra de negócio já estabelecida. Já o segundo ponto, ocorre no final do componente. Nessa etapa, os genes passam por todos os processamentos de análise de rede de interação e alguns grafos são gerados, dando ao usuário a possibilidade de escolher os melhores pontos.



Fonte: VIEIRA et al., (2021).

Figura 13. *Workflow* abstrato apresentando somente o componente *Network Analysis* expandido.



Fonte: VIEIRA et al., (2021).

Figura 14. Pontos de interação com o workflow abstrato.

Uma vez que, a interação não é uma funcionalidade e, também não é uma responsabilidade do SGWfC, optou-se por desenvolver uma interface *web* utilizando Django e React JS. Por serem independentes do SGWfC, a escolha dessas tecnologias possibilita qualquer interação de ser implementada e hospedada separadamente, e assim, simplificar o acesso do usuário final. É exatamente nessa interface de interação - *Gene Graph Interaction,* que foi desenvolvido o Projeto 2, a fim de ter uma interface adequada para que usuários finais pudessem visualizar e escolher os melhores grafos de interação gerados.

5.2.2 A escolha do *workflow Prefect*

Para que essas interações sejam possíveis, foi necessário a escolha de um *SGWfC* que atendesse as demandas para essa funcionalidade, bem como de outros requisitos do projeto.

Vieira et al., (2021) avaliou cinco diferentes SGWfC em diferentes requisitos (Integração Externa, Agendado Escalável, Cache de Tarefa, Paralelismo e Controle de Versão), sendo eles: *Airflow, Nextflow, Galaxy, Prefect, Luigi, Toil e Cromwell*. O resultado da comparação dos SGWfCs promoveu a escolha do *Prefect* para o desenvolvimento do projeto, uma vez que, foi o SGWfC que mais se adequou aos requisitos esperados.

5.2.3 Arquitetura do *Workflow* SGWfc-Gene

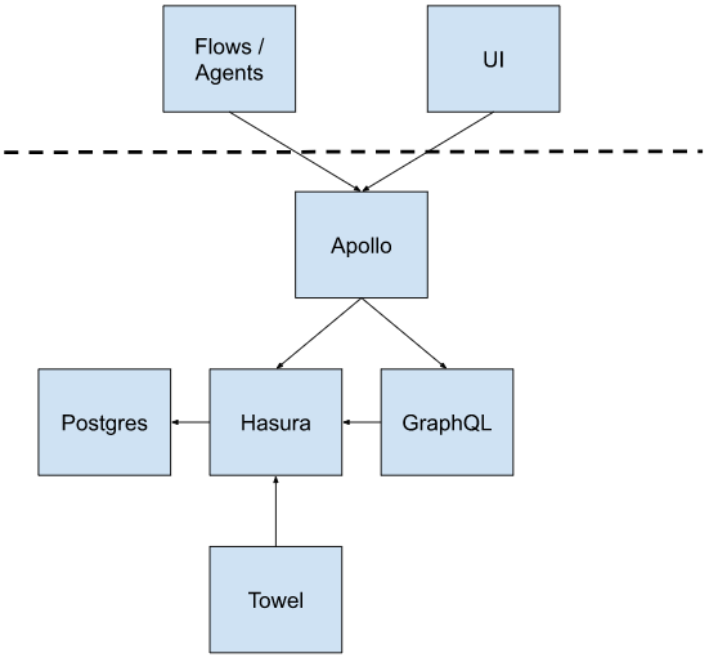
A arquitetura do *Prefect* é composta por sete elementos principais. Abaixo é apresentado uma breve descrição de cada elemento:

- **UI**: Uma interface web com *dashboards* e acompanhamento de dados sobre os *workflows* registradores.
- **Flows/Agents**: São *workflows* independentes ou agentes distribuídos que rodam os *workflows*.
- **Apollo**: API para interação como o servidor que gerencia os *workflows*.
- **PostgreSQL**: Banco de Dados para armazenar metadados sobre *o workflow*.
- **Hasura**: Projeto que cria uma API GraphQL em cima do *PostgreSQL*.
- **GraphQL**: API com mutações de lógica de negócio definidas.
- **Towel**: Serviço que cuida de agendamento de *workflows* e o seu monitoramento.

Prefect é um projeto *open-source* sua documentação completa é disponibilizada no através do link abaixo:

<https://docs.prefect.io/orchestration/server/architecture.html>

A Figura 15 apresenta a representação da arquitetura do *Prefect* por meio de um diagrama. Como podemos observar, para que o sistema que guarda as informações de *workflows* permaneça seguro, há a necessidade de autenticação entre a UI e o *Apollo*, assim como, entre os Agentes e o *Apollo* .



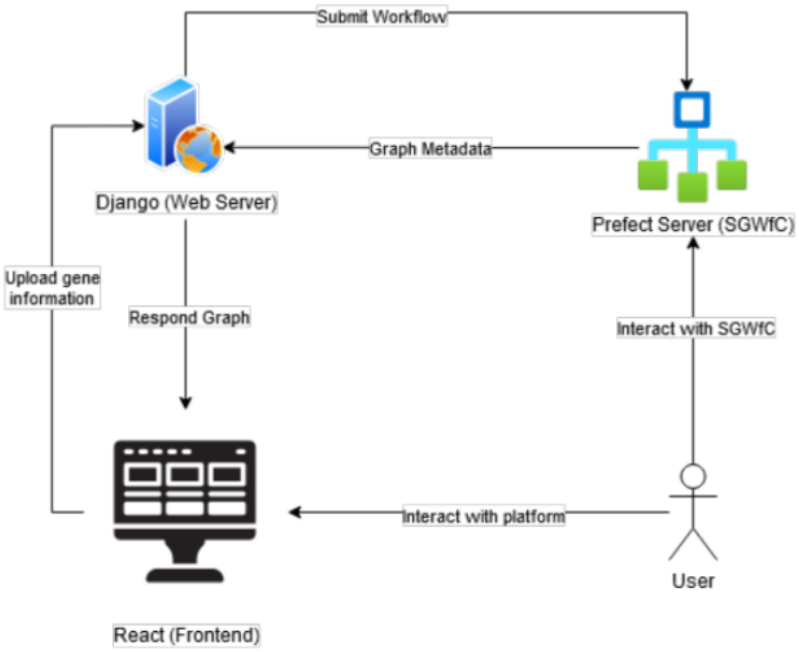
Fonte: Prefect Docs

Figura 15. Arquitetura do *Prefect*.

Uma vez estabelecido a arquitetura do *SGWfC* com a opção de operar *workflow* manualmente, é preciso adicionar agora como essa interação é feita nas saídas das tarefas.

O *Django Framework* foi escolhido como platataforma de desenvolvimento web para as interações com o *workflow*. Além disso, foi escolhido o *React* como tecnologia de *front-end*, pela sua simplicidade e por ter alguns componentes visuais open-source disponíveis para vizualização de grafos (a exemplo, cytoscape-react). O *Cytoscape* foi escohlido para manipular grafos pois, possui a mesma interface tanto para *Javascript,* como para exportação de saída no *Python* pelo *NetworkX*.

A Figura 16 representa a interação entre o usuário e o SGWfC. O usuário pode interagir através da UI do SGWfC somente executando o *workflow* ou utilizar a interface que permite interatividade entre os passos do *workflow*. Caso a opção seja utilizar a interface interativa, essa interface se comunica diretamente com o servidor *Django* (*Web Server*) que faz a requisição ao *Prefect* (SGWfC) para que ele execute aquele passo do *workflow*. Após a execução, o SGWfC retorna ao *Web Server* a resposta com ao dados de processamento e os grafos resultantes. Esses grafos são enviados ao *React* (*fron-end*) que renderiza o resultado no navegador do usuário por meio do componente *Cytoscape* (Vieira et al, 2021).



Fonte: Vieira et al., (2021)

Figura 16. Representação de interação entre o usuário e o *workflow*.

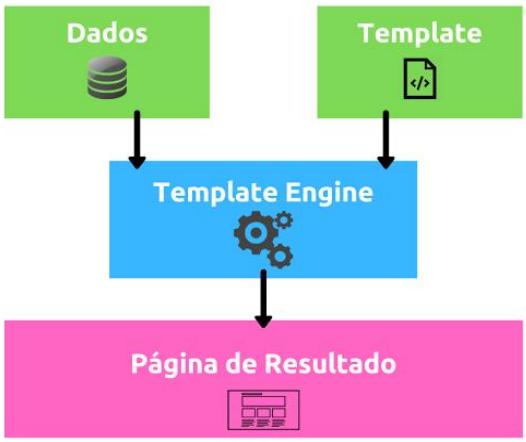
No Projeto 2, além dos conhecimentos aplicados e desenvolvidos no Projeto 1, também tive a oportunidade de estudar novas ferramentas que não tinha conhecimento. Dentre elas, o *framework React,* para o desenvolvimento do *front-end, e o Redux,* para gerenciar os estados dos componentes do *React.* Além disso, foi necessário o estudo da ferramenta *Cytoscape, a qual* é uma plataforma de *software* para bioinformática para visualizar grafos de interações moleculares e, também o funcionamento do *workflow Prefect*.

6. Implementação e avaliação

6.1 Implementação do Projeto 1 (*workflow* PIPE-MB)

No tópico 5.1.6 foram especificados os tipos de estudos possíveis a serem realizados pelo *workflow* PIPE-MB. O desenvolvimento da interface web responsável pelos preenchimentos dos parâmetros de entrada do *workflow* foi a primeira etapa de desenvolvimento iniciada.

Esse projeto foi desenvolvido utilizando o padrão MTV (*Model*, *Template*, *View*) do *Django Framework,* no qual usamos o conceito de *Template Engine* ou View Engine, em que é possível gerar páginas HTML ou trechos de códigos HTML a partir do *back-end* para serem renderizados no navegador. A Figura 17 apresenta a representação da estrutura do *Template Engine*. No *Django* usamos o *Template Engine* chamado de *Django Template*, que foi utilizado para gerar toda a estrutura do formulário da página *web* de entrada dos parâmetros do *workflow*.



Fonte: (TreinaWeB)

Figura 17. Estrutura do *Template Engine*.

A primeira etapa do desenvolvimento dessa página web de entrada dos parâmetros do *workflow* foi a especificação de cerca de 170 parâmetros de entrada diferentes (Figura 18), os quais deveriam ser mostrados na tela de acordo com a opção de estudo selecionada pelo usuário. A definição e validação desses parâmetros foi bastante trabalhosa pelo grande número de parâmetros necessários. Cada um desses parâmetros foi definido através do arquivo da classe ‘F*orm’* do *Django,* no qual é possível especificar o tipo do parâmetro (*char*, *boolean*, *integer*, *float*, entre outros) e outras informações relevantes para os campos de formulários HTML tais como *label*, tipo de *input* HTML, se o campo é obrigatório ou não, e também possíveis atributos adicionais.

Figura 18. Especificação de alguns dos parâmetros de entrada utilizado no *workflow*.

Depois da especificação dos 170 parâmetros de entrada do *workflow* no arquivo '*forms.py*' do *Django*, foi necessário a inserção da chamada dos mesmos na página HTML, que irá renderizar esses campos do formulário através do *Django Template*. A Figura 19 mostra como a chamada foi implementada.

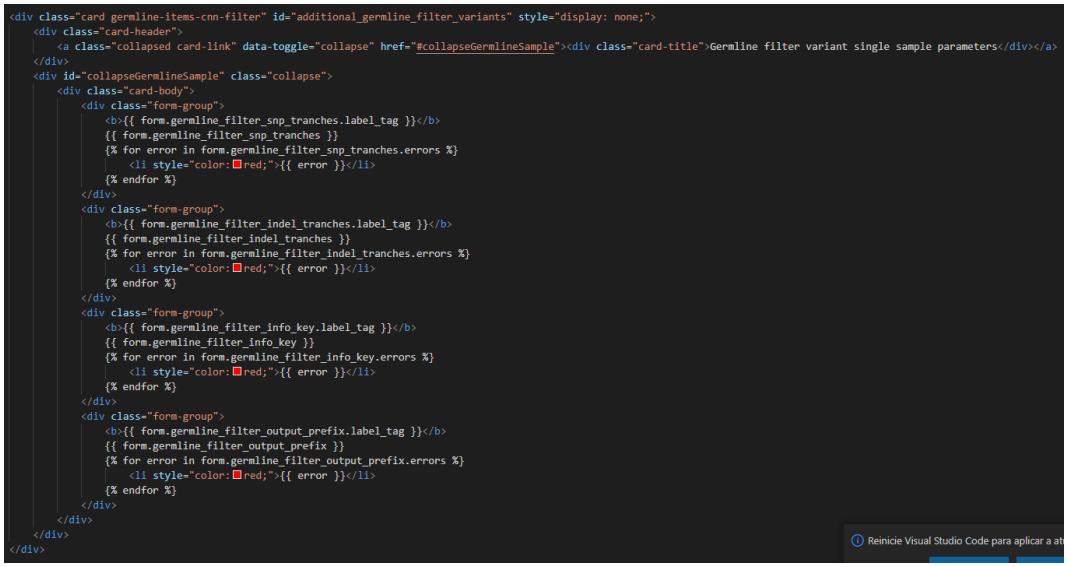


Figura 19. Definição dos parâmetros especificados na página HTML.

6.1.1 Primeira versão da página *web* de entrada do *workflow* PIPE-MB

Depois da definição dos parâmetros de entrada foi possível visualizar uma primeira versão da *página web*. Essa versão da tela de entrada do *workflow* é composta por um formulário dividido em caixas dinâmicas, as quais são posicionadas em sequência.

Essas caixas apresentam a opção de expandir ou colapsar, além de estar visível ou não, de acordo com os parâmetros marcados pelo usuário, como mostrado nas Figuras 20 e 21.

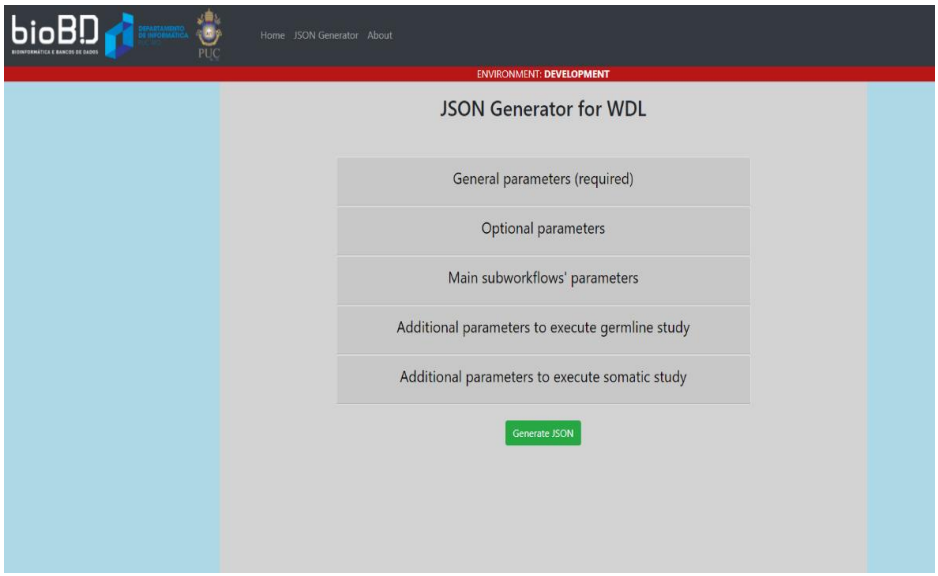


Figura 20. Interface dos parâmetros de entrada do *workflow* (caixas colapsadas).

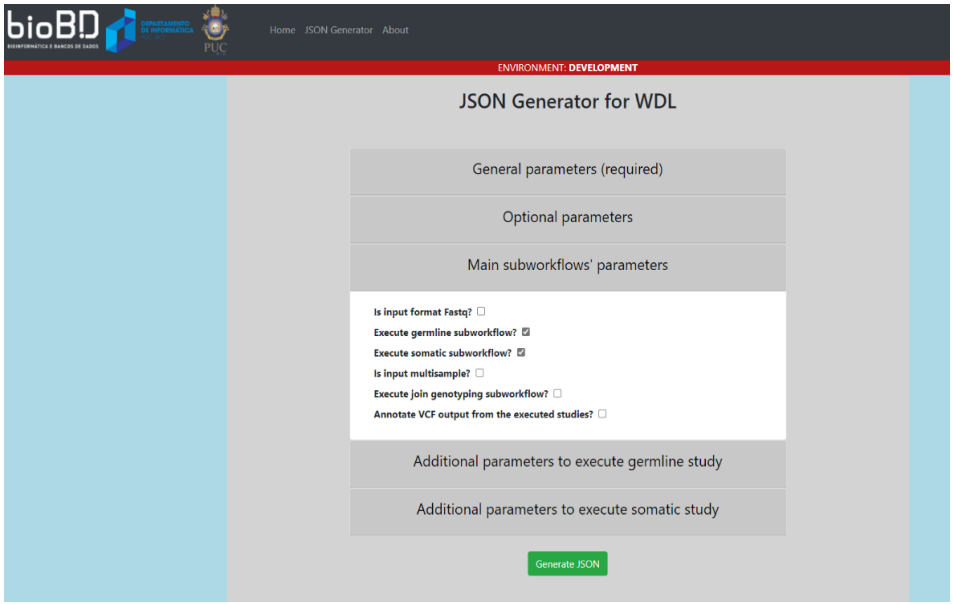


Figura 21. Interface dos parâmetros de entrada do *workflow* (caixas expandidas).

A Figura 22 apresenta algumas mensagens obtidas no console do *Devtools* do Chrome, após uma tentativa de submissão do formulário que apresentavam alguns parâmetros obrigatórios e não preenchidos em caixas ocultas ou colapsadas.

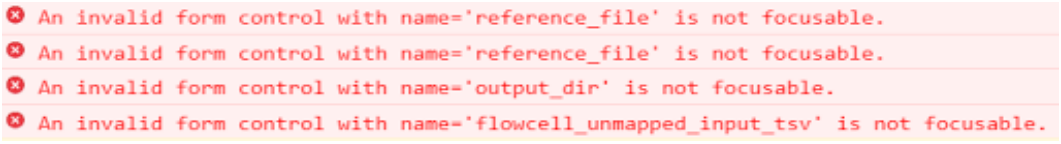


Figura 22. Mensagem de erro no console do *Devtools* do Chrome.

Esse evento ocorria, pois no momento da definição dos parâmetros por meio do '*forms.py*' do *Django,* estabelecemos um estado de obrigatoriedade ou não dos campos de maneira estática, independentemente do tipo de estudo escolhido pelo usuário no momento do preenchimento do formulário. No entanto, a obrigatoriedade de um campo deveria ser dinâmica de acordo com o tipo de estudo selecionado pelo usuário.

Uma solução para contornar esses problemas, seria o desenvolvimento de uma lógica de controle de obrigatoriedade ou não dos campos, por meio do *Javascript* no *front-end, no entanto, isso* aumentaria ainda mais a complexidade do tratamento dos 170 parâmetros, os quais poderiam apresentar comportamento diferentes. Uma solução alternativa seria uma refatoração sobre a definição dos parâmetros de entrada no *front-end* para a utilização de tecnologias como *React,* com algum Gerenciador de Estado como o *Redux,* para controlar o dinamismo de obrigatoriedade dos campos do formulário. Entretanto, chegou-se à decisão de deixar a princípio como responsabilidade do usuário o preenchimento ou não de grande parte dos parâmetros de entrada.

Após algumas reuniões com os usuários do INCA sobre a primeira versão da página web de entrada, chegou-se à conclusão de que essa estrutura citada não era a mais adequada para apresentar os parâmetros para todos os tipos de usuários em níveis diferentes de conhecimento. Além disso, ficava muito difícil a visualização de todos os campos que estavam sendo preenchidos (Figura 8). Sendo assim, direcionamos o trabalhado para uma nova proposta de *layout* para a apresentação dos parâmetros, a fim de atender melhor os diferentes usuários do INCA.

6.1.2 Segunda versão da página *web* de entrada do *workflow* PIPE-MB

Perante as dificuldades encontradas na primeira versão da página *web* de entrada do *workflow* PIPE-MB, como por exemplo a forma no qual eram exibidas as caixas dos parâmetros em sequência: a página poderia se expandir muito na horizontal o que dificultaria a visualização e o controle pelo usuário sobre os parâmetros no qual estavam sendo preenchidos.

Buscamos então, uma nova estrutura da página *web* de parâmetros de entrada (*JSON*) do *workflow* PIPE-MB. Nessa nova estrutura, temos uma organização melhor e mais compacta da exibição dos parâmetros a serem preenchidos pelo usuário. A proposta é uma estrutura bem mais compacta corrigindo os problemas encontrados na primeira versão, como mostrado na Figura 23.

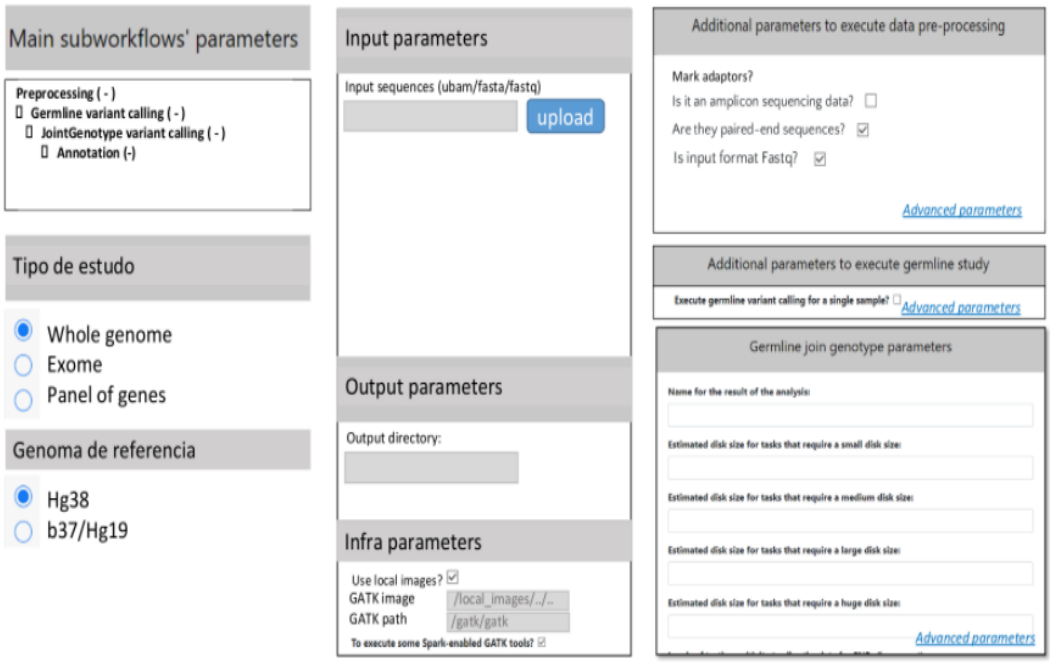


Figura 23. Nova estrutura de *layout* para especificação dos parâmetros de entrada do *workflow*.

Seguindo a nova proposta de *layout*, até o momento foram feitas mudanças parciais para chegar na versão final da nova estrutura, conforme mostrado na Figura 24.

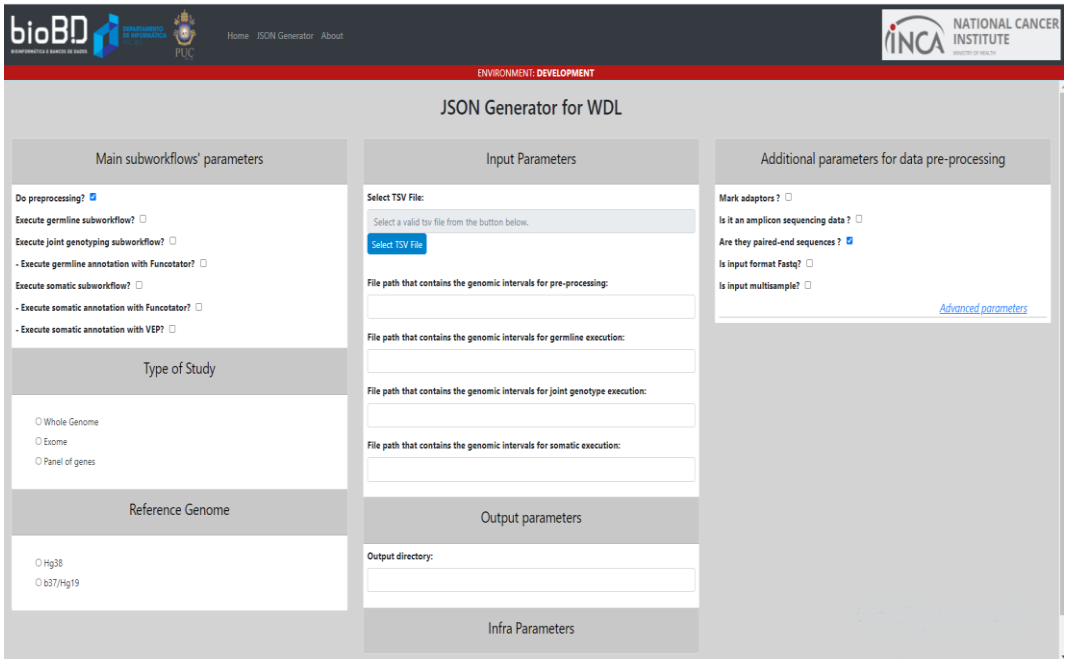


Figura 24. *Layout* da versão atual da interface de entrada do *workflow*.

Como podemos observar (Figura 24), a segunda opção de *layout* ficou mais compacta e centralizada para configuração. Nela, temos uma exibição inicial padrão, com a expansão de algumas caixas de configuração e de outras não. Assim, de acordo com a opção de estudo selecionada pelo usuário, na seção '*Main subworkflows parameters',* cada tipo de caixa ou itens correspondentes ao estudo selecionado é expandida e fica visível ao usuário, já caixas e itens que não estão relacionados ao tipo de estudo escolhido pelo usuário, ficam ocultos.

6.1.3 Modularização do código-fonte Javascript do projeto

Inicialmente, todo código *Javascript front-end* estava sendo implementado em um único arquivo. No entanto, conforme o projeto foi crescendo, tornou-se necessário a divisão do *Javascript* em módulos, com um módulo para cada tela. Uma vez que o projeto está sendo desenvolvido utilizando a biblioteca *JQuery* no *front-end*, foi então adotado um padrão '*Core-Presenter-Controller*' que pode ser visto como um similar ao padrão *MVP* (*Model-View-Presenter*), o qual tem a finalidade de separar a camada de apresentação das camadas de regras de negócio. Abaixo é apresentado as funções de cada objeto do módulo seguindo o padrão 'Core-Presenter-Controller':

- **Core**: toda parte lógica do módulo. É um objeto *Javascript* que recebe como dependência uma instância do objeto *Presenter* e do objeto *Controller* na sua inicialização e, também pode receber instâncias de outros objetos que sejam necessários em sua lógica de implementação.

- **Presenter**: é onde se encontra toda parte de manipulação de eventos *Javascript* associados a iteração com os elementos da página web HTML com acesso direto ao DOM do HTML para adição ou remoção de elementos da tela, alteração de atributos de elementos da tela, visibilidade dos elementos e outras aplicações relacionadas aos elementos da página.

- **Controller**: é onde as funções responsáveis por realizar requisições *Ajax* ao servidor web estão implementadas. Ele faz com que o módulo receba e envie dados ao servidor *web* e retorne as respostas para alguma função *call-back* do objeto '*Core*' do módulo passado como referência, e assim, essa função pode fazer o tratamento adequado para essas respostas do servidor *web*.

Como podemos observar na Figura 25, foi criado um arquivo chamado '*init.js*', no qual é instanciado os módulos e em seguida inicializados.

Figura 25. Definição do arquivo de inicialização dos módulos.

Podemos verificar na Figura 25 que foram instanciados dois módulos '*EntryWorkFlow*' e '*InputTSVFile*', no qual são passadas referências dos objetos necessários para suas lógicas de implementação. Verificamos que em seguida é chamado o método 'init()' implementado em a cada um desses módulos ('*EntryWorkFlow*' e '*InputTSVFile*') para inicializá-los e carregar os eventos e o estado inicial dos mesmos.

Além disso, é necessário que os arquivos *Javascript* de cada módulo sejam carregados no arquivo HTML da página que ele vai ser utilizado. A ordem de chamada desses arquivos referentes ao carregamento desses módulos é importante na chamada HTML, pois essa leitura é feita de forma sequencial e seguindo a ordem de carregamento correta evitamos erros de chamadas de variáveis que ainda não foram definidas, ou seja, os módulos devem ser carregados antes do arquivo 'init.js' para que eles sejam instanciados depois de suas definições, como podemos ver na Figura 26.

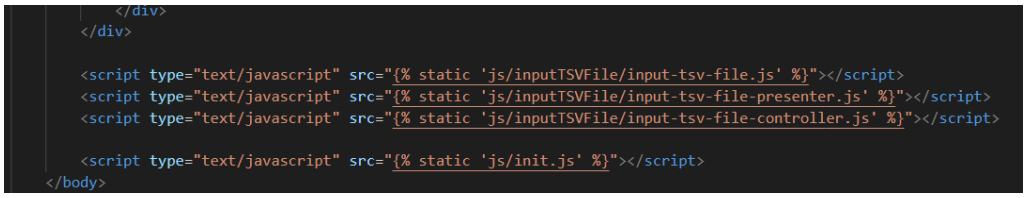


Figura 26. Carregamento dos arquivos Javascript no arquivo HTML.

Uma opção viável seria usar o *Webpack* que serve como um empacotador de módulos estáticos para aplicações *Javascript*. Com ele é possível uma melhor gerência dos pacotes e, também, a opção de juntá-los em um único arquivo *Javascript* de forma minificada (processo de remover elementos desnecessários e reescrever o código para reduzir o tamanho do arquivo) no momento do carregamento da página HTML.

6.1.4 Visualização dos itens da tela de acordo com tipo de estudo escolhido

Como já mencionado anteriormente, somente caixas ou itens correspondentes ao estudo selecionado é expandida e fica visível ao usuário. Para isso, foi implementando no módulo '*EntryWorkFlow*' a lógica de visibilidade dos itens e caixas na tela. A cada evento de clique do *mouse* em um dos *checkboxs* da página HTML (sessão '*Main subworkflows parameters'),* é acionado uma das funções mostrada na Figura 27. Essas funções são passadas como *call-back* para cada uma das opções de estudos possíveis e, então, é

passado para essa função o estado do *checkbox* como um parâmetro booleano, informando se, ele está ou não, marcado na suposta ação de clique do usuário. Para cada um desses estados é acionado um *toggle* do *Jquery* (biblioteca *Javascript*), que permite a alternância de visibilidade do item na página HTML.

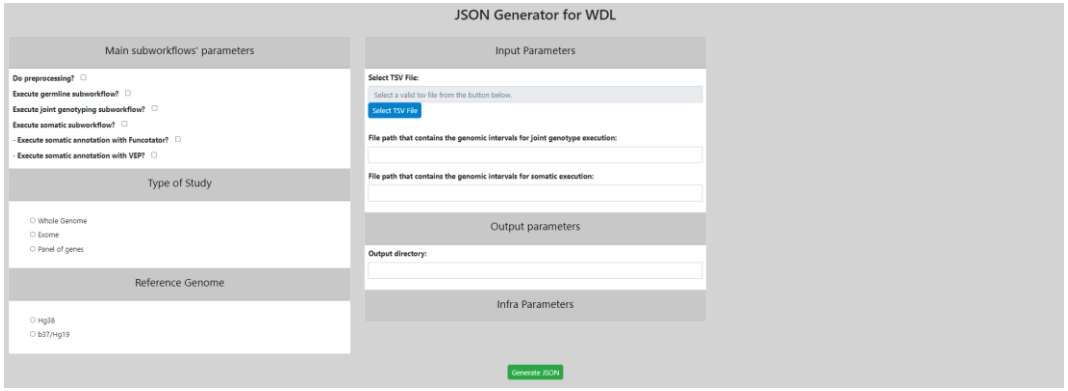
Por meio dessa função *toggle do Jquery* é possível associar itens específicos a cada tipo de estudo referenciado através de marcadores de classes e IDs como propriedades dos elementos HTML, e assim, alternar a visibilidade dinamicamente dos elementos do formulário de acordo com o tipo de estudo selecionado pelo usuário no formulário. Isso simplificou bastante o processo de preenchimento desses campos do formulário, diminuindo a quantidade de parâmetros visível ao usuário.

```
/* Show/Hide itens Do Preprocessing */
displayDoPreprocessingItens: function(isChecked){
    var isVisible = isChecked;
    $('.preprocessing-items').toggle(isVisible);
    $('#preprocessing_intervals_list_item_form').toggle(isVisible);
},
/* Show/Hide itens Execute Germline Subworkflow */
displayExecuteGermlineItens: function(isChecked){
    var isVisible = isChecked;

    $('.germline-items').toggle(isVisible);
    $('#main-subworkflow-execute-germline-annot-funcotator').toggle(isVisible);
    $('#main-subworkflow-execute-germline-annot-vep').toggle(isVisible);
    $('#germline_intervals_list_item_form').toggle(isVisible);
},
/* Show/Hide itens Execute Germline Annotation Funcotator Subworkflow */
displayExecuteGermlineAnnotFuncotatorItens: function(isChecked){
    var isVisible = isChecked;
    $('.germline-items-annotation-funcotator').toggle(isVisible);
},
/* Show/Hide itens Execute Somatic Annotation Funcotator Subworkflow */
displayExecuteSomaticAnnotFuncotatorItens: function(isChecked){
    var isVisible = isChecked;
    $('.somatic-items-annotation-funcotator').toggle(isVisible);
},
/* Show/Hide itens Execute Germline Annotation Funcotator Subworkflow */
displayExecuteGermlineAnnotVEPItens: function(isChecked){
    var isVisible = isChecked;

    $('.germline-items-annotation-vep').toggle(isVisible);
},
/* Show/Hide itens Execute Somatic Annotation VEP Subworkflow */
displayExecuteSomaticAnnotVEPItens: function(isChecked){
    var isVisible = isChecked;
    $('.somatic-items-annotation-vep').toggle(isVisible);
},
/* Show/Hide itens Execute Joint Genotyping Subworkflow */
displayExecuteJointGenotypingItens: function(isChecked){
    var isVisible = isChecked;
    $('.joint-genotyping-items').toggle(isVisible);
    $('#main-subworkflow-execute-germline-annot-funcotator').toggle(isVisible);
    $('#main-subworkflow-execute-germline-annot-vep').toggle(isVisible);
    $('#jointgenot_intervals_list_item_form').toggle(isVisible);
},
/* Show/Hide itens Execute Somatic Subworkflow */
displayExecuteSomaticItens: function(isChecked){
    var isVisible = isChecked;
    $('.somatic-items').toggle(isVisible);
    $('#main-subworkflow-execute-somatic-annot-funcotator').toggle(isVisible);
    $('#main-subworkflow-execute-somatic-annot-vep').toggle(isVisible);
    $('#somatic_intervals_list_item_form').toggle(isVisible);
},
```

Figura 27. Implementação de funções Javascript para controle de visibilidade dos itens na tela.

Por meio dessa funcionalidade, deixamos a tela de preenchimento do formulário de entrada do *workflow* mais flexível e compacta para uso do usuário (Figura 28).

a)



b)

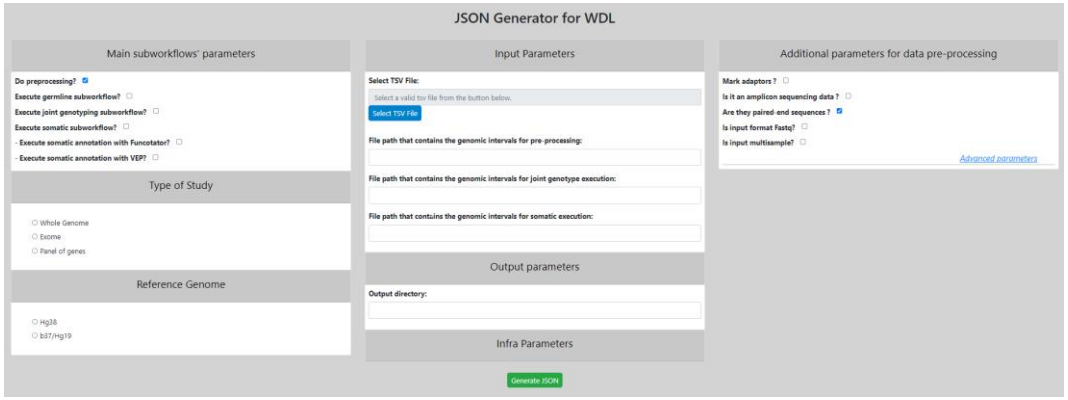


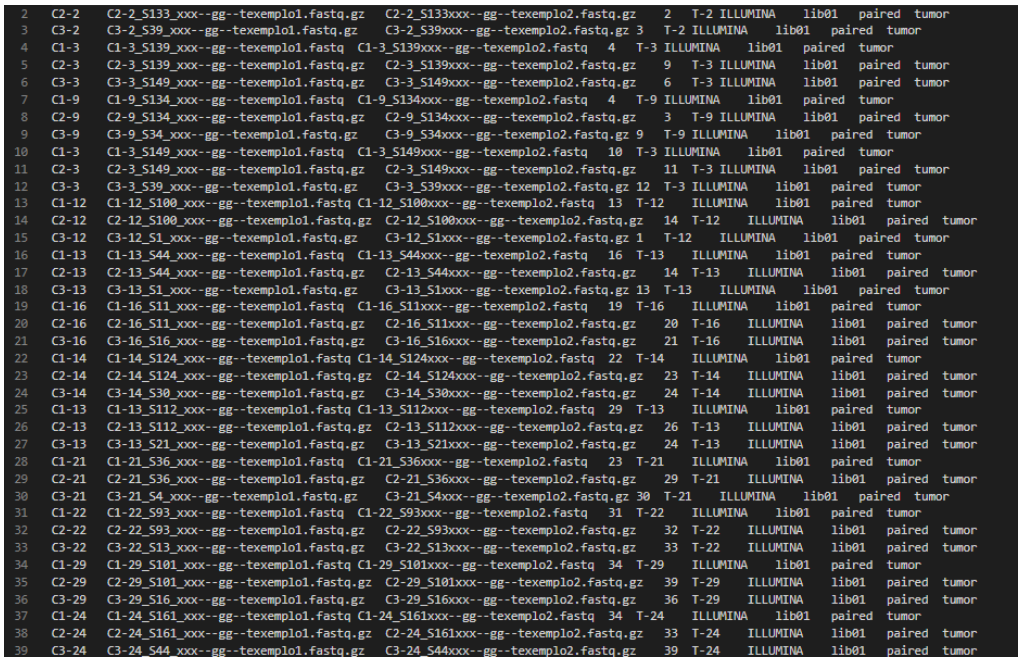
Figura 28. Versão atual da tela de entrada da página web do workflow PIPE-MB. a) estado em que nenhum tipo de estudo é selecionado e, b) estado em que o pré-processamento é selecionado.

6.1.5 Implementação da Tela de Gerenciamento de Arquivo TSV

Um arquivo TSV é um arquivo de valores separados por tabulação. É um formato de arquivo bastante utilizados em dados matemáticos, científicos ou estatísticos. Na bioinformática é comum o uso de arquivo TSV para armazenar dados de amostras genéticas.

Uma dificuldade relatada pelos usuários do INCA, refere-se à edição dos arquivos TSV para o usuário comum. Há certa dificuldade para edição e pareamento de amostras, uma vez que, esse processo era feito diretamente no arquivo TSV tabular. Como é mostrado na Figura 29 o processo de edição do arquivo se torna difícil para usuários comuns e com grande risco de erro durante

sua edição em um editor de texto comum, além de possuir uma visualização ruim dos dados.



C2-2	C2-2_S133_xxx--gg--texemplo1.fastq.gz	C2-2_S133xxx--gg--texemplo2.fastq.gz	2	T-2	ILLUMINA	lib01	paired	tumor		
C3-2	C3-2_S39_xxx--gg--texemplo1.fastq.gz	C3-2_S39xxx--gg--texemplo2.fastq.gz	3	T-2	ILLUMINA	lib01	paired	tumor		
C1-3	C1-3_S139_xxx--gg--texemplo1.fastq	C1-3_S139xxx--gg--texemplo2.fastq	4	T-3	ILLUMINA	lib01	paired	tumor		
C2-3	C2-3_S139_xxx--gg--texemplo1.fastq.gz	C2-3_S139xxx--gg--texemplo2.fastq.gz	9	T-3	ILLUMINA	lib01	paired	tumor		
C3-3	C3-3_S149_xxx--gg--texemplo1.fastq.gz	C3-3_S149xxx--gg--texemplo2.fastq.gz	6	T-3	ILLUMINA	lib01	paired	tumor		
C1-9	C1-9_S134_xxx--gg--texemplo1.fastq	C1-9_S134xxx--gg--texemplo2.fastq	4	T-9	ILLUMINA	lib01	paired	tumor		
C2-9	C2-9_S134_xxx--gg--texemplo1.fastq.gz	C2-9_S134xxx--gg--texemplo2.fastq.gz	3	T-9	ILLUMINA	lib01	paired	tumor		
C3-9	C3-9_S34_xxx--gg--texemplo1.fastq.gz	C3-9_S34xxx--gg--texemplo2.fastq.gz	9	T-9	ILLUMINA	lib01	paired	tumor		
C1-3	C1-3_S149_xxx--gg--texemplo1.fastq	C1-3_S149xxx--gg--texemplo2.fastq	10	T-3	ILLUMINA	lib01	paired	tumor		
C2-3	C2-3_S149_xxx--gg--texemplo1.fastq.gz	C2-3_S149xxx--gg--texemplo2.fastq.gz	11	T-3	ILLUMINA	lib01	paired	tumor		
C3-3	C3-3_S39_xxx--gg--texemplo1.fastq.gz	C3-3_S39xxx--gg--texemplo2.fastq.gz	12	T-3	ILLUMINA	lib01	paired	tumor		
C1-12	C1-12_S100_xxx--gg--texemplo1.fastq	C1-12_S100xxx--gg--texemplo2.fastq	13	T-12	ILLUMINA	lib01	paired	tumor		
C2-12	C2-12_S100_xxx--gg--texemplo1.fastq.gz	C2-12_S100xxx--gg--texemplo2.fastq.gz	14	T-12	ILLUMINA	lib01	paired	tumor		
C3-12	C3-12_S1_xxx--gg--texemplo1.fastq.gz	C3-12_S1xxx--gg--texemplo2.fastq.gz	1	T-12	ILLUMINA	lib01	paired	tumor		
C1-13	C1-13_S44_xxx--gg--texemplo1.fastq	C1-13_S44xxx--gg--texemplo2.fastq	16	T-13	ILLUMINA	lib01	paired	tumor		
C2-13	C2-13_S44_xxx--gg--texemplo1.fastq.gz	C2-13_S44xxx--gg--texemplo2.fastq.gz	14	T-13	ILLUMINA	lib01	paired	tumor		
C3-13	C3-13_S1_xxx--gg--texemplo1.fastq.gz	C3-13_S1xxx--gg--texemplo2.fastq.gz	13	T-13	ILLUMINA	lib01	paired	tumor		
C1-16	C1-16_S11_xxx--gg--texemplo1.fastq	C1-16_S11xxx--gg--texemplo2.fastq	19	T-16	ILLUMINA	lib01	paired	tumor		
C2-16	C2-16_S11_xxx--gg--texemplo1.fastq.gz	C2-16_S11xxx--gg--texemplo2.fastq.gz	20	T-16	ILLUMINA	lib01	paired	tumor		
C3-16	C3-16_S16_xxx--gg--texemplo1.fastq.gz	C3-16_S16xxx--gg--texemplo2.fastq.gz	21	T-16	ILLUMINA	lib01	paired	tumor		
C1-14	C1-14_S124_xxx--gg--texemplo1.fastq	C1-14_S124xxx--gg--texemplo2.fastq	22	T-14	ILLUMINA	lib01	paired	tumor		
C2-14	C2-14_S124_xxx--gg--texemplo1.fastq.gz	C2-14_S124xxx--gg--texemplo2.fastq.gz	23	T-14	ILLUMINA	lib01	paired	tumor		
C3-14	C3-14_S30_xxx--gg--texemplo1.fastq.gz	C3-14_S30xxx--gg--texemplo2.fastq.gz	24	T-14	ILLUMINA	lib01	paired	tumor		
C1-13	C1-13_S112_xxx--gg--texemplo1.fastq	C1-13_S112xxx--gg--texemplo2.fastq	29	T-13	ILLUMINA	lib01	paired	tumor		
C2-13	C2-13_S112_xxx--gg--texemplo1.fastq.gz	C2-13_S112xxx--gg--texemplo2.fastq.gz	26	T-13	ILLUMINA	lib01	paired	tumor		
C3-13	C3-13_S21_xxx--gg--texemplo1.fastq.gz	C3-13_S21xxx--gg--texemplo2.fastq.gz	24	T-13	ILLUMINA	lib01	paired	tumor		
C1-21	C1-21_S36_xxx--gg--texemplo1.fastq	C1-21_S36xxx--gg--texemplo2.fastq	23	T-21	ILLUMINA	lib01	paired	tumor		
C2-21	C2-21_S36_xxx--gg--texemplo1.fastq.gz	C2-21_S36xxx--gg--texemplo2.fastq.gz	29	T-21	ILLUMINA	lib01	paired	tumor		
C3-21	C3-21_S4_xxx--gg--texemplo1.fastq.gz	C3-21_S4xxx--gg--texemplo2.fastq.gz	30	T-21	ILLUMINA	lib01	paired	tumor		
C1-22	C1-22_S93_xxx--gg--texemplo1.fastq	C1-22_S93xxx--gg--texemplo2.fastq	31	T-22	ILLUMINA	lib01	paired	tumor		
C2-22	C2-22_S93_xxx--gg--texemplo1.fastq.gz	C2-22_S93xxx--gg--texemplo2.fastq.gz	32	T-22	ILLUMINA	lib01	paired	tumor		
C3-22	C3-22_S13_xxx--gg--texemplo1.fastq.gz	C3-22_S13xxx--gg--texemplo2.fastq.gz	33	T-22	ILLUMINA	lib01	paired	tumor		
C1-29	C1-29_S101_xxx--gg--texemplo1.fastq	C1-29_S101xxx--gg--texemplo2.fastq	34	T-29	ILLUMINA	lib01	paired	tumor		
C2-29	C2-29_S101_xxx--gg--texemplo1.fastq.gz	C2-29_S101xxx--gg--texemplo2.fastq.gz	39	T-29	ILLUMINA	lib01	paired	tumor		
C3-29	C3-29_S16_xxx--gg--texemplo1.fastq.gz	C3-29_S16xxx--gg--texemplo2.fastq.gz	36	T-29	ILLUMINA	lib01	paired	tumor		
C1-24	C1-24_S161_xxx--gg--texemplo1.fastq	C1-24_S161xxx--gg--texemplo2.fastq	34	T-24	ILLUMINA	lib01	paired	tumor		
C2-24	C2-24_S161_xxx--gg--texemplo1.fastq.gz	C2-24_S161xxx--gg--texemplo2.fastq.gz	33	T-24	ILLUMINA	lib01	paired	tumor		
C3-24	C3-24_S44_xxx--gg--texemplo1.fastq.gz	C3-24_S44xxx--gg--texemplo2.fastq.gz	39	T-24	ILLUMINA	lib01	paired	tumor		

Figura 29. Exemplo de um arquivo TSV em um editor de texto padrão.

Para solucionar esse problema, foi implementado o módulo '*InputTSVFile',* no qual foi criada uma tela de gerenciamento de arquivo TSV, com opção de *upload* de um arquivo TSV, edição e pareamento de amostras (este nos casos do usuário selecionar estudo '*Somatic*').

Esse arquivo TSV é obrigatório para o usuário como entrada do *workflow*, então é necessário que o usuário faça o upload do arquivo TSV (mesmo que não ocorra alterações) e clicar no botão '*Choose this TSV File'* no pop-up de Gerenciamento do Arquivo TSV, para que a variável associada no JSON gerado receba o path (caminho) desse arquivo TSV que foi salvo em uma pasta do servidor, para que ele seja enviado para a execução do *workflow*.

A Figura 30, apresenta a tela de Gerenciamento de Arquivo TSV. Nessa tela temos um botão para *upload* de arquivo TSV, o qual será carregado em uma tabela HTML dinâmica. A partir dessa tabela dinâmica é possível realizar facilmente várias edições diretamente nas células da tabela, marcando as linhas, as quais serão modificadas para cor de fundo verde. Outra função criada é a opção de preenchimento automático e customizado para alterar uma coluna inteira de uma única vez, além disso, é possível inserir e remover linhas facilmente. Essas

alterações deram maior flexibilidade e segurança para o usuário durante edição dos arquivos

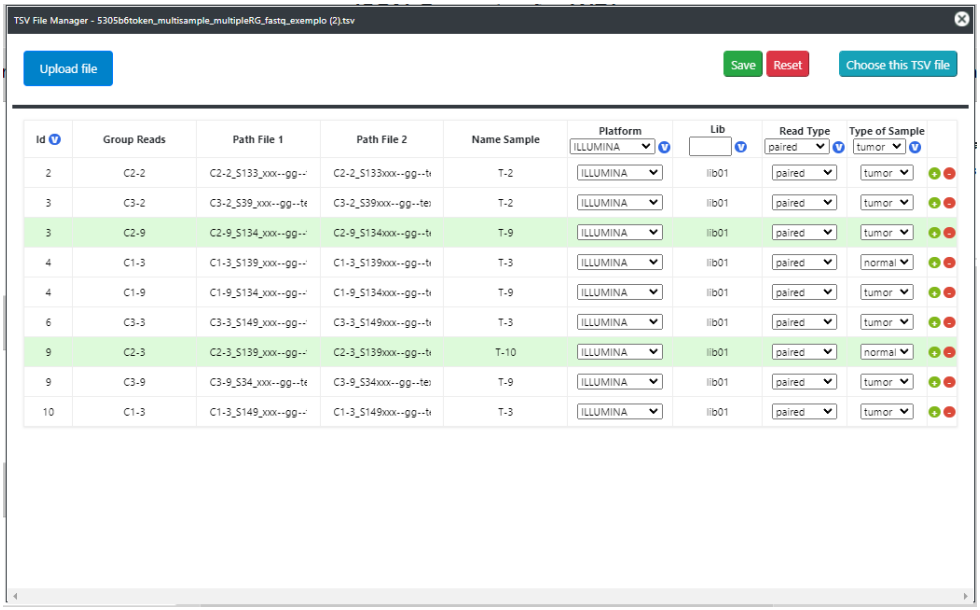


Figura 30. Tela de gerenciamento de arquivo TSV desenvolvida.

Ao editar um arquivo TSV e salvá-lo, toda validação necessária é realizada e, possíveis IDs repetidos ou células sem preenchimentos são destacadas com sua cor de fundo vermelha, com o intuito de chamar a atenção do usuário. Caso validado corretamente, o arquivo é salvo em uma pasta no servidor do *back-end.* Futuramente, a ideia é que cada usuário do *workflow* tenha um *workspace* com seus arquivos de *upload* enviados para um servidor do INCA e gerenciados pelo usuário através do protocolo LDAP (*Lightweight Directory Access Protocol*).

No final da edição do arquivo TSV, o usuário deverá clicar no botão '*Choose this TSV File'* do *pop-up e,* então, o arquivo TSV carregado no momento é preenchido no campo *input* do formulário '*Select TSV File*' na caixa '*Input Parameters*', como é mostrado na Figura 31.

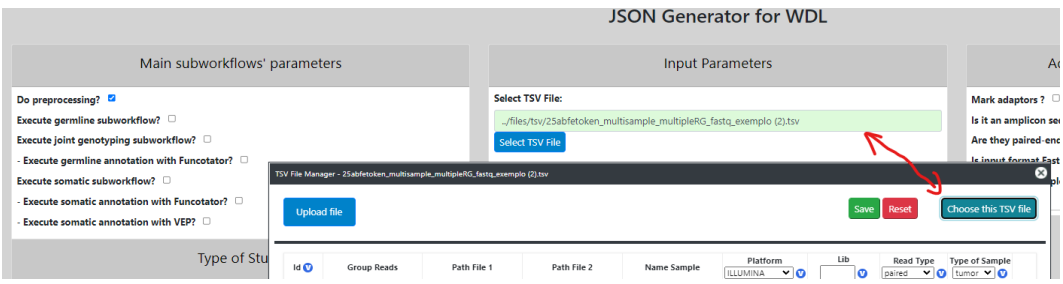


Figura 31. Passo executado pelo usuário para selecionar arquivo TSV escolhido como entrada do estudo a ser executado pelo *workflow*.

Uma funcionalidade adicional foi desenvolvida para caso o usuário selecione um estudo '*Somatic*'. Neste caso, haverá a possibilidade de o usuário parear amostras Normais x Tumoral. Na tela de gerenciamento de arquivo TSV, o usuário poderá preencher o pareamento de amostras referente as amostras do arquivo TSV carregado na tela. Com o pareamento preenchido pelo usuário e clicando no botão '*Apply*' uma coluna adicional ao arquivo TSV é preenchida de acordo com esse pareamento. Por exemplo, se o usuário escolher o pareamento entre as amostras 'normal C-1' e 'tumor C-10', no momento que o usuário clicar no botão '*Apply*' em todas as linhas do arquivo TSV que tiverem o '*Name Sample*' = 'C-1' e '*Type of Sample*' = 'normal', na coluna adicional é adiciona o valor 'C-10'(tumor), referente ao tumor pareado. Para o inverso, a lógica é a mesma. Casos em que for encontrado o valor 'C-10' o valor 'C-1' será adicionado na coluna adicional, obedecendo o pareamento (Figura 32).

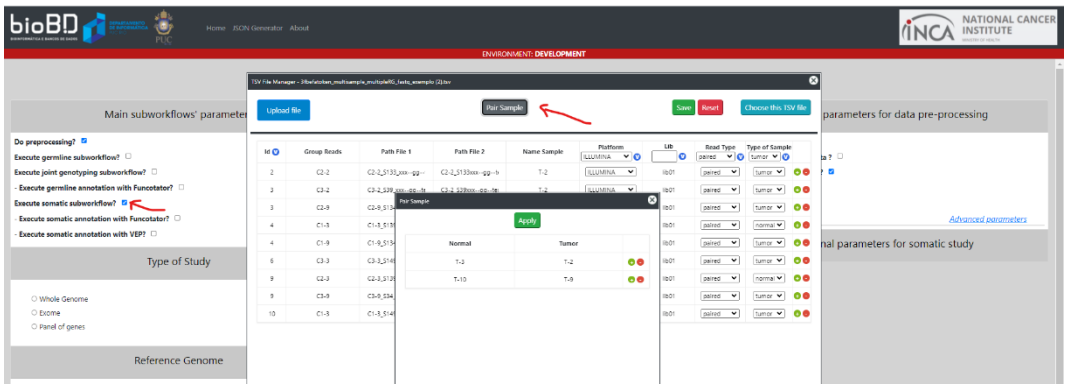


Figura 32. Tela para definição de pareamento de amostras.

6.2 Implementação do Projeto 2 (*SGWfC-Gene*)

No projeto 2, foram implementados 3 componentes principais utilizando a tecnologia *React* no *front-end*. Inicialmente, para o desenvolvimento dos componentes foi preciso um estudo conceitual sobre o *framework React,* o qual tem muitos recursos que facilitam o desenvolvimento, tais como *React Hooks*, JSX e *Redux*.

React Hooks é uma funcionalidade adicionada no *React* a partir da versão 16.8, que tem como objetivo oferecer formas de se definir estados sem a necessidade de ter uma classe (*stateful componente*).

O *JSX* é uma extensão de sintaxe para o *Javascript* e permite criar elementos para serem utilizados como *templates* de componentes *React*.

Basicamente, os elementos criados com o JSX são bem similares a código HTML e fornecem aos desenvolvedores uma forma mais simples e intuitiva de criar os componentes de uma aplicação. Porém, apesar de muito similar ao HTML, o JSX não é interpretado pelo navegador. Por este motivo, deve-se utilizar um *transpiler Javascript* para essa conversão. Atualmente, o mais conhecido deles é o Babel, que faz esse trabalho de converter um código *React* em código *Javascript* puro e interpretado pela *engine* do navegador.

Na Figura 33, é apresentado um exemplo da definição de um componente *React* chamado de '*Example*', utilizando uma função padrão do *Javascript* para a sua definição. Neste componente, é declarada uma variável *count* e em seguida é passada a função *setCount* que irá alterar o estado dessa variável. A variável *count* é inicializada utilizando o *Hook useState* com um valor inicial. Logo em seguida, no retorno da função, temos a definição do *template* do componente no qual é utilizado a sintaxe do *JSX* que facilita bastante a implementação, pois permite utilizar tags HTML para a definição dos elementos. Nesse *template* de exemplo, é inserido um botão (*tag button* do HTML), onde é passado com um atributo chamado '*onClick*', o qual é um evento (ação de clique do mouse) do *React.* Esse botão, ao ser clicado a função passada como *call-back* é executada e nela há uma chamada da função *setCount* que tem a tarefa de mudar o estado da variável *count* por meio do *Hook useState. A partir de*sse exemplo, foi possível compreender um pouco da estrutura básica de desenvolvimento, utilizando o *framework React* como ferramenta de *front-end*.

React é um projeto *open-source* desenvolvido pelo Facebook e sua documentação é disponibilizado através do link abaixo:

<https://pt-br.reactjs.org/docs/getting-started.html>

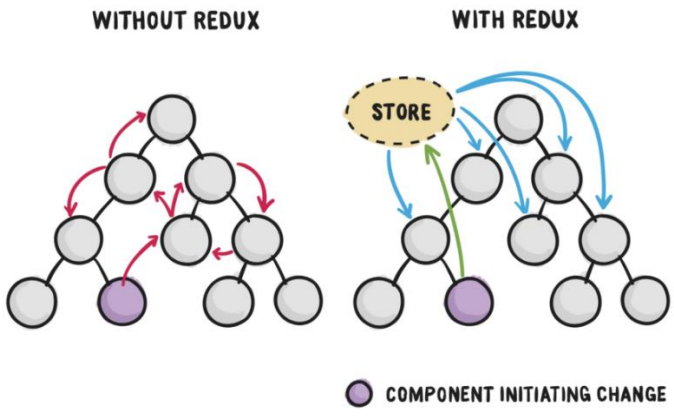
Redux é um projeto *open-source* e sua documentação é disponibilizado através do link abaixo:

<https://redux.js.org/>

```jsx
import React, { useState } from 'react';

function Example() {
  // Declare uma nova variável de state, a qual chamaremos de "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Figura 33. Exemplo da implementação de um componente *React*.

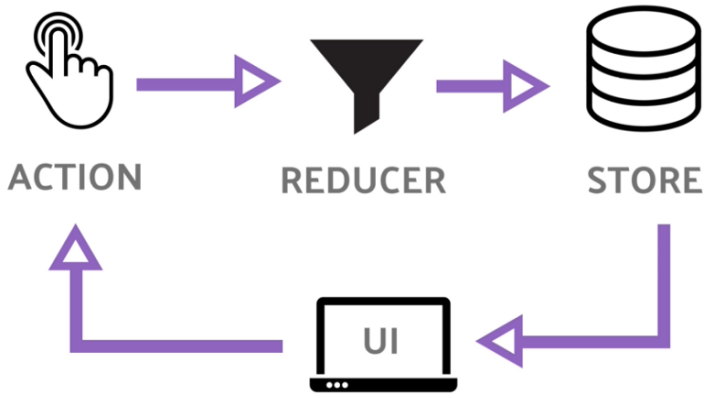
Além da definição de componentes, outro conceito importante do *React* é como os componentes trocam dados entre si. No próprio *React* existem alguns recursos que permite a comunicação entre um componente-pai e um componente-filho e vice-versa (comunicação em duplo-sentido). No entanto, quando temos componentes irmãos essa comunicação pode ser difícil de ser implementada e gerenciada. Portanto, é recomendado a utilização de um Gerenciador de Estados. Um dos mais conhecidos *open-source* é o *Redux*. Como podemos ver na Figura 34. o *Redux* busca melhorar a transmissão de dados entre os componentes de forma a centralizar os estados dos componentes, o qual é chamado de *Store*. Um *Store* é um container que armazena e centraliza o estado geral da aplicação (React Brasil, 2018). Podemos ter uma *Store* por aplicação, ou seja, ela é a “Única Fonte de Verdade” (*Single Source of Truth*).



Fonte: (React Brasil, 2018)

Figura 34. Comparação entre a comunicação de componente com e sem uso do *Redux.*

A Figura 35 apresenta um diagrama que mostra o funcionamento do fluxo de uma evolução de um estado com o *Redux.* As *Actions* são fontes de informações que são enviadas da aplicação para o *Store*. *Reducers* recebem e tratam as informações para que sejam ou não enviadas à *Store*. Basicamente a cada ação(evento) do usuário que altera o estado de algum componente, um método do *Redux* que implementa o Padrão de Projeto *Observer* faz uma chamada do próprio componente para uma *Action* no *Store.* Em seguida, por meio do *Reducer* o estado componente é atualizado no *Store*. No momento que o estado de um componente X é atualizado no *Store,* os outros componentes Y que dependem do estado do componente X são então 'avisados' sobre essa mudança e podem receber um novo valor gerado no componente X e, então podem tomar alguma ação dependendo da sua lógica de implementação. No projeto 2, foi utilizado o *Redux* como Gerenciador de Estados para permitir a comunicação entre os componentes implementados (React Brasil, 2018).



Fonte: (React Brasil, 2018)

Figura 35. Fluxograma de funcionamento do *Redux*.

6.2.1 Componente para *upload* de arquivo no formato CSV.

Foi desenvolvido um componente que permite ao usuário fazer o *upload* do arquivo no formato CSV. Ao fazer o *upload*, o documento é enviado para uma pasta no servidor *web*. Foi utilizado o *SQLite* que é um banco de dados relacional embutido na aplicação. Nele, foi criado uma tabela *Documents* para salvar o *path*(caminho) dos arquivos CSV carregados pelo usuário. Essa tabela foi definida com os campos de ID do usuário, data de *upload* do arquivo, *token* do arquivo e o seu *path.* Foi utilizado o *ORM* (*Object-Relational Mapping*) do *Django* para fazer a persistência dos dados.

6.2.2 Componente da exibição de tabela de arquivos carregados pelo usuário

Uma vez que o usuário faça o *upload* de um arquivo, o *Redux* é então acionado e, então, os dados das informações sobre arquivo carregado é recebido por um outro componente responsável por exibir uma tabela HTML que mostra o histórico de arquivos carregados pelo usuário. Essa tabela (Figura 4), contém um botão '*Run*' para cada arquivo carregado. No momento que um botão '*Run*' da tabela for clicado é disparado uma *Action* que fará uma mudança de estado no *Story* do *Redux*, passando esse *token* para outro componente que é responsável pela exibição do grafo gerado pelo *workflow*.

6.2.3 Componente da exibição do Grafo utilizando o *Cytoscape*

Ao receber o *token* do arquivo CSV selecionado na tabela, o componente de exibição do grafo, faz uma nova requisição ao *back-end* (servidor *web Django*) enviando esse *token*. No *back-end*, ao receber o *token*, é verificado na tabela *Documents* do *SQLite* qual o *path* do arquivo CSV referenciado, e esse *path* é passado para o agente do *Prefect* por meio de uma API, o agente então irá utilizar o arquivo *CSV* como entrada da execução do *workflow*. Depois de executado, o resultado do *workflow* é gerado no formato de um grafo por meio da ferramenta do *NetworkX*. Por fim, essa resposta é recebida pelo componente do *React* que utiliza outro componente *open-source chamado Cytoscape-react* para a exibição do grafo (Figura 5). Na Figura 36, é apresentado um exemplo básico da utilização do componente *Cytoscape-react*. Nele é definido um objeto *Javascript* chamado *elements* no qual é definido cada nó do grafo e sua posição.

React-cytoscape é um projeto *open-source* e sua documentação é disponibilizado através do link abaixo:

<https://github.com/plotly/react-cytoscapejs>

```
import React from 'react';
import ReactDOM from 'react-dom';
import CytoscapeComponent from 'react-cytoscapejs';

class MyApp extends React.Component {
  constructor(props){
    super(props);
  }

  render(){
    const elements = [
       { data: { id: 'one', label: 'Node 1' }, position: { x: 0, y: 0 } },
       { data: { id: 'two', label: 'Node 2' }, position: { x: 100, y: 0 } },
       { data: { source: 'one', target: 'two', label: 'Edge from Node1 to Node2' } }
    ];

    return <CytoscapeComponent elements={elements} style={ { width: '600px', height: '600px' } } />;
  }
}
```

Fonte: React-cytoscape

Figura 36. Exemplo do uso do componente *Cytoscape-react.*

Alguns problemas foram encontrados na tentativa de carregar o grafo resultante da execução do *workflow* no navegador. O primeiro foi que os nós estavam chegando sem posição pré-definida, fazendo com que todos os nós fossem renderizados no mesmo ponto. Em uma primeira tentativa de solucionar esse problema, foi utilizado uma opção de *layout* do *Cytoscape* em que é possível utilizar um algoritmo de espalhamento nas posições dos nós. Foi utilizado o formato de *layout fcose*, que aplica um algoritmo de espalhamento dos nós. A princípio o problema foi solucionado. No entanto, o grafo resultante do *workflow* apresenta, em geral, uma média de 9 mil nós o que tornava inviável a visualização do grafo inteiro pelo usuário, além de sobrecarregar o navegador do usuário para aplicar um algoritmo de espalhamento em 9 mil nós. Solucionar esse problema ainda é um desafio em aberto para a sequência do projeto.

7. **Considerações Finais**

O trabalho de Projeto Final contou com a participação do aluno em projetos de aplicação real em contribuição ao BioBD (Laboratório de Bioinformática e Bancos de Dado da PUC-Rio) e ao INCA (Instituto Nacional do Câncer). O trabalho envolveu a participação em dois projetos diferentes com objetivo de conhecer o funcionamento de SGWfCs (Sistemas Gerenciados de Workflows Científico) e no desenvolvimento de interfaces para os *workflows* dos projetos envolvidos.

O trabalho possibilitou um contato com várias ferramentas de desenvolvimento de *software* para *web*, tais como, *Django Framework* e *ReactJS.* Também foi apresentado o conceito de *workflows* científicos e sua importância em análises e estudos científicos. Além disso, o aluno foi apresentado há alguns aspectos relacionados ao ramo de Bioinformática e tecnologias comumente utilizadas nesse segmento, dentre elas podemos destacar o *NetworkX* e o *Cytoscape*.

O Projeto Final, como mencionado envolveu a participação em dois projetos já em andamento e com tecnologias já definidas. Então, foi um grande desafio além de desenvolver as demandas solicitadas, também ter a flexibilidade de estudar diferentes tecnologias e o funcionamento dos diferen*tes workf*lows abordados. Mas, a experiência foi válida no âmbito de aprendizado com os conceitos abordados nos dois projetos, além de tido a possibilidade de compartilha novos conhecimentos com a equipe do *BioBD* envolvida nos projetos.

Ambos os projetos ainda não foram finalizados. Os próximos passos do projeto 1 (*workflow* PIPE-MB) é o desenvolvimento de uma página *web* para mostrar o resultado gerado pelo *workflow* para o usuário, além de criar uma tela de acompanhamento de execução do *workflow* com os logs de execução sendo exibidos e, também a criação de cadastro e gerência do usuário por meio de autenticação com o sistema do INCA e, também a criação de um *workspace* para os usuários para facilitar a gerência de arquivos envolvidos no uso da ferramenta. Já no projeto 2 (SGWfC-Gene) há um desafio de conseguir exibir e gerenciar a visualização do grafo gerado como resultado da execução do *workflow* no navegador do usuário de forma flexível.

8. **Referências**

[1] BRAGHETTO, R. K.; CORDEIRO, D. Capítulo 1 - Introdução à Modelagem e Execução de Workflows Científicos. In: SALGADO, A. C.; LÓSCIO, B. F.; ALCHIERI, E.; BARRETO, P. S. **Atualizações em Informática**. Sociedade Brasileira de Computação, 1ed. Porto Alegre, Brasil: SBC. 2014.p. 1-40. Disponível em: <https://www.researchgate.net/publication/265122654_Introducao_a_Modelagem_e_Execucao_de_Workflows_Cientificos>. Acessado em: 11/04/2021.

[2] BROAD INSTITUTE. Getting started with GATK4. 2021a. Disponível em: < https://gatk.broadinstitute.org/hc/en-us/articles/360036194592-Getting-started-with-GATK4>. Acessado em: 18/06/2021.

[3] BROAD INSTITUTE. Picard. 2021b. Disponível em: <http://broadinstitute. github.io/picard/>. Acessado em: 18/06/2021.

[4] CUEVAS-VICENTTIN, V., DEY, S. C., KÖHLER, S., RIDDLE, S., AND LUDÄSCHER, B. Scientific workflows and provenance: Introduction and research opportunities. **Datenbank-Spektrum**, 12(3):193–203. (2012).

[5] STANCEY, E. What Humans Can Learn From 'Human In The Loop' Learning. Forbes, 2021. Disponível em: <https://www.forbes.com/sites/edstacey/2021/04/09/what-humans-can-learn-from-human-in-the-loop-learning/?sh=5a18a2086aca>. Acessado em: 25/11/2021.

[6] FRED HUTCH. Cromwell Workflow Manager and WDL Workflows. 2020. Disponível em: <https://sciwiki.fredhutch.org/compdemos/Cromwell/>. Acessado em: 18/06/2021.

[7] [INCA] Instituto Nacional de Câncer. Pesquisa. **Instituto Nacional de Câncer**. 2020. Disponível em: < https://www.inca.gov.br/pesquisa>. Acessado em: 11/04/2021.

[8] [INCA] Instituto Nacional de Câncer. Estimativa de Câncer no Brasil, 2020. Disponível em: < https://www.inca.gov.br/numeros-de-cancer >. Acessado em: 17/11/2021.

[9] JSON ORG. Introdução ao JSON. 2021. Disponível em: < https://www.json.org/ json-pt.html>. Acessado em: 18/06/2021.

[10] GENOMICA. Como a bioinformática auxilia no diagnóstico de doenças genéticas na Genomika. **Hospital Israelita Albert Einstein**. 2014. Disponível em: <https://www.genomika.com.br/blog/como-a-bioinform%C3%A1tica-auxilia-nodiagn%C3%B3stico-de-doen%C3%A7as-gen%C3%A9ticas-na-genomika/>. Acessado em: 11/04/2021.

[11] LUDÄSCHER, B.; ALTINTAS, I.; BERKLEY, C.; HIGGINS, D.; JAEGER, E.; JONES, M.; LEE, E. A.; TAO, J. and ZHAO, Y. Scientific workflow management and the Kepler system. **Concurrency and Computation: Practice and Experience**, 18 (10):1039–1065. 2006.

[12] Open WDL. About OpenWDL. Disponível em < https://openwdl.org/>. Acessado em: 11/04/2021.

[13] React Brasil. Iniciando com Redux em 9 passos. 2018. Disponível em < https://medium.com/reactbrasil/iniciando-com-redux-c14ca7b7dcf>. Acessado em: 02/11/2021.

[14] SILVA, L. A. M. **Workflows Científicos**. MONOGRAFIA (Bacharel em Ciência da Computação) - Instituto de Ciências Exatas, Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora. Juiz de Fora. p.66. 2007. Disponível em <http://www.monografias.ice.ufjf.br/tcc-web/exibePdf?id=3>. Acessado em: 11/04/2021.

[15] STANFORD UNIVERSITY. Humans in the Loop: The Design of Interactive AI Systems. Human-Centered Artificial Intelligence.2019. Disponível em < https://hai.stanford.edu/news/humans-loop-design-interactive-ai-systems>. Acessado em: 25/11/2021.

[16] VIEIRA, T. A. S. C.; CASANOVA, M. A. Execução Flexível de Workflows. Rio de Janeiro. 2005. 429p. Tese de Doutorado. Departamento de Informática - Pontifícia Universidade Católica do Rio de Janeiro.

[17] VIEIRA, D. M.; ALEGRIA, A. L.; LANNA, C. A.; BORONI, M.; LIFSCHITZ, S. Interação com grafos gênicos utilizando Sistema Gerenciador de Workflows Científicos. Monografias em Ciência da Computação, Carlos José Pereira de Lucena. ISSN: 0103-9741, 2021.

[18] [WHO] World Health Organization. WHO report on câncer: setting priorities, investing wisely and providing care for all**. World Health Organization**. 2020. Disponível em: < https://apps.who.int/iris/rest/bitstreams/1267643/retrieve >. Acessado em: 11/04/2021.