UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

GREGOR YANNIS WEY

APLICAÇÃO WEB PARA ANÁLISE DE DADOS GENÉTICOS DO WORKFLOW CIENTÍFICO C-GEMIS

CAXIAS DO SUL

2023

GREGOR YANNIS WEY

APLICAÇÃO WEB PARA ANÁLISE DE DADOS GENÉTICOS DO WORKFLOW CIENTÍFICO C-GEMIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Daniel Luis Notari

CAXIAS DO SUL

2023

GREGOR YANNIS WEY

APLICAÇÃO WEB PARA ANÁLISE DE DADOS GENÉTICOS DO WORKFLOW CIENTÍFICO C-GEMIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado(a) em 30/11/2023

BANCA EXAMINADORA

Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul - UCS

Prof. Dra. Scheila de Avila e Silva
Universidade de Caxias do Sul - UCS

Dr. Marcos Vinicius Rossetto
Universidade de Caxias do Sul - UCS

AGRADECIMENTOS

Agradeço aos meus pais por me possibilitarem esta oportunidade e por todo apoio que me deram ao longo de minha vida. Agradeço ao Professor Notari por ter me guiado durante o desenvolvimento deste trabalho e pela paciência. Agradeço a mim mesmo por não ter desistido.

"O mal que existe no mundo provém quase sempre da ignorância, e a boa vontade, se não for esclarecida, pode causar tantos danos quanto a maldade."

Albert Camus

RESUMO

Workflows científicos tem se popularizado com o aumento de dados provenientes das áreas da genética. O grande volume de dados gerado por estas aplicações exige ferramentas capazes de realizarem a análise. Entre os *workflows* desenvolvidos é possível destacar o C-Gemis. C-Gemis é uma aplicação que tem como objetivo realizar a análise de genes na busca de biomarcadores do câncer gástrico. O projeto C-Gemis envolve diversas camadas de aplicação, abrangendo o processamento analítico de dados até a interação *web* com o usuário. Permite ao usuário informar genes, disponibilizando os resultados da análise. Entretanto em seu antigo estado não era possível obter os resultados, a possibilidade de *download* por parte do usuário era uma necessidade fundamental a ser implementada. Este trabalho teve por objetivo apresentar uma proposta de solução para uma nova interface gráfica *web* utilizando *React* e *Typescript*. Foi proposto a refatoração da camada *web* e da arquitetura geral do C-Gemis. Uma arquitetura com serviços isolados com a utilização de uma Application Programming Interface (API) desenvolvida em *NodeJS*. A escalabilidade da aplicação e a necessidade de futuras implementações foram levadas em consideração na proposta de solução. Foi possível desenvolver uma aplicação funcional, atendendo as necessidades requisitadas. O *download* dos resultados foi disponibilizado com sucesso. Foi implementado o *Google Analytics*, permitindo realizar análise da quantidade de acesso e entender o comportamento do usuário. Utilizando a Amazon Web Services (AWS) o C-Gemis foi disponibilizado para o uso do público alvo na *internet*.

Palavras-chave: Workflow. API. C-Gemis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API Application Programming Interface
REST Representational state transfer
RNA-seq Sequenciamento de Ácido ribonucleico
RNA Ácido ribonucleico
DNA Ácido desoxirribonucleico
SOAP Simple Object Access Protocol
HTTP Hypertext Transfer Protocol
JSON JavaScript Object Notation
XML Extensible Markup Language
HTML HyperText Markup Language
PCA Principal Component Analysis
OMS Organização Mundial de Saúde
GCP Google Cloud Platform
RPC Remote Procedure Call
SOA Service-oriented architecture
UDDI Universal Description, Discovery and Integration
WSDL Web Services Description Language
CRUD Create, Read, Update and Delete
URL Uniform Resource Locator
TCP Transmission Control Protocol
TCGA The Cancer Genome Atlas Program
GEO Gene Expression Omnibus
SPA Single Page Application
WHO World Health Organization
INCA Instituto Nacional do Câncer
SQL Structured Query Language
HUGO Human Genome Organisation
SHA256 Secure Hash Algorithm
AWS Amazon Web Services
SSL Secure Sockets Layer
IP Internet Protocol
EC2 Amazon Elastic Compute Cloud

SUMÁRIO

1 INTRODUÇÃO

O câncer segundo a Organização Mundial de Saúde (OMS) em 2020 é responsável por uma a cada seis mortes mundialmente[1]. Apenas no ano de 2020 foram cerca de 10 milhões de óbitos. O tipo mais comum foi o câncer de mama com 2.260 milhões de casos. O câncer de pulmão foi o mais letal com 1.800 milhões de mortes registradas, seguido pelo estomacal. De acordo com o Instituto Nacional do Câncer (INCA), entre 2023 a 2025 são esperados 704 mil novos casos de câncer no Brasil, destes o câncer gástrico sendo responsável por 13 mil casos (SANTOS *et al.*, 2023).

Segundo o Instituto Nacional de Câncer americano, a doença pode ser definida como um crescimento desordenado de células pelo corpo [2]. Durante o processo de divisão celular pode ocorrer a criação de tumores que se espalham sem controle. De acordo com dados divulgados em 2022 pela Organização de Pesquisa do Câncer do Reino Unido, cerca de 60% das pessoas diagnosticadas com câncer no pulmão em estágio inicial sobrevivem por mais de 5 anos, em estágios avançados a taxa cai para menos de 10% [3]. A taxa de sobrevivência do câncer gástrico diagnosticado em estágio inicial é de 65%, enquanto para estágios avançados é menor de 15% (D'ANGELO; RIENZO; OJETTI, 2014).

A descoberta em estágios iniciais do câncer gástrico potencializa a taxa de sobrevivência. Métodos atuais de prevenção podem ser invasivos ao paciente e financeiramente inviáveis em países menos desenvolvidos (D'ANGELO; RIENZO; OJETTI, 2014), exemplo da endoscopia. Técnicas para descoberta das expressões genéticas podem apresentar uma nova solução para o combate da doença (YASUI *et al.*, 2004).

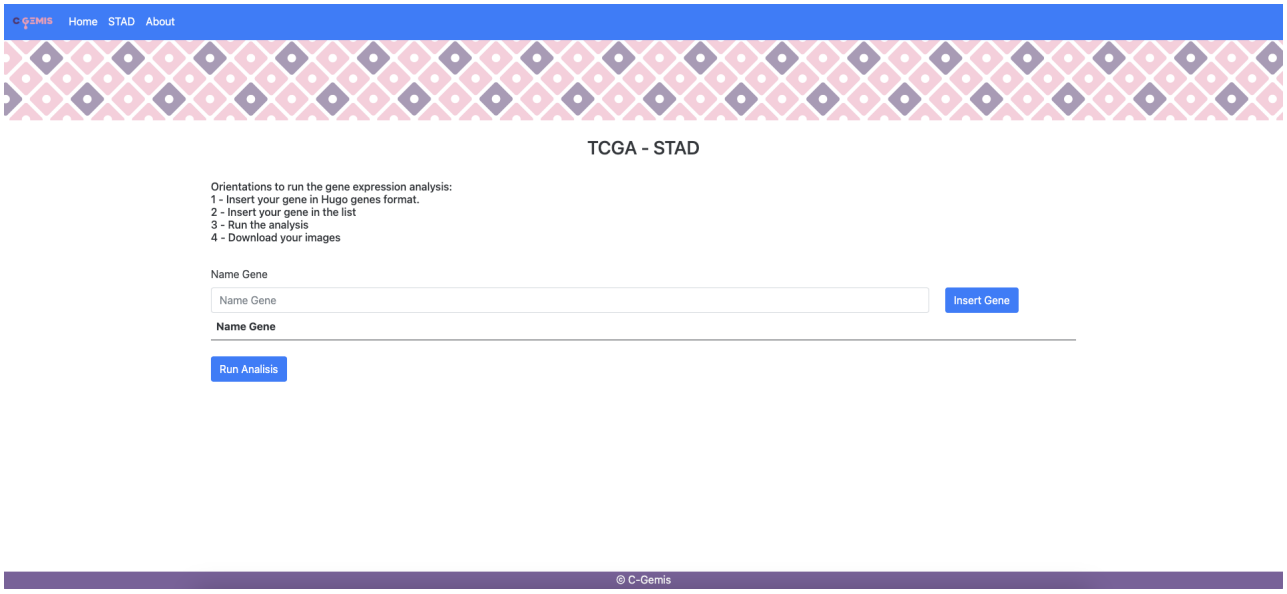
As técnicas de Microarranjo e *Sequenciamento de Ácido ribonucleico (RNA-seq)* geram dados que necessitam de ferramentas de análise (ROSSETTO, 2021). Neste contexto a aplicação C-Gemis (ROSSETTO, 2021) foi desenvolvida para a análise de dados de expressão gênica do câncer gástrico. Utilizando o ambiente R[4] e dados de bancos públicos como o *The Cancer Genome Atlas Program (TCGA)*[5] e *Gene Expression Omnibus (GEO)*[6]. É realizada a análise e busca de biomarcadores do câncer gástrico.

O C-Gemis possuía uma camada *web* em *Angular* que não atendia as necessidades do projeto, conforme interface apresentada na Figura 1. Não permitia salvar os resultados ou selecionar outros tipos de câncer. O desenvolvimento de uma nova camada *web* era necessária. A

[1] Disponível em https://www.who.int/news-room/fact-sheets/detail/cancer
[2] Disponível em https://www.cancer.gov/about-cancer/understanding/what-is-cancer
[3] Disponível em https://www.cancerresearchuk.org/about-cancer/cancer-symptoms/why-is-early-diagnosis-important
[4] Disponível em https://www.r-project.org/
[5] Disponível em https://www.cancer.gov/ccg/research/genome-sequencing/tcga
[6] Disponível em https://www.ncbi.nlm.nih.gov/geo/

aplicação em R também era responsável pelos *endpoints* da API, função disponível apenas com o uso de bibliotecas. Foi proposto a criação de uma API seguindo os padrões *Representational state transfer (REST)* em uma nova camada para comunicação com a aplicação R e o banco de dados (LI *et al.*, 2016; EHSAN *et al.*, 2022).

Figura 1 – Interface C-Gemis



Fonte: Captura de tela

1.1 OBJETIVOS

Os seguintes objetivos visam ser alcançados.

1.1.1 OBJETIVO GERAL

Este trabalho visou aprimorar a ferramenta C-Gemis através do desenvolvimento de uma nova interface gráfica *web* que permita escalabilidade e *download* de resultados.

1.1.2 OBJETIVOS ESPECÍFICOS

1. Apresentar uma proposta de arquitetura *web* válida para o problema.
2. Desenvolver uma API para a comunicação entre a interface *web* e a aplicação em R.
3. Permitir a manipulação de imagem na camada *web*.
4. Instalar, testar e validar a aplicação.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado da seguinte forma:

- O Capítulo 1 apresenta a introdução a este trabalho, suas motivações e objetivos.
- O Capítulo 2 apresenta o contexto do trabalho. Uma apresentação sobre o câncer, C-Gemis, arquitetura de *software* e trabalhos relacionados.
- O Capítulo 3 apresenta a proposta de solução baseada na pesquisa e análise do Capítulo 2.
- O Capítulo 4 apresenta o desenvolvimento da aplicação proposta no capítulo anterior
- O Capítulo 5 apresenta o resultado de testes definidos no Capítulo 3.
- O Capítulo 6 apresenta o resultado final do projeto

2 REFERENCIAL TEÓRICO

Este capítulo tem por objetivo aprofundar os principais conceitos apresentados no Capítulo 1. Entre eles serão abordados: O câncer, sua definição, evolução da doença e métodos para descobrir biomarcadores. O projeto C-Gemis e seu funcionamento. A Arquitetura Cliente-Servidor para aplicações *web* e trabalhos relacionados.

2.1 O CÂNCER

A palavra câncer vem do grego *karkínos*, significando caranguejo (MEDRADO, 2015). Conhecido desde a antiguidade pelos gregos e egípcios é uma doença que acompanha a humanidade a séculos. Hoje é uma das maiores causas de mortes no mundo, segundo a OMS no Brasil é a segunda maior causa de mortes (OPPERMANN, 2014). Estima-se que cerca de 40% dos casos mundiais poderiam ser evitados (MEDRADO, 2015).

O risco do surgimento do câncer depende de vários fatores. Desde as condições de vida do indivíduo e o seu contexto social até a estrutura biológica genética. Entre fatores variáveis, as condições de trabalho estão entre as principais. Existem ao menos 19 tipos de tumores ligados a atividades que lidam com substâncias cancerígenas (MEDRADO, 2015). O trabalho com minérios ou agrotóxicos são ligados a um risco elevado.

O tabagismo representa 20% das causas de morte preveníveis (OPPERMANN, 2014), é a principal causa evitável de óbitos. Cerca de 90% das mortes de câncer de pulmão são derivadas do tabagismo. O uso do tabaco e derivados pode desencadear outras doenças como aneurisma arterial, trombose vascular, úlceras, infecções respiratórias e impotência sexual (MEDRADO, 2015).

O sedentarismo ligado a uma dieta de alimentos com densidade energética elevada e açucaradas acarreta em problemas de obesidade. A obesidade pode acarretar no maior risco para o câncer endométrio, rim, vesícula biliar e mama. Dietas baseadas em gorduras saturadas podem elevar o risco de desenvolver os cânceres de mama, cólon, próstata e esôfago (MEDRADO, 2015). A incidência de luz ultravioleta acarreta na maior chance do desenvolvimento do câncer de pele, câncer este tido como o mais comum (MEDRADO, 2015).

Dentre os fatores imutáveis, o aumento da expectativa de vida possibilitou notar uma correlação com a incidência de câncer em pessoas de idade. Não é completamente explicado o motivo desta correlação, o acumulo de mutações genéticas é considerado um fator (MEDRADO, 2015).

Entre outros fatores podem ser destacados as consequências da senescência, processo natural de envelhecimento e seus efeitos sobre o corpo. O surgimento de radicais livres, menor

capacidade do corpo no reparo do DNA, encurtamento dos telômeros e a hipometilação do DNA (MEDRADO, 2015).

A herança genética é responsável por uma parte minoritária do câncer. Os genes durante a vida de uma pessoa podem passar por modificações, sejam estas devido a fatores externos ou internos. Estes genes tendem a ser passados para o feto durante a gestação. Indivíduos afetados apresentam o câncer em idade precoce (MEDRADO, 2015).

2.1.1 Tumores

A multiplicação celular é uma característica de células do tipo lábeis e estáveis. O problema começa a ocorrer com o crescimento desordenado destas células. O acúmulo em regiões específicas é denominado de tumor (MEDRADO, 2015).

Os tumores malignos são conhecidos como neoplasias malignas. Sua característica é o crescimento invasivo. Este processo pode levar a metástase, quando o tumor atinge outros órgãos além do seu de origem (OPPERMANN, 2014). Este processo pode ocorrer quando as células conseguem chegar ao sistema sanguíneo ou linfático (MEDRADO, 2015).

Os tumores podem ser divididos em duas categorias, os malignos e benignos. Tumores benignos em diversos casos apresentam um crescimento que permite diferenciar as células no tecido do estroma. O modo como se assemelham e a sua organização harmônica permitem a observação (MEDRADO, 2015).

A velocidade da multiplicação é outra característica importante. Tumores malignos tendem a se multiplicar em uma taxa mais elevada. Este processo acelerado pode prejudicar a capacidade dos tecidos vizinhos realizarem o encapsulamento do tumor. Este encapsulamento é formado pelo estroma do tecido que está sendo invadido, tem como objetivo isolar as células modificadas (MEDRADO, 2015).

Existem casos que tumores malignos são envoltos no encapsulamento e conseguem realizar a metástase. Esta metástase oculta está presente em cerca de 20% dos diagnósticos, enquanto cerca de 30% apresentam sinais visíveis da metástase (MEDRADO, 2015).

Tumores malignos possuem características como menos moléculas de caderinas, responsáveis pela adesão celular. Produção de metaloproteinases e gelatinases, quebrando a estrutura dos tecidos adjacentes. Conseguem realizar a angiogênese, processo de formação de vasos capilares proveniente de vasos já existentes perto da região afetada. A angiogênese provém os nutrientes necessários para o crescimento do tumor.

As mortes por câncer tem caído nos últimos anos (BROOKS, 2012). Devido a mudanças de hábitos, novas técnicas de cura e ao tratamento precoce. A descoberta do câncer em estágios iniciais enfrenta a dificuldade da doença não demonstrar sintomas. A detecção de biomarcadores, entretanto, se apresenta como uma aliada na busca deste objetivo.

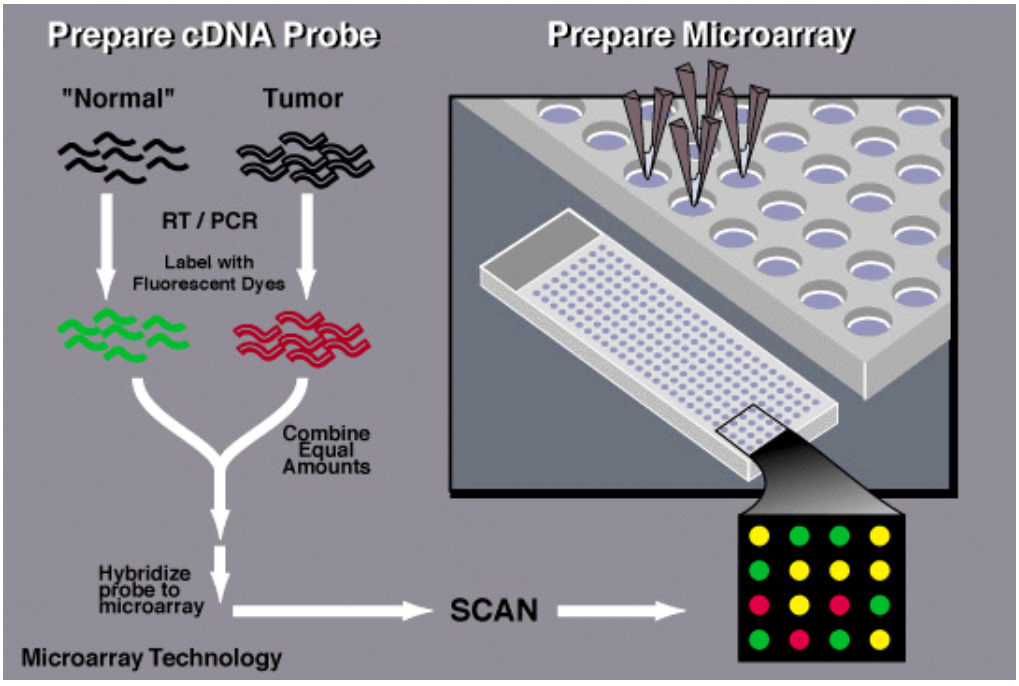
Biomarcadores são indicadores de mudanças e do estágio celular (LORENZI, 2020). Sua composição pode ser baseada em Ácido desoxirribonucleico (DNA), Ácido ribonucleico (RNA) ou proteínas (ROSSETTO, 2021). Cada composição apresenta características próprias que exigem técnicas de análise. Atualmente quase todos os biomarcadores utilizados clinicamente foram descobertos acidentalmente (LORENZI, 2020).

2.1.2 Técnicas para descoberta de Biomarcadores

Entre técnicas para a descoberta dos biomarcadores, pode ser destacada o microarranjo de DNA (LORENZI, 2020), processo descrito na Figura 2. Fitas simples de DNA são dispostas em formato matricial em uma placa onde ocorrerá a hibridização por complementaridade com nucleotídeos provenientes de uma sonda. As fitas de DNA provêm de células normais e com tumores. Utilizando fluoróforo as fitas saudáveis são marcadas pela cor verde, as fitas com tumor são marcadas de vermelho (ROSSETTO, 2021).

O processo de hibridização poderá alterar a cor das fitas, indicando presença de expressão gênica. Esta técnica gera um grande volume de dados, exigindo métodos de análise robustos. Entre os problemas do microarranjo podem ser destacados a baixa sensibilidade e especificidade para alguns genes e a necessidade de conhecer a estrutura dos genomas para popular a sonda (HRDLICKOVA; TOLOUE; TIAN, 2017).

Figura 2 – Processo de microarranjo de DNA



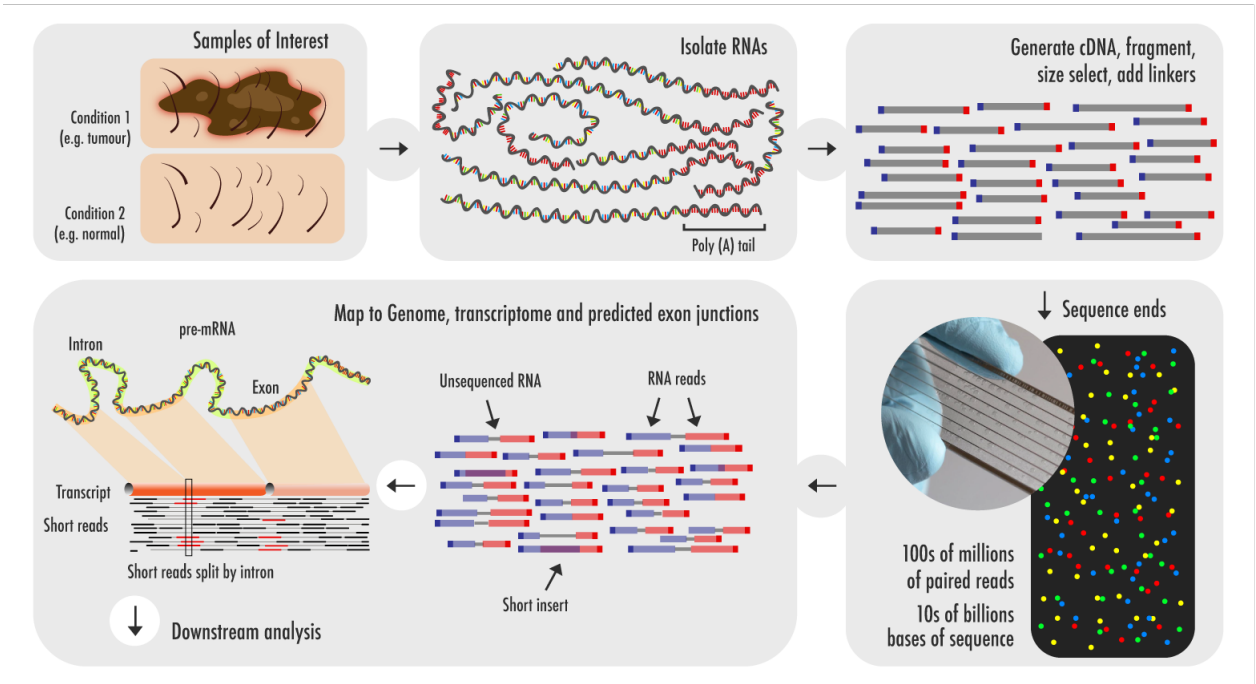
Fonte: National Human Genome Research Institute (2020).

Outra técnica mais recente utilizada na descoberta de expressão gênica é o RNA-seq, como mostra a Figura 3. Esta técnica se caracteriza por permitir uma faixa maior de detecção e menor ruído de sinal (HRDLICKOVA; TOLOUE; TIAN, 2017). Uma grande vantagem do RNA-seq

é a capacidade de sequenciar transcriptomas de sequências genômicas ainda desconhecidas (WANG; GERSTEIN; SNYDER, 2009).

O RNA é isolado e fragmentado em uma quantidade pré determinada de nucleotídeos. Estas fitas simples são transformadas em cDNA e nas extremidades são adicionadas sequências adaptativas (ROSSETTO, 2021). É realizada a análise utilizando sequenciamento, que pode ser simples ou pareado (WANG; GERSTEIN; SNYDER, 2009).

Figura 3 – Processo de RNA-seq



Fonte: Technology Networks (2018).

2.2 C-GEMIS

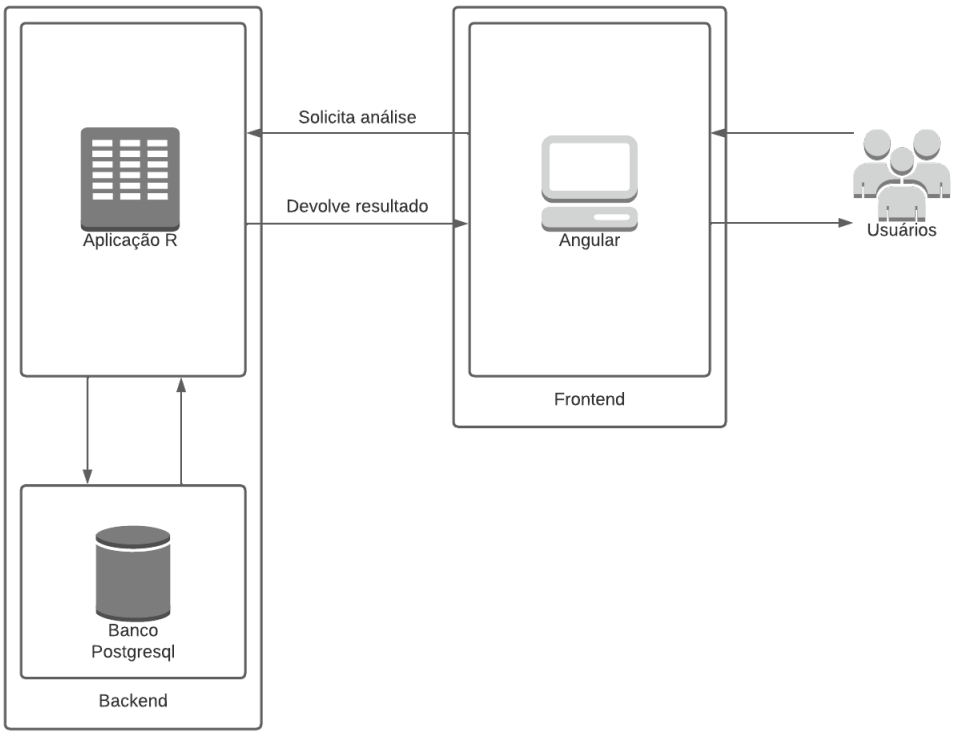
C-Gemis é a aplicação desenvolvida para a análise de dados do câncer gástrico (ROSSETTO, 2021). Separada em duas camadas, a camada de análise de dados desenvolvida em R e a camada *web* de visualização desenvolvida utilizando *Angular*. A arquitetura do projeto pode ser visualizada na Figura 4.

A camada de análise de dados foi desenvolvida na linguagem R, o fluxo é definido na Figura 5. O objetivo do R é a análise computacional e visualização gráfica de dados (TEAM, 2000). A utilização de bibliotecas permite que funcionalidades sejam agregadas ao projeto e facilitem o processo de desenvolvimento e análise.

Os dados necessário para a análise foram obtidos de repositórios públicos como TCGA[1] e GEO[2]. Estes dados são provenientes da análise RNA-seq (Seção 2.1.2). A biblioteca *GEO-*

[1] Disponível em https://www.cancer.gov/ccg/research/genome-sequencing/tcga
[2] Disponível em https://www.ncbi.nlm.nih.gov/geo/

Figura 4 – Arquitetura C-Gemis



Fonte: Elaborado pelo autor (2023)

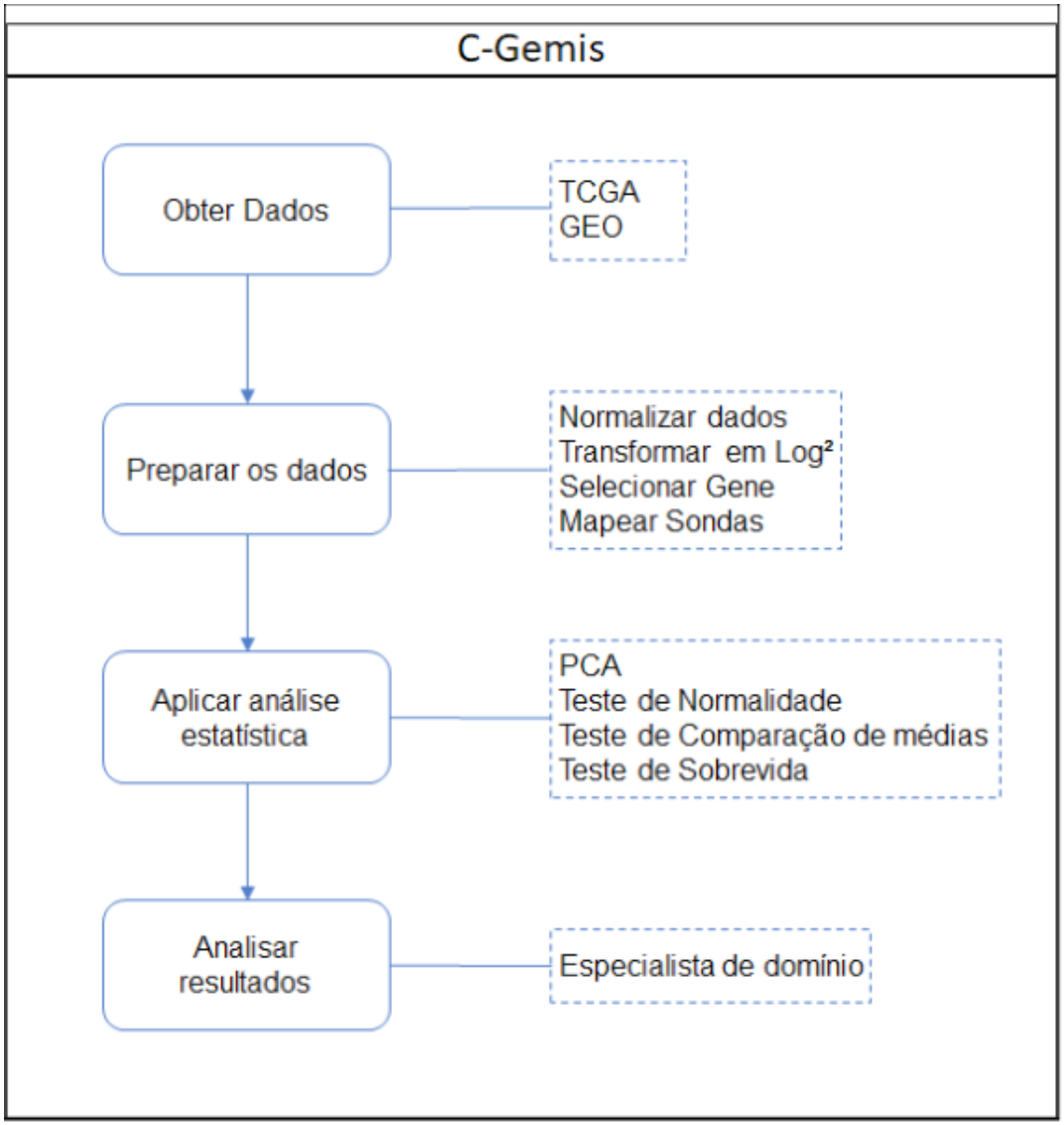
query foi utilizada no *download* de dados do GEO (DAVIS; MELTZER, 2007). Estes dados são preparados pelo R utilizando métodos do repositório *BioConductor*, como o *TCGAbiolinks*.

Após a obtenção dos dados, os mesmos são preparados para a análise. Os dados são normalizados utilizando a análise de conteúdo de GC e filtrado apenas o resultado entre o primeiro e terceiro quartil para evitar viés (ROSSETTO, 2021). Ocorre a transformação logarítmica dos dados, técnica esta que tem como objetivo realizar a distribuição de dados uniformemente (MCKNIGHT *et al.*, 2019).

Após estes processos é iniciado a análise estatística. As amostras de câncer gástrico e de tecidos não cancerígenos passam pela Principal Component Analysis (PCA) (DONG *et al.*, 2022). Genes provenientes do repositório GEO necessitam do mapeamento da sonda devido a codificação. É aplicado o teste de Shapiro-Wilk (YAMANAPPA *et al.*, 2018) para observar a distribuição entre os dados de tecido tumoral e não tumoral e determinar se devem ser utilizados métodos paramétricos. A última etapa é a análise de sobrevida, utilizando as classificações de Lauren (ROSSETTO, 2021). Cinco comparações são realizadas para análise de métricas e resultados: entre os tumores utilizando a classificação de Lauren (XUE *et al.*, 2019), a alta e baixa expressão de gene do tumor difuso, a alta e baixa expressão de gene do tumor intestinal, a classificação do TCGA e a classificação da *World Health Organization (WHO)*.

A aplicação *web* do C-Gemis realiza a comunicação com o ambiente em R. Envia a soli-

Figura 5 – Fluxograma da Aplicação R do C-Gemis



Fonte: (ROSSETTO, 2021)

citação ao servidor e recebe a resposta, assim como especificado na arquitetura cliente-servidor (Seção 2.3.1). Os resultados gerados pela análise devem ser disponibilizados para *download*. Esta funcionalidade atualmente não é suportada, sua implementação exige conhecimento entre a comunicação de aplicações isoladas que muitas vezes são estruturadas em tecnologias diferentes.

2.3 ARQUITETURA DE SOFTWARE WEB

Um dos maiores desafios da *web* é garantir o funcionamento harmônico entre diversas tecnologias que se comunicam entre si. A tecnologia utilizada no desenvolvimento de um serviço ou aplicação é apenas a ferramenta para o produto final. É fundamental compreender e utilizar os padrões de arquitetura de *software*, comunicação e protocolos correspondentes.

Com o advento do paradigma da orientação a objetos, modelar objetos baseado em representações da vida real, surgiu a arquitetura orientada a serviços, Service-oriented architecture (SOA). A SOA permitiu o desacoplamento de funcionalidades, evitando desenhos arquiteturais confusos e frágeis (GAMMA *et al.*, 2000). Seu uso se tornou difuso entre empresas por permitir integração entre sistemas legados e simplificar processos de negócios (CHAVES, 2019). O fundamento desta arquitetura é o barramento de serviço, meio de comunicação em comum para diversas tecnologias. Poder ser utilizado o Extensible Markup Language (XML) ou algum *web service* para comunicação (GAMMA *et al.*, 2000). O serviço deve ser projetado para se tornar um componente, permitindo ser integrado facilmente (SORENSEN, 2020).

O provedor de serviços utiliza o barramento para publicar os serviços disponíveis no padrão Universal Description, Discovery and Integration (UDDI) e na especificação Web Services Description Language (WSDL) (GAMMA *et al.*, 2000). Estes registros criados com informações sobre o serviço podem ser acessados por um cliente a fim de obter as especificações. A comunicação com o cliente é realizada utilizando o protocolo Simple Object Access Protocol (SOAP) ou REST. O XML foi utilizado durante um longo período de tempo como o padrão da indústria para estruturar dados a serem transitados (SORENSEN, 2020).

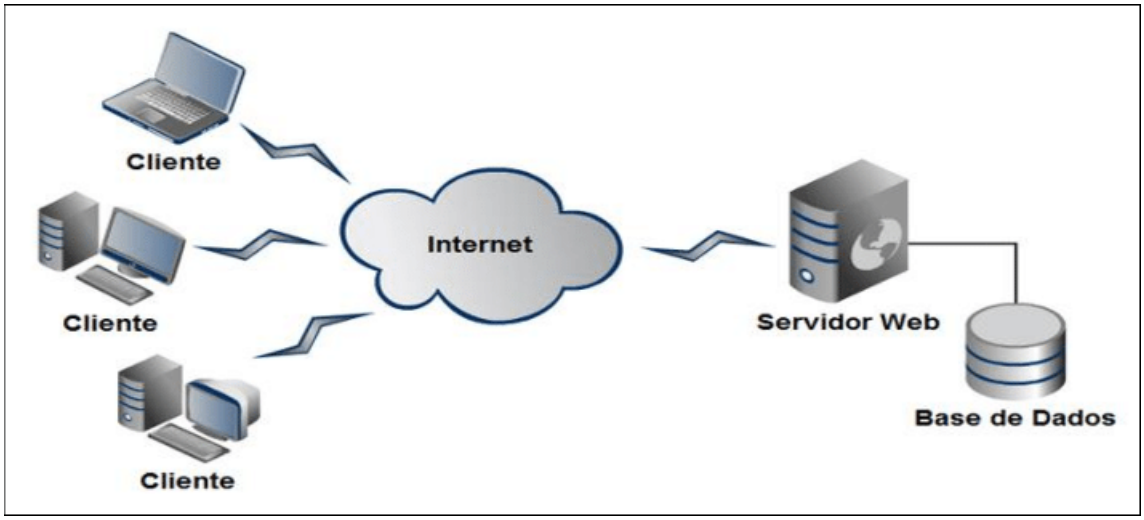
A orientação a serviços possibilitou o paradigma cliente-servidor. Clientes realizam solicitações ao Servidor (FOROUZAN; MOSHARRAF, 2013). O paradigma é implementado pela arquitetura cliente-servidor (Seção 2.3.1). A necessidade de comunicação entre processos deu origem a definição de API (Seção 2.3.2). Uma interface que define conjuntos de instruções entre a camada de aplicação e o sistema operacional (FOROUZAN; MOSHARRAF, 2013).

2.3.1 Arquitetura Cliente-Servidor

Dentre as arquiteturas de serviço *web* a mais utilizada é a arquitetura cliente-servidor (GAMMA *et al.*, 2000), muitas vezes implementando o modelo REST. A arquitetura cliente-servidor é fundamentada em serviços distribuídos (MARINO, 2016). Um conjunto de servidores disponibilizam seus serviços para acesso de requisições dos clientes (ANDREW; DAVID; FEAMSTER, 2021), conforme mostrado na Figura 6. Os clientes podem ser definidos como instâncias de execução do programa. A comunicação entre cliente e servidor é feita via Remote Procedure Call (RPC) que implementa um protocolo solicitação resposta (MARINO, 2016). Atualmente utilizada por aplicações web e que implementam por padrão o protocolo Hypertext Transfer Protocol (HTTP) para comunicação. Sua vantagem é a capacidade de escalabilidade e alta disponibilidade (MARINO, 2016). Por outro lado é suscetível a falhas de rede ou ataques, por consequência causando indisponibilidade do serviço.

REST é o modelo de *web service* mais utilizado (GAMMA *et al.*, 2000). Surge como uma alternativa mais rápida e leve ao protocolo SOAP (BELQASMI *et al.*, 2012). O protocolo de comunicação para dados é de escolha do desenvolvedor, mas o método mais comum é o HTTP. O retorno de solicitações pode utilizar diversos formatos como HyperText Markup Lan-

Figura 6 – Exemplo de arquitetura cliente servidor



Fonte: Researchgate (2013)

guage (HTML), XML ou o JavaScript Object Notation (JSON). No modelo REST não existe a dependência de estados, uma requisição deve conter todos os dados exigidos pelo servidor para processar a chamada (GAMMA *et al.*, 2000).

Serviços *web* que implementam este modelo são chamados de *RESTful*. As principais operações que o REST define são baseadas no princípio Create, Read, Update and Delete (CRUD) (GAMMA *et al.*, 2000). O protocolo HTTP as implementam utilizando *POST*, *GET*, *PUT* e *DELETE*, respectivamente e como pode ser observado na Tabela 1 de métodos.

O protocolo HTTP é o protocolo fundamental da camada de aplicação da *web* (KUROSE; ROSS; ZUCCHI, 2005), atualmente o predominante (ANDREW; DAVID; FEAMSTER, 2021). Servidores *web* disponibilizam objetos *web* para acesso de clientes, estes objetos podem ser páginas HTML, fotos, vídeos ou o agregado de arquivos. O acesso via HTTP é feito utilizando *Uniform Resource Locator (URL)* (KUROSE; ROSS; ZUCCHI, 2005). URL tem como propósito definir um nome para a página que permita acessos na *web*. Podem ser divididas em três partes, o protocolo, o endereço da máquina hóspede do *site* e o caminho interno (ANDREW; DAVID; FEAMSTER, 2021). Para ocorrer a troca de informações entre um cliente e um servidor, é necessário o estabelecimento de uma conexão. Esta conexão segue os princípios do protocolo de transporte Transmission Control Protocol (TCP) (KUROSE; ROSS; ZUCCHI, 2005).

O HTTP suporta dois tipos de conexões, persistentes e não persistentes. Conexões não persistentes podem sobrecarregar o servidor (FOROUZAN; MOSHARRAF, 2013), para cada objeto solicitado é necessário alocar *buffers* TCP (KUROSE; ROSS; ZUCCHI, 2005). Conexões persistentes permitem o envio de páginas *web* e todos os objetos que as compõem, com o estabelecimento de apenas uma conexão TCP. O servidor fecha a conexão após um tempo de inatividade pré definido (KUROSE; ROSS; ZUCCHI, 2005). Dentre as conexões persistentes existem duas variações, com e sem paralelismo. Na variação sem paralelismo, novas requisições do cliente precisam

GET	Solicitação sem informações no corpo da mensagem
HEAD	Solicitação de informações sobre a página
PUT	Envio de documento
POST	Envio com informações no corpo da mensagem
TRACE	Servidor devolve a solicitação do cliente
DELETE	Remoção de documento
CONNECT	Reservado
OPTIONS	Retorna opções disponíveis da página

Tabela 1 – Tabela de métodos HTTP.

1XX	Informativos
2XX	Sucesso da solicitação
3XX	Redirecionamento a outra URL
4XX	Falha relacionada ao cliente
5XX	Falha relacionada ao servidor

Tabela 2 – Tabela de status HTTP.

esperar a resposta do servidor das solicitações já enviadas. Por padrão o HTTP utiliza o paralelismo. Requisições são feitas de modo sequencial pelo cliente, o servidor responde com objetos sequencialmente (KUROSE; ROSS; ZUCCHI, 2005).

Dois formatos de mensagens são definidos, um para resposta e outra para a solicitação. Na mensagem de solicitação três campos importantes são definidos na primeira linha; o método, a URL e a versão (FOROUZAN; MOSHARRAF, 2013). O método é responsável por declarar a intuição da solicitação, os métodos disponíveis podem ser vistos na Tabela 1. O campo URL corresponde ao endereço *web*, a versão diz a respeito do HTTP, atualmente na 1.1.

O cabeçalho permite definir propriedades a serem enviadas ao servidor. O cliente pode informar ao servidor qual padrão de carácter aceita, mídias, esquemas de codificação ou idiomas que suporta (FOROUZAN; MOSHARRAF, 2013). Caso o método utilizado seja *POST* ou *PUT* é possível enviar informações no corpo da requisição. A mensagem de resposta possui estrutura semelhante. Na primeira linha entretanto o primeiro campo diz a respeito da versão HTTP, o segundo campo corresponde ao *status* resposta da solicitação e o terceiro campo a descrição textual do *status* (FOROUZAN; MOSHARRAF, 2013), exemplificados na Tabela 2.

2.3.2 API

Application Programming Interface pode ser definido como uma interface entre componentes de *software* (LAMOTHE; GUÉHÉNEUC; SHANG, 2021). Uma API contém documentação especificando as requisições que são aceitas e a estrutura da resposta. Esta interface permite implementar diversos protocolos e arquiteturas, por muito tempo o PHP foi uma das ferramentas mais utilizadas no desenvolvimento (ANDREW; DAVID; FEAMSTER, 2021). API SOAP utiliza o XML para trocar dados, seu uso se encontra em decadência. API *Web Sockets* permite a comu-

nicação bilateral, o servidor não fica restrito a solicitação do cliente. A versão abordada neste trabalho é a API REST.

No artigo (PRAYOGI *et al.*, 2020), os autores apresentam o desenvolvimento e a comparação entre duas propostas de REST API, uma utilizando a tecnologia PHP e a outra *NodeJS* (Seção 3.4.3). O ambiente foi configurado da seguinte forma. Um banco de dados relacional contendo uma tabela com 3920 dados. Utilização do *Apache Web Server* [3] para os hospedar ambas API. Ambas API apresentam dois *endpoints*, um para retornar todos os dados da tabela e outro para retornar específicos mediante parâmetros.

Para medir os resultados foram utilizadas duas medidas de performance. O tempo de resposta entre o momento que a solicitação do cliente é enviada até o momento que é respondida. A outra medida definida foi a capacidade de processamento de requisições em um período fixo de tempo e a porcentagem de requisições perdidas. Foi possível observar um resultado melhor pela API em *NodeJS* (PRAYOGI *et al.*, 2020). O tempo de resposta médio para o *endpoint* de todos os dados da tabela da API em PHP variou entre 244ms a 266ms. Por outro lado a implementação em *NodeJS* apresentou resultados melhores entre 138ms a 156ms. É possível observar pelos resultados como o *NodeJS* lida melhor com requisições paralelas. Com 300 requisições simultâneas a API em PHP começa a apresentar dificuldade em atender as solicitações, apenas 90% das requisições são respondidas. Em casos com 1000 requisições simultâneas este número piora para 48,70%, a implementação em *NodeJS* por outro lado consegue responder 100% dos casos (PRAYOGI *et al.*, 2020).

2.4 TRABALHOS RELACIONADOS

O processo de automatização e integração de dados de análise tem ganhado destaque em diversos campos de pesquisa, são denominados *workflows* científicos (KHAN *et al.*, 2019). Entre as áreas que trabalham com grandes dados, o domínio da genética é considerado o de maior demanda (KHAN *et al.*, 2019). O rápido crescimento da produção de dados levou ao desenvolvimento de inúmeros *workflows* científicos. Para análise foram considerados apenas *workflows* com interface gráfica e gratuitos, podendo ser plataformas *web* ou programas nativos *desktop*.

Sarek (GARCIA *et al.*, 2020) é um *workflow* desenvolvido para computadores POSIX, podendo ser instalado em servidores e ambientes UNIX. O desenvolvimento foi realizado utilizando *Nextflow*, linguagem dedicada para a criação de *pipelines* científicos. Há suporte nativos para *container* como Google Cloud e AWS. Os pacotes e dependências são geridos pelo ambiente Conda e disponibilizados no *Docker* (GARCIA *et al.*, 2020). Como entrada são aceitos arquivos no formato FASTQ com as sequências das amostras e um arquivo no formato TSV com a descrição dos arquivos de sequência.

Devido a ser uma aplicação que roda localmente em máquina é necessário um grande

[3] Disponível em https://httpd.apache.org/

poder computacional. É recomendado ao menos 16 *cores*, 128 GB de memória RAM e ao mínimo 4 TB de armazenamento. Em análises completas de casos de tumores de Meduloblastoma foram necessárias mais de 48 horas. A vantagem desta ferramenta é a automação do processo que começa na instalação de dependências até a geração dos arquivos resultantes da análise (GARCIA *et al.*, 2020).

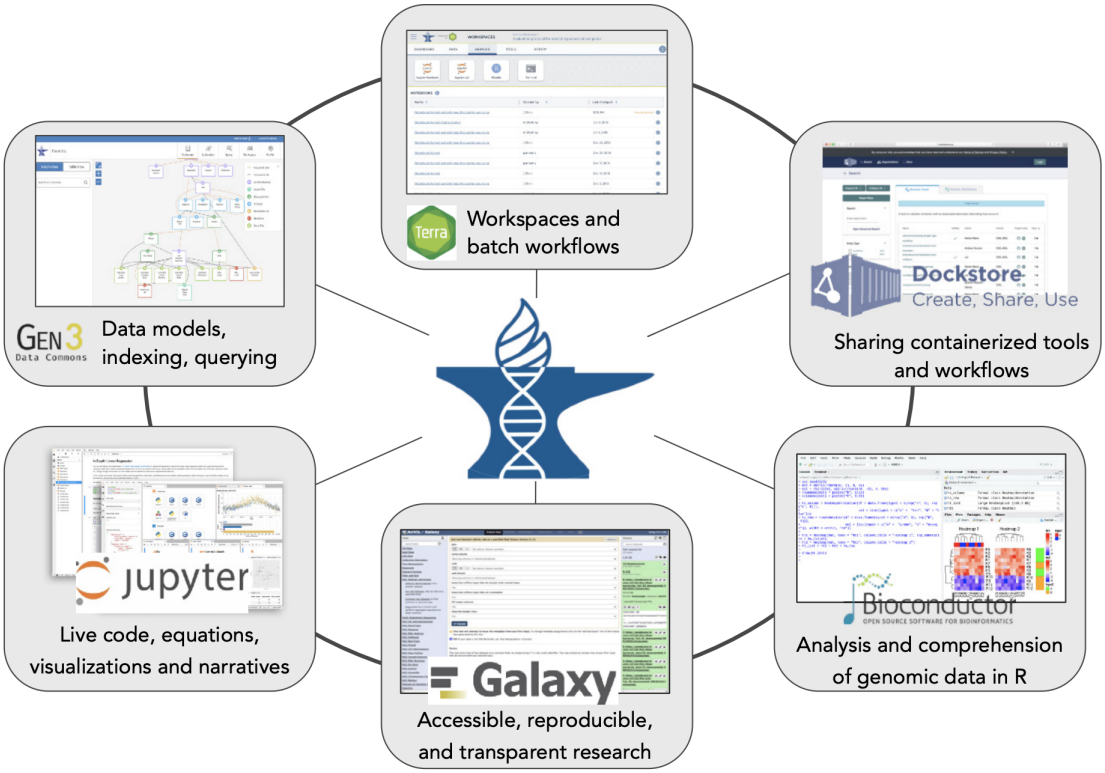
Entre as ferramentas de *workflow* científico de genomas desenvolvidas, pode ser citada a *GO FEAT*(ARAUJO *et al.*, 2018). Comparada a outras ferramentas já existentes, a *GO FEAT* é completamente gratuita, implementa interface gráfica de uso fácil e apresenta os resultados de maneira amigável. O *back-end* da ferramenta foi desenvolvido utilizando PHP. Para análise dos dados de *input* é realizado a comunicação com diversas API. As seguintes API são chamadas: EMBL-EBI (ARAUJO *et al.*, 2018), UniProt (ARAUJO *et al.*, 2018), QuickGo (ARAUJO *et al.*, 2018) e SEED (ARAUJO *et al.*, 2018). Os resultados são salvos no banco de dados MySQL[4] e apresentados em gráficos e tabelas. A camada *web* foi desenvolvida com o uso de HTML5, CSS3 e *Javasript*. Utilizando uma sequência aleatória, GO FEAT levou cerca de 4 minutos para análise, 10 minutos a menos que a ferramenta Blast2GO. Arquivos grandes podem representar uma limitação na análise completa dos genomas, devido a necessidade de comunicar com interfaces e API. A análise completa do genoma da *Drosophila melanogaster* leva cerca de 30 horas (ARAUJO *et al.*, 2018).

Outra plataforma desenvolvida foi a NHGRI *Genomic Data Science Analysis, Visualization, and Informatics Lab-space* (AnVIL) (SCHATZ *et al.*, 2022). AnVIL é uma arquitetura proposta para o armazenamento de dados genéticos, análise e compartilhamento de resultados baseado em computação na nuvem. O ambiente de computação está hospedado na Google Cloud Platform (GCP). O AnVIL apresenta acesso a aplicações individuais que executam tarefas distintas, mas que se comunicam por meio de API e diretório de arquivos, arquitetura exemplificada na Figura 7.

As aplicações podem ser acessadas através do site do portal [5]. O número de dados de genomas humanos disponível é maior que 280.000, dados estes provenientes de mais de 240 repositórios (SCHATZ *et al.*, 2022). Entre as aplicações disponíveis estão o Gen3, Terra, Dockstore, Jupyter notebook, RStudio, Galaxy e Bioconductor (SCHATZ *et al.*, 2022). Podem ser destacados o Jupyter Notebook, *software* de código aberto *web* que permite a execução de código e análise em tempo real. O Jupyter suporta principalmente as linguagens Python e R. Permite integração com outras aplicações por possuir suporte ao Terra. O RStudio é um ambiente em R que permite a execução de análises avançadas e uma visualização gráfica detalhada, é executado através do Terra. Bioconductor é uma poderosa ferramenta de métodos de análise estatísticos e gráficos para dados de genomas na linguagem R. Galaxy é uma plataforma *web* que disponibiliza diversos métodos para análise (SCHATZ *et al.*, 2022).

[4] Disponível em https://www.mysql.com/
[5] Disponível em https://anvilproject.org/

Figura 7 – Arquitetura AnVIL



Fonte: (SCHATZ *et al.*, 2022)

O projeto AnVIL pretende democratizar o acesso a ferramentas de análise, *datasets* protegidos e os recursos para realizar a análise (SCHATZ *et al.*, 2022). Entretanto, os custos da computação na nuvem se apresentam como a maior desvantagem do projeto atualmente. Diversas instituições estão investindo na tecnologia, mas a replicação de grandes *datasets* para a nuvem ainda é uma preocupação.

A plataforma *web* GenSAS permite o envio de arquivos *assembly* no formato FASTA para análise (HUMANN *et al.*, 2019). É recomendado o uso da ferramenta BUSCO para estimar a qualidade da amostra antes de a enviar para análise. Arquivos auxiliares com evidências também são suportados no formato GFF3. O ambiente utiliza integração com o JBrowse a o *plugin* Apollo para visualizar e realizar a curadoria dos dados. GenSAS oferece seis ferramentas para realizar a anotação funcional, BLAST+, Diamond, InterProScan, Pfam, SignalP e Target P (HUMANN *et al.*, 2019). Os arquivos finais podem ser exportados nos formatos GFF3, FASTA e formatos de texto.

3 PROPOSTA DE SOLUÇÃO

Esta seção tem como objetivo contextualizar e apresentar a solução proposta para os requisitos e necessidades do projeto. Serão apresentados diagramas com a arquitetura, protótipos de telas e requisitos. As tecnologias envolvidas no desenvolvimento serão especificadas.

3.1 REQUISITOS

Conforme apresentado o C-Gemis (Seção 2.2), foram levantados requisitos que o projeto não atende atualmente. A possibilidade do usuário baixar o resultado da análise é atualmente o maior déficit da aplicação, a tornando inoperacional. A estruturação de uma nova arquitetura do projeto para permitir a escalabilidade e implementação de novas ferramentas é fundamental. A criação de uma documentação padronizada também se faz necessária.

Na camada *web* os seguintes requisitos foram levantados;

1. *Input* do usuário: Por meio da página *web* o usuário deve conseguir informar o gene escolhido a uma lista, este seguindo a nomenclatura HUGO Genes[1] (ROSSETTO, 2021). Ao disparar a ação a API será chamada para resolver a solicitação.

2. Visualização dos dados: Após a análise do gene o resultado deve ser disponibilizado ao usuário. O resultado deverá ser apresentado em tela, a possibilidade de baixar os arquivos também deve estar disponível.

3. Informações sobre o projeto: O usuário deve conseguir localizar informações sobre a aplicação e seus criadores, caso deseje realizar contato.

4. Integração com a plataforma *Google Analytics* para fornecedor dados a respeito de uso e acessos da plataforma

5. Utilização da ferramenta *ReCaptcha* para proteger a aplicação contra *spam* e tráfego artificial de robôs.

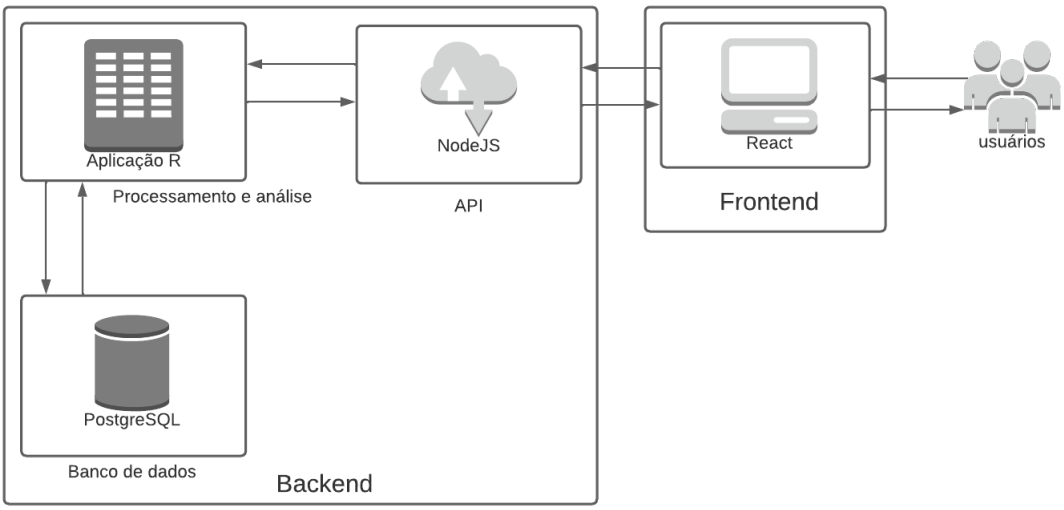
3.2 ARQUITETURA DE SOFTWARE

Baseado na arquitetura cliente-servidor, é proposto uma nova arquitetura ao projeto C-Gemis, disponível na Figura 8. No *Frontend* foi optado pelo uso da biblioteca *React* (Seção 3.4.2) com a linguagem *Typescript* (Seção 3.4.1). A API será desenvolvida utilizando *NodeJS* (Seção 3.4.3) ambiente em *javascprit* assíncrono orientado a eventos para servidores que

[1] Disponível em https://www.genenames.org/

permite alta escalabilidade (CASCIARO; MAMMINO, 2020). O processamento e análise de dados permanece no ambiente em R, assim como o banco de dados permanece no ambiente *Postgresql*.

Figura 8 – Arquitetura proposta



Fonte: Elaborado pelo autor (2023)

No atual projeto do C-Gemis as requisições são feitas diretamente ao ambiente em R. Arquitetura esta que leva solicitações ao banco de dados passar pela aplicação em R, gerando dependência não necessária e complexidade para futuras mudanças. Estes *endpoints* serão abstraídos e implementados na API proposta. A separação entre camadas permitirá isolar as aplicações e restringir os ambientes as suas funcionalidades básicas, além de permitir maior disponibilidade. Os seguintes *endpoints* serão necessários:

1. *GetGene*: Requisição do tipo *POST*. O corpo da solicitação conterá um campo informando o gene. A API retornará como verdadeiro ou falso se o gene já está registrado no banco de dados.

2. *GetImage*: Requisição do tipo *POST*. Deverá retornar o nome da imagem, será informado o seu caminho.

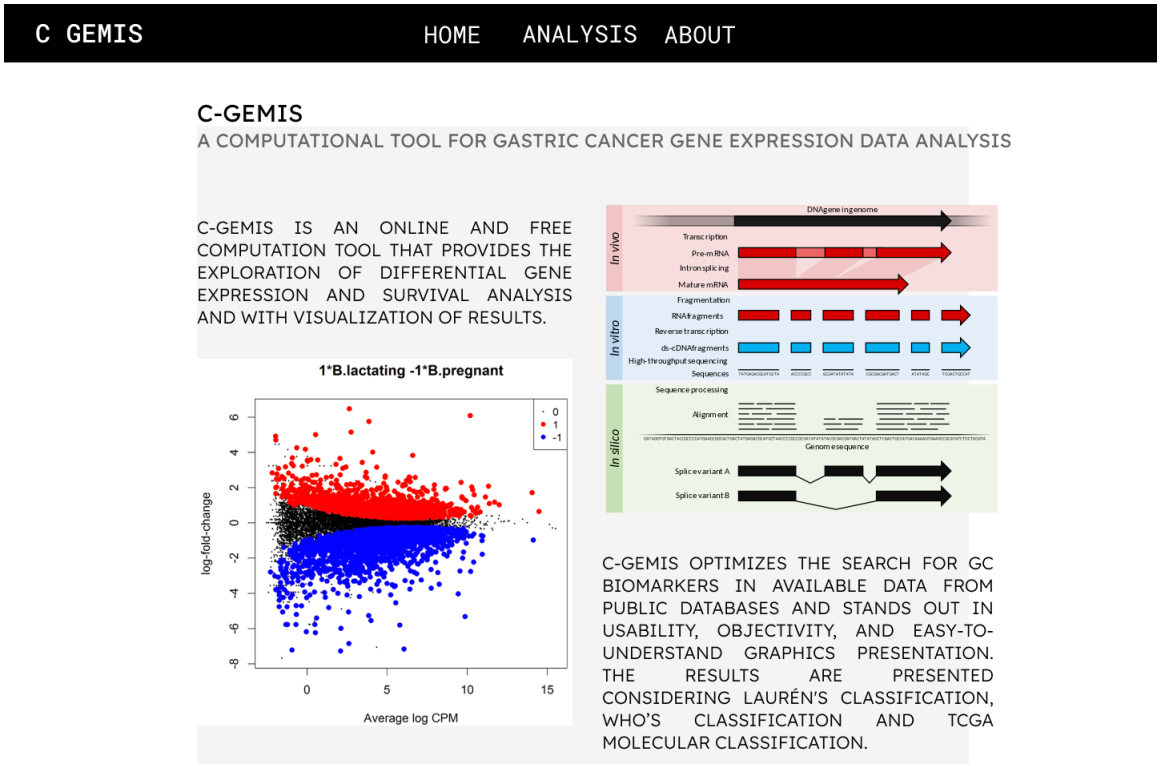
3. *DownloadImage*: Requisição do tipo *POST*. As imagens informadas no corpo da solicitação serão agregadas a um arquivo compactado que será disponibilizado para *download* ao usuário.

4. *RunGene*: Requisição do tipo *POST*. A solicitação irá conter uma lista de genes. Para cada gene informado será realizado a análise. A resposta da requisição deverá conter uma lista com os resultados, associando o nome da imagem e seu caminho respectivamente.

3.3 PROTÓTIPOS DE TELA

A Figura 9 apresenta a tela de introdução da aplicação. As imagens utilizadas nesta tela são figurativas a apresentam propósito ilustrativo. Utilizando o menu no topo da página é possível navegar entre os recursos do site.

Figura 9 – Tela *home*

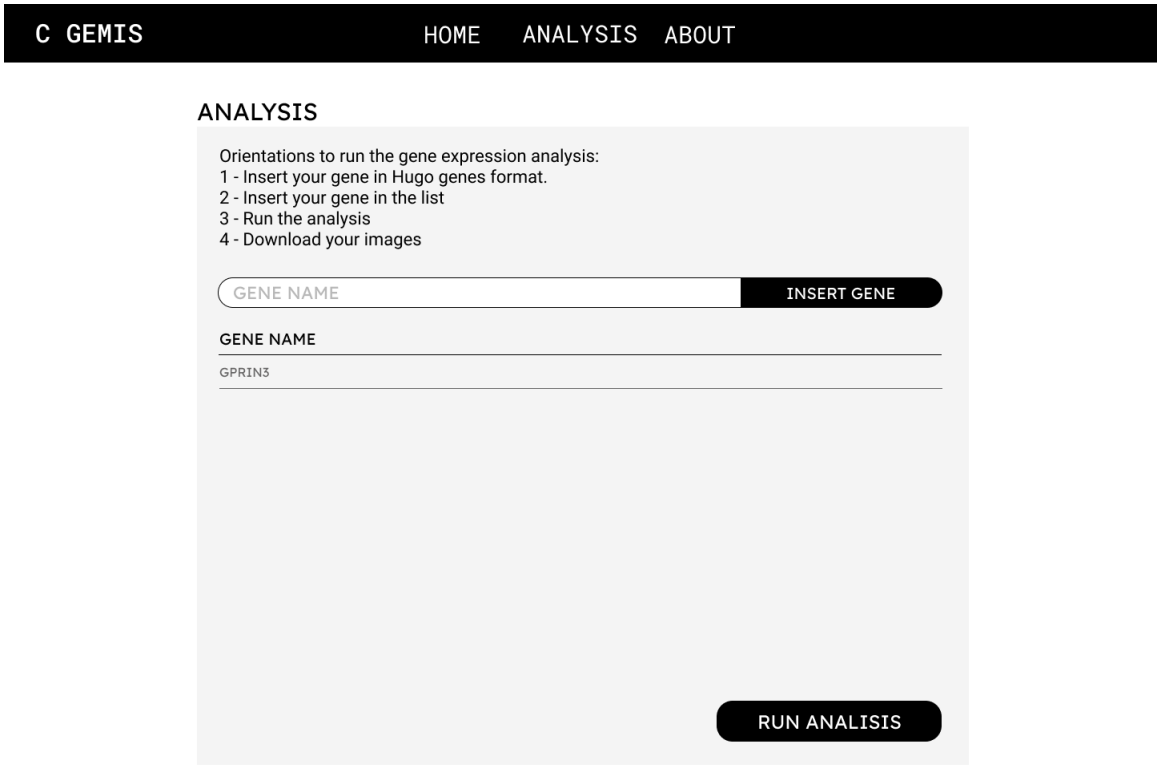


Fonte: Elaborado pelo autor (2023)

O *input* de dados pelo usuário é realizado na tela representada na Figura 10. A tela é acessível utilizando a opção *analysis* no menu superior . O usuário informará o gene, o adicionará na lista clicando em *Insert gene*. Após adicionar os genes desejados é necessário clicar em *Run analysis* para começar o processamento. Quando finalizado a análise o usuário será redirecionado a tela de resultados, representada na Figura 11.

Após processamento os resultados deverão ser mostrados em telas utilizando-se de imagens. Os arquivos resultantes do processamento podem ser baixados para a máquina do usuário. A captura de arquivos poderá ser feita clicando no botão *download*, conforme apresentado na Figura 11.

Figura 10 – Tela de *input*



Fonte: Elaborado pelo autor (2023)

3.4 FERRAMENTAS

3.4.1 Typescript

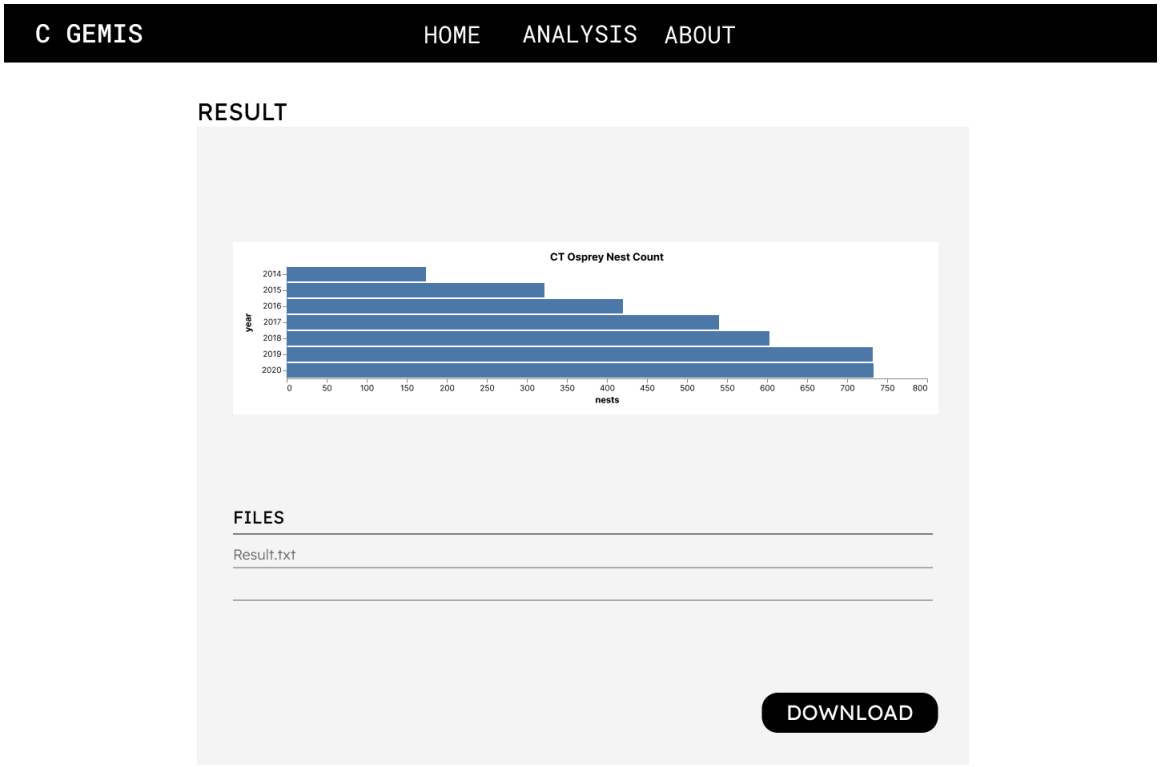
Typescript é uma linguagem de alto nível desenvolvida pela *Microsoft*, é definida como um *superset* ao *Javascript*. O *Javascript* surgiu como uma linguagem simples com intuito de executar pequenos *scripts* nos navegadores [2]. Com o passar do tempo sua aplicação se tornou difusa e usada amplamente em projetos de grande escala. O seu uso em grandes projetos acarretou na preocupação com a segurança e comportamentos inesperados do código. O *Typescript* adota a tipagem estática, verificando o código na pré execução e identificando erros. Segundo dados disponibilizados pelo *GitHub* [3], *Typescript* foi a quarta linguagem mais utilizada em 2022. O aumento de uso em relação a 2021 foi de 37,8%.

3.4.2 React

React é uma biblioteca para *Javascript* e *Typescript* focada na renderização de *User Interface* utilizando componentes [4]. A filosofia do desenvolvimento *React* está no uso de com-

[2] Disponível em https://www.typescriptlang.org/
[3] https://octoverse.github.com/2022/top-programming-languages
[4] Disponível em https://react.dev/learn/describing-the-ui

Figura 11 – Tela de resultado



Fonte: Elaborado pelo autor (2023)

ponentes, trechos de códigos que podem ser reutilizados tendo funções específicas, evitando repetição de código. Seu uso se tornou popular pela ascensão de Single Page Application (SPA), páginas de carregamento único (WIERUCH, 2020). Este tipo de aplicação realiza uma requisição inicial, o servidor entrega o HTML básico e código *Javascript*. Futuras interações na aplicação são resolvidas pelo *framework* com o código *Javascript*, tornando o uso mais rápido e fluído ao usuário. A carga de requisições ao servidor é diminuída pois cada iteração não exige que um novo arquivo HTML seja entregue ao cliente (WIERUCH, 2020).

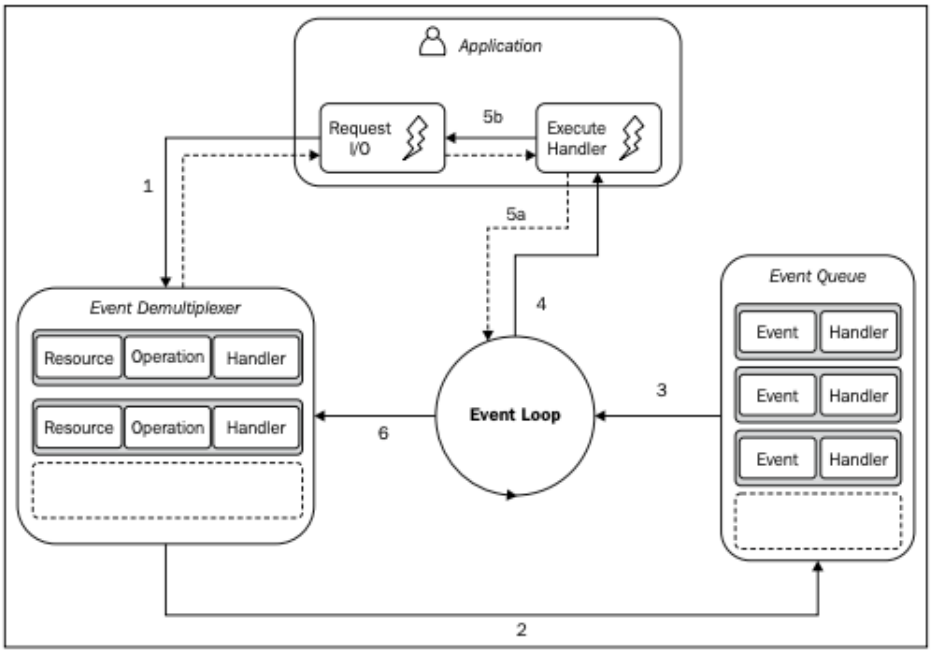
Segundo dados fornecidos pela *npm trends React* foi o *framework* para *javascript* com maior número de *downloads*. Entre junho de 2022 a junho de 2023 foram mais de 20 milhões de *downloads* [5]. Dados de análise do *Stackoverflow* mostram que *React* é responsável por cerca de 5% das perguntas realizadas na plataforma nos últimos meses. Um número expressivo e maior que os *frameworks* concorrentes *Angular* e *Vue*, com 2% e 1% respectivamente [6].

[5] Disponível em https://npmtrends.com/angular-vs-react-vs-vue
[6] Disponível em https://insights.stackoverflow.com/trends

3.4.3 NodeJS

NodeJs é um ambiente de execução *javascript* baseado na orientação a eventos assíncrono [7]. O *NodeJs* tem como característica rodar como uma aplicação *single thread*, exigindo menos recursos que servidores tradicionais *multi threads*. O princípio do *NodeJs* está nos conceitos de não ser bloqueante e ser assíncrono. As requisições recebidas são colocadas na fila de eventos. O *Event loop*, processo rodando na *thread* é que realiza a gerência desta fila, conforme Figura 21. Utilizando a biblioteca *libuv* [8] as operações são enviadas ao sistema operacional de forma assíncrona. O *Event loop* é responsável por disparar o *callback* com a resposta da requisição quando finalizada (CASCIARO; MAMMINO, 2020). As vantagens do *NodeJs* estão em não ocorrer *deadlocks* e na melhor eficiência comparado a serviços *multi threads* [7].

Figura 12 – *NodeJS* funcionamento



Fonte: (CASCIARO; MAMMINO, 2020)

3.4.4 Material UI

Material UI é uma biblioteca de componentes de código aberto desenvolvida para *React* [9]. Oferece ao desenvolvimento *frontend* uma grande gama de componentes prontos para uso. Estes componentes permitem customização e agilizam o processo de escrita de códigos *boilerplate*. Códigos *boilerplate* podem ser caracterizados como trechos que são repetidos diversas vezes no projeto com mínima alteração.

[7] Disponível em https://nodejs.org/en/about
[8] Disponível em https://libuv.org/
[9] Disponível em https://mui.com/material-ui/

Material UI implementa o *Material* desenvolvido pela *Google*. O *Material* é uma linguagem visual de *design* criada com objetivo de padronizar interfaces visuais [10]. No ano de 2023 a *Google* utiliza o Material em grande parte de suas aplicações, como o *Gmail*, *Youtube*, *Google Drive* e *Google Maps*. O *Material* fornece diversas ferramentas para o desenvolvimento visual, como o ícones, fontes, escalas de cores, formas e escalas de texto.

3.5 TESTES

Para validação das funcionalidades do projeto foram estabelecidos testes. Os seguintes testes estão organizados em ordem cronológica de desenvolvimento.

1. **Teste de comunicação a API**: Ao enviar uma solicitação de teste, a API deverá retornar o status HTTP 200, indicando sucesso.
2. **Teste de obtenção de resultado da aplicação R**: A API deverá chamar a aplicação R para obter um resultado. A aplicação R irá enviar como resposta uma imagem de teste pré definida. A API deverá salvar esta imagem no diretório e retornar o caminho.
3. **Teste envio de gene**: Através da interface *web* uma lista de genes deve ser enviada a API. Esta lista deve ser identificada pela API.
4. **Teste *display* de resultado**: Arquivos de teste deverão ser recuperados do banco e corretamente mostrados no *front-end*.
5. **Teste da geração correta de resultado**: Será realizada a análise de um gene X no atual e no novo C-Gemis. O resultado deverá ser o mesmo para validar a funcionalidade análise.
6. **Teste de disponibilização do aplicação**: A aplicação já com suas funcionalidades implementadas será disponibilizada para acesso na internet

[10] Disponível em https://m2.material.io/design

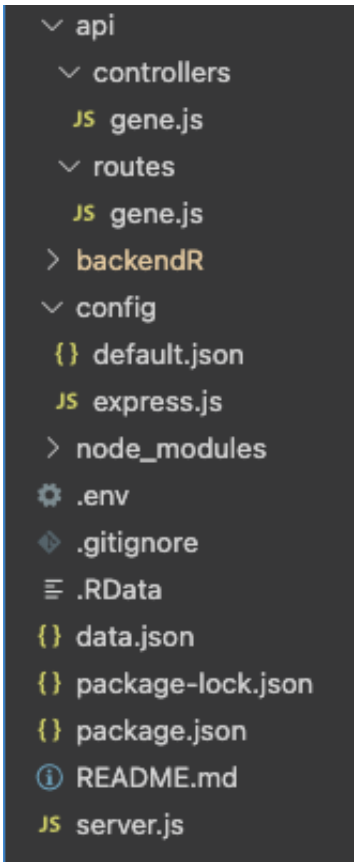
4 DESENVOLVIMENTO

Este capítulo tem como objetivo apresentar o desenvolvimento do projeto. Será explicado como foi criado as partes que compõem a aplicação, a API e o *frontend* e como foi realizado a hospedagem. Trechos de códigos serão apresentados que justifiquem o funcionamento além de explicações sobre decisões que foram necessárias serem tomadas durante o desenvolvimento. É apresentado as telas e suas funcionalidades implementadas.

4.1 API

A estrutura da API pode ser observada conforme Figura 13. A declaração de rotas foi isolada das funções de controle, seguindo boas práticas do desenvolvimento em *NodeJs*. Dados sensíveis como acesso ao banco de dados, dados referentes a caminhos internos e a configuração do servidor devem ser armazenados no arquivo .env [1].

Figura 13 – Estrutura organizacional da API



Fonte: Elaborado pelo autor (2023)

[1] Disponível em https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

Na Figura 14 é possível observar a estrutura de uma função que executa uma consulta ao banco de dados. Definidas como assíncronas para evitar o bloqueio da aplicação ao usuário enquanto aguarda a resposta. Seguindo boas práticas do *NodeJs* referente a tratamento de erros e exceções, é utilizado os comandos *try catch* (CASCIARO; MAMMINO, 2020). A consulta deve retornar a resposta com *status* 200 indicando sucesso, conforme visto sobre o método HTTP (Seção 2.3.1). Qualquer outro cenário além do sucesso na consulta, retornará *status* 500, indicando erro.

Figura 14 – Exemplo de requisição da API

```
const getGenes = (async (req, res) => {

   try{
      const sql = await pool.query('SELECT * FROM gene')
      res.status(200).json(sql.rows)
   } catch (err) {
      res.status(500).send({ error: { message:'unable to get data' }})
   }

})
```

Fonte: Elaborado pelo autor (2023)

Para a integração entre o ambiente *NodeJs* e o ambiente *R* foram testados dois módulos populares entre a comunidade, o *r-integration* [2] e o *r-script* [3]. Ambos apresentam suporte a chamadas assíncronas, entretanto durante os testes com o *r-script*, não foi possível validar tais chamadas. Os constantes erros e a documentação significativamente menor foram determinantes para a escolha do outro módulo, o *r-integration*.

Se tornou necessário modificar a aplicação *R* para integrar com a nova API. No antigo molde a aplicação *R* realizava a consulta diretamente ao banco *PostgreSQL*, processo este agora delegado a API. Devido a quantidade de dados que deve trafegar entre as aplicações e as limitações das funções do *R* com parâmetros dinâmicos, foi escolhido utilizar arquivos JSON. Dados retirados da consulta em Structured Query Language (SQL) são armazenados no arquivo JSON do gene, o nome deste arquivo é passado para a aplicação *R* que realiza a leitura, criando o *dataset* necessário para análise. Se utilizou o mesmo processo para devolver o resultado. A aplicação *R* devolve a API um arquivo JSON contendo o caminho, nome e descrição das imagens da análise de cada gene.

Durante o desenvolvimento do projeto foi notado a possibilidade de simplificar e diminuir o número de *endpoints*. Inicialmente seria feito uma requisição para verificar se já foi realizada a análise de determinado gene e se os arquivos existem, caso não, seria feito outra requisição para verificar se o banco de dados possui registros para realizar a análise, por fim a

[2] Disponível em https://github.com/FabrizioSandri/r-integration
[3] Disponível em https://github.com/joshkatz/r-script

requisição para gerar a análise. Foi possível simplificar este processo a um único *endpoint*, o *runGene*, como pode ser visto na Figura 15.

Figura 15 – *Endpoint runGene*

```
const runGene = (async (req, res) => {
  try{
    const { genes } = req.body
    var files = [];

    for await (const gene of genes){

      if(await existFile(gene)){
        var file = fs.readFileSync(process.env.PATH_R+"/images/"+gene+"/analysis.json",'utf8');
        var formatedFile = await formatResponse(file);
        files.push({[gene]:formatedFile});
        continue
      }

      const sql = await pool.query('SELECT * FROM dados AS d INNER JOIN pacientes AS p ON p.paciente_id = d.paciente WHERE d.gene = $1',[gene])

      if(sql.rows.length < 1) continue

      const fileData = JSON.stringify(sql.rows)
      var filename =  gene + "|" + Date.now().toString() + ".json"

      await fs.writeFile(process.env.PATH_R+"/"+filename,fileData, (err) =>
        err && console.error(err)
      );

      await callMethodAsync(process.env.PATH_R+"/runAnalysis.R", "x", {jsonFile:filename}).then(async (result) => {
        var file = fs.readFileSync(process.env.PATH_R+"/"+result[0],'utf8');
        var formatedFile = await formatResponse(file);
        files.push({[gene]:formatedFile});
      }).catch((error) => {
        console.log(error)
      })
    };
    res.status(200).send(files);
  }catch(err){
    res.status(500).send({ error: { message:'unable to save data' + err }})
  }
```

Fonte: Elaborado pelo autor (2023)

Para cada gene da lista recebida é primeiro verificado se já existe o resultado com as imagens verificando a pasta com o nome do gene. Em casos positivos a pasta é lida e seu conteúdo é retornado através de uma função que normaliza dados. O fluxo segue realizando a busca de dados do banco, casos com retorno vazio são ignorados. Com os dados do banco obtidos é realizado a chamada da aplicação em R que devolverá o resultado que será normalizado. O arquivo JSON normalizado pode ser visto na Figura 16. É retornado uma lista contendo um objeto com o nome do gene, e as imagens dividias em três categorias. As imagens foram dividias em imagens sobre a sobrevivência, expressão gênica e o panorama geral.

APIs usadas em ambientes de produção tendem a usar métodos de segurança como autenticação e cadastro de usuário para evitar o acesso indevido de dados. Devido a necessidade deste projeto, não é necessário implementar um sistema de cadastro e login para obter o *token* de autorização. Um *token* aleatório encriptado utilizando Secure Hash Algorithm (SHA256) foi gerado. Este *token* foi armazenado nas variáveis de ambiente da API e do *frontend*. Toda requisição deve enviar em seu cabeçalho o campo *Authorization* contendo o *token* pre gerado. Aplicações terceiras que não possuem a chave são impossibilitadas de adquirir dados mesmo sabendo o endereço URL da API.

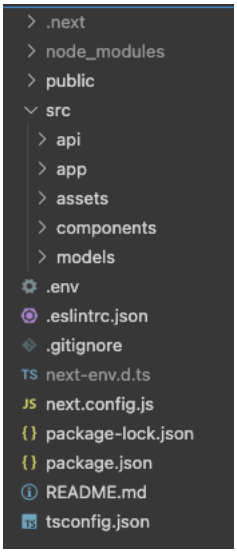
Figura 16 – Arquivo JSON de resposta

Fonte: Elaborado pelo autor (2023)

4.2 FRONTEND

A estrutura da camada *frontend* pode ser observada conforme Figura 13. As páginas da aplicação são organizadas dentro da pasta *app*, seguindo a estrutura de organização em diretórios presente na versão 13 do *NextJs* [4]. Componentes reutilizáveis são organizados na pasta *components*, exemplo da barra de navegação superior. Constantes de uso geral se encontram na pasta *assets*. Objetos utilizados na aplicação foram organizados na pasta *models*.

Figura 17 – Estrutura organizacional do Frontend



Fonte: Elaborado pelo autor (2023)

[4] Disponível em https://nextjs.org/docs/app/building-your-application/routing

O desenvolvimento de componentes visuais utiliza o Material UI, conforme apresentado na Seção 3.4.4. O esquema de cores utilizado pode ser visto na Figura 18, conforme solicitado o *design* implementou a identidade já desenvolvida previamente.

Figura 18 – Esquema visual

Fonte: Designer Center UCS (2018)

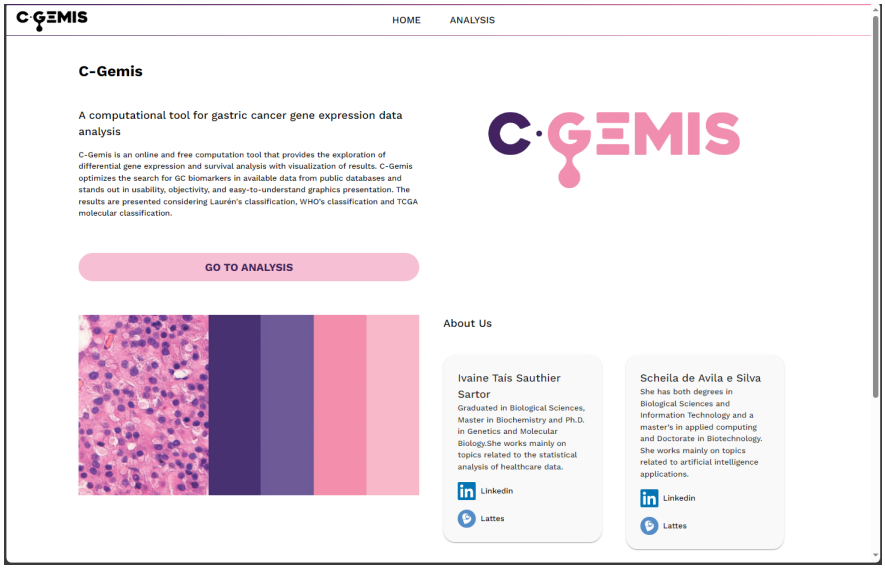
Na Figura 19 pode ser observado a tela inicial do *frontend*. É apresentado uma descrição breve explicando o C-Gemis, informações sobre as pessoas envolvidas no projeto foram disponibilizadas como *cards*. Os textos presentes nesta tela foram adaptados do antigo *frontend*. Na segunda tela ocorre a interação com o usuário, conforme Figura 20. É apresentado as instruções que o usuário deve seguir e ao lado a lista com os genes selecionados para análise. Foi disponibilizado um *hiperlink* para o *site* do Human Genome Organisation (HUGO) [5]. A lista ao ser enviada provocará com que a página apresente uma tela de carregamento enquanto aguarda o retorno da API. A tela de carregamento consiste em dois círculos partindo do mesmo ponto e movendo-se em direções opostas, simulando o processo de divisão celular, como pode ser observado na Figura 21.

Para realizar a comunicação com a API, é utilizado o *Axios* [6]. Este pacote fornece um cliente HTTP baseado em promessa, permite ser usado tanto no *frontend* como no *backend* (ZOU; TANG; LV, 2022). Na Figura 22 é apresentado a requisição feita a API para realizar a análise dos genes. O caminho base da API foi armazenado no arquivo .env, tornando inacessível para o lado do cliente. Assim como mostrado na seção anterior da API, os comandos *try catch* são utilizados para tratar erros evitando a inoperabilidade da aplicação e informar o usuário a resposta da solicitação.

O resultado da análise é transformado no objeto *Analysis*, moldado conforme o JSON

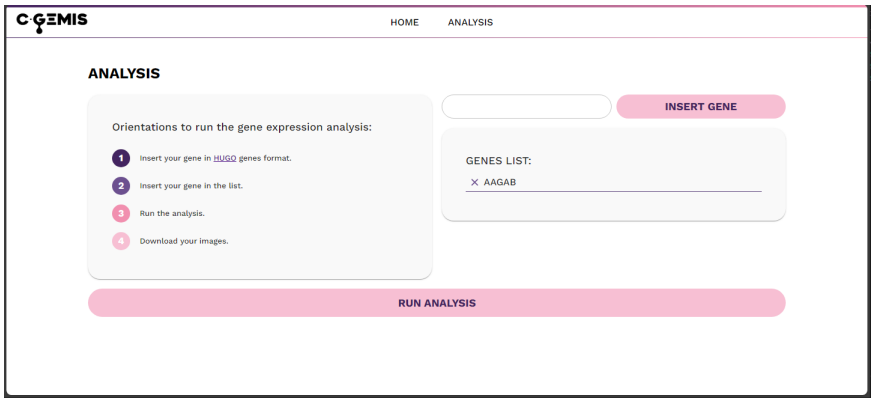
[5] https://www.genenames.org/
[6] Disponível em https://axios-http.com/docs/intro

Figura 19 – Tela inicial



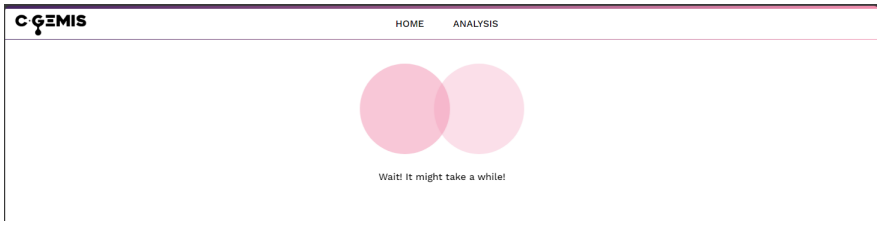
Fonte: Elaborado pelo autor (2023)

Figura 20 – Tela inserção de genes



Fonte: Elaborado pelo autor (2023)

Figura 21 – Tela de carregamento



Fonte: Elaborado pelo autor (2023)

apresentado anteriormente. A tela de resultados pode ser visto na Figura 23. Para cada gene é apresentado a possibilidade de realizar o *download* completo da análise, ou realizar o *download* de análises específicas.

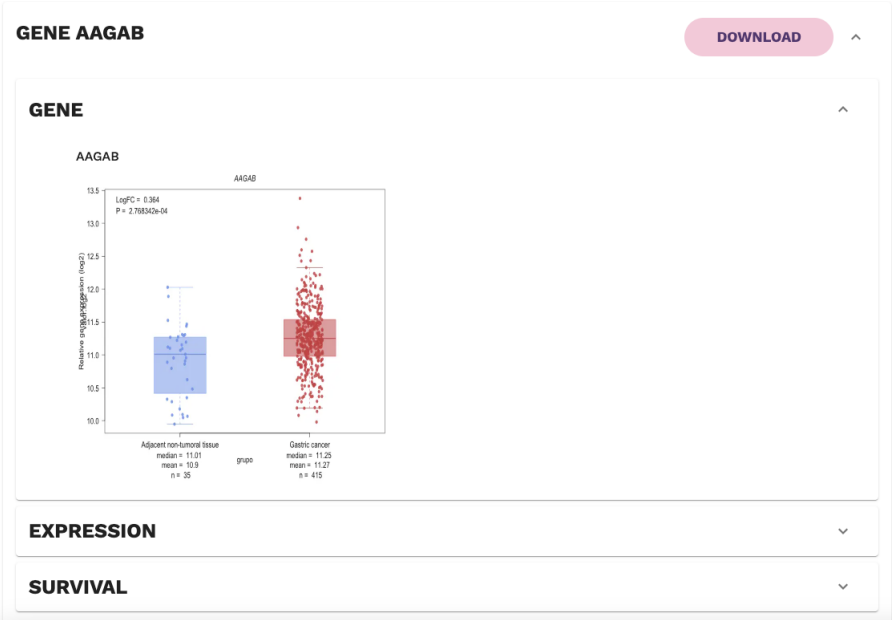
Dentre os requisitos do desenvolvimento da ferramente pode ser citado acompanhar o número de acessos ao site, dado importante que justifica o projeto. Para obter esta informação

Figura 22 – Requisição para a API

```
export async function runAPIGene(genes: String[]) {
    const response = await instance.post(`/run`,{"name":genes} ,getHeaders()).then((response) => {
          console.log(response)
          return response.data;
    }).catch(error =>{
          return "Não foi possível realizar a análise, " + error.message;
    })
    return response;
}
```

Fonte: Elaborado pelo autor (2023)

Figura 23 – Tela resultados



Fonte: Elaborado pelo autor (2023)

foi utilizado o *Google Analytics* [7]. A *Google* fornece um pequeno pedaço de código, que é implementado ao código fonte. Este código realizada a integração com a conta, fornecendo dados chaves sobre acesso e comportamento dos usuários.

4.3 HOSPEDAGEM

A disponibilização do C-Gemis a acesso geral na *internet* foi realizada utilizando a AWS [8]. A AWS oferece uma grande gama de serviços baseado na computação na nuvem. Neste caso foi utilizado o serviço Amazon Elastic Compute Cloud (EC2), que disponibiliza criar computadores virtuais, chamados de instâncias. A EC2 fornece um painel de gerenciamento simplificado e de fácil uso para configurar interfaces de rede, direcionamento de tráfego e outras configurações.

Foi criada uma instância para o C-Gemis, esta foi associada a um Internet Protocol (IP)

[7] Disponível em https://marketingplatform.google.com/about/analytics/
[8] Disponível em https://aws.amazon.com/pt/

fixo disponibilizado pela *Amazon* para realizar o acesso HTTP. Nesta instância foi configurado o banco de dados, as bibliotecas R e demais necessidades para que a aplicação funcionasse. Para gerenciar o processo do *Frontend* e da API foi utilizado o PM2 [9]. O PM2 permite a execução do *Node JS* em segundo plano, além de auto inicializar os processos em caso de interrupção.

O ambiente da AWS permite a integração com diversos outros serviços para futura escalabilidade do projeto. O serviço *Route 53* fornece o gerenciamento de domínio. É possível a criação de outras instâncias para aumentar o poder de processamento horizontalmente. O serviço *Elastic Load Balancer* realizaria gestão de tráfego e o redirecionamento para as diversas instância com intuito de obter melhor performance e menor tempo de resposta, evitando a sobrecarga.

[9] Disponível em https://pm2.keymetrics.io/

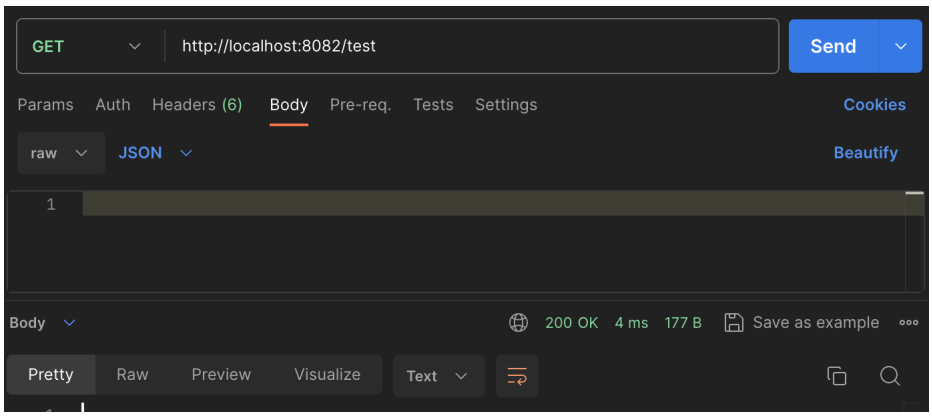
5 TESTES

Este capítulo tem como finalidade apresentar os resultados obtidos com os testes definidos na Seção 3.5. A ferramenta *Postman*[1] foi utilizada para auxiliar a realizar requisições HTTP.

5.1 COMUNICAÇÃO A API

A API foi instanciada em desenvolvimento local na porta 8082. Utilizando a ferramenta *Postman* foi possível chamar o *endpoint test* com sucesso, obtendo a resposta com *status* 200 indicando sucesso, conforme Figura 24.

Figura 24 – Resultado teste de comunicação



Fonte: Elaborado pelo autor (2023)

5.2 OBTENÇÃO DE RESULTADO DA APLICAÇÃO R

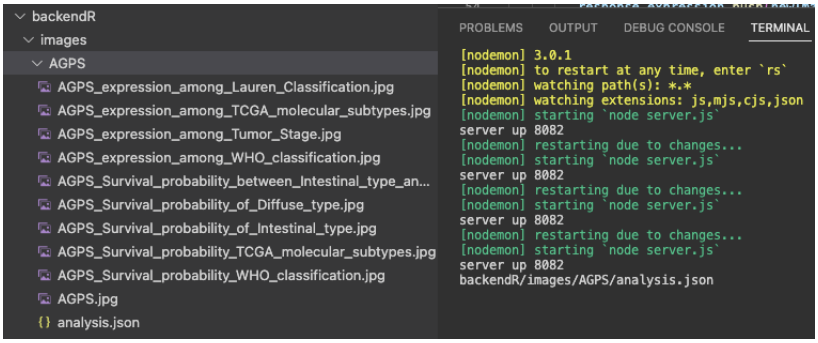
Conforme Figura 25, é possível observar o diretório salvo com as imagens e o arquivo contendo as informações sobre estes arquivos. No *console* de desenvolvimento a API retornou o caminho deste diretório informado pelo R. O que significa que a aplicação R conseguiu responder a requisição corretamente.

5.3 ENVIO DE GENE

Conforme Figura 26, é possível notar que a API reconheceu a lista enviada a ela via requisição HTTP enviada do *frontend*.

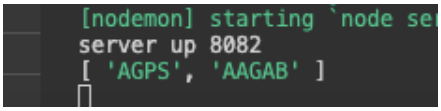
[1] Disponível em https://www.postman.com/

Figura 25 – Resultado segundo teste



Fonte: Elaborado pelo autor (2023)

Figura 26 – Resultado lista de genes



Fonte: Elaborado pelo autor (2023)

5.4 OBTENÇÃO DE RESULTADO

Neste teste foi realizada uma requisição no *frontend* buscando pelo gene AAGAB. O resultado com as fotos foi possível obter no navegador, conforme mostra a Figura 27 uma das imagens.

Figura 27 – Resultado imagem no frontend

Fonte: Elaborado pelo autor (2023)

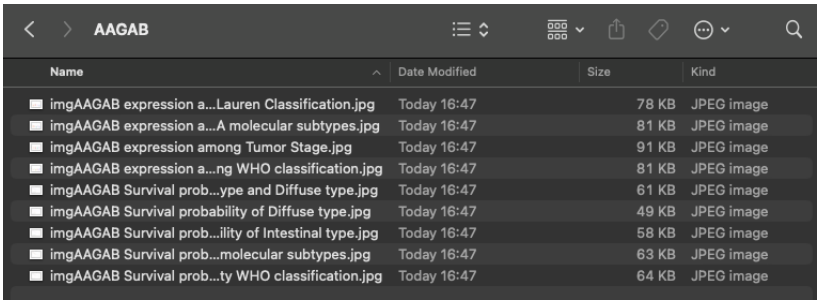
5.5 GERAÇÃO CORRETA DE RESULTADO

Foi realizado a análise do gene AAGAB no antigo C-Gemis e no atual. Os resultados foram comparados, constando serem os mesmos.

5.6 DISPONIBILIZAÇÃO DA APLICAÇÃO

A aplicação foi hospedada na AWS e disponibilizada a uso geral. Na Figura 28 é possível observar os arquivos baixados de uma consulta teste.

Figura 28 – Arquivos baixados



Fonte: Elaborado pelo autor (2023)

6 CONSIDERAÇÕES FINAIS

Dado os requisitos inicias apresentados ao projeto e estabelecidos na introdução do trabalho, pode se concluir que os objetivos foram atingidos com sucesso. Após pesquisas serem realizadas, a arquitetura planejada e desenvolvida o C-Gemis pode ser disponibilizado na *internet*. O requisito fundamental de *download* foi implementado com sucesso, permitindo assim o usuário ter em sua máquina o resultado que buscava. A API apresentou bom tempo de resposta e cumpriu seus requisitos.

Durante o desenvolvimento o ponto crítico ao projeto foi a comunicação entre a API e a aplicação em R. Poucas bibliotecas que gerenciem esta comunicação foram encontradas. Destas apresentavam problemas com chamadas assíncronas, com pouca documentação ou suporte, ou apresentavam problemas com a integração e a estrutura de dados dos objetos utilizados como parâmetros.

A disponibilização do C-Gemis foi possível utilizando a AWS. O serviço EC2 fornece um gerenciamento simplificado e de fácil manuseio na parte de interfaces de rede da instância. A AWS fornece um IP fixo para uso e acesso HTTP. Para acesso HTTPS entretanto será necessário a aquisição de um domínio e utilizar um certificado Secure Sockets Layer (SSL). Devido ao plano utilizado da AWS ser o gratuito, existem diversas limitações quanto a escalabilidade, exigindo a assinatura de um plano pago em caso de futura demanda.

REFERÊNCIAS

ANDREW, T.; DAVID, W.; FEAMSTER, N. **Computer networks sixth edition**. [S.l.]: pearson, 2021.

ARAUJO, F. A. *et al.* Go feat: a rapid web-based functional annotation tool for genomic and transcriptomic data. **Scientific reports**, Nature Publishing Group UK London, v. 8, n. 1, p. 1794, 2018.

BELQASMI, F. *et al.* Soap-based vs. restful web services: a case study for multimedia conferencing. **IEEE internet computing**, IEEE, v. 16, n. 4, p. 54–63, 2012.

BROOKS, J. D. Translational genomics: the challenge of developing cancer biomarkers. **Genome research**, Cold Spring Harbor Lab, v. 22, n. 2, p. 183–187, 2012.

CASCIARO, M.; MAMMINO, L. **Node. js Design Patterns: Design and implement production-grade Node. js applications using proven patterns and techniques**. [S.l.]: Packt Publishing Ltd, 2020.

CHAVES, J. T. F. Service-oriented architecture (soa), agile development methods and quality assurance (qa): a case study. 2019.

DAVIS, S.; MELTZER, P. S. Geoquery: a bridge between the gene expression omnibus (geo) and bioconductor. **Bioinformatics**, Oxford University Press, v. 23, n. 14, p. 1846–1847, 2007.

DONG, W. *et al.* Denoising aggregation of graph neural networks by using principal component analysis. **IEEE Transactions on Industrial Informatics**, IEEE, v. 19, n. 3, p. 2385–2394, 2022.

D'ANGELO, G.; RIENZO, T. D.; OJETTI, V. Microarray analysis in gastric cancer: a review. **World journal of gastroenterology: WJG**, Baishideng Publishing Group Inc, v. 20, n. 34, p. 11972, 2014.

EHSAN, A. *et al.* Restful api testing methodologies: Rationale, challenges, and solution directions. **Applied Sciences**, MDPI, v. 12, n. 9, p. 4369, 2022.

FOROUZAN, B. A.; MOSHARRAF, F. **Redes de computadores: uma abordagem top-down**. [S.l.]: AMGH Editora, 2013.

GAMMA, E. *et al.* **Padrões de Projeto–Soluções Reutilizáveis de Software Orientado a Objetos**. [S.l.]: Bookman–Porto Alegre, 2000.

GARCIA, M. *et al.* Sarek: A portable workflow for whole-genome sequencing analysis of germline and somatic variants. **F1000Research**, Faculty of 1000 Ltd, v. 9, 2020.

HRDLICKOVA, R.; TOLOUE, M.; TIAN, B. Rna-seq methods for transcriptome analysis. **Wiley Interdisciplinary Reviews: RNA**, Wiley Online Library, v. 8, n. 1, p. e1364, 2017.

HUMANN, J. L. *et al.* Structural and functional annotation of eukaryotic genomes with gensas. **Gene prediction: methods and protocols**, Springer, p. 29–51, 2019.

KHAN, F. Z. *et al.* Sharing interoperable workflow provenance: A review of best practices and their practical application in cwlprov. **GigaScience**, Oxford University Press, v. 8, n. 11, p. giz095, 2019.

KUROSE, J. F.; ROSS, K. W.; ZUCCHI, W. L. **Redes de Computadores ea Internet: uma abordagem top-down**. [S.l.]: Pearson Addison Wesley, 2005.

LAMOTHE, M.; GUÉHÉNEUC, Y.-G.; SHANG, W. A systematic review of api evolution literature. **ACM Computing Surveys (CSUR)**, ACM New York, NY, v. 54, n. 8, p. 1–36, 2021.

LI, L. *et al.* Design patterns and extensibility of rest api for networking applications. **IEEE Transactions on Network and Service Management**, IEEE, v. 13, n. 1, p. 154–167, 2016.

LORENZI, N. P. C. **Autocoleta cervicovaginal no rastreamento do câncer do colo do útero: aceitabilidade, detecção de Papilomavírus Humano de alto risco oncogênico e pesquisa de biomarcadores**. Tese (Doutorado) — Universidade de São Paulo, 2020.

MARINO, A. G. G. **Arquitetura de Software**. [S.l.]: Editora Pearson, 2016.

MCKNIGHT, D. T. *et al.* Methods for normalizing microbiome data: an ecological perspective. **Methods in Ecology and Evolution**, Wiley Online Library, v. 10, n. 3, p. 389–400, 2019.

MEDRADO, L. **Carcinogenese: Desenvolvimento, Diagnostico e Tratamento das Neoplasias**. [S.l.]: Sao Paulo, Erica, 2015.

OPPERMANN, C. P. **Entendendo o câncer**. [S.l.]: Artmed Editora, 2014.

PRAYOGI, A. *et al.* Design and implementation of rest api for academic information system. In: IOP PUBLISHING. **IOP Conference Series: Materials Science and Engineering**. [S.l.], 2020. v. 875, n. 1, p. 012047.

ROSSETTO, M. V. Análise de expressão diferencial de genes: uma solução computacional para identificação de biomarcadores de tumores gástricos em humanos. 2021.

SANTOS, M. de O. *et al.* Estimativa de incidência de câncer no brasil, 2023-2025. **Revista Brasileira de Cancerologia**, v. 69, n. 1, 2023.

SCHATZ, M. C. *et al.* Inverting the model of genomics data sharing with the nhgri genomic data science analysis, visualization, and informatics lab-space. **Cell Genomics**, Elsevier, v. 2, n. 1, 2022.

SORENSEN, F. L. **Enterprise Architecture and Service-oriented Architecture**. [S.l.]: Nova Science Publishers, 2020.

TEAM, R. C. R language definition. **Vienna, Austria: R foundation for statistical computing**, v. 3, n. 1, 2000.

WANG, Z.; GERSTEIN, M.; SNYDER, M. Rna-seq: a revolutionary tool for transcriptomics. **Nature reviews. Genetics**, NIH Public Access, v. 10, n. 1, p. 57, 2009.

WIERUCH, R. **The road to react: Your journey to master plain yet pragmatic react. js**. [S.l.]: Robin Wieruch, 2020.

XUE, K. *et al.* Application of ct texture analysis in predicting preoperative lauren classification of gastric cancer. In: IEEE. **2019 IEEE International Conference on Mechatronics and Automation (ICMA)**. [S.l.], 2019. p. 2185–2189.

YAMANAPPA, W. *et al.* Non-local means image denoising using shapiro-wilk similarity measure. **IEEE Access**, IEEE, v. 6, p. 66914–66922, 2018.

YASUI, W. *et al.* Search for new biomarkers of gastric cancer through serial analysis of gene expression and its clinical implications. **Cancer science**, Wiley Online Library, v. 95, n. 5, p. 385–392, 2004.

ZOU, Y.; TANG, P.; LV, T. Design and implementation of ship shore power data analysis system based on doris data warehouse. In: IEEE. **2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)**. [S.l.], 2022. p. 367–370.