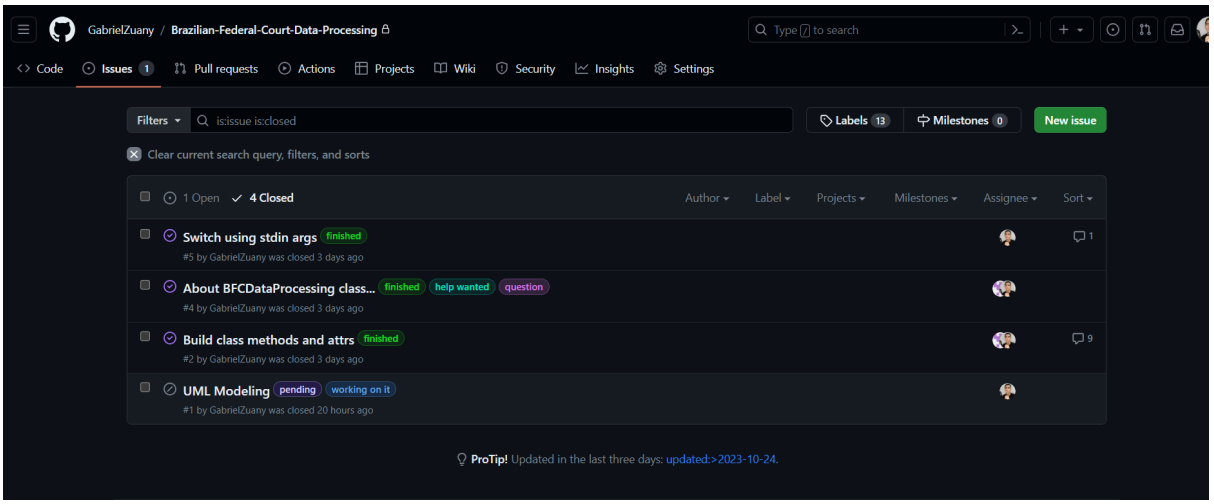
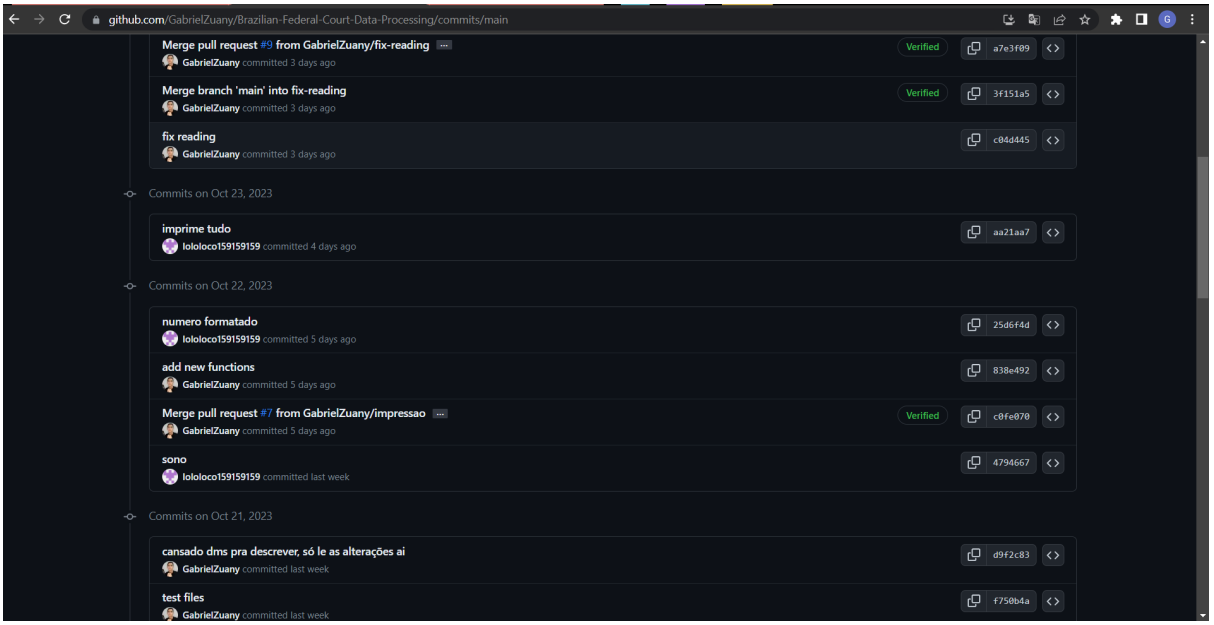
Relatório Trabalho 1 POO

Introdução:

O código foi feito em conjunto usando a extensão Live Share disponível no VScode e ferramentas de versionamento de código (GIT). Eventualmente, pequenas implementações eram feitas isoladamente, de acordo com as notas deixadas na aba de *Issues* do GitHub pelos desenvolvedores, mas claramente explicadas e comentadas ao realizar os commits. A maior parte da divisão de tarefas se deu quando existiram erros de leitura e necessidade da implementação das impressões para debug, ficando um dos integrantes para corrigir a leitura e outro para imprimir as saídas. Ao final, nos juntamos para revisar o projeto, realizar otimizações e buscar bugs e entender todo o trabalho realizado.





Testes:

Para a implementação inicial, nos baseamos nos arquivos fornecidos nos links presentes no documento de especificação (pdf). Foram realizados testes separados para cada método correspondente a uma funcionalidade solicitada e em seguida todos juntos, para corrigir e averiguar possíveis discrepâncias entre a nossa saída e a saída esperada.
Para otimizar o processo de teste utilizamos alguns comandos simples em shell para fazer a build completa do código e execução:

```
$ ant clean
& ant compile
& ant jar
& java -cp bin App --estadual trabalho1-poo-script/testes/AL/in/candidatos.csv trabalho1-poo-script/testes/AL/in/votaca.csv 02/10/202
& diff compare/meu.txt trabalho1-poo-script/testes/AL/out/output-estadual.txt
```

No final, usamos os testes executados pelo script para garantir que, nos demais casos, o processamento dos dados estava correto (não houve erro em nenhuma saída desde a primeira execução do script).

```
zuany@gzuany:/mnt/c/Users/gabri/OneDrive/Área de Trabalho/Developer/WindowsWorkEnvironment/testePOO/pastaTeste$ ./test.sh
Script de teste PROG OO - Trabalho 1

[I] Testando deputados-Gabriel...
[I] Testando deputados-Gabriel: +- teste AC
[I] Testando deputados-Gabriel: |  teste AC, tudo OK (estadual)
[I] Testando deputados-Gabriel: |  teste AC, tudo OK (federal)
[I] Testando deputados-Gabriel: +- teste AL
[I] Testando deputados-Gabriel: |  teste AL, tudo OK (estadual)
[I] Testando deputados-Gabriel: |  teste AL, tudo OK (federal)
[I] Testando deputados-Gabriel: +- teste MG
[I] Testando deputados-Gabriel: |  teste MG, tudo OK (estadual)
[I] Testando deputados-Gabriel: |  teste MG, tudo OK (federal)
[I] Testando deputados-Gabriel: +- teste PE
[I] Testando deputados-Gabriel: |  teste PE, tudo OK (estadual)
[I] Testando deputados-Gabriel: |  teste PE, tudo OK (federal)
[I] Testando deputados-Gabriel: +- teste RS
[I] Testando deputados-Gabriel: |  teste RS, tudo OK (estadual)
[I] Testando deputados-Gabriel: |  teste RS, tudo OK (federal)
[I] Testando deputados-Gabriel: +- pronto!


zuany@gzuany:/mnt/c/Users/gabri/OneDrive/Área de Trabalho/Developer/WindowsWorkEnvironment/testePOO/pastaTeste$
```

Implementação:

O código foi separado em classes e subclasses para facilitar sua implementação e entendimento. Como principal classe, criamos a `BFCDataProcessing` responsável por armazenar os candidatos (`HashMap<String,Candidate>`) e os partidos (`HashMap<String,ElectoralParty>`), além de armazenar a quantidade total de candidatos eleitos (usada para evitar processamento desnecessário nas funções de impressão) e a data da eleição.
O `Candidate` armazena suas informações relevantes da eleição e seu `ElectoralParty`, enums foram usados nas informações para facilitar a leitura e entendimento do código.

```java
package bfcdp.enums;

public enum EnumCandidateType {
    FEDERAL,
    STATE
}
```

```java
package bfcdp.enums;

public enum EnumGender {
    MALE,
    FEMALE
}
```

```java
package bfcdp.enums;

public enum EnumVoteType{
    LEGENDA,
    NOMINAL
}
```

```java
package bfcdp.enums;

public enum EnumApplication {
    APPROVED,
    REJECTED
}
```

```java
package bfcdp.enums;

public enum EnumResult {
    WIN,
    LOSE
}
```

Usamos a classe abstrata candidatos para criar subclasses (`FederalCandidate` e `StateCandidate`) que herdam a implementação dos métodos da superclasse para evitar replicação de código.

O `ElectoralParty` armazena suas informações sobre o partido e uma `List<String>` com o Id dos candidatos (Usado como key no `HashMap<String,Candidate>` armazenado na estrutura `BFCDataProcessing` para ter acesso aos seus candidatos sem precisar armazenar sua estrutura inteira).

Diagrama UML (Apenas os atributos de classes)

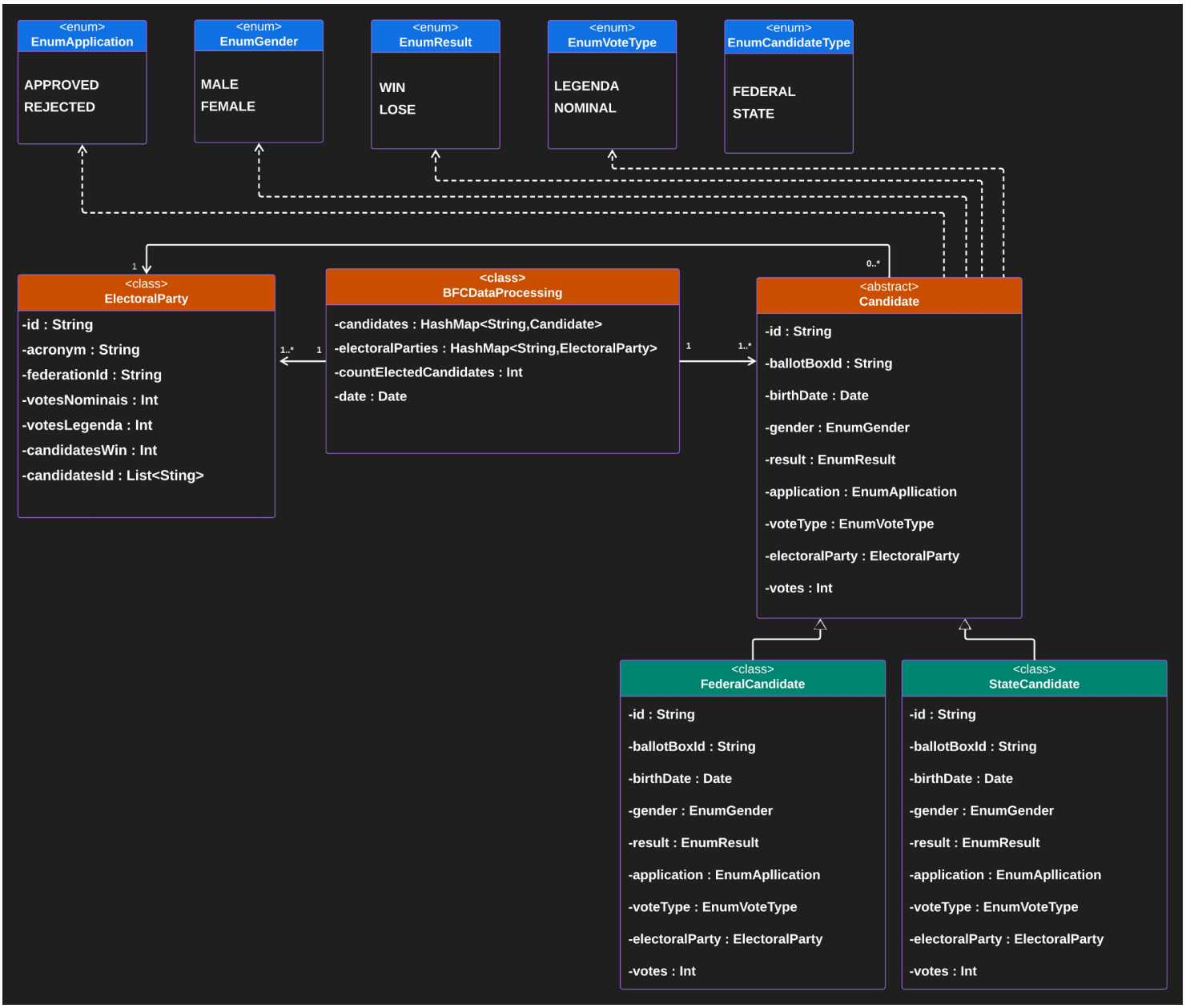
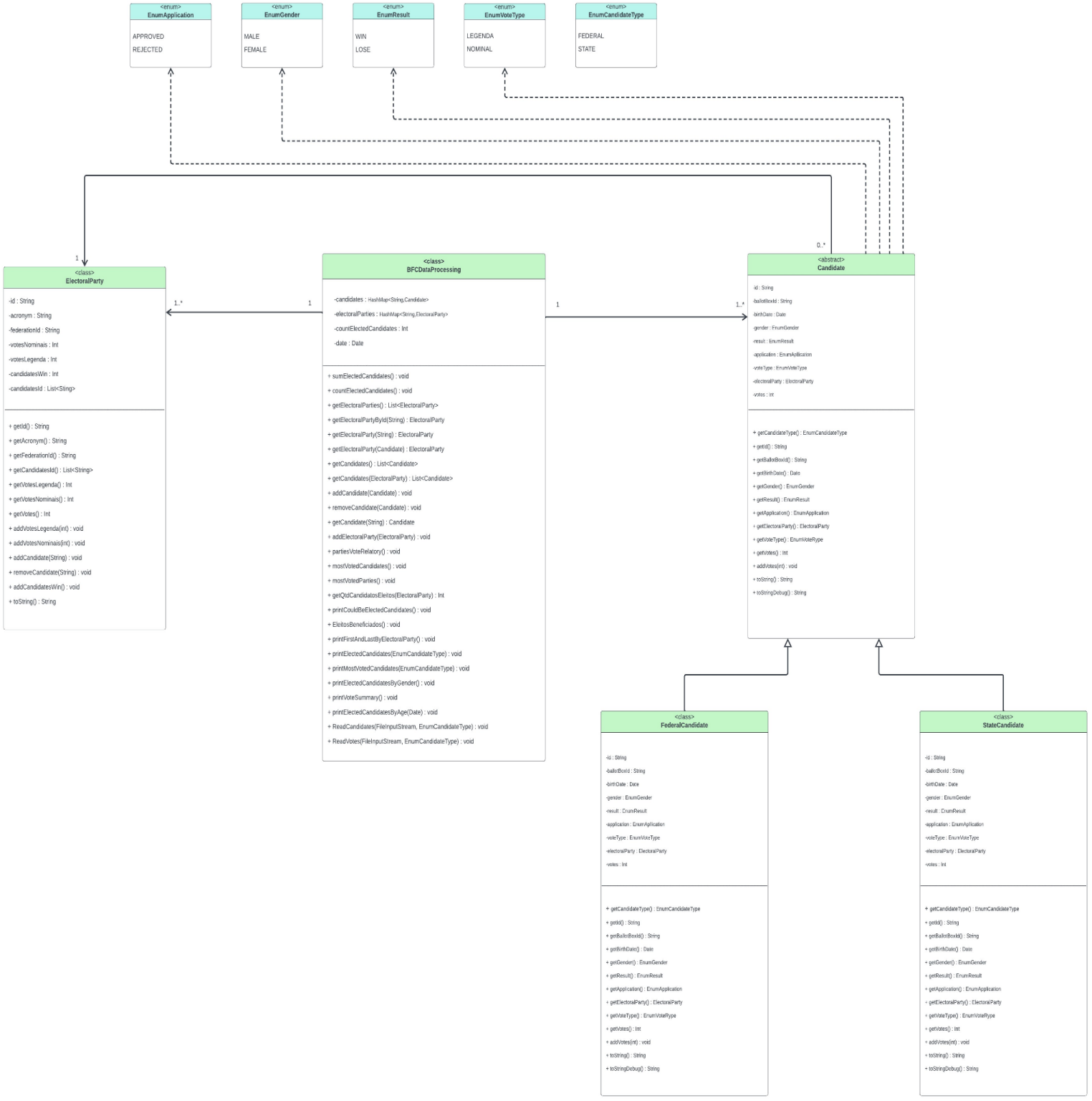
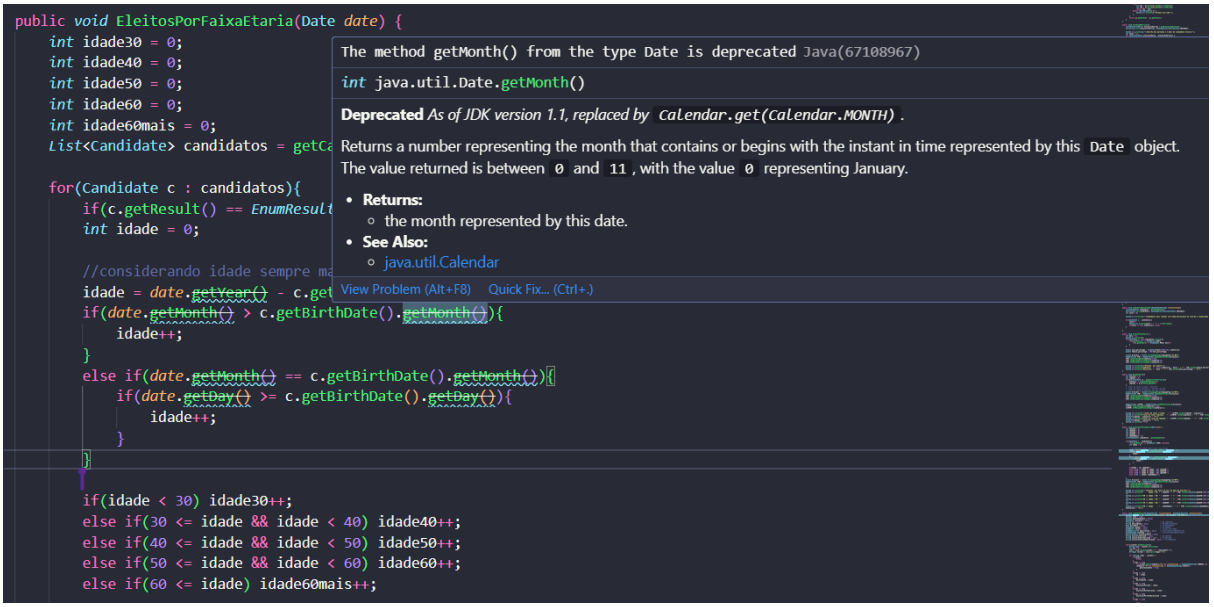


Diagrama UML Completo:



Bugs:

Para realizar a conta da idade dos candidatos, foi usado uma função antiga do Java, sua implementação está marcada como incompleta pelos desenvolvedores e pode acarretar em erros em casos específicos, não ocorreu erro durante a execução, mas é importante ressaltar que existe essa possibilidade.
Tivemos também grandes problemas na leitura dos candidatos se tratando de casos específicos, como candidatos com mesmo id ou zero votos, corrigimos esse erro com a implementação de uma flag que apontava a necessidade ou não de armazenar o candidato após processarmos todas suas informações e armazená-las quando necessário. Entretanto, sua implementação não foi arquitetada previamente e mesmo processando “melhor” que anteriormente, apresentava alguns erros. Dessa forma, apesar da correção dos erros mediante a adição de condicionais, é um grande potencial para futuros bugs, já que sua adição não era prevista.



Desenvolvido por:
- Gabriel Zuany Duarte Vargas - 2022100865
- Lorenzo Rizzi Fiorot - 2022100892