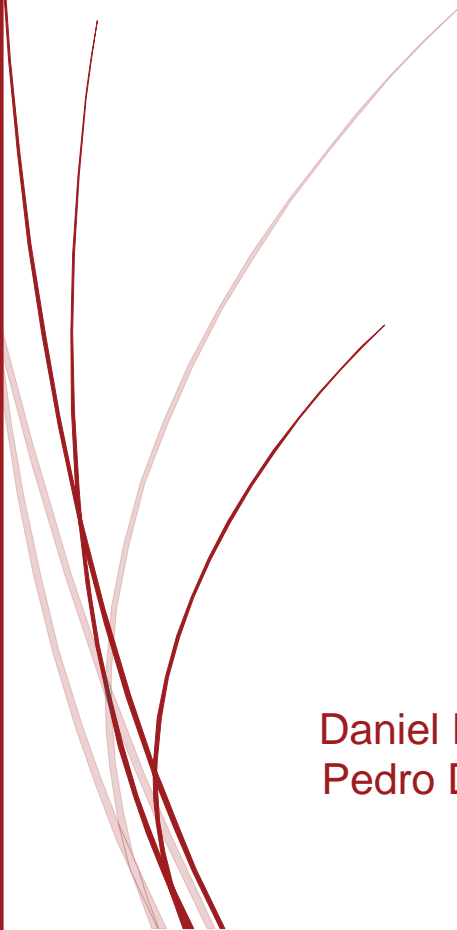


19-06-2022

# Relatório Trabalho Prático

Programação Avançada



Daniel Duarte Dias Ferreira Albino - 2020134077  
Pedro David Avelar Ramos - 2019130867

# Índice

Introdução .....	2
Decisões/Opções .....	3
Diagrama da Máquina de Estados .....	4
Descrição sucinta das classes .....	6
Descrição das Classes .....	8
Funcionalidades Implementadas .....	12
Aspetos a melhorar .....	15
Conclusão .....	16

## Introdução

Nesta cadeira foi-nos proposto, para trabalho prático, a implementação de um programa, em java, uma aplicação de apoio ao processo de gestão de projetos e estágios do Departamento de Engenharia Informática e de Sistemas do ISEC. A aplicação será alimentada com diversos tipos de dados consoante a fase em que se encontra o processo de gestão.

## Decisões/Opções

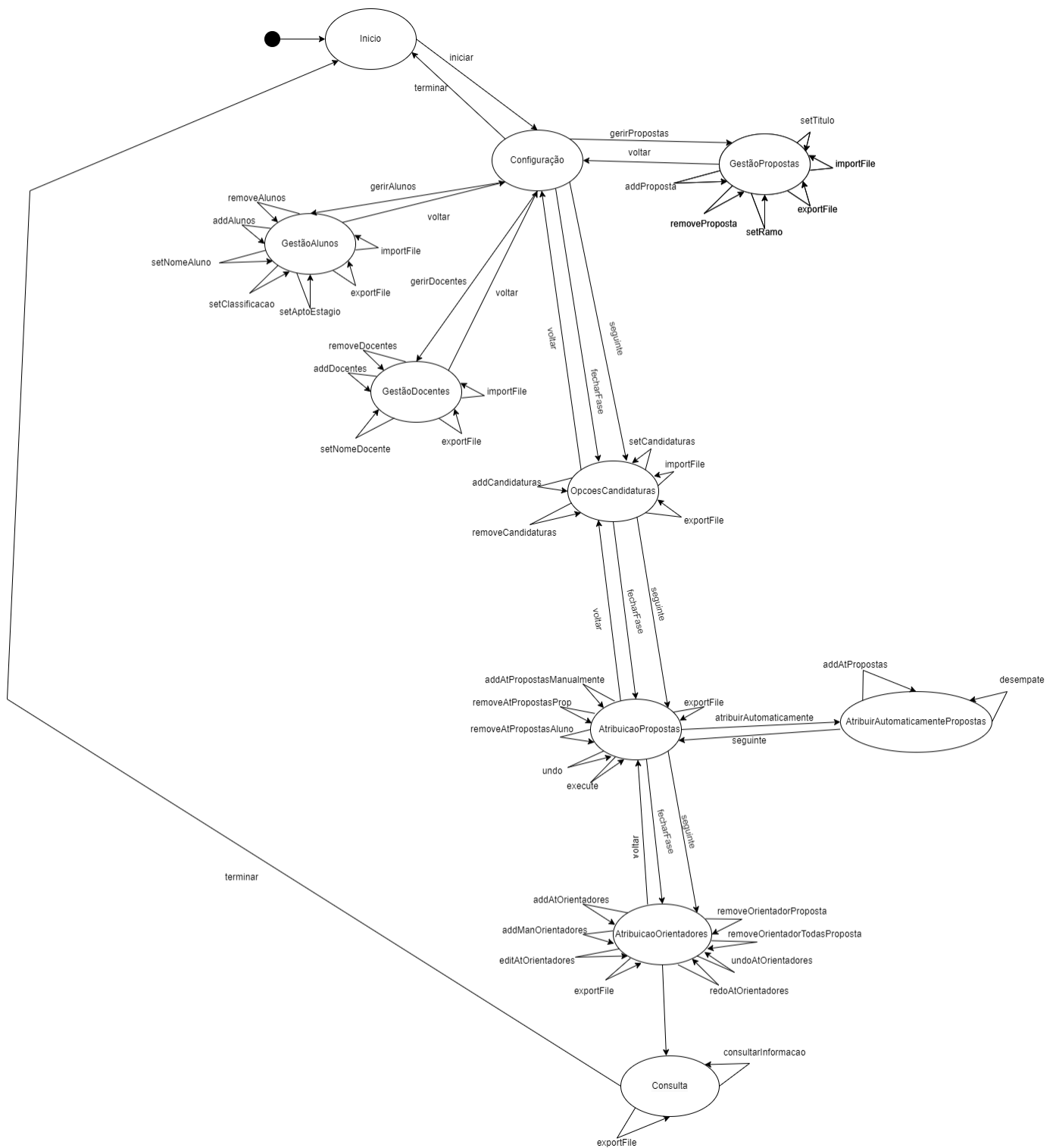
Neste projeto, além do proposto e das normas que um programador tem de seguir para que o código esteja organizado, não tivemos opções muito relevantes. Mais especificamente decidimos utilizar uma classe “data” que continha todas as funções e o código relativo á gestão das Candidaturas, Docentes, Alunos, Propostas, Atribuição de Propostas e Atribuição de Orientadores.

Isto devido principalmente ao facto de não querermos sobrecarregar o projeto com classes “a mais”. Para além disso, tendo tudo numa só classe, temos acesso a outras funções que já tinham sido implementadas, o que facilita bastante fazer certas verificações, como ter acesso ao “Map” das propostas atribuídas.

Com isto, percorrer o mesmo para agregar-lhes orientadores, assim “passamos” á frente uma verificação, não havendo código repetitivo (que já é feita, aquando da agregação dos alunos às propostas). Isto porque logicamente esse map apenas terá guardado dados de propostas ainda não atribuídas o que facilita em futuras verificações.

Além destas decisões, optamos também por usar “Maps” para guardar todos os dados, dado que “Maps” em relação aos arrays já tem funções pré-definidas para percorrer, remover, entre outros e não temos de nos preocupar com o tamanho máximo, visto que ele é dinâmico originalmente.

## Diagrama da Máquina de Estados



*Figura 1 - Diagrama de Estados*

Como todas as alterações de dados devem ser feitas no contexto de uma transição de dados, inserimos os estados “**Gestão de Alunos**” que contém tudo relativo aos alunos, o que acontece também para os outros estados (adicionar, editar, remover, consultar), “**Gestão de Docentes**” e “**Gestão de Propostas**”. Estes estados estão todos relacionados com o estado “**Configuração**” uma vez que sem os dados referidos anteriormente a aplicação não funciona consoante o objetivo desenvolvido, e assim, só se pode avançar após esses dados terem sido corretamente inseridos.

Logo a seguir ao estado “**Configuração**” temos os estados de atribuição de “**Opções Candidaturas**”, “**Atribuição de Propostas**” e “**Atribuição de Orientadores**”. Os mesmos irão utilizar os dados relativos aos estados anteriores principais e irão ser a base deste programa.

O estado “**Atribuição Automática de Propostas**” é responsável por atribuir automaticamente as propostas aos alunos. Para além disso, é também responsável por detetar empates e resolvê-los ou caso isso seja impossível avisar o utilizador para que o próprio o resolva.

Por fim temos o estado “**Consulta**”, que, como o nome indica, é o estado em que o utilizador pode filtrar tudo o que sucedeu nos estados passados, e “escolher” da forma que deseja visualizar a informação.

## Descrição sucinta das classes

Todo o nosso projeto está dividido por packages, sendo que cada um tem o seu objetivo e apenas contém dados que compreendem a esse objetivo.

O package “**model**” contém todas as classes referentes ao modelo de dados e da máquina de estados do programa. Assim, este é dividido pelo package “**data**” e “**fsm**” e contém duas classes “**DataManager**” e “**Manager**”.

O package “**data**” contém todas classes que manipulam os dados, mais concretamente, neste package podemos encontrar:

- A classe “**Alunos**”, “**Docentes**”, “**Propostas**”, “**Autopropostas**”, “**Estágios**”, “**Projetos**” e “**Data**”;
- O enum “**Erros**”;
- Os packages responsáveis pela funcionalidade “**undo**” e “**redo**”:
  - command:
    - A classe “**AddAtPropostasCommad**”, “**CommandAdapter**”, “**RemoveAtPropostasAlunoCommand**” e “**RemoveAtPropostasPropCommand**”;
    - A interface “ **ICommand**”;
  - memento:
    - A classe “**CareTaker**” e “**Memento**”;
    - A interface “**IMemento**” e “**IOrieginator**”.

O package “**fsm**” que contém todas as classes referentes à máquina de estados, ou seja, neste package podemos encontrar:

- A classe “**AtribuicaoOrientadores**”, “**AtribuicaoAutomaticaPropostas**”, “**AtribuicaoPropostas**”, “**Configuração**”, “**Consulta**”, “**GestaoAlunos**”, “**GestaoDocentes**”, “**GestaoPropostas**”, “**GPEContext**”, “**Inicio**” e “**OpcoesCandidatura**”;
- O enum “**GPEState**”;
- A interface “**IGPEState**”.

No package “**UI**” podemos encontrar dois packages “**text**” referente à interface de texto e o “**gui**” referente à interface em JavaFX.

O package “**text**” contém apenas uma classe “**UI**” que contém toda a interface de texto.

O package “**utils**” contém duas classes “**PAInput**” e “**TratamentoErros**” usadas pela interface de texto.

O package “**gui**” contém as classes “**MainJFX**” e “**RootPane**” e está dividida em vários packages:

- O package “**resources**” que contém todos os recursos utilizados pelo programa, mais especificamente, as fontes das letras (package “**fonts**”) e as imagens (package “**images**”) e as classes “**FontManager**” e “**ImageManager**” disponibilizadas pelos docentes;
- O package “**states**” contém um package “**utils**” com uma classe que gera um popup e todas as “janelas” de cada estado. Cada estado tem o seu próprio package, mais concretamente, o package “**alunos**” que contém todas as janelas usadas pelo estado “**GestaoAlunos**”.

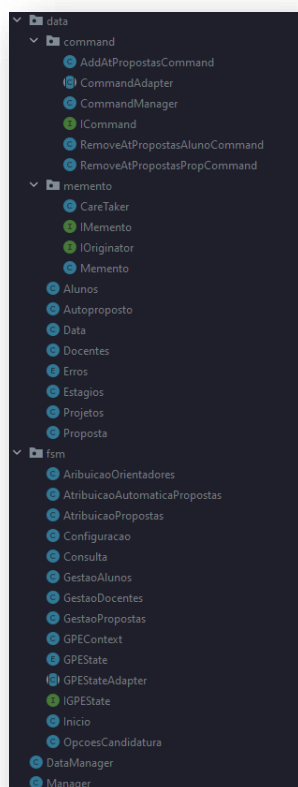


Figura 2 - Package “data”

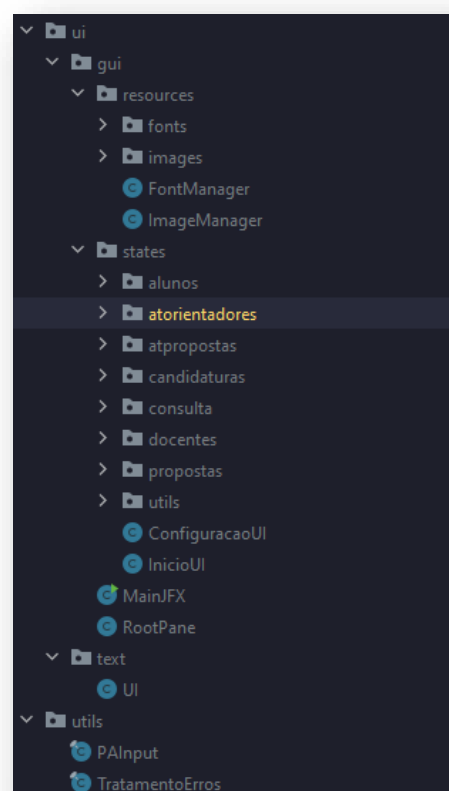


Figura 3 - Package “ui”



## Descrição das Classes

**Classe Alunos:** esta classe representa o objeto aluno integrando a informação relativa a esse aluno, como o seu número, nome, email, curso, ramo, classificação, se está apto a ir a estágio e os vários métodos get, set e toString.

**Classe Docentes:** esta classe representa o objeto docente onde tem a informação relativa a esse docente, como o seu email, nome e os vários métodos get, set e toString.

**Classe Proposta:** esta classe representa o objeto proposta que integra a informação relativa a essa proposta, como o código, tipo, ramo, título e os vários métodos get, set e toString. Esta é a classe principal de onde derivam as seguintes classes:

- **Estágios:** que guarda a informação de um estágio e tem como parâmetros o Id da empresa, o Id do aluno e os vários métodos get, set e toString;
- **Projetos:** que guarda a informação de um projeto e tem como parâmetros o email do docente, o Id do aluno e os vários métodos get, set e toString;
- **Autoproposto:** que guarda a informação de uma autoproposta e tem como parâmetros o email do docente, o Id do aluno e os vários métodos get, set e toString.

**Classe Data:** esta classe é responsável pela gestão de todos os dados, ou seja, nesta classe é possível guardar vários alunos, docentes, propostas (dos 3 tipos mencionados em cima), candidaturas e atribuir propostas e orientadores. Para além disso, é nesta classe que percebemos se um estado está fechado ou não. Toda esta informação é guardada por “Maps”.

**Classe Data Manager:** esta classe é o “intermediário” entre a máquina de estados e os dados. A mesma tem uma referência para a class Data. Nesta classe temos todos os métodos que temos na classe Data.

**Classes da máquina de estados:** a máquina de estados está dividida por várias classes, são elas “AtribuiçãoOrientadores”, “AtribuiçãoPropostas”, “Configuração”, “GestãoAlunos”, “GestãoDocentes”, “GestãoPropostas”, “AtribuiçãoAutomaticaPropostas”, “Inicio” e “OpçõesCandidatura”. Nas mesmas

podemos encontrar todas as funções que fazem sentido nestes estados de acordo com o diagrama de estados apresentado anteriormente.

A classe “**GPEStateAdapter**” contém todos os métodos presentes nas classes da máquina de estados com um comportamento default.

A classe “**GPEContext**” tem uma referência para os dados e outra para o estado e é nela que toda a máquina de estados começa. Temos também lá presente as funções de save e load.

**Interface: IGPEState** é a interface que engloba todos os métodos da máquina de estados, neste caso só a declaração dos mesmos. **Classe UI:** Nesta classe podemos encontrar toda a User Interface.

**Classe PAInput:** classe disponibilizada pelos docentes com algumas alterações importantes para o nosso trabalho.

**Classe TratamentoErros:** classe responsável por imprimir no ecrã todas as frases quando ocorre algum erro. A mesma só faz sentido na interface de texto.

**Classe MainText:** é nesta classe onde a interface de texto começa. Na mesma temos uma referência para a classe GPEContext.

**Classe MainJFX:** é nesta classe onde a interface gráfica começa, tem uma referência para o UI, esta classe encontra-se junto à classe MainText e nada tem que ver com a que esta no package “gui”.

**Enum:** neste projeto temos dois Enum, o **GPEState** que guarda os nomes de todos os estados e tem um método que é uma fábrica de estados “createState” e o **Erros** que neste momento está inacabado e que serve para dar alguns erros específicos ao utilizador para que este se aperceba do que fez errado.

#### **Classes da interface gráfica:**

- **Classes dos recursos:** as classes “**FontManager**” e “**ImageManager**” são duas classes disponibilizadas pelos docentes para que seja possível, de uma forma mais fácil, carregar as fontes de letras e imagens para a interface;

- **Classes dos estados:**

- As classes “**AddAlunosForm**”, “**EditAlunosForm**”, “**GestaoAlunosUI**”, “**RemoveAlunosForm**” e “**TableViewAlunos**” são responsáveis por criar as janelas do estado “**GestãoAlunos**”. As classes “**AddAlunosForm**”, “**EditAlunosForm**” e “**RemoveAlunosForm**” são responsáveis por criar um formulário para adicionar, editar e remover alunos, respetivamente. A classe “**GestaoAlunosUI**” é responsável por ter os botões para abrir as janelas de adição, remoção, e edição e os botões de importação e exportação dos dados. A classe “**TableViewAlunos**” é responsável por criar uma table view para a apresentação dos dados dos alunos;
- As classes “**AddAtOrientadoresForm**”, “**EditAtOrientadoresForm**”, “**RemoveAtOrientadoresForm**” são responsáveis por criar formulários para a adição, edição e remoção dos orientadores, respetivamente. E a classe “**AtOrientadoresUI**” é responsável por ter os botões para abrir as janelas de adição, remoção, e edição e o botão de exportação dos dados.
- As classes “**AddAtPropostasForm**”, “**RemoveAtPropostasForm**” são responsáveis por criar formulários para a adição e remoção da atribuição das propostas, respetivamente. A classe “**AtPropostasUI**” é responsável por ter os botões para abrir as janelas de adição e remoção, o botão de exportação dos dados e o botão de atribuir automaticamente, sendo este um estado em que houve a necessidade de criar uma nova janela, que se encontra na classe “**AddAutoAtPropostas**” localizada no mesmo package das classes deste ponto.
- As classes “**AddCandidaturasForm**”, “**EditCandidaturasForm**”, “**RemoveCandidaturasForm**” são responsáveis por criar formulários para a adição, edição e remoção da gestão das candidaturas, respetivamente. A classe “**GestaoCandidaturasUI**” responsável por ter os botões para abrir as janelas de adição, remoção, e edição e os botões de exportação e importação dos dados. E a classe “**Miseno**” responsável por criar uma janela com todas as opções para visualizar os dados.

- Na classe “**ConsultaUI**” é possível visualizar todas as informações relativas às propostas atribuídas e não atribuídas, número de orientações por docente, entre outras;
- As classes “**AddDocentesForm**”, “**EditDocentesForm**”, “**RemoveDocentesForm**” são responsáveis por criar formulários para a adição, edição e remoção dos docentes, respetivamente. A classe “**GestaoDocentesUI**” responsável por ter os botões para abrir as janelas de adição, remoção, e edição e os botões de exportação e importação dos dados. E a classe “**TableViewDocentes**” responsável por criar uma table view para a apresentação dos dados dos docentes;
- As classes “**AddPropostasForm**”, “**EditPropostasForm**”, “**RemovePropostasForm**” são responsáveis por criar formulários para a adição, edição e remoção das propostas, respetivamente. A classe “**GestaoPropostasUI**” responsável por ter os botões para abrir as janelas de adição, remoção, e edição e os botões de exportação e importação dos dados. E a classe “**TableViewPropostas**” responsável por criar uma table view para a apresentação dos dados das propostas;
- As classes “**ConfiguracaoUI**” e “**InicioUI**” responsáveis por criar as janelas do estado “**configuracao**” e “**inicio**”.
- **Classe MainJX:** é nesta classe onde toda a interface do programa começa.
- **Classe RootPane:** apresenta uma referência para os dados e é esta que inicializa todas as janelas de cada estado.

## Funcionalidades Implementadas

Funcionalidades	Cumprido	Implementado parcialmente	Não cumprido	Observações
Adicionar aluno (manualmente)	x			
Remover aluno	x			
Importação de alunos (CSV)	x			
Exportação de alunos (CSV)	x			
Obter dados de um aluno ou de todos	x			
Adicionar docente (manualmente)	x			
Remover docente	x			
Importação de docentes (CSV)	x			
Exportação de docentes (CSV)	x			
Obter dados de um docente ou de todos	x			
Adicionar propostas (manualmente)	x			
Remover propostas	x			
Importação de propostas (CSV)	x			
Exportação de propostas (CSV)	x			
Obter dados de uma proposta ou de todas	x			
Adicionar Candidaturas (manualmente)	x			
Remover Candidatura	x			
Importar candidaturas (CSV)	x			
Exportar candidaturas (CSV)	x			

Obter informação das candidaturas (Alunos com autoproposta, com candidatura já registada, sem candidatura registada, Autopropostas de alunos, Propostas de docentes, Propostas de candidaturas, Propostas sem candidaturas e todas as candidaturas)	x			
Atribuir Propostas (automaticamente com desempate)	x			
Adicionar Propostas (manualmente)	x			
Remover Propostas	x			
Exportação de propostas (CSV)	x			
Apresentar as propostas atribuídas	x			
Atribuir Orientadores (automaticamente)	x			
Atribuir Orientadores (manualmente)	x			
Remover Orientadores	x			
Editar Orientadores	x			
Exportação (CSV)	x			
Lista de estudantes com propostas atribuídas	x			
Conjunto de propostas disponíveis	x			

Lista de estudantes sem propostas atribuídas e com opções de candidatura	x			
Conjunto de propostas atribuídas	x			
Número de orientações por docente, em média, mínimo, máximo, e por docente especificado	x			
Exportação das informações anteriores (CSV)	x			
Fecho das fases	x			
Gravação do estado	x			
Carregamento	x			
Undo e Redo (Atribuição de propostas e de orientadores)	x			
Consulta da distribuição dos estágios por ramos	x			
Consulta da percentagem das propostas atribuídas e não atribuídas		x		Nesta parte não conseguimos fazer com que o gráfico apresentasse as percentagens.
Consulta das empresas com mais estágios (top 5)	x			
Consulta dos docentes com mais orientações (top 5).	x			
Código comentado usando JavaDoc			x	Não tivemos tempo para implementar.

## Aspetos a melhorar

Neste tópico iremos apresentar alguns aspetos que poderíamos melhorar no nosso projeto.

- O gráfico que apresenta a distribuição dos estágios por ramos não aparece com cores;
- Os gráficos apresentarem as percentagens;
- A classe “Data” está demasiado extensa, o que torna a visibilidade do código bastante difícil.



## Conclusão

Com esta meta aprendemos a criar uma aplicação bem constituída em Java, usando muitas das funcionalidades e funções que o Java oferece para Orientar Objetos e a criar interfaces usando o JavaFX. Foi algo desafiador, pois fez-nos pensar além dos pequenos projetos que desenvolvemos, o que nos obrigou a organizar, estruturar e avaliar o código, para que fosse atingida a meta que o professor desejava.