UCalculator

Relatório do Trabalho Prático 1

Diogo Silva (a76407)
João Barreira (a73831)
Rafael Costa (a61799)

Dezembro 2018

Universidade do Minho
Escola de Engenharia

Processamento de Dados com Streams de JAVA
Mestrado Integrado em Engenharia Informática – 4.º Ano
Universidade do Minho

1 Introdução

O objetivo deste trabalho consiste em explorar as funcionalidades que são fornecidas pela *Date-Time API* da linguagem *JAVA*.

Dentro deste contexto, foi desenvolvida uma calculadora universal com três modos de funcionamento.

O primeiro diz respeito a uma calculadora de datas (modo *LocalDateCalculator*). O segundo é referente a uma calculadora de fusos horários (modo *TimeZoneCalculator*). Por fim, foi também desenvolvido um modo de agendamento de reuniões (modo *Schedule*).

Estes três modos de utilização encontram-se descritos detalhadamente nas seguintes secções deste relatório.

2 Arquitetura *MVC*

A aplicação desenvolvida está assente na arquitetura *MVC* (*Model View Controller*). Este tipo de arquitetura permite uma divisão do projeto em componentes reutilizáveis. A seguinte figura ilustra a arquitetura desenvolvida, assim como os diversos componentes e ligações entre os mesmos.

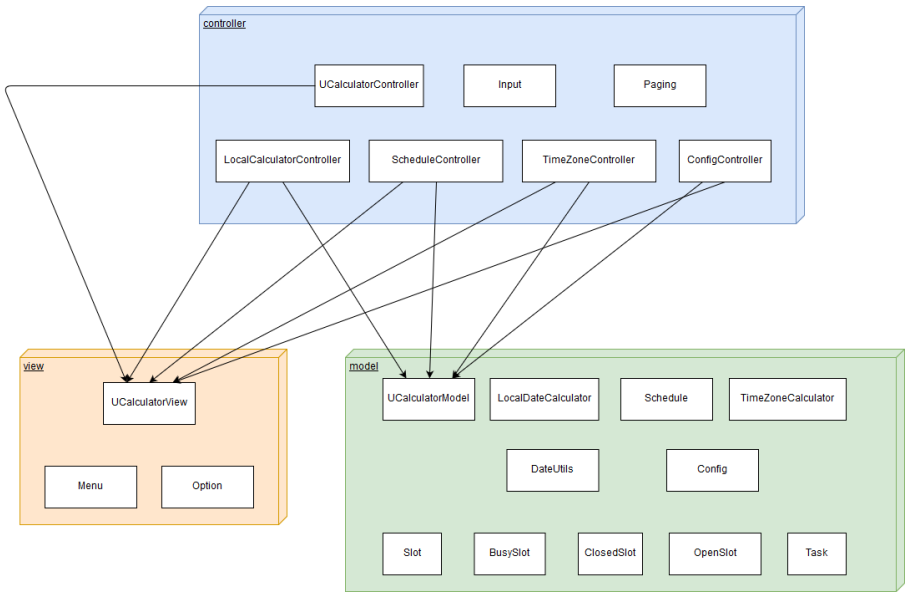

Figura 1: Arquitetura *MVC* da aplicação

2.1 *Model*

O *Model* traduz-se como sendo a lógica da aplicação. É aqui que são efetuados todos os cálculos necessários para as diversas operações entre datas desenvolvidas, assim como todas as conversões de fusos horários. Grande parte de toda a lógica associada a estas operações está presente numa classe denominada *DateUtils*. Esta classe apresenta um conjunto de métodos genéricos que tiram partido da *API DateTime* de *Java 8*.

É também no *Model* que se trata de toda a lógica associada à implementação de uma agenda com um conjunto de *slots*, isto é, que se trata de toda a lógica associada às operações de adição/remoção de um evento na agenda, assim como a sua posterior editação e consulta.

Finalmente, o *Model* trata também de toda a configuração do sistema, permitindo mesmo que um utilizador a modifica em *runtime*.

Todas estas funcionalidades são acedidas por uma camada de abstração chamada *UCalculatorModel* que consiste numa *façade* que esconde a implementação a eventuais processos que utilizem este *Model* desenvolvido. Isto permite uma maior versatilidade caso se pretenda modificar outras componentes da aplicação (como a *View*) sem que se necessite de modificar a estrutura interna do *Model*.

2.2 *View*

A *View* consiste num conjunto de componentes que permitem uma ilustração da aplicação de um modo compreensível ao utilizador. No âmbito desta aplicação, a *View* é responsável por apresentar um conjunto de menus com diversas opções ao utilizador no modo de consola de texto. Além disso, possui métodos que permitem uma representação visual dos resultados obtidos pelas calculadoras desenvolvidas, assim como métodos que ilustram um conjunto de *slots* pertencentes a uma determinada data.

Este componente não comunica com os restantes componentes do sistema, é apenas acedido pelo *Controller* através de uma *façade* chamada *UCalculatorView*.

2.3 *Controller*

O *Controller* é o componente que controla o fluxo da aplicação. É aqui que através dos pedidos efetuados pelo utilizador que se efetua a ligação entre a lógica necessária para o processamento desses pedidos e a posterior ilustração dos resultados ao utilizador. Ou seja, sempre que o *Controller* recebe um pedido do utilizador (após efetuar uma pré-validação do seu *Input*), reencaminha esse pedido para o *Model* e, após a receção dos resultados, pede à *View* que os represente de um modo intuitivo ao utilizador.

Após o arranque da aplicação, apenas a *façade* do *Controller* é acedida, isto é, a classe *UCalculatorController*. Esta *façade* reencaminha, conforme a decisão do utilizador, o processo para um dos quatro *subcontrollers*, sendo eles

o *LocalCalculatorController*, *TimeZoneController*, *ScheduleController* e o *ConfigController*. Pode-se então dizer que a aplicação está dividida em diversas componentes independentes entre si, sendo que cada *subcontroller* apenas acede aos métodos que necessita.

Ao longo deste relatório explicar-se-á, com detalhe, as implementação destas quatro diferentes funcionalidades implementadas nesta aplicação.

3 *LocalDateCalculator*

Esta componente da aplicação é controlada pelo *LocalCalculatorController* e divide-se em três subcomponentes. Estes consistem num processo que efetua um conjunto de somas/subtrações de unidades de tempo a uma data (*Date Calculator*), num processo que determina o intervalo de tempo decorrido entre quaisquer duas datas e outro processo que efetua operações relacionadas com semanas. O *controller* inicia-se pelo método *startFlow* que invoca um dos componentes mencionados conforme a intenção do utilizador.

3.1 *Date Calculator*

Tal como foi mencionado, esta componente permite a adição/subtração de um determinado valor de tempo a uma data. O *controller* possui uma variável construída à custa da estrutura *Stack* de *Java* denominada de *state*. Esta variável permite manter um conjunto de *strings* que representam o estado de um conjunto de operações efetuadas a uma data. Assim, pode-se enviar este conjunto de *strings* à *View* que ilustra este aspeto ao utilizador. Tomou-se esta decisão de modo a tornar a interface de texto mais intuitiva ao utilizador. Um exemplo da ilustração desta variável ao utilizador é o seguinte:

```
*** SELECT OPERATION ***
Add ------ 1
Subtract - 2
Solve ---- 3
Back ----- 4
Cancel --- 0
State:  12-01-2018 + 12 days
Insert option:
```

Figura 2: Estado atual de um conjunto de operações efetuadas a uma data

Como se pode verificar na figura, o utilizador até ao momento introduziu a data do dia 12 de Janeiro de 2018 e adicionou 12 dias a essa data. Daqui o utilizador pode continuar a adicionar ou subtrair unidades de tempo através das opções de *Add* e *Subtract*, ou obter o resultado final da operação através da opção *Solve*. A implementação da variável à custa de uma *Stack* é bastante útil para a opção *Back*. Para o caso apresentado acima, a aplicação anularia o valor de 12 dias e voltaria a pedir um novo valor de tempo ao utilizador para somar à data inicial. Internamente, à variável *state* basta apenas efetuar-se a operação *pop()* para remover o estado anterior.

Todos os processos mencionados acima são efetuados pelo *controller* por serem processos de validação/interação com o utilizador. No entando, toda a lógica associada aos cálculos propriamente ditos são efetuados pelo *model*. No *model* está presente uma classe chamada *DateUtils* que consiste numa classe utilitária que possui um conjunto de funções genéricas que manipulam datas. Certas funções são um pouco mais específicas, como por exemplo a adição ou remoção de dias úteis ou de quinzenas. Todas as funções criadas para esta calculadora de datas consistem em expressões *lambda*. Esta decisão permite uma maior abstração do código já que estas funções podem ser passadas como parâmetro a outras eventuais classes auxiliares do *model*.

Uma outra classe bastante relevante pertencente ao *model* é a classe *LocalDateCalculator*. Esta classe está também implementada como sendo uma *Stack* de *LocalDates*. Aqui faz-se o *push* de uma nova operação e também o *pop* quando for necessário. Sempre que o utilizador pede para adicionar um conjunto de dias a uma data, por exemplo, esta classe recebe uma função como parâmetro a aplica-a ao último resultado obtido, isto é, à *LocalDate* presente no topo da *Stack*. Após o cálculo do novo resultado, adiciona este mesmo resultado ao topo *Stack*. Novamente, este tipo de implementação é bastante útil quando o utilizador pretende efetuar o *undo* de operações. Como foi explicado acima, a *DateUtils* tem presente um conjunto de expressões *lambda* que efetuam adições ou subtrações a uma data. Isto permitiu que esta classe *LocalDateCalculator* não tenha que conhecer diretamemte a *DateUtils* já que pode receber as expressões *lambda* por parâmetro. De notar que é a *façade UCalculatorModel* que faz a ligação entre estas duas classes. Evidentemente que o *controller* não conhece esta implementação, ele apenas comunica com a *API* fornecida pela *façade* do *model*.

3.2 *Interval Calculator*

Nesta componente, o *controller* pede ao utilizador que introduza duas datas. Após a receção das mesmas (efetuando um processo de validação) o *controller* pergunta ao utilizador como pretende receber o resultado desta operação, dando a escolher ao utilizador um conjunto de unidades ou a opção de receber o resultado como um *Period*. Quando recebe a opção pretendida do utilizador, reencaminha as datas para a *façade UCalculatorController*. O resultado é calculado através das funções *intervalInUnits* para o caso da escolha de uma unidade em específico ou através da função *intervalBetweenDates* para a escolha

do resultado como sendo um *Period*. Existem também funções mais específicas chamadas *intervalInWorkingDays* e *IntervalInFortnights* que devolvem, respetivamente, o resultado do intervalo em dias úteis e em quinzenas. Todas estas funções estão presentes na classe *DateUtils*.

3.3 *Weeks Calculator*

Esta calculadora contém três modos distintos. Um destes modos consiste em calcular o número da semana do ano a que uma determinada data pertence. Aqui o *controller* pede uma data ao utilizador e após a sua validação pede ao *model* que a calcule através da função *weekNumberOfLocalDate* presente na classe *DateUtils*.

Outra funcionalidade consiste em calcular as datas de início e de fim da semana, recebendo como *input* o número da semana do ano e o ano à qual esta pertence. Esta funcionalidade é calculada com recurso ao método *dateOfWeekNumber* da classe *DateUtils*.

Quanto à restante funcionalidade, esta permite calcular as datas de um determinado dia da semana de um determindo mês e ano. Por exemplo, determinar a data da terceira Segunda-feira do mês de Dezembro de 2016. Além de permitir este cálculo para uma determinada semana do mês, também permite o cálculo de todas as Segundas-feiras presentes num mês, isto para o caso exemplificado. Neste caso o resultado é uma lista que contém todas essas datas. Estas operações são calculadas com recurso aos métodos *dateOfDayOfWeekInMonth* e *getAllDaysOfWeekInMonth* presentes na classe *DateUtils*.

De notar que todas as funções mencionadas não são diretamente acedidas pelo *controller*, este apenas comunica com a *façade* do *model*.

4 *TimezoneCalculator*

Esta componente da aplicação é controlada pelo *TimezoneController* e divide-se em dois subcomponentes (i.e. operações).

O método principal do controlador do modo *TimezoneCalculator* é o *startFlow*. Este método é responsável por exibir o menu principal deste modo ao utilizador, através da *UCalculatorView*. Através do *UCalculatorModel*, inicializa a variável do respetivo submodelo *TimezoneCalculator* responsável por guardar os dados relativos à lista de fusos horários disponíveis e que serão utilizados para as conversões posteriores. Após o *input* do utilizador, é também responsável por chamar os métodos responsáveis pelo funcionamento das duas operações suportadas: *Timezone Converter* e *Travel Calculator*.

4.1 *Timezone Converter*

A operação *Timezone Converter* permite ao utilizador saber qual o dia e a hora atuais numa qualquer parte do mundo.

Em primeiro lugar, o controlador pede ao utilizador, através da *View*, que indique a localização correspondente ao fuso horário que pretende conhecer. Esta inserção pode também ser feita através de uma listagem paginada (cf. secção 4.3).

Após a receção do *input* (pré-validado pela classe auxiliar *Input*), o controlador envia um pedido ao método *getTimeZone* do *model*, por forma a obter a hora atual no fuso horário correspondente à localização inserida pelo utilizador.

Por sua vez, este método chama o utilitário presente na *DateUtils*. Esta possui uma função chamada *convertToTimezone* que, recebendo um identificador do fuso horário e uma *LocalDateTime*, converte essa mesma data/hora para o fuso horário recebido.

Por fim, o resultado desta operação é mostrado ao utilizador através da *View*, como indica a Figura 3. Este resultado obedece às configurações efetuadas pelo utilizador relativamente ao formato de exibição das datas e horas.

```
*** TIMEZONE CALCULATOR ***
Timezone Converter - 1
Travel Calculator -- 2
Back ---------------- 0
Insert option: 1
Enter a location (0 to cancel): Macao
Current timezone at Asia/Macao is 06-12-2018 03:30.
Press ENTER to continue...
```

Figura 3: Resultado da operação *Timezone Converter*

4.2 *Travel Calculator*

A operação *Travel Calculator* possibilita ao utilizador fazer o planeamento de uma viagem de avião. Permite indicar a localização do destino, apresentando qual a data/hora dos locais a que irá chegar. Existe ainda a possibilidade de inserir múltiplas escalas ao voo.

Primeiramente, o controlador pede ao utilizador, através da *View*, que insira a data e a hora de início da viagem.

Depois, é exibido um menu em que poderá escolher adicionar uma escala ou calcular o resultado. Caso seja selecionada a opção de calcular o resultado sem que tenha sido adicionada pelo menos uma escala, o controlador, através da *View*, envia uma mensagem de erro.

De seguida, caso o utilizador indique que pretende inserir uma nova escala, o controlador envia um pedido à *View* para que seja exibido um novo submenu. Aqui, é perguntado ao utilizador qual a localização da conexão, bem como qual o tempo total de voo. Caso não se trate da primeira escala a ser adicionada, é também perguntado qual o tempo entre voos. Esta informação pretende tornar

a operação mais realista visto que as trocas de aviões/voo não são imediatas nestas situações.

Todas as inserções de localizações podem ser também efetuadas pelo utilizador com recurso a uma listagem paginada (cf. secção 4.3). Além disso, todos os pedidos e exibições de datas/horas são feitos obedecendo às configurações efetuadas pelo utilizador relativamente ao seu formato.

Como é necessária uma noção de ordem devido às sucessões das escalas, estas informações são guardadas num *LinkedHashMap* auxiliar. Este mapeamento, associa a localização da escala a um par (i.e. *SimpleEntry*) correspondente ao tempo do voo e ao tempo entre o voo anterior e o atual. Para o caso da primeira conexão, é dado tempo entre voos nulo (i.e. 0 horas e 0 minutos).

Para o cálculo do resultado final, é feita uma simples iteração por este mapeamento. Nesta iteração são calculadas as horas de chegada aos sucessivos destinos, sendo enviados pedidos ao método *getArrivalTime* do *model*, por forma a fazer as conversões para os fusos horários locais. Por sua vez, o *model* chama a função *convertToTimezone* da classe auxiliar *DateUtils* (descritas na secção anterior) para converter a data/hora de partida para o fuso horário local. Depois, apenas lhe soma o tempo de voo. Para a seguinte iteração, bastará somar o tempo entre voos a este valor para ter a nova data/hora de partida.

Estas atualizações dos tempos de início e fim dos voos de cada uma das escalas são feitas pelo método *getTravelTimes* do *model* e respetiva função utilitária da classe *DateUtils*. Este método recebe o *LocalDateTime* correspondente à data/hora de fim do voo anterior, o tempo entre voos da escala e a duração do voo atual. Desta forma, consegue calcular as *LocalDateTimes* de início e de fim do voo de cada uma das escalas. O *LocalDateTime* de início corresponde à soma do tempo entre voos ao *LocalDateTime* de fim do voo anterior. O *LocalDateTime* de fim corresponde à soma da duração do voo à data/hora de início.

Finalmente, o resultado desta operação é mostrado ao utilizador através da *View*, como indica a Figura 4.

```
    *** ADD CONNECTION ***
Add Connection -------------- 1
Calculate Total Travel Time - 2
Cancel ---------------------- 0
Insert option: 2
Arrival at Europe/Moscow at 01-01-2019 15:30 (01-01-2019 18:30 local)
Arrival at Asia/Shanghai at 02-01-2019 00:50 (02-01-2019 08:50 local)
Press ENTER to continue...
```

Figura 4: Resultado da operação *Travel Calculator*

4.3 Paginação

Esta classe é pertencente à camada do controlador da aplicação. De um ponto de vista geral, a sua função é exibir um conjunto possivelmente grande de

entradas de uma lista em várias páginas. Desta forma, possibilita uma exibição mais clara dos elementos dessa lista ao utilizador.

Mais concretamente, esta classe é utilizada pela nossa aplicação para permitir ao utilizador inserir localizações – correspondente aos inúmeros fusos horários mundiais –, de uma forma mais clara.

Para isso, esta classe possui as seguintes variáveis de instância:

- *elements* – lista de todos os elementos a serem exibidos
- *currentPage* – índice da página atual
- *pageSize* – número de elementos a exibir por página
- *totalPages* – número total de páginas

Por forma a permitir a visualização posterior da listagem, possui métodos responsáveis pela geração de uma lista de elementos correspondentes aos elementos da página atual, da página seguinte e da página anterior. Estes métodos são depois utilizados ao nível do *TimezoneController* para gerar os elementos das sucessivas páginas e exibi-las ao utilizador, através do método *displayPage* da *UCalculatorView*.

Além disso, esta classe possibilita a obtenção de um elemento específico da página atual, através do seu índice na página atual. Com recurso a este método, o controlador poderá, então, obter o elemento da lista escolhido pelo utilizador.

Do ponto de vista do utilizador, este tipo de listagem paginada será mostrada caso este, ao ser-lhe pedido que insira uma localização/fuso horário, pressione a tecla *enter* ou caso insira um nome faça *match* com vários fusos horários. Na figura 5 podemos ver esta listagem.

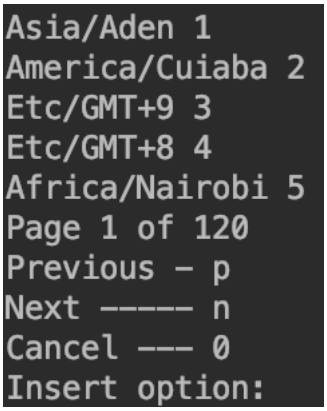

Figura 5: Visualização das localizações (i.e. fusos horários) através de listagem paginada

Será também exibida uma listagem paginada ao utilizador caso este indique, por exemplo, um nome de um continente (e.g. *"Europe"*), como mostra a figura 6. Na verdade, isto acontece porque é feita uma procura pela *string* inserida na totalidade dos elementos, devolvendo-os numa listagem. Isto significa que não é obrigatório que se insira o nome de um continente, mas outra qualquer *string* a procurar.

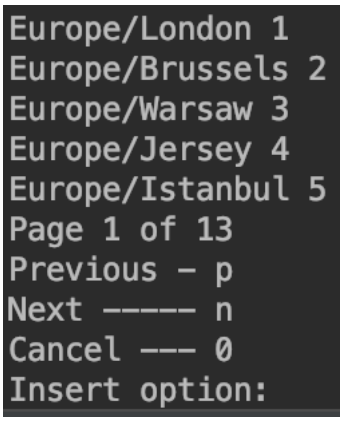

Figura 6: Visualização das localizações (i.e. fusos horários) de um continente através de listagem paginada

Depois, em ambas estas listagens, o utilizador possuirá a opção de selecionar um dos elementos da página (opção *1* a *5*), retroceder para a página anterior (opção *p*) ou avançar para a página seguinte (opção *n*). Poderá ainda cancelar (opção *0*), voltando para o menu anterior.

5 *Schedule*

A Schedule é a classe que abarca a funcionalidade de agendamento da aplicação. A sua estrutura interna é organizada por dias (fazendo uso de um mapa cuja chave é uma instância de LocalDate) dividido em slots (cujo valor do mapa é uma lista desses). Portanto, traduzindo-se em Map<LocalDate, List<Slot». A classe Slot é abstrata pelo que se concretiza em três estados distintos BusySlot, OpenSlot, e ClosedSlot tal que cada uma destas classes extende Slot. OpenSlot e ClosedSlot acrescentam semântica ao Slot indicando se este se encontra disponível ou indisponível. Por outro lado, BusySlot indica que o Slot se encontra ocupado ao mesmo tempo que contém umas instância de Task. Por sua vez, Task representa a tarefa contida no slot que consiste num título/descrição e numa lista de indivíduos com a qual esta é partilhada. Quanto a este módulo, algumas decisões foram tomadas. Os slots têm um tamanho dinâmico contudo este não pode ser alterado sem que implique reinicializar a estrutura. Ou seja, a duração do slot pode ir desde 1 minuto por slot até 1 dia inteiro (1440 minutos). O valor por defeito na primeira utilização da aplicação é de 60 minutos podendo ser alterado no ficheiro de configuração ou em Runtime através do menu para o efeito. Além do tamanho do slot, também é possível alterar a hora de início do horário assim como a hora final. Estes parâmetros podem ser alterados sem obrigar a estrutura a ser novamente inicializada. Sempre que for efetuada uma inserção de uma tarefa ou simplesmente a mudança do estado de um slot de aberto para fechado toda a lista de slots desse dia é inicializada. Significa isto que sempre que um utilizador consultar o dia de uma agenda que ainda não esteja inicializado, este poderá visualizar todos os slots desse dia no estado aberto apesar de não estarem ainda na estrutura respetiva da agenda. Assim, é possível poupar memória ao apenas ser armazenado na agenda os slots de dias que tenham sofrido efetivamente alterações. Uma decisão tomada que também merece ser referida passa pelo facto dos ids dos slots externos diferirem dos internos. Isto é, o utilizador visualizará os slots sempre desde o id 0 independentemente da hora de início e fim definidas por ele. Para efeitos de computação, este id externo é convertido para o id efetivo. O intuito deste pormenor passa por abstrair o estado interno da agenda quando o utilizador efetuar operações. Por fim, outra tomada de decisão importante encontra-se na operação de definir a hora de início ou fim da agenda. Isto é, para a operação de definir a nova hora inicial do horário, caso não seja possível definir o slot para aquela especificada pelo utilizador (devido a incompatibilidade), o slot imediatamente anterior passa a ser o efetivo. Por exemplo, assumindo uma duração do slot de 60 minutos, e o utilizador especificar uma hora de inicio às 02:30, então a hora efetiva passa a ser às 02:00 pois é a anterior compatível com a duração do slot. Neste caso, traduzir-se-à numa agenda que começa efetivamente no slot cujo id é 2 internamente. Ao definir a hora de fim da agenda, ocorre o análogo tal que passa a ficar efetivo o slot imediatamente a seguir à hora especificada. Por fim, o utilizador pode efetuar 6 operações distintas na agenda.

1. Add Task: esta operação consiste em adicionar uma nova tarefa à agenda

para isso basta serem fornecidos o dia, o slot inicial e a duração desta. Denote-se que apenas é possível preencher slots livres (OpenSlot) pelo que se a tarefa intersetar com algum slot ocupado ou fechado não permite inserir.

2. Open Slots: esta operação tem como intuito abrir slots previamente fechados. Isto é, passar slots do estado indisponível para disponível novamente. Caso intersete com slots no estado ocupado ou livre, estes não sofrerão alterações.

3. Close Slots: esta operação é análoga a Open Slots exceto no facto de não ser possível fechar slots caso a duração especificada do utilizador intersete com algum slot ocupado, ou seja, que contenha uma tarefa.

4. Consult: esta operação permite visualizar o estado da agenda. Para tal basta ser especificado o dia respetivo. Ainda, é possível navegar na mesma fazendo uso das teclas 'p' (previous) e 'n' (next). Para efeitos de computação, o utilizador está efetivamente a visualizar uma cópia dos slots visto a função responsável por este use case em Schedule efetuar a operação de clone de cada slot.

5. Edit Task: esta operação permite editar uma tarefa em qualquer um dos seus campos. Isto é, é possível alterar o seu título assim como pessoas envolvidas tal como a data em que se efetua, o slot inicial e a duração desta.

6. Remove Task: esta operação encarrega-se por remover uma tarefa tal que apenas é preciso indicar um slot em que esteja esteja inserida e todos os restantes slots serão removidos. Ou seja, passarão para o estado OpenSlot. É importante salientar que todas as tarefas estão presentes num espaço contíguo de slots não podendo ser dividida.

6 *Config*

A Config é a classe responsável por gerir as configurações assim como persistir a instância da classe Schedule. Aqui, o grupo decidiu fazer uso de um Singleton visto ser pretendida a existência de apenas uma instância desta classe assim como efetuar o carregamento das configurações uma vez ao inicializar de forma a manter a consistência entre o estado do programa em memória e das configurações e Schedule persistidas. Apesar de não ser necessário neste contexto, ainda foram tomadas medidas de forma a evitar a sua instanciação por reflexão de Java e prevenção de multi-instanciação simultânea num contexto multi-threaded (quando chamado pela primeira vez o método getInstance). Denote-se que apesar de não ser necessário nesta aplicação, o grupo acrescentou estas contra-medidas por considerar uma boa prática. Por conseguinte, ao utilizador é dada a possibilidade de alterar o padrão utilizado na definição de data e hora, apenas data ou apenas a hora. Além disso, também é possível alterar o tamanho do slot assim como a hora de inicio e fim do horário. Existem dois mecanismos possíveis para efetuar estas alterações: através do ficheiro de configurações existente denominado por "UCalculator.config" na raiz do projeto ou em Runtime usando o menu para o efeito. Se porventura alguma configuração não estiver correta ou consistente, a configuração por defeito será a utilizada. Por conseguinte, alterar o tamanho do slot implica reinicializar a estrutura do Schedule visto a duração anterior das tarefas não se transportarem por vezes para a nova duração dos slots. Por fim, todas as alterações efetuadas no ficheiro de configuração serão carregadas para a aplicação. Assim, caso a agenda sofra uma nova hora de início ou fim, a estrutura ao ser carregada do disco sofrerá logo a seguir essas alterações de forma a manter a consistência. Por fim, as alterações introduzidas pelo utilizador através do menu para efeito na aplicação serão imediatamente persistidas e tomadas efetivas em Runtime inclusivamente a mudança da duração dos slots.

7 Conclusões e Análise Crítica

Este projeto permitiu adquirir e reforçar conhecimento relativamente à utilização de uma arquitetura *MVC* que consiste em decompor uma aplicação em três camadas distintas: *Model*, *View* e *Controller*. Desta forma, este padrão arquitetural tem como intuito a separação da aplicação em camadas independentes que interajam mutuamente. Através da divisão por camadas, o código desenvolvido revela-se mais robusto seja pela independência da lógica inerente seja na facilidade em detetar eventuais *bugs* e a sua correção.

O foco principal deste projeto consistiu na exploração da *API Date-Time* de *Java* introduzida na versão 8. Esta *API* assenta num conjunto extenso de métodos e operações sobre datas e tempo que amplificam imenso as funcionalidades de outras *APIs* mais antigas, como é exemplo a *API* do *GregorianCalendar*, e que permitem a simplificação do seu uso no desenvolvimento de aplicações que dependam destas.

Por fim, o grupo considera ter feito um bom uso da *API Date-Time* e que construiu um sistema de agendamento relativamente versátil voltado ao utilizador. Contudo, a *User Interface* deveria ter sido desenvolvida com recurso a uma ferramenta gráfica como o *Swing*, já que uma interface em modo de texto tende a não ser tão intuitiva principalmente se possuir um vasto número de funcionalidades. Outro aspeto que se deve considerar em melhorar no futuro consiste em tornar a agenda ainda mais versátil, podendo dar ao utilizador a opção de acrescentar mais detalhes às tarefas inseridas. Por último, outro mecanismo poderia ser desenvolvido de modo a que se possa modificar a duração do *slot* sem que todos os dados do horário sejam perdidos, visto a estrutura ser inicializada novamente nesta versão da aplicação.