Code Inspection Report

'Bom Dia Academia' Software Development Project

Group: 01
77794, Rita Tomás, EIC2
77959, Hugo Martins, EIC1PL
78174, Joaquim Rocha, EIC1PL
79086, Ana Nunes, EIC2

BSc/MSc in [LEI | LIGE | METI]
Academic Year 2018/2019 - 1st Semester
Software Engineering I

ISCTE-IUL, Instituto Universitário de Lisboa
1649-026 Lisbon
Portugal

November 2018

Table of Contents

Introduction 3
Code inspection – BDA 5
Code inspection checklist 6
Found defects 8
Corrective measures 8
Conclusions of the inspection process 8
Code inspection – Email 9
Code inspection checklist 9
Found defects 11
Corrective measures 12
Conclusions of the inspection process 12
Code inspection – Facebook 12
Code inspection checklist 12
Found defects 15
Corrective measures 15
Conclusions of the inspection process 15
Code inspection – Twitter 15
Code inspection checklist 16
Found defects 18
Corrective measures 18
Conclusions of the inspection process 19

Introduction

O software desenvolvido consiste na elaboração de uma aplicação, designada de "Bom Dia Academia", que integra várias fontes de informação académica.

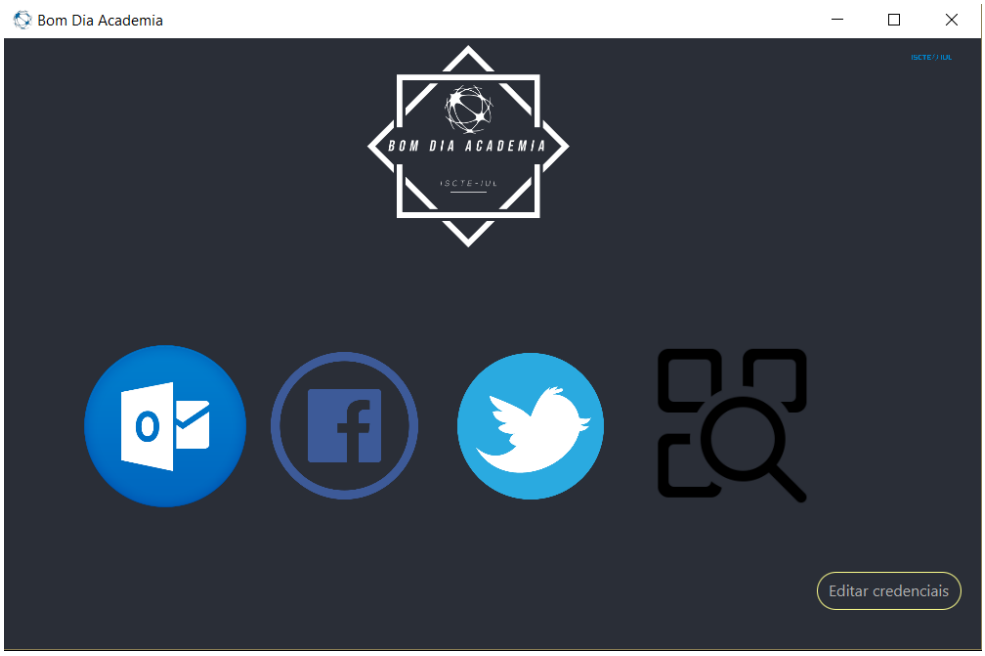


Fig.1 – Página principal da aplicação

O código gerado permite executar funcionalidades para consulta de informação académica através de determinados serviços: Facebook, Twitter e Email. É a partir destes três canais que o utilizador pode consultar e enviar dados dependendo do canal que escolher (enviar mensagens, responder a emails, fazer retweets, etc...).

Para consultar os vários tipos de serviços da aplicação, implementou-se uma GUI (interface gráfica) do tipo Timeline, onde as informações/mensagens são disponibilizadas de forma cronológica. Também é possível visualizar e obter os dados essenciais das mesmas, nomeadamente a data e hora, o título e a fonte.

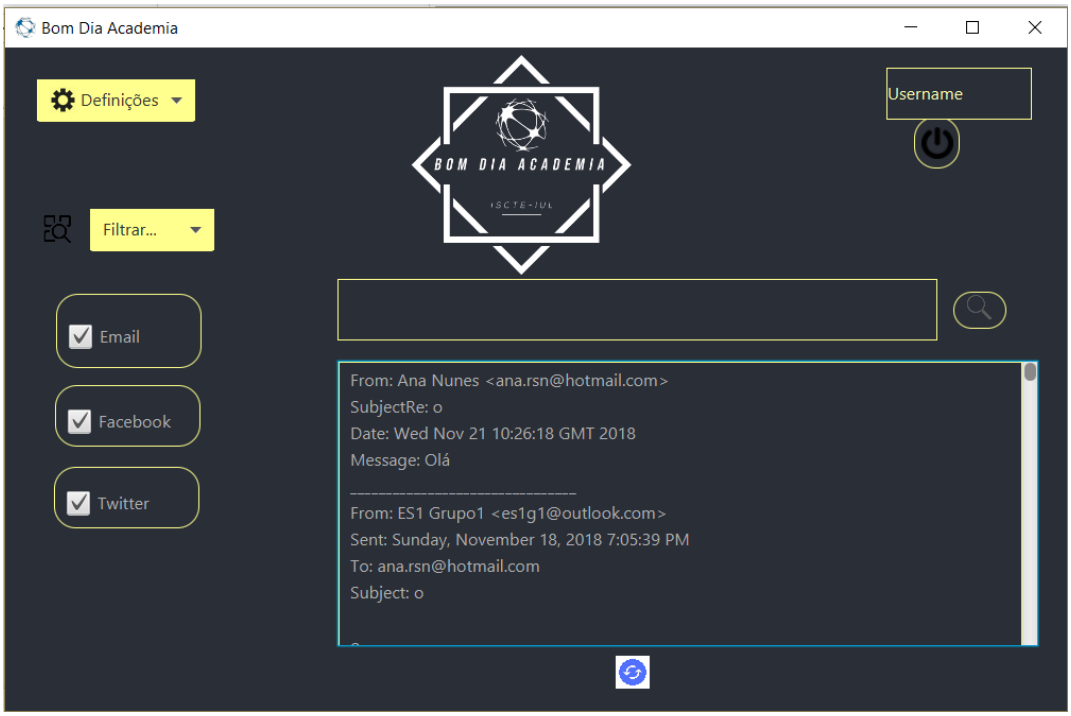


Fig.2 – Timeline

Em relação aos conteúdos propriamente ditos, com a aplicação é possível filtrá-los segundo determinados parâmetros e também tendo em conta as características da API de cada canal: filtragem por palavra-passe, filtragem por periocidade no tempo (conteúdos mais recentes, conteúdos das últimas 24 horas, da última semana ou do ultimo mês) e também filtragem de conteúdos de determinados utilizadores/contas.

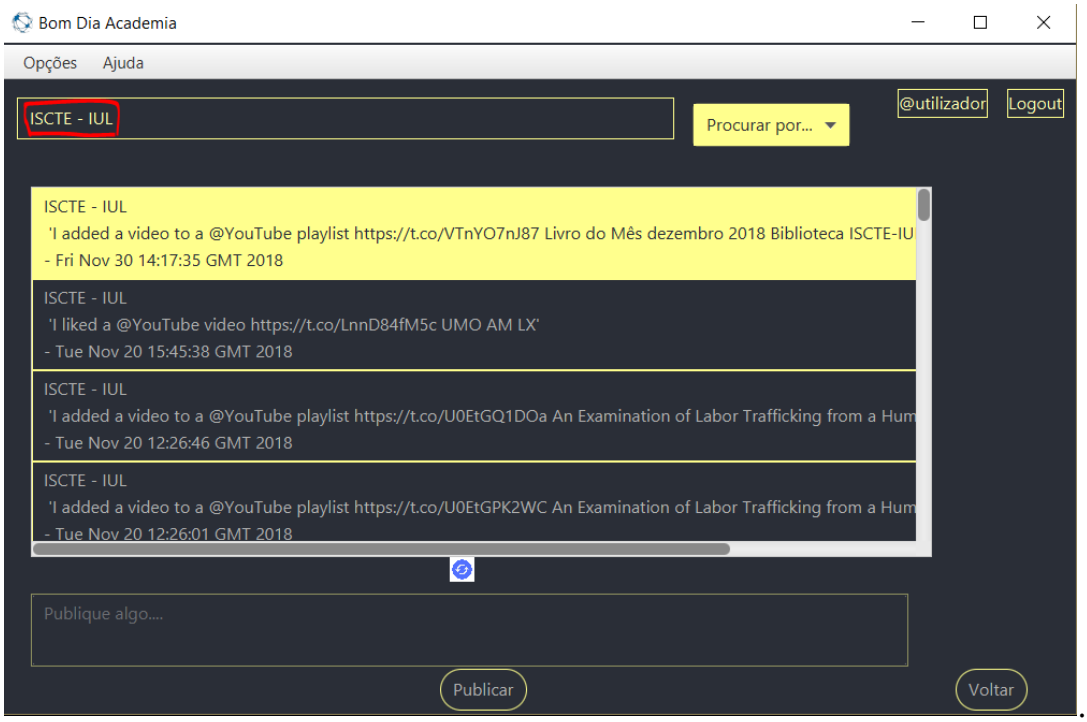


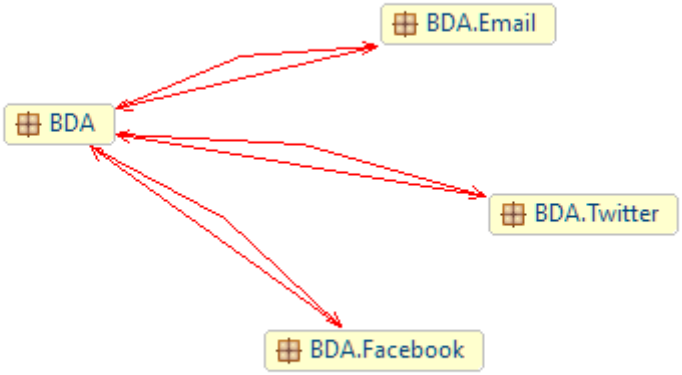
Fig.3 – Exemplo da filtragem de conteúdos (Twitter)

O ficheiro “config.xml” contém toda a documentação necessária dos critérios em uso para os filtros de cada serviço. É também neste ficheiro que se encontra toda a informação das configurações de cada serviço, tais como os dados dos tokens de acesso ou autenticação, nomes de utilizadores/contas, palavras-passe, Consumer Keys, entre outros. Estes dados poderão, posteriormente, ser manipulados (removidos, modificados, criados) através do GUI da aplicação.

Para prevenir eventuais constrangimentos na utilização da aplicação (má ligação com a rede, desconexão acidental com a aplicação, …), esta permite o seu funcionamento em modo offline, de modo a que o utilizador possa consultar toda a informação académica obtida pelos serviços anteriormente.

Através do diagrama de dependência de pacotes gerado pela ferramenta ModelGoon e apresentado abaixo concluímos que cada serviço é unido através do pacote BDA, sendo neste que implementamos a timeline conjunta dos três serviços.

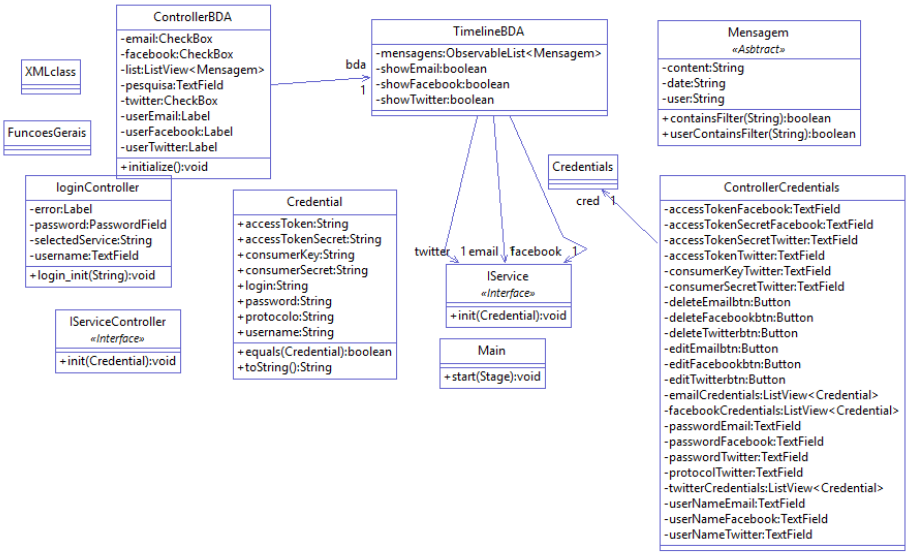
Observámos também que cada serviço consegue funcionar separadamente pois cada pacote é independente sendo apenas reunido aos restantes pelo pacote BDA.



Code inspection – BDA

Package que agrega as principais classes, nomeadamente main, as interfaces IService, a super classe Mensagem, XMLClass(a responsavel por guardar as credenciais no fixeiro config.xml), as classes e o controlador onde existe a possibilidade de alterar credênciais, o controlador de login que é utilizado em todos os serviços, o controlador da janela inicial e também o controlador correspondente à janela onde podemos vizualizar uma timeline correspondente a todos os serviços utilizados. Dentro deste package podemos encontrar também os ficheiros fxml correspondentes a cada controlador,

Diagrama de classes:



Meeting date:	*5/12/2018*
Meeting duration:	*60 minutes*
Moderator:	*Rita Caveirinha*
Producer:	*Hugo Martins e Ana Nunes*
Inspector:	*Rita Tomás*
Recorder:	*Joaquim Rocha*
Component name (Package/Class/Method):	*BDA*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*56,6*

Code inspection checklist

1. Variable, Attribute, and Constant Declaration Defects (VC)
   - ✗ Are descriptive variable and constant names used in accord with naming conventions?
   - ✓ Are there variables or attributes with confusingly similar names?
   - ✓ Is every variable and attribute correctly typed?
   - ✓ Is every variable and attribute properly initialized?
   - ✓ Could any non-local variables be made local?
   - ✓ Are all for-loop control variables declared in the loop header?
   - ✓ Are there literal constants that should be named constants?
   - ✓ Are there variables or attributes that should be constants?
   - ✓ Are there attributes that should be local variables?
   - ✓ Do all attributes have appropriate access modifiers (private, protected, public)?
   - ✓ Are there static attributes that should be non-static or vice-versa?
2. Method Definition Defects (FD)
   - ✓ Are descriptive method names used in accord with naming conventions?
   - ✗ Is every method parameter value checked before being used?
   - ✓ For every method: Does it return the correct value at every method return point?
   - ✓ Do all methods have appropriate access modifiers (private, protected, public)?
   - ✓ Are there static methods that should be non-static or vice-versa?
3. Class Definition Defects (CD)
   - ✓ Does each class have appropriate constructors and destructors?

- ✓ Do any subclasses have common members that should be in the superclass?
- ✓ Can the class inheritance hierarchy be simplified?

4. Data Reference Defects (DR)
   - ✓ For every array reference: Is each subscript value within the defined bounds?
   - ✓ For every object or array reference: Is the value certain to be non-null?
5. Computation/Numeric Defects (CN)
   - ✓ Are there any computations with mixed data types?
   - ✓ Is overflow or underflow possible during a computation?
   - ✓ For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
   - ✓ Are parentheses used to avoid ambiguity?
6. Comparison/Relational Defects (CR)
   - ✓ For every boolean test: Is the correct condition checked?
   - ✓ Are the comparison operators correct?
   - ✓ Has each boolean expression been simplified by driving negations inward?
   - ✓ Is each boolean expression correct?
   - ✓ Are there improper and unnoticed side-effects of a comparison?
   - ✓ Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

7.Control Flow Defects (CF)

- ✓ For each loop: Is the best choice of looping constructs used?
- ✓ Will all loops terminate?
- ✓ When there are multiple exits from a loop, is each exit necessary and handled properly?
- ✓ Does each switch statement have a default case?
- ✓ Are missing switch case break statements correct and marked with a comment?
- ✓ Do named break statements send control to the right place?
- ✓ Is the nesting of loops and branches too deep, and is it correct?
- ✓ Can any nested if statements be converted into a switch statement?
- ✓ Are null bodied control structures correct and marked with braces or comments?
- ✓ Are all exceptions handled appropriately?
- ✓ Does every method terminate?

8.Input-Output Defects (IO)

- ❑ Have all files been opened before use?
- ❑ Are the attributes of the input object consistent with the use of the file?
- ❑ Have all files been closed after use?
- ❑ Are there spelling or grammatical errors in any text printed or displayed?
- ❑ Are all I/O exceptions handled in a reasonable way?

9.Module Interface Defects (MI)

- ✓ Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?
- ✓ Do the values in units agree (e.g., inches versus yards)?
- ✓ If an object or array is passed, does it get changed, and changed correctly by the called method?

10.Comment Defects (CM)

- ✗ Does every method, class, and file have an appropriate header comment?
- ✗ Does every attribute, variable, and constant declaration have a comment?
- ✓ Is the underlying behavior of each method and class expressed in plain language?
- ✓ Is the header comment for each method and class consistent with the behavior of the method or class?
- ✓ Do the comments and code agree?
- ✓ Do the comments help in understanding the code?
- ✗ Are there enough comments in the code?

- ✓ Are there too many comments in the code?

11.Layout and Packaging Defects (LP)

- ✓ Is a standard indentation and layout format used consistently?
- ✓ For each method: Is it no more than about 60 lines long?
- ✓ For each compile module: Is no more than about 600 lines long?

12.Modularity Defects (MO)

- ✓ Is there a low level of coupling between modules (methods and classes)?
- ✓ Is there a high level of cohesion within each module (methods or class)?
- ✓ Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
- ✓ Are the Java class libraries used where and when appropriate?

13.Storage Usage Defects (SU)

- ✓ Are arrays large enough?
- ✓ Are object and array references set to null once the object or array is no longer needed?

14.Performance Defects (PE)

- ✓ Can better data structures or more efficient algorithms be used?
- ✓ Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
- ✓ Can the cost of recomputing a value be reduced by computing it once and storing the results?
- ✓ Is every result that is computed and stored actually used?
- ✓ Can a computation be moved outside a loop?
- ✓ Are there tests within a loop that do not need to be done?
- ✓ Can a short loop be unrolled?
- ✓ Are there two loops operating on the same data that can be combined into one?
- ✓ Are frequently used variables declared register?
- ✓ Are short and commonly called methods declared inline?

Found defects

Found defect Id	Package, Class, Method, Line	Defect category	Description
1			
2			
3			
...	...	...	...

Corrective measures

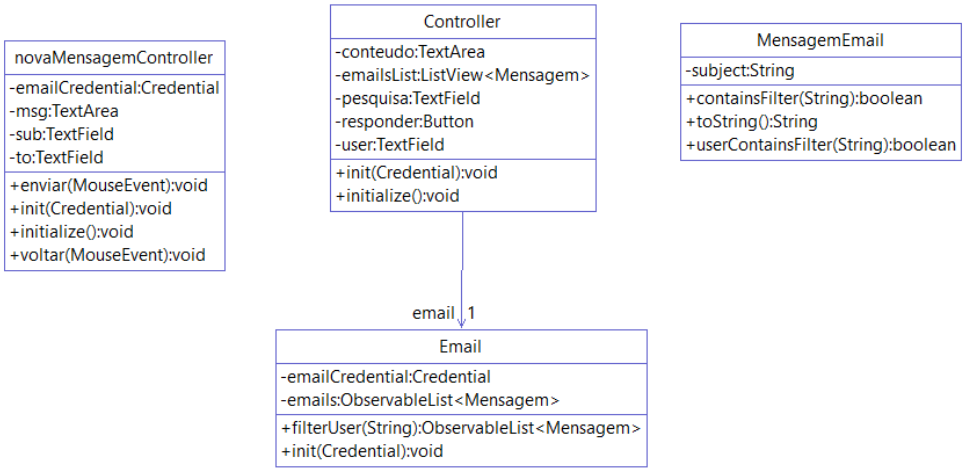
Nomes dos packages não estão de acordo com convenções, pois estão em letra maiusculo. Faltam comentários em alguns atributos e metodos, e Javadocs. A interface gráfica podia estar mais agradável, alguns elementos estão desalinhados. Penso que deveria também haver concordância ao longo do projeto em relação à lingua utilizado, visto que estamos a alternar entre português e inglês.

Conclusions of the inspection process

Apenas pequenas alterações.

Code inspection – Email

Serviço de email, onde é possível receber e enviar mensagens, consultar as pastas “Caixa de entrada” e “Itens enviados” e também fazer pesquisas filtradas dentro das mesmas.



Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*04/12/2018* *90 minutes* *Joaquim Rocha* *Ana Nunes* *Joaquim Rocha* *Joaquim Rocha*
Component name (Package/Class/Method):	*BDA.Email/**
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*43,5%*

Code inspection checklist

1. Variable, Attribute, and Constant Declaration Defects (VC)
   - ✓ Are descriptive variable and constant names used in accord with naming conventions?
   - ✗ Are there variables or attributes with confusingly similar names?
   - ✓ Is every variable and attribute correctly typed?
   - ✓ Is every variable and attribute properly initialized?
   - ✗ Could any non-local variables be made local?
   - ✓ Are all for-loop control variables declared in the loop header?
   - ✗ Are there literal constants that should be named constants?
   - ✗ Are there variables or attributes that should be constants?

- ✗ Are there attributes that should be local variables?
- ✓ Do all attributes have appropriate access modifiers (private, protected, public)?
- ✗ Are there static attributes that should be non-static or vice-versa?

2. Method Definition Defects (FD)
   - ✓ Are descriptive method names used in accord with naming conventions?
   - ✓ Is every method parameter value checked before being used?
   - ✓ For every method: Does it return the correct value at every method return point?
   - ✓ Do all methods have appropriate access modifiers (private, protected, public)?
   - ✗ Are there static methods that should be non-static or vice-versa?
3. Class Definition Defects (CD)
   - ✓ Does each class have appropriate constructors and destructors?
   - ✗ Do any subclasses have common members that should be in the superclass?
   - ✗ Can the class inheritance hierarchy be simplified?
4. Data Reference Defects (DR)
   - ✓ For every array reference: Is each subscript value within the defined bounds?
   - ✗ For every object or array reference: Is the value certain to be non-null?
5. Computation/Numeric Defects (CN)
   - ✗ Are there any computations with mixed data types?
   - ✗ Is overflow or underflow possible during a computation?
   - ✗ For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
   - ✓ Are parentheses used to avoid ambiguity?
6. Comparison/Relational Defects (CR)
   - ✓ For every boolean test: Is the correct condition checked?
   - ✓ Are the comparison operators correct?
   - ✓ Has each boolean expression been simplified by driving negations inward?
   - ✓ Is each boolean expression correct?
   - ✗ Are there improper and unnoticed side-effects of a comparison?
   - ✗ Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

7.Control Flow Defects (CF)

- ✓ For each loop: Is the best choice of looping constructs used?
- ✓ Will all loops terminate?
- ✓ When there are multiple exits from a loop, is each exit necessary and handled properly?
- ❑ Does each switch statement have a default case?
- ❑ Are missing switch case break statements correct and marked with a comment?
- ❑ Do named break statements send control to the right place?
- ✓ Is the nesting of loops and branches too deep, and is it correct?
- ✗ Can any nested if statements be converted into a switch statement?
- ✗ Are null bodied control structures correct and marked with braces or comments?
- ✓ Are all exceptions handled appropriately?
- ✓ Does every method terminate?

8.Input-Output Defects (IO) (não se aplica)

- ❑ Have all files been opened before use?
- ❑ Are the attributes of the input object consistent with the use of the file?
- ❑ Have all files been closed after use?
- ❑ Are there spelling or grammatical errors in any text printed or displayed?
- ❑ Are all I/O exceptions handled in a reasonable way?

9.Module Interface Defects (MI)

- ✓ Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?
- ✓ Do the values in units agree (e.g., inches versus yards)?
- ✓ If an object or array is passed, does it get changed, and changed correctly by the

called method?

10.Comment Defects (CM)

- ✗ Does every method, class, and file have an appropriate header comment?
- ✓ Does every attribute, variable, and constant declaration have a comment?
- ✓ Is the underlying behavior of each method and class expressed in plain language?
- ✓ Is the header comment for each method and class consistent with the behavior of the method or class?
- ✓ Do the comments and code agree?
- ✓ Do the comments help in understanding the code?
- ✗ Are there enough comments in the code?
- ✗ Are there too many comments in the code?

11.Layout and Packaging Defects (LP)

- ✓ Is a standard indentation and layout format used consistently?
- ✗ For each method: Is it no more than about 60 lines long?
- ✓ For each compile module: Is no more than about 600 lines long?

12.Modularity Defects (MO)

- ✗ Is there a low level of coupling between modules (methods and classes)?
- ✓ Is there a high level of cohesion within each module (methods or class)?
- ✗ Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
- ✓ Are the Java class libraries used where and when appropriate?

13.Storage Usage Defects (SU)

- ✓ Are arrays large enough?
- ✗ Are object and array references set to null once the object or array is no longer needed?

14.Performance Defects (PE)

- ✗ Can better data structures or more efficient algorithms be used?
- ✓ Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
- ✗ Can the cost of recomputing a value be reduced by computing it once and storing the results?
- ✓ Is every result that is computed and stored actually used?
- ✓ Can a computation be moved outside a loop?
- ✗ Are there tests within a loop that do not need to be done?
- ✗ Can a short loop be unrolled?
- ✗ Are there two loops operating on the same data that can be combined into one?
- ✗ Are frequently used variables declared register?
- ✓ Are short and commonly called methods declared inline?

Found defects

Identify and describe found defects, opinions and suggestions.

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	BDA.Email, MensagemEmail.java, 34/90/95	CM(10)	Adicionar JavaDocs e retirar comentários para 2 métodos inexistentes
2	BDA.Email, Email.java, 50/487	CM(10)	Adicionar JavaDoc

3	BDA.Email, Controller, 75	CM(10)	Adicionar JavaDoc
4	BDA.Email, Controller, initialize(), 72	FD(2)	Método vazio
5	BDA.Email, Email, getTimeline(), 100 a 181	LP(11)	Método com mais de 60 linhas

Corrective measures

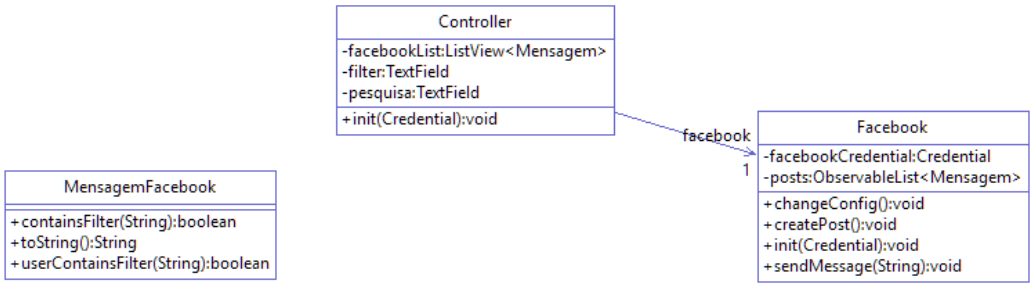
1) expandir botões de ordenar emails e de nova mensagem para não cortar as palavras (interface gráfica);
2) botões da barra superior ainda não funcionam (Opções e Ajuda);
3) diminuir o tamanho do getTimeline() seria o ideal.

Conclusions of the inspection process

Necessita apenas de correções mínimas

Code inspection – Facebook

Serviço Facebook onde é possível obter a timeline cronológica do mesmo, e filtrar o seu conteúdo.



Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*04/12/2018* *90 minutes* *Ana Nunes* *Hugo Martins* *Ana Nunes* *Joaquim Rocha*
Component name (Package/Class/Method):	*BDA.Facebook/**
Component was compiled:	
Component was executed:	*all*
Component was tested without errors:	*all*
Testing coverage achieved:	*36,9%*

Code inspection checklist

1. Variable, Attribute, and Constant Declaration Defects (VC)

- ✓ Are descriptive variable and constant names used in accord with naming conventions?
- ✗ Are there variables or attributes with confusingly similar names?
- ✓ Is every variable and attribute correctly typed?
- ✓ Is every variable and attribute properly initialized?
- ✗ Could any non-local variables be made local?
- ✓ Are all for-loop control variables declared in the loop header?
- ✗ Are there literal constants that should be named constants?
- ✗ Are there variables or attributes that should be constants?
- ✗ Are there attributes that should be local variables?
- ✓ Do all attributes have appropriate access modifiers (private, protected, public)?
- ✗ Are there static attributes that should be non-static or vice-versa?

2.Method Definition Defects (FD)

- ✓ Are descriptive method names used in accord with naming conventions?
- ✓ Is every method parameter value checked before being used?
- ✓ For every method: Does it return the correct value at every method return point?
- ✓ Do all methods have appropriate access modifiers (private, protected, public)?
- ✗ Are there static methods that should be non-static or vice-versa?

3.Class Definition Defects (CD)

- ✓ Does each class have appropriate constructors and destructors?
- ✗ Do any subclasses have common members that should be in the superclass?
- ✗ Can the class inheritance hierarchy be simplified?

4.Data Reference Defects (DR)

- ✓ For every array reference: Is each subscript value within the defined bounds?
- ☹ For every object or array reference: Is the value certain to be non-null?

5.Computation/Numeric Defects (CN)

- ✗ Are there any computations with mixed data types?
- ✗ Is overflow or underflow possible during a computation?
- ✗ For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
- ✓ Are parentheses used to avoid ambiguity?

6.Comparison/Relational Defects (CR)

- ✓ For every boolean test: Is the correct condition checked?
- ✓ Are the comparison operators correct?
- ✓ Has each boolean expression been simplified by driving negations inward?
- ✓ Is each boolean expression correct?
- ✗ Are there improper and unnoticed side-effects of a comparison?
- ✗ Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

7.Control Flow Defects (CF)

- ✓ For each loop: Is the best choice of looping constructs used?
- ✓ Will all loops terminate?
- ✓ When there are multiple exits from a loop, is each exit necessary and handled properly?
- ✗ Does each switch statement have a default case?
- ✗ Are missing switch case break statements correct and marked with a comment?
- ✓ Do named break statements send control to the right place?

- ✓ Is the nesting of loops and branches too deep, and is it correct?
- ✗ Can any nested if statements be converted into a switch statement?
- ✗ Are null bodied control structures correct and marked with braces or comments?
- ✓ Are all exceptions handled appropriately?
- ✓ Does every method terminate?

8.Input-Output Defects (IO) (não se aplica)

- ❑ Have all files been opened before use?
- ❑ Are the attributes of the input object consistent with the use of the file?
- ❑ Have all files been closed after use?
- ❑ Are there spelling or grammatical errors in any text printed or displayed?
- ❑ Are all I/O exceptions handled in a reasonable way?

9.Module Interface Defects (MI)

- ✓ Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?
- ✓ Do the values in units agree (e.g., inches versus yards)?
- ✓ If an object or array is passed, does it get changed, and changed correctly by the called method?

10.Comment Defects (CM)

- ☹ Does every method, class, and file have an appropriate header comment?
- ☹ Does every attribute, variable, and constant declaration have a comment?
- ✓ Is the underlying behavior of each method and class expressed in plain language?
- ☹ Is the header comment for each method and class consistent with the behavior of the method or class?
- ✓ Do the comments and code agree?
- ✓ Do the comments help in understanding the code?
- ☹ Are there enough comments in the code?
- ✗ Are there too many comments in the code?

11.Layout and Packaging Defects (LP)

- ✓ Is a standard indentation and layout format used consistently?
- ✓ For each method: Is it no more than about 60 lines long?
- ✓ For each compile module: Is no more than about 600 lines long?

12.Modularity Defects (MO)

- ✗ Is there a low level of coupling between modules (methods and classes)?
- ✓ Is there a high level of cohesion within each module (methods or class)?
- ✗ Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
- ✓ Are the Java class libraries used where and when appropriate?

13.Storage Usage Defects (SU)

- ✓ Are arrays large enough?
- ✗ Are object and array references set to null once the object or array is no longer needed?

14.Performance Defects (PE)

- ✗ Can better data structures or more efficient algorithms be used?
- ✓ Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
- ✗ Can the cost of recomputing a value be reduced by computing it once and storing the results?

- ✓ Is every result that is computed and stored actually used?
- ✓ Can a computation be moved outside a loop?
- ✗ Are there tests within a loop that do not need to be done?
- ✗ Can a short loop be unrolled?
- ✗ Are there two loops operating on the same data that can be combined into one?
- ✗ Are frequently used variables declared register?
- ✓ Are short and commonly called methods declared inline?

Found defects

Identify and describe found defects, opinions and suggestions.

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	BDA.Facebook, getTimeLine	DR (4)	Mensagem no caso de não existirem posts
2	BDA.Facebook	CM (10)	Melhorar JavaDocs
3			
...	...	...	...

Corrective measures

Interface:
Email e twitter desnecessários
Barra no inicio da interface, não entendo a função
@username devia conter o username
Config.xml:
User e password são as do twitter, deveriam ser as do facebook

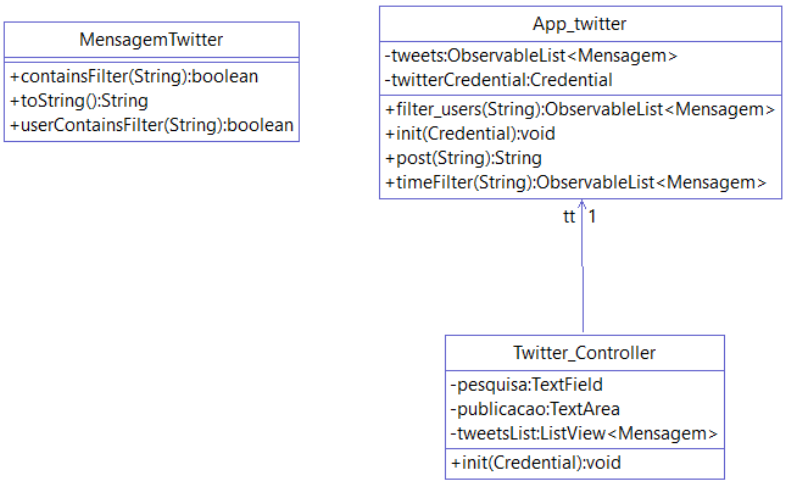
1. Cada vez que se vai buscar posts deveria de se verificar se há posts, ou seja, que o array não é nulo.
2. Alguns metodos, atribudos não estão devidamente comentados pelas regras do javadoc.

Conclusions of the inspection process

Apenas necessita de pequenas melhorias.

Code inspection – Twitter

Serviço do Twitter, onde é possível publicar tweets, pesquisar pelos mesmos, filtrando-os por tweets, utilizador e por periocidade (Últimas 24 horas e Último mês).



Meeting date:	*04/12/2018*
Meeting duration:	*90minutes*
Moderator:	*Joaquim Rocha*
Producer:	*Rita Tomás*
Inspector:	*Joaquim Rocha*
Recorder:	*Joaquim Rocha*
Component name (Package/Class/Method):	*BDA.Twitter/**
Component was compiled:	
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*24,9%*

Code inspection checklist

1. Variable, Attribute, and Constant Declaration Defects (VC)
   - ✓ Are descriptive variable and constant names used in accord with naming conventions?
   - ✗ Are there variables or attributes with confusingly similar names?
   - ✓ Is every variable and attribute correctly typed?
   - ✓ Is every variable and attribute properly initialized?
   - ✗ Could any non-local variables be made local?
   - ✓ Are all for-loop control variables declared in the loop header?
   - ✗ Are there literal constants that should be named constants?
   - ✗ Are there variables or attributes that should be constants?
   - ✗ Are there attributes that should be local variables?
   - ✓ Do all attributes have appropriate access modifiers (private, protected, public)?
   - ✗ Are there static attributes that should be non-static or vice-versa?
2. Method Definition Defects (FD)
   - ✓ Are descriptive method names used in accord with naming conventions?
   - ✓ Is every method parameter value checked before being used?
   - ✓ For every method: Does it return the correct value at every method return point?
   - ✓ Do all methods have appropriate access modifiers (private, protected, public)?
   - ✗ Are there static methods that should be non-static or vice-versa?
3. Class Definition Defects (CD)
   - ✓ Does each class have appropriate constructors and destructors?
   - ✗ Do any subclasses have common members that should be in the superclass?
   - ✗ Can the class inheritance hierarchy be simplified?

4. Data Reference Defects (DR)

❑ For every array reference: Is each subscript value within the defined bounds?

✗ For every object or array reference: Is the value certain to be non-null?

5. Computation/Numeric Defects (CN)

✗ Are there any computations with mixed data types?

✗ Is overflow or underflow possible during a computation?

✗ For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?

✓ Are parentheses used to avoid ambiguity?

6. Comparison/Relational Defects (CR)

✓ For every boolean test: Is the correct condition checked?

✓ Are the comparison operators correct?

✓ Has each boolean expression been simplified by driving negations inward?

✓ Is each boolean expression correct?

✗ Are there improper and unnoticed side-effects of a comparison?

✗ Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

7.Control Flow Defects (CF)

✓ For each loop: Is the best choice of looping constructs used?

✓ Will all loops terminate?

✓ When there are multiple exits from a loop, is each exit necessary and handled properly?

❑ Does each switch statement have a default case?

❑ Are missing switch case break statements correct and marked with a comment?

❑ Do named break statements send control to the right place?

✓ Is the nesting of loops and branches too deep, and is it correct?

✗ Can any nested if statements be converted into a switch statement?

✗ Are null bodied control structures correct and marked with braces or comments?

✓ Are all exceptions handled appropriately?

✓ Does every method terminate?

8.Input-Output Defects (IO) (não se aplica)

❑ Have all files been opened before use?

❑ Are the attributes of the input object consistent with the use of the file?

❑ Have all files been closed after use?

❑ Are there spelling or grammatical errors in any text printed or displayed?

❑ Are all I/O exceptions handled in a reasonable way?

9.Module Interface Defects (MI)

✓ Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?

✓ Do the values in units agree (e.g., inches versus yards)?

✓ If an object or array is passed, does it get changed, and changed correctly by the called method?

10.Comment Defects (CM)

✗ Does every method, class, and file have an appropriate header comment?

✗ Does every attribute, variable, and constant declaration have a comment?

✓ Is the underlying behavior of each method and class expressed in plain language?

✓ Is the header comment for each method and class consistent with the behavior of the method or class?

✓ Do the comments and code agree?

✓ Do the comments help in understanding the code?

✗ Are there enough comments in the code?

✗ Are there too many comments in the code?

11.Layout and Packaging Defects (LP)

- ✓ Is a standard indentation and layout format used consistently?
- ✓ For each method: Is it no more than about 60 lines long?
- ✓ For each compile module: Is no more than about 600 lines long?

12.Modularity Defects (MO)

- ✗ Is there a low level of coupling between modules (methods and classes)?
- ✓ Is there a high level of cohesion within each module (methods or class)?
- ✗ Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
- ✓ Are the Java class libraries used where and when appropriate?

13.Storage Usage Defects (SU)

- ❑ Are arrays large enough?
  - ✗ Are object and array references set to null once the object or array is no longer needed?

14.Performance Defects (PE)

- ✗ Can better data structures or more efficient algorithms be used?
- ✓ Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
- ✗ Can the cost of recomputing a value be reduced by computing it once and storing the results?
- ✓ Is every result that is computed and stored actually used?
- ✓ Can a computation be moved outside a loop?
- ✗ Are there tests within a loop that do not need to be done?
- ✗ Can a short loop be unrolled?
- ✗ Are there two loops operating on the same data that can be combined into one?
- ✗ Are frequently used variables declared register?
- ✓ Are short and commonly called methods declared inline?

Found defects

Identify and describe found defects, opinions and suggestions.

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	BDA.Twitter, App_twitter.java, filter_users(String), timeFilter(String), getCredential(), 175/202/267	CM(10)	Adicionar JavaDoc a estes métodos
2	BDA.Twitter, App_twitter.java, 36	CM(10)	Atributo não documentado (twitterCredential)
3	BDA.Twitter, MensagemTwitter.java	CM(10)	Header comment em falta
4	BDA.Twitter, Twitter_Controller.java	CM(10)	Header comment em falta
5	BDA.Twitter, MensagemTwitter.java, containsFilter(String)/userContainsFilter(String), 34/43	CM(10)	Adicionar JavaDoc a estes métodos
6	BDA.Twitter, Twitter_Controller.java, 44/73/79/84	CM(10)	Adicionar JavaDoc a estes métodos

Corrective measures

1) botões da barra superior ainda não funcionam (Opções e Ajuda);
2) comentar os métodos/atributos em falta (JavaDoc)

Conclusions of the inspection process

Necessita apenas de correções mínimas