Code Inspection Report

'Bom Dia Academia' Software Development Project

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

Group 8
77823, Diogo Faustino, EIC2
78400, Gonçalo Cruz, EIC2
, ,Gonçalo Duarte, EIC2
78208, Miguel Pedroso, EIC2

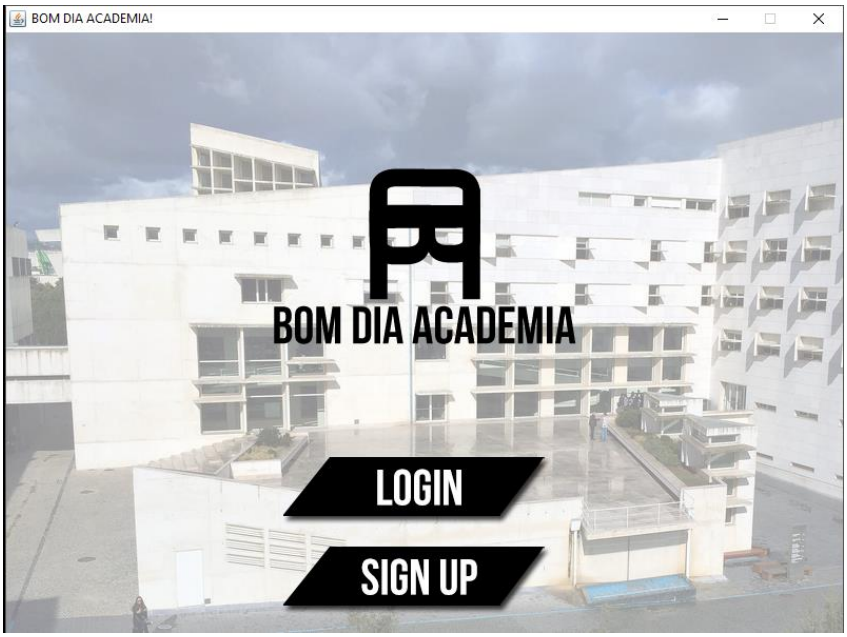
ISCTE-IUL, Instituto Universitário de Lisboa
1649-026 Lisbon
Portugal

November 2018

Table of Contents

Introduction

O projeto tem como todo nome BDA (Bom Dia Academia), e tem como objetivo permitir o acesso à informação académica disponibilizada através de vários meios informáticos, como Email, Facebook e Twitter. Para consultar o utilizador terá de criar uma conta e fazer login na aplicação. Para alem de poder consultar a informação o utilizador poderá também que o utilizador envie informação através desses meios.



Code inspection – Name of the component being inspected

Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*07/12/2017* *60 minutes* *Diogo Faustino* *Gonçalo Duarte* *Diogo Faustino, Gonçalo Cruz, Gonçalo Duarte,Miguel Pedroso* *Miguel Pedroso*
Component name (Package/Class/Method):	*BDA*
Component was compiled:	*yes*
Component was executed:	*yes*
Component was tested without errors:	*yes*
Testing coverage achieved:	97 %

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?
x	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 Java Inspection Checklist, Page 2

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)

x	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?
x	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?
x	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? Java Inspection Checklist, Page 3

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/Tweet/18	VC	Não tem encapsulamento apropriado, devia ser private
2	BDA/Tweet/22	VC	Não tem encapsulamento apropriado, devia ser private
3	BDA/Tweet/26	VC	Não tem encapsulamento apropriado, devia ser private
4	BDA/FacebookPost/10-13	CM	Não tem comentário header apropriado
5	BDA/FacebookPost/53	LP	Método com mais de 60 linhas
6	BDA/Facebook/38-41	CM	Não tem comentário header apropriado
7	BDA/ChangePass/56	CM	Não tem comentário header apropriado
8	BDA/ChangePass/64	CM	Não tem comentário header apropriado
9	BDA/ChangeMail/193	CM	Não tem comentário header apropriado
10	BDA/ChangeMail/201	CM	Não tem comentário header apropriado
11	BDA/ChangeMail/45	CM	Descrição da função do construtor mal feita
12	BDA/ChangeName/57	CM	Não tem comentário header apropriado
13	BDA/ChangeName/65	CM	Não tem comentário header apropriado
14	BDA/ChangeName/47	CM	Descrição da função do construtor mal feita
15	BDA/Home/25	VC	Não tem encapsulamento apropriado, devia ser private
16	BDA/Home/48	LP	Método com mais de 60 linhas

Corrective measures

– Diogo Faustino

1-Alterar o encapsulamento para private – Diogo Faustino
2-Alterar o encapsulamento para private – Diogo Faustino
3-Alterar o encapsulamento para private – Diogo Faustino

– Miguel Pedroso

4 - Refazer o comentario no header de modo a que fique mais compreensivel
5 - Eliminar codigo em excesso de modo a que o metodo fique com menos de 60 linhas
6 - Refazer o comentario no header de modo a que fique mais compreensivel

– Gonçalo Cruz

7 - Refazer o comentario no header de modo a que fique mais compreensivel
8 - Refazer o comentario no header de modo a que fique mais compreensivel
9 - Refazer o comentario no header de modo a que fique mais compreensivel
10 - Refazer o comentario no header de modo a que fique mais compreensivel
11 - Refazer o comentario no header de modo a que fique mais compreensivel
12 - Refazer o comentario no header de modo a que fique mais compreensivel
13 - Refazer o comentario no header de modo a que fique mais compreensivel

14 - Refazer o comentario no header de modo a que fique mais compreensivel

– Gonçalo Duarte

15 - Refazer o comentario no header de modo a que fique mais compreensivel
16 - Eliminar codigo em excesso de modo a que o metodo fique com menos de 60 linhas

Conclusions of the inspection process

O projeto foi verficado e o codigo foi inspecionado, acreditamos que o software está pronto a ser entregue ao cliente e apenas necessita de pequenas mudanças no código que não afetam de algo modo a performance ou alguma funcionalidade da aplicação.