Projeto

Na disciplina de Engenharia de Software será usada a aplicação Quizzes Tutor para os alunos responderem a perguntas de escolha múltipla sobre as matérias da unidade curricular.

Adicionalmente, no projeto da disciplina iremos enriquecer esta aplicação com 3 novas *features*. O projecto é open source disponível num repositório GitHub.

Para facilitar o desenvolvimento estão disponíveis vídeos sobre a aplicação, suas funcionalidades. modelos e código. Adicionalmente, todas as funcionalidades podem ser experimentadas usando as interfaces **DEMO** (Student, Teacher e Admin) disponibilizadas.

Grupos e Ramos

O projeto é realizado por grupos de 6 alunos, subdivididos em grupos de 2 alunos. Cada subgrupo de 2 alunos fica responsável por uma das funcionalidades.
Deve ser criado um ramo para cada subgrupo, em que o seu nome corresponde à funcionalidade (*feature*) de que o grupo é responsável. Assim, vão existir os seguintes ramos:

- *master* - onde são efetuadas as entregas finais do código integrado de cada uma das partes do projeto, a partir do código em *develop*
- *develop* - onde é integrado o código de cada um dos subgrupos, essa integração deve ser efetuada frequentemente, com a granularidade de história, seguindo uma política de *pull request*, em que o código de um subgrupo deve ser obrigatoriamente revisto e aceite por pelo menos um elemento de cada um dos outros dois subgrupos
- *ere*, *prd* e *pri*, onde se desenvolve, respetivamente, as funcionalidades de Estatísticas de Respostas, Perguntas de Resposta Difícil, Perguntas de Resposta Incorreta. A colaboração nestes ramos de sub-grupo deve seguir a seguinte estratégia:
    -
    - os dois elementos do subgrupo seguem uma política de integração contínua com granularidade de tarefa
    - para cada tarefa terminada deve ser feito um commit, pelo aluno que implementou a tarefa
    - os elementos podem dividir as tarefas de uma história ou, em alternativa, cada um deles implementa todas as tarefas de uma história
    - quando o sub-grupo decide dividir as tarefas de uma história, **cada aluno deve implementar tarefas de tipo diverso**, por exemplo, não pode implementar apenas tarefas de teste.
    - quando uma história é terminada deve ser submetido um *pull request* ao *develop* que tem de ser aprovado por um aluno de cada um dos outros subgrupos

Entregas

O projeto possui 3 partes:

1. (30%) Domínio, testes de serviços e JPA - entrega a **18 de março de 2022 às 17:00**
2. (30%) Lógica de negócio de Serviçoa, Testes de serviços, Spring-boot e Web Services - entrega a **1 de abril de 2022 às 17:00**
3. (40%) JMeter, VueJs e Testes End-to-end - entrega a **22 de abril de 2022 às 17:00**

As entregas são efetuadas no GitHub usando *tags* da forma *Feature-Parte* e *Parte*

- As *tags Feature-Parte* são colocados nos ramos de *feature*. Por exemplo, *pri-p1* corresponde à primeira entrega da *feature* de Perguntas de Resposta Incorreta no ramo pri
  -
  - git checkout pri
  - git push origin
  - git tag -a pri-p1 -m "first delivery pri"
  - git push origin pri-p1
- As *tags Parte* são colocadas no ramo *master* depois de ter integrado o ramo *develop*. Por exemplo, p2 corresponde à entrega, relativa à parte 2, da integração das 3 *features* no ramo *develop*
  -
  - git checkout master
  - git merge develop
  - git push origin
  - git tag -a p1 -m "first delivery"
  - git push origin p1

Funcionalidades

As funcionalidades são apresentadas de seguida. **Em cada aula de laboratório são descritas as tarefas necessárias para resolver o projeto.**

Student Dashboard

Cada aluno possui um dashboard associado a cada uma das suas disciplinas execução. Esse dashboard fornece informação ao aluno sobre 3 elementos: as perguntas que respondeu incorretamente, os resultados obtidos semanalmente e as perguntas mais difíceis. O aluno pode pedir explicitamente para atualizar a informação de um particular elemento do dashboard para incluir os novos dados até ao momento atual.

Perguntas com Resposta Incorreta (PRI)

O aluno pode visualizar no seu dashboard as perguntas que respondeu incorretamente. Por resposta incorreta entende-se aquela que o aluno errou ou não respondeu num quiz que já esteja terminado. É necessário distinguir explicitamente as duas situações: não respondeu e respondeu errado. O aluno pode remover uma resposta incorreta do conjunto de respostas incorretas, não sendo esta resposta nunca mais adicionada ao conjunto, a não ser que o aluno peça explicitamente para adicionar as suas respostas

incorretas num certo período. Uma resposta incorreta apenas pode ser removida 5 dias após ter sido adicionada ao conjunto de respostas incorretas.

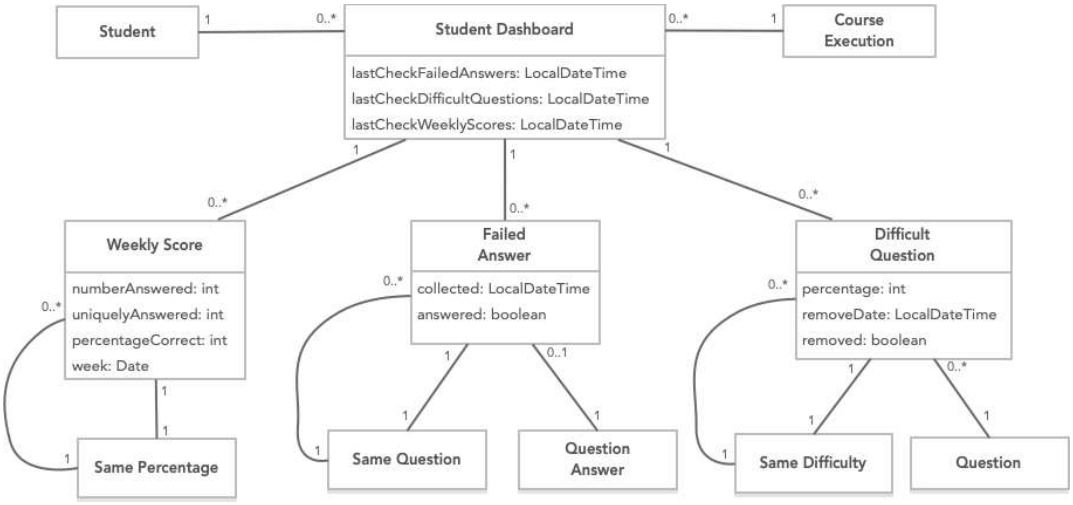
Perguntas com Resposta Difícil (PRD)

O aluno pode visualizar no seu dashboard as perguntas que foram respondidas na última semana (a semana é calculada como os 7 dias anteriores à data atual) e possuem um grau de dificuldade, ou seja, menos de 25% das respostas dadas são corretas. O aluno pode remover uma pergunta difícil do conjunto de perguntas difíceis, sendo ela adicionada de novo apenas se, passada uma semana após a data da remoção, a pergunta voltar a ter uma taxa de resposta correta menor de 25%. Apenas podem surgir perguntas que estão no estado "removed == false".

Estatísticas de Respostas (ERE)

O aluno pode visualizar um conjunto de estatísticas sobre as suas respostas na última semana (as semanas são calculadas com início no domingo): número total de perguntas respondidas, número total de perguntas respondidas sem repetição, percentagem de respostas corretas. Estas estatísticas aplicam-se a quiz answers que já podem ser públicas. Adicionalmente, o dashboard apresenta a mesma informação para cada uma das semanas em que o aluno respondeu a perguntas, de domingo a sábado. Esta informação é persistente e não necessita de ser recalculada. Apenas sendo recalculada a informação relativa à semana atual, sempre que solicitado. O aluno pode remover as estatísticas de uma dada semana, mas não a atual, não sendo possível repor essa informação.

Modelo de Domínio



Onde *SameCONDITION* deve ser interpretada como associando a instância a que está ligada pela relação de 1 para 1 com N instâncias que possuem a mesma condição (exclui a própria). A condição de *SamePercentage* são os *WeeklyScore* que possuem a mesma *percentageCorrect*. A condição *SameQuestion* são as *FailedAnswer* cuja resposta é da mesma

pergunta. A condição *SameDifficulty* são as *DifficultyQuestion* que possuem a mesma dificuldade, *percentage*.

Primeira Parte

Objetivo

Implementar e testar os serviços de criar e remover para cada uma das funcionalidades. Implica também implementar as classes de domínio persistentes. Pode ser usado o código disponibilizado, acrescentando as classes de domínio em falta e estendendo os testes fornecidos por forma a alcançar uma cobertura de 100%. A inclusão do código fornecido deve ser efetuada de acordo com a divisão em tarefas associada ao desenvolvimento de cada uma das funcionalidades, ou seja, deve ser incluído em commits separados. Para os testes funcionarem pode ser necessário acrescentar algumas instruções em algumas das superclasses ou em classes de configuração, e.g. SpockTest.groovy e BeanConfiguration.groovy.

Perguntas com Resposta Incorreta (PRI)

Devem ser respeitadas as interfaces das classes de domínio e serviço.**Domínio:**

```
public FailedAnswer(Dashboard dashboard, QuestionAnswer
questionAnswer, LocalDateTime collected) {...
```

Serviço:

```
public FailedAnswerDto createFailedAnswer(int dashboardId, int
questionAnswerId) {...
public void removeFailedAnswer(int failedAnswerId) {...
```

Perguntas com Resposta Difícil (PRD)

Devem ser respeitadas as interfaces das classes de domínio e serviço.**Domínio:**

```
public DifficultQuestion(Dashboard dashboard, Question question, int
percentage) {...
```

Serviço:

```
public DifficultQuestionDto createDifficultQuestions(int dashboardId,
int questionId, int percentage) {...
public void removeDifficultQuestion(int difficultQuestionId) {...
```

Estatísticas de Respostas (ERE)

Devem ser respeitadas as interfaces das classes de domínio e serviço.**Domínio:**

```
public WeeklyScore(Dashboard dashboard, LocalDate week) {...
```

Serviço:

```
public WeeklyScoreDto createWeeklyScore(Integer dashboardId) {...
public void removeWeeklyScore(Integer weeklyScoreId) {...
```

Sugestões de Planeamento

1. Ler com atenção o enunciado
2. Analisar o modelo de domínio

3. Analisar o código fornecido como suporte
4.
    1. Domínio
    2. Serviços
5. Analisar o código de partida
6.
    1. Domínio
    2. Serviços
7. Listar as diferenças entre as duas bases de código e o objetivo final do projeto
8.
    1. Entidades de domínio que necessitam de ser implementas
    2. Testes que necessitam de ser estendidos
    3. Novos testes que necessitam de ser implementados
9. Planear
10.
    1. Definir as histórias (*create* e *remove*) para cada uma das funcionalidades
    2. Decompor as histórias em tarefas de modo a haver um desenvolvimento incremental que permita validação constante. Para cada história definir tarefas para (cada tarefa estará associada a um *issue*, um elemento do subgrupo e um *commit* a realizar por esse elemento):
    3.
        1. Aplicar o domínio definido no código fornecido
        2. Aplicar o serviço definidos no código fornecido
        3. Aplicar os testes definidos no código fornecido, correr os testes
        4. Definir uma tarefa para implementar as entidades de domínio (idealmente também poderá se efetuado de forma incremental, primeiro a classe *Same...* e depois a sua relação como a lista de classes que contém, isto aplica-se também às seguintes tarefas)
        5. Definir uma tarefa para cada teste a estender
        6. Definir uma tarefa para cada novo teste

Dado que parte da história será implementada com o código fornecido, pode ser feito *pull-request* quando essa parte já estiver a funcionar (10.2.1-3).

Critérios de Avaliação

Parte1

Entrega

- Ficheiro markdown de acordo com a matriz

Sprint 1	Total
Totals for group and subgroup grades	**20.00**
Actual group grade [max 4] and subgroup grade [max 14]	
[subgroup] Project management / Github	**2.00**
Github project with subgroup sprint stories and tasks	1.00
Subgroup github issues created for tasks	1.00
[subgroup] Feature Development	**12.00**
Create Test Cases	1.50
Create Implemented Tests	1.50
Create Service Implementation	1.00
Remove Test Cases	1.50
Remove Implemented Tests	1.50
Remove Service Implementation	1.00
Object-Oriented Quality	1.50
Domain Model Implemented in JPA	0.50
Coverage of the tests on the new code	1.00
Feature branch compiles	0.50
All service tests run	0.50
[group] Code integration and Group Work	**4.00**
Compilation (master branch): no errors	0.50
Run tests (master branch): no errors or failures	0.50
Markdown file with submission summary	1.00
Most recent versions of features integrated into develop branch	1.00
Quality of approved pull-requests	1.00
[subgroup] Bonus & Penalties	
Penalty: Garbage (unnecessary files, println, non-english)	
[individual] Individual grades	**2.00**
Commits/Issues (1 per task, quality of messages: professional language, consistent, issues referenced) [max = 0.5]	0.50
Sprint involvement (appropriate amount of work) [max=1.5]	1.50

Segunda Parte

Objetivo

Desenvolvimento de serviços de acordo com a abordagem *Test First*, implementação e teste de serviços REST.

São fornecidos os testes de serviços, que devem ser copiados para o projeto e **não podem ser alterados**. Este testes de serviço servem como especificação da funcionalidade a implementar. São também fornecidos os casos de testes de serviços REST para completar. São ainda indicadas algumas alterações ao domínio, a efetuar conforme os testes forem sendo integrados no projeto.

É necessário implementar, para cada uma das três funcionalidades, os serviços REST *get*, *update* e *delete*, que correspondem aos respetivos serviços referidos nos testes de serviço fornecidos (em que *delete* corresponde a *remove* nos serviços). Os dois primeiros serviços, *get* e *update*, recebem o identificador do *dashboard*, o terceiro, *delete*, recebe o identificador da entidade a apagar. Para cada um dos serviços REST devem ser implementados três casos de testes: (1) sucesso, retorna 200, em que deve ser verificado se o resultado possui os dados corretos, erro de acesso, código 403, (2) quando um docente tenta invocar o serviço, ou (3) um outro aluno inscrito na mesma disciplina execução tenta invocar o serviço.

Neste segunda parte a consistência do domínio é verificada sem considerar as classes SAME. No entanto esse código não deve ser removido, ou seja, os atuais testes que verificam essa parte do domínio devem continuar a executar com sucesso.

Perguntas com Resposta Incorreta (PRI)

Devem ser efetuadas as seguintes alterações (mas elas devem ser efetuadas apenas conforme se for introduzindo os testes de serviço):

- O atributo *lastCheckFailedAnswers* de *Dashboard* é inicializado a *null*, quando o *dashboard* é criado.

Perguntas com Resposta Difícil (PRD)

Devem ser efetuadas as seguintes alterações (mas elas devem ser efetuadas apenas conforme se for introduzindo os testes de serviço):

- O atributo *lastCheckDifficultQuestions* de *Dashboard* é inicializado a *null*, quando o *dashboard* é criado.
- A lógica de negócio do serviço de remoção de pergunta difícil é alterado de acordo com os testes de serviço fornecidos. Por exemplo, apenas se aplica às perguntas respondidas em *quizzes* cujo *resultDate* é nos últimos 7 dias, e entidade não é apagada, mas apenas marcada como para apagar.

Estatísticas de Respostas (ERE)

Devem ser efetuadas as seguintes alterações (mas elas devem ser efetuadas apenas conforme se for introduzindo os testes de serviço):

- O atributo *lastCheckWeeklyScores* de *Dashboard* é inicializado a *null*, quando o *dashboard* é criado.
- Na classe *WeeklyScore* deve ser adicionado o atributo *boolean closed*, e os métodos (fornecidos) *computeStatistics()*, *getWeeklyQuizAnswers()*, *isAnswerWithinWeek(QuizAnswer quizAnswer)* e *setClosed(boolean closed)*. O atributo *closed* indica se todas as respostas a perguntas no período já se encontram públicas.

Sugestões de Planeamento

Cada subgrupo:

1. Para cada um dos testes de serviços fornecidos (será uma história cada um deles, e uma tarefa para cada caso de teste), e por esta ordem, *create*, *remove*, *get* e *update*:
2.
    1. Copiar o teste de serviço fornecido para o projeto.
    2. Correr o teste e alterar o restante código, para assegurar que o teste passa.
3. Para cada um dos serviços REST (será uma história para cada um deles, e uma tarefa para cada caso de teste), e por esta ordem, *get*, *delete*, *update*:
4.
    1. Implementar o serviço REST.
    2. Testar o serviço REST.

Critérios de Avaliação

Entrega

- Ficheiro markdown de acordo com a matriz

Sprint 2	Total
Totals for group and subgroup grades	**20.00**
Actual group grade [max 4] and subgroup grade [max 14]	
[subgroup] Project management / Github	**1.00**
Github project with subgroup sprint stories and tasks	0.50
Subgroup github issues created for tasks	0.50
[subgroup] Test First Service Development	**7.5**
1. create service	0.5
2. get service	2.5
3. update service	4
4. remove service	0.5
[subgroup] REST Service Implementation and Test	**4.50**
1. get REST service	0.5
2. update REST service	0.5
3. delete REST service	0.5
1. get REST service test	1
2. update REST service test	1
3. delete REST service test	1
[subgroup] Feature branches	**1.00**
Compilation (feature branches): no errors	0.50
Run tests (feature branches): no errors or failures	0.50
[group] Code integration and Group Work	**4.00**
Compilation (master branch): no errors	0.50
Run tests (master branch): no errors or failures	0.50
Markdown file with submission summary	1.00
Most recent versions of features integrated into develop branch	1.00
Quality of approved pull-requests	1.00
[subgroup] Penalties	
Penalty: Garbage (unnecessary files, comments, println, non-english)	
[individual] Individual grades	**2.00**
Commits (1 per task/issue, quality of messages: professional language, consistent, issues referenced) [max = 0.5]	0.50
Sprint involvement (appropriate amount of work) [max=1.5]	1.50

Terceira Parte

Objetivo

Testes de regressão para facilitar a alteração de código, testes de carga, implementação de apresentação *client-side*, teste *end-to-end*.

Alterações ao *Backend*

Remover as classes *SamePercentage*, *SameDifficulty* e *SameQuestion*. Alterar código e testes de forma a que todos os testes passem.

Testes de Carga

Os testes de carga em JMeter devem estender o *script* disponibilizado. Para cada funcionalidade devem ser implementados dois testes: (1) vários estudantes respondem a um *quiz* e de seguida acedem ao seu *dashbord* de funcionalidade específica efetuando um *update* seguido de um *get*; (2) (OPCIONAL/BÓNUS) cada estudante responde a vários *quizzes*, em que os *quizzes* podem partilhar perguntas, sendo pré-definido em variáveis o número de estudantes, de perguntas, de *quizzes* que cada estudante responde e de perguntas por *quiz*, devendo a atribuição de perguntas a *quizzes* ser feita de forma aleatória, assim como os *quizzes* que cada aluno responde. Para implementar (1) apenas é necessário estender o *script* fornecido com um novo *thread group*. Para implementar (2) é necessário alterar o *script* fornecido por forma a considerar a definição de vários quizzes.

Implementação do Frontend

Implementar o frontend de cada uma das funcionalidades. Sugere-se a estrutura apresentada na parte final dos seguintes vídeos: WeeklyScores, DifficultQuestions, FailedAnswers (visualizar a 1/4 da velocidade). Para visualizar a pergunta, nos dois últimos casos, usar o componente *StudentViewDialog* como é usado em *QuestionsView*.
Usar o componente Vuetify Data Table, e sugere-se seguir a estrutura usada em *QuestionsView.vue*. Os três componentes a desenvolver serão subcomponentes de *DashboardView.vue* e devem ser declarados da seguinte forma:

```
<weekly-scores-view :dashboardId="dashboardId" v-
on:refresh="onWeeklyScoresRefresh">
</weekly-scores-view>
<failed-answers-view :dashboardId="dashboardId"
:lastCheckFailedAnswers="lastCheckFailedAnswers" v-
on:refresh="onFailedAnswersRefresh">
</failed-answers-view>
<difficult-questions-view :dashboardId="dashboardId" v-
on:refresh="onDifficultQuestionsRefresh">
</difficult-questions-view>
```

A atualização das datas (*lastCheck*) no dashboard deve ser feito através da emissão de um evento pelos seus subcomponentes (*v-on:refresh*).

Pode ser necessário efetuar alterações ao *backend*, conforme forem sendo identificadas, aquando da implementação do *frontend*. Por exemplo pode ser necessário considerar *QuizAnswers* apenas se estiverem no estado *completed*, na lógica da funcionalidade *Difficult Question*.

Testes *End-to-End*

Efetuar um teste *end-to-end* para cada uma das funcionalidades, de acordo com a sequência dos seguintes vídeos: *WeeklyScores*, *DifficultQuestions*, *FailedAnswers* (visualizar a 1/4 da velocidade).
A sequência tem três parte: (1) o *demo teacher* faz *login* e cria 2 perguntas e um *quiz*; (2) o *demo student* responde ao *quiz*; (3) o *demo student* acede ao *dashboard*. Para (1) e (2) sugere-se copiar, adaptar e estender o *script* de teste *discussion.js*. Na parte (3) o *demo student* acede ao *dashboard* específico da funcionalidade, faz *refresh* para obter os dados mais recentes, e apaga as entidades. Nos casos de *difficult question* e *failed answer* também visualiza a pergunta. No caso de *failed answer*, ao tentar apagar a primeira vez ocorre um erro e a

mensagem de erro é fechada com *cy.closeErrorMessage()*. No caso *weekly scores* a mensagem de erro também tem de ser fechada quando se tenta apagar o *weekly score* da semana corrente.
Os seguintes *scripts* devem ser adicionados a *database.js*. Cada um dos testes de funcionalidade deve incluir:

```
beforeEach(() => {
  cy.deleteQuestionsAndAnswers();
  ...
}
afterEach(() => {
  cy.deleteFailedAnswers();
  cy.deleteQuestionsAndAnswers();
});
```

Adicionalmente, no teste *failedAnswers.js* deve ser invocado *cy.setFailedAnswersAsOld()* após se verificar que não é possível apagar uma *failed answer* com menos de 5 dias, para se alterar a sua data e depois testar que quando têm mais de 5 dias podem ser apagadas. Também no teste *weeklyScores.js* deve ser invocado *cy.createWeeklyScore()* após aceder ao *dashboard*, para ser adicionado à base de dados um *weekly score* relativo a uma semana anterior, e assim poder ser apagado.

Os scripts podem ser executados isoladamente.

Sugestões de Planeamento

1. Definir uma história, de uma única tarefa, para cada uma das remoções da classes *Same*.
2. Definir uma história para cada um dos testes de carga. As tarefas estão associadas às invocações de serviços web.
3. Definir uma história para cada funcionalidade, em que haverá tarefas para cada uma das invocações remotas ao *backend*, de definição do modelo, e para cada umas das ações da vista (*get*, *refresh* e *delete*).
4. Definir uma história para o teste *end-to-end* de cada uma das funcionalidades. Há pelo menos duas tarefas, na primeira é adaptado o script sugerido, e na segunda implementado o restante teste.

Critérios de Avaliação

Entrega

- Ficheiro markdown de acordo com a matriz

Sprint 3	Total
Totals for group and subgroup grades (INCLUDES BONUS)	**22**
Actual group grade [max 4] and subgroup grade [max 14]	
[subgroup] Project management / Github	**1**
Github project with subgroup sprint stories and tasks	0.5
Subgroup github issues created for tasks	0.5
[subgroup] Remove Same Domain Entity	**1**
Remove entity, clean code and tests	1
[subgroup] JMeter load test	**4**
Basic sequence	2
Multiple questions and quizzes (BONUS)	2
[subgroup] Frontend (Vue)	**7**
Get action	2
Refresh action	2
Delete action	2
Quality and modularity of components	1
[subgroup] End-to-end tests (Cypress)	**2.5**
Adapt given script	0.5
Get action	0.5
Refresh action	1
Delete action	0.5
[subgroup] Feature branches	**0.5**
Compilation (feature branches): no errors	0.25
Run tests (feature branches): no errors or failures	0.25
[group] Code integration and Group Work	**4**
Compilation (master branch): no errors	0.5
Run tests (master branch): no errors or failures	0.5
Markdown file with submission summary	1
Most recent versions of features integrated into develop branch	1
Quality of approved pull-requests	1
[subgroup] Penalties	
Penalty: Garbage (unnecessary files, comments, println, non-english)	
[individual] Individual grades	**2**
Commits (1 per task/issue, quality of messages: professional language, consistent, issues referenced) [max = 0.5]	0.5
Sprint involvement (appropriate amount of work) [max=1.5]	1.5

Avaliação

A avaliação tem três componentes:

- Cada aluno aluno é avaliado pelo seu contributo no contexto do subgrupo
- Cada subgrupo é avaliado sobre a *feature* que lhe está atribuída
- O grupo é avaliado sobre a integração das *features* no repositório global

O *feedback* qualitativo da avaliação é dado num ramo de correção pelo docente, *pN-evaluation*, em que *N* é a parte do projeto. Em alguns casos, quando a integração não contem todo o código, poderá ser criado um ramo de subgrupo, por exemplo, *prdN-evaluation*, que corresponde à parte *N* da avaliação das Perguntas de Resposta Difícil.

É obrigatório entregar um ficheiro markdown com um resumo da entrega de acordo com as matrizes disponibilizadas.