UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO

GRUPO 7

LUÍZA RUBIO - 10786121

LUCAS GABRIEL MENDES MIRANDA - 10265892

LEONARDO FONSECA PINHEIRO - 11219241

CESAR HENRIQUE DE ARAUJO GUIBO - 11218705

LUCA FARIA CURCIO - 10295502

TRABALHO DE SSC0124 – PARTE 2

SÃO CARLOS – SP

2020

SUMÁRIO

1 INTRODUÇÃO

A segunda parte do projeto focou na revisão dos artefatos da parte 1 do projeto, bem como na finalização da modelagem do sistema de software e na geração do código fonte do sistema utilizando os modelos desenvolvidos.

2 APRESENTAÇÃO GERAL DO SISTEMA

2.1 ALTERAÇÕES EM RELAÇÃO À APRESENTAÇÃO DA PARTE 1

O grupo não recebeu nenhum feedback negativo em relação à apresentação geral do sistema contida na parte 1. Por esse motivo, decidimos mantê-la inalterada.

2.2 SOBRE A APLICAÇÃO

Com a pandemia de COVID-19, ficou ainda mais evidente a necessidade de aplicações capazes de auxiliar educadores e alunos a manterem suas atividades educacionais à distância. Nesse documento, e em seus anexos, é descrito o funcionamento de uma dessas aplicações. A *Ensinet*, como é chamada, é uma aplicação web que tem como objetivo disponibilizar conteúdo educacional produzido por um educador. Para melhor detalhar o seu funcionamento, consideremos os dois principais tipos de usuário do sistema: *estudantes* e *educadores*.

2.3 ESTUDANTES

Carlos é um estudante universitário de 20 anos de idade que está sempre interessado em aprender coisas novas. Apesar de estar matriculado no curso de Ciências da Computação, ele também gosta de estudar outros assuntos, como História e Sociologia. Com a pandemia de COVID-19, foi forçado a continuar seus estudos a partir de casa. Por esse motivo, surgiu a necessidade de acompanhar as disciplinas que estava cursando (e as aulas dos outros assuntos que ele tem interesse) através de uma plataforma online. Dessa forma, ele pode se cadastrar na Ensinet como um *estudante.*

Estudantes são os usuários que consomem o conteúdo educacional disponível no site. Um estudante pode:

- Se matricular em disciplinas públicas ou privadas através de um código disponibilizado pelo educador responsável;
- Visualizar e consumir o conteúdo de disciplinas públicas ou de disciplinas privadas em que estão matriculados.

Nota-se que, quando um estudante está matriculado em uma disciplina, ele tem acesso a conteúdo adicional, que consiste de provas e trabalhos. Além disso,

estudantes matriculados recebem uma pontuação de 0 a 10, atribuída pelo educador, que corresponde ao seu rendimento na disciplina em questão.

2.4 EDUCADORES

Isabela é uma professora que precisa disponibilizar suas aulas em uma plataforma online. Ela precisa publicar conteúdo gravado, transmitir aulas ao vivo e organizar trabalhos e provas para a avaliação de seus alunos. Assim como Carlos, ela também pode se cadastrar na *Ensinet*, mas como uma educadora.

Educadores são usuários, que além de possuírem as funcionalidades de alunos, são responsáveis por distribuir conteúdo educacional através da plataforma. Tal conteúdo pode consistir de:

- Videoaulas gravadas;
- Videoaulas ao vivo;
- Trabalhos;
- Provas.

O conteúdo disponibilizado por um educador deve ser agrupado em disciplinas. Por exemplo: se um educador deseja publicar uma vídeoaula sobre autovetores, ele deve associá-la à disciplina de Álgebra Linear, que deve ter sido criada por ele anteriormente. Um educador deve poder criar várias disciplinas e escolher se cada uma delas será pública ou privada. Disciplinas públicas estarão disponíveis a todos os usuários do site, enquanto que disciplinas privadas estarão disponíveis apenas para usuários matriculados nela.

2.5 INFORMAÇÕES ADICIONAIS

Considerando o cenário de uso apresentado, pode-se sintetizar o sistema como uma plataforma semelhante à Netflix, com as áreas do conhecimento (como Geografia, Biologia e Matemática) sendo equivalentes aos gêneros de filmes. Ou seja, uma plataforma que junta videoaulas, aulas de dúvida ou qualquer outro tipo de vídeo educacional da matéria e disponibiliza para o usuário de forma rápida e de fácil acesso.

Os vídeos serão armazenados na nuvem, sendo esse armazenamento feito com o auxílio de serviços como o *Amazon Web Services*, por exemplo. A transmissão deles será feita por *streaming*, inicialmente sem legendas, sendo possível realizar downloads.

3 DIAGRAMA DE CASOS DE USO

Também não recebemos nenhum feedback negativo em relação ao diagrama de casos de uso apresentado na parte anterior. Porém, foram feitas quatro modificações em relação à versão anterior do diagrama:

- O caso de uso “tornar-se educador” foi adicionado, pois notamos que, na versão anterior, não existia nenhuma maneira de registrar educadores no sistema. Sendo assim, definimos agora que *Estudantes* podem escolher se tornarem *Educadores*;
- Existiam alguns casos de uso que englobavam duas ações em apenas um caso, como “disponibilizar/remover prova”. Esses casos de uso foram divididos em dois novos na nova versão. Por exemplo: “disponibilizar/remover prova” foi dividido em “disponibilizar prova” e “remover prova”;
- O caso de uso “fazer login” foi adicionado;
- O caso de uso “escolher se uma disciplina deve ser pública ou privada” foi removido, pois entendemos que o caso de uso “criar uma disciplina” já deve incluir essa opção.

A imagem com o modelo de casos de uso se encontra no diretório *./casos_de_uso/casos_de_uso.png.*

4 MODELO CONCEITUAL DO SISTEMA

O modelo conceitual foi alterado na segunda parte do projeto. A alteração se deu porque os conceitos devem ser pensados como substantivos, o que não foi feito na parte anterior. A imagem com o diagrama se encontra no diretório *./modelo_conceitual/modelo_conceitual.png.*

5 DIAGRAMAS DE INTERAÇÃO

Os diagramas de interação foram feitos com base nos últimos exercícios realizados no semestre e no *feedback* que recebemos. Os casos de uso que representamos nesses diagramas foram:

- “Realizar cadastro”;
- “Fazer matrícula”;
- “Assistir aula”;
- “Disponibilizar aula”.

Para os dois primeiros, utilizamos diagramas de sequência, e, para os dois últimos, diagramas de comunicação. Abaixo, temos uma breve discussão sobre as mensagens de cada diagrama, para facilitar o entendimento delas.

5.1 DIAGRAMAS DE SEQUÊNCIA

5.1.1 Caso de uso “realizar cadastro”

O diagrama em questão se encontra no diretório *./diagramas_interacao/diagramas_sequencia/realizar_cadastro.png.* Nele, temos cinco mensagens representadas:

- **1. *realizarCadastro(nome, email, senha)*:** trata-se da mensagem principal do caso de uso. O ator *Visitante* envia ela para o objeto *user,* que representa esse ator. Ele faz isso para se cadastrar no sistema, por isso o seu nome, email e senha são passados como parâmetro. Retorna um inteiro, que deve ser um código de erro ou sucesso, a depender de como a operação terminou;
- **1.1. *validarNome(nome)*:** mensagem responsável por validar o nome de usuário. Ele deve ser validado checando na base de dados se um usuário com o mesmo nome já existe, ou não. Em caso positivo, o nome é inválido. Retorna *True* se *nome* é válido, e *False*, caso contrário;
- **1.2. *validarEmail(email)*:** mensagem responsável por validar o email do usuário. Ele deve ser validado checando se a string enviada pelo *Visitante* corresponde a um endereço de email válido. Em caso negativo, o email é inválido. Retorna *True* se *email* é válido, e *False*, caso contrário;
- **1.3. *validarSenha(senha)*:** mensagem responsável por validar a senha do usuário. Ela deve ser validada checando se a string enviada pelo *Visitante* possui mais de seis caracteres. Em caso negativo, a senha é inválida. Retorna *True* se *senha* é válida, e *False*, caso contrário;
- **1.4. *inserirUsuarioNaBD(nome, email, senha)*:** mensagem responsável por criar um novo usuário com os dados correspondentes na base de dados. Retorna *True*, se a operação na base de dados foi terminada com sucesso, e *False*, caso contrário.

5.1.2 Caso de uso “fazer matrícula”

O diagrama em questão se encontra no diretório *./diagramas_interacao/diagramas_sequencia/fazer_matricula.png.* Nele, temos três mensagens representadas:

- **1. *fazerMatricula(idDisciplina)*:** trata-se da mensagem principal do caso de uso. O ator *Estudante* envia ela para o objeto *user,* que representa esse ator. Ele faz isso para se matricular em uma disciplina, por isso o id da disciplina em questão é passado como parâmetro. Retorna um inteiro, que deve ser um código de erro ou sucesso, a depender de como a operação terminou;
- **1.1 *existeDisciplina(idDisciplina)*:** antes de efetivar a matrícula no banco de dados, é necessário verificar se a disciplina em que o *Estudante* deseja se

matricular existe. Essa mensagem é responsável por isso. Retorna *True* se existe uma disciplina com o id *idDisciplina,* e *False*, caso contrário;
- **1.2 *inserirDisciplinaMatriculadaNaBd(idDisciplina)*:** responsável por efetivar a matrícula do *Estudante* na base de dados. Retorna *True* se a operação na base de dados for bem-sucedida, e *False*, caso contrário.

5.2 DIAGRAMAS DE COMUNICAÇÃO

5.2.1 Caso de uso "assistir aula"

O diagrama em questão se encontra no diretório *./diagramas_interacao/diagramas_comunicacao/assistir_aula.png.* Nele, temos dez mensagens representadas:

- **1. *assistirAula(aula)*:** trata-se da mensagem principal. É responsável por desencadear a cadeia de eventos que leva a exibição do vídeo correspondente a uma aula. O seu único parâmetro, *aula*, corresponde a um objeto da classe Aula, e é aula a ser assistida;
- **1.1. *getIdAula()*:** responsável por retornar o id de um objeto da classe Aula;
- **1.2. *getIdDisciplina()*:** responsável por retornar o id da disciplina da qual um objeto da classe Aula faz parte;
- **1.3. *existeDisciplina(idDisciplina):*** retorna *True* se uma disciplina com id *idDisciplina* existe, e *False*, caso contrário;
- **1.4. *existeAula(idAula)*:** retorna *True* se uma aula com id *idAula* existe, e *False*, caso contrário;
- **1.5. *estaCadastrado()*:** retorna *True* se o objeto da classe *Visitante* está cadastrado no sistema. Em outras palavras, retorna *True* se o objeto em questão for uma instância da classe *Estudante*, e *False*, caso contrário;
- **1.6. *verificaMatriculado(idDisciplina):*** retorna *True* se o objeto da classe *Estudante* está matriculado na disciplina de id *idDisciplina,* e *False,* caso contrário;
- **1.7. *verificaSePrivada()*:** retorna *True* se o objeto da classe *Aula* é privada, e *False*, caso contrário;
- **1.8. *exibirVideo(video):*** exibe o video de um objeto da classe Aula;
- **1.9. *exibirMsgErro(msg)*:** exibe uma mensagem de erro determinada pela *String msg*.

5.2.2 Caso de uso "disponibilizar aula"

O diagrama em questão se encontra no diretório *./diagramas_interacao/diagramas_comunicacao/disponibilizar_aula.png.* Nele, temos quatro mensagens representadas:

- **1. *disponibilizarAula(nomeDaAula, descricao, video, idDisciplina):*** trata-se da mensagem principal do caso de uso. É responsável por disponibilizar uma aula, por isso, tem como parâmetros os dados a respeito dessa aula. Um dos parâmetros é o *idDisciplina*, que corresponde ao id da disciplina da qual essa aula faz parte;
- **1.1. *existeDisciplina(idDisciplina):*** antes de criar uma nova aula, é preciso verificar se a disciplina da qual a aula faz parte existe. Essa mensagem é responsável por isso. Retorna *True* se a disciplina em questão existe, e *False,* caso contrário;
- **1.2. *verificaFormatoVideo(video)*:** antes de disponibilizar uma aula, é preciso verificar se o arquivo de vídeo passado como parâmetro é válido. Essa mensagem é responsável por fazer isso. Retorna *True* se o arquivo passado como parâmetro é um arquivo de vídeo válido, e *False,* caso contrário;
- **1.3. *criar(nomeDaAula, descricao, video, idDisciplina, idEducador):*** é responsável por criar a nova aula. Pode ser interpretada como um construtor da classe Aula.

6 DIAGRAMA DE CLASSES

O diagrama em questão se encontra no seguinte diretório: *./diagrama_classes/diagrama_classes.png.* É importante ressaltar alguns pontos a seu respeito:

- Seguindo a notação UML, usamos sublinhado para indicar métodos estáticos. Por exemplo: no diagrama, o método *validarEmail* está sublinhado*,* por isso, ele representa um método estático;
- Para casos em que uma classe utiliza um método estático de outra classe, consideramos que existe uma dependência “usa” entre as duas classes. Por exemplo: pelo seu diagrama de sequência, observa-se que o método *realizarCadastro* da classe Visitante utiliza os métodos estáticos *validarNome*, *validarEmail* e *validarSenha* da classe Estudante. Por esse motivo, no diagrama de classes, representamos uma dependência “usa” entre as classes Visitante e Estudante;
- Casos em que ocorre visibilidade entre classes por parâmetro, também são representados por dependências “usa”;
- Casos em que ocorre visibilidade entre classes localmente declaradas são representados por dependências “cria”;
- Casos em que ocorre visibilidade entre classes por atributo são representados por associações.

7 CÓDIGO GERADO

O código gerado (na linguagem de programação Java) se encontra no diretório *./codigo_gerado/ensinet.* Para produzi-lo, utilizamos a ferramenta CASE *GenMyModel (*que foi recomendada pela docente), sendo o gerador utilizado denominado *advancedUML2Java*. Uma das dificuldades que tivemos para utilizar essa ferramenta, foi a necessidade de refazer o diagrama de classes usando ela. Isso foi necessário porque esse diagrama foi inicialmente feito através de outra ferramenta (*PlantUML),* e a *GenMyModel* requer a refacção do diagrama para que o código seja gerado.

Outra dificuldade foi o fato do código ter sido gerado com algumas funções com parâmetros faltando, mesmo com esses parâmetros tendo sido especificados no diagrama. Além disso, tivemos algumas dificuldades em relação aos tipos de dados *File* e *Long,* e às funções que retornam *Void.* O gerador utilizado não reconhecia essas palavras-chave em todas as instâncias, resultando em algumas anomalias no código. Tivemos, então, que consertar esses erros manualmente.

Observou-se que a ferramenta gerou os *setters* e *getters* dos atributos, bem como as assinaturas dos métodos, mas não gerou os construtores de cada classe. As associações do diagrama resultaram em atributos referenciais. Por exemplo: no diagrama, a classe *Disciplina* está associada à classe *Educador*, por isso, *Disciplina* possui um atributo referencial a *Educador*. As multiplicidades e bidirecionalidade das associações foram traduzidas para o código de maneira correta. Chegamos à conclusão de que o código foi gerado de maneira relativamente completa (com exceção dos erros citados no segundo parágrafo desta seção). Isso se deu pela completude do diagrama de classes construído, o que demonstra a importância de se fornecer um diagrama completo. Ou seja, quanto mais completo o diagrama de classes fornecido, mais completo será o código gerado, e menor será o esforço humano necessário para consertar o código.

Por fim, cabe ressaltar que, como o nosso projeto possui uma elevada quantidade de casos de uso, um esforço considerável seria necessário para chegarmos a uma implementação completa. Além dos casos de uso, teríamos ainda que modelar e implementar uma interface gráfica para a aplicação, visto que trata-se de uma aplicação web.

8 DISCUSSÕES FINAIS

Algumas dificuldades já foram descritas na elaboração do trabalho 1, entre elas: decidir uma ideia concreta para o projeto e desenvolver a arquitetura do sistema, onde tivemos algumas dificuldades para formular a melhor forma de representar efetivamente nosso projeto.

De forma geral, a segunda etapa desse desenvolvimento, ou seja, após o trabalho 1, se mostrou mais simples. Conseguimos navegar os exercícios sem tantas dificuldades, em parte por estarmos mais habituados com a matéria e, principalmente, por estarmos cada vez mais familiarizados com o projeto e com a ideia que pensamos em executar.

No geral, esse projeto nos proporcionou muitas experiências importantes, abrangendo desde o próprio conteúdo e os conceitos de *APOO* aplicados à Ensinet até a comunicação e trabalho em grupo com pessoas que, até o começo do trabalho, eram desconhecidas(grupo formado por meio do Discord).

E por conta de todo esse conhecimento adquirido ao longo do projeto, o grupo considera que a trajetória na matéria chega a um fim extremamente satisfatório, com um projeto final que gostamos e com um ponto final muito mais rico que o inicial.

9 REFERÊNCIAS BIBLIOGRÁFICAS

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3ª edição. Addison-Wesley Professional, 2012.

LARMAN, C. **Utilizando UML e Padrões**: Uma Ferramenta à Análise e ao Projeto Orientado a Objetos e ao Processo Unificado. 3ª edição, Bookman, Porto Alegre, 2007.