Universidade do Minho

Mestrado Integrado em Engenharia Informática

Desenvolvimento de Sistemas de Software

Sistema de Gestão de Turnos

Cesário Perneta

A73883

Duarte Freitas

A63129

Pedro Marta

A75587

Tiago Fraga

A74092

Ricardo Cardante

A75368

30 de Dezembro de 2017

Conteúdo

Capítulo 1

Introdução

O seguinte trabalho prático advém da necessidade de aplicação e consequente cimentação de conhecimentos adquiridos nas aulas relativas à unidade curricular de Desenvolvimento de Sistemas de Software, visando componentes essenciais à cadeira, nomeadamente, modelação do sistema proposto e consequente desenvolvimento.

A tarefa visa a elaboração de um sistema de gestão dos turnos práticos de um curso (mestrado ou licenciatura), tendo sido apresentado, numa primeira fase, modelo de domínio, diagramas de use cases e consequentes especificações, agora complementados com os restantes diagramas inerentes conclusão do exercício prático, nomeadamente:

- Diagramas de Sequência de Use Case
- Diagramas de Sequência de Subsistemas
- Diagramas de Sequência de Implementação
- Diagramas de Máquinas de Estado
- Diagrama de *Packages*
- Diagrama de Classes de Especificação
- Diagrama de Classes de Implementação

No final deste trabalho, é esperado que tenham sido corretamente aplicados conceitos nucleares de Desenvolvimento de Sistemas de Software, de um modo organizado e estruturado, enquadrado com o problema proposto.

Capítulo 2

Diagramas e Modelação

2.1 Modelo de Domínio (ver Figura 5.1)

O Modelo de Domínio inclui todas as entidades que, segundo o grupo, foram consideradas como necessárias para o projeto, nomeadamente: Unidade Curricular, Turno, Sala, Aulas, Lugar, Alunos, Professor, Diretor de Curso, entre outras.

No modelo podemos destacar a existência de quatro entidades nucleares ao projeto, como é o caso do Aluno, Professor, Unidade Curricular e Turno, uma vez que como constataremos nos seguintes pontos, serão estes os conceitos fortes a integrar os temas dos vários serviços a especificar segundo Use Case.

2.1.1 Aluno

O *Aluno*, um dos utilizadores do sistema, deverá estar inscrito numa série de *Unidades Curriculares*, e, consequentemente, num singular *Turno* por cada uma delas. O mesmo poderá requerer a troca de turno, que poderá ser levada a cabo por proposta direta a um outro aluno (ficando à mercê da sua aceitação/rejeição) ou por submissão de intenção de troca numa lista de trocas, onde as prioridades se deverão reger, em primeiro lugar, pela obediência a estatuto especial de aluno e em segundo lugar, pela ordem de entrada.

Um determinado aluno, inscrito num determinado turno, poderá ser expulso do mesmo, perdendo o

seu *Lugar*, caso veja um dado *Limite de Faltas* ultrapassado.

2.1.2 Professor

O *Professor*, um dos utilizadores do sistema, terá sob a sua alçada várias unidades curriculares onde poderá exercer a sua ação de docente. Poderá inclusivé ser *Coordenador* de qualquer uma delas, posição que lhe permite a edição da informação da dita *Unidade Curricular*. Além do mais, todo e qualquer professor tem a capacidade de marcar falta a um dado aluno, no âmbito de uma unidade curricular onde lecione e num turno que lhe esteja atribuído.

2.1.3 Unidade Curricular

A **Unidade Curricular** é composta por vários turnos de modo a suprir as diferentes necessidades horárias dos alunos. Estes turnos dizem ainda respeito a diferentes tipos de aulas, uma vez que as mesmas podem apresentar diferente caráter, nomeadamente, teórica, teórico-práticas e pratico-laboratoriais. Cada Turno apresenta uma Capacidade, nº que delimita a quantidade de Alunos inscritos no mesmo. Trocas poderão ser levadas a cabo mediante a cedência de um Lugar previamente alocado a um Aluno.

2.2 Diagrama de Use Case (ver Figura 5.2)

O Diagrama de Use Case permite, na fase de modelação, identificar tanto atores (quem vai interagir com o sistema), como os próprios Use Case, unidades coerentes de funcionalidade. Tais unidades definem o comportamento do sistema sem revelar a sua estrutura interna, mostrando apenas a comunicação entre aplicação e os vários atores. Assim, com o conjunto de todos os Use Case, definimos a funcionalidade do sistema no seu todo.

No contexto do problema proposto, o grupo encontrou como atores principais o Aluno e Professor, cada um capaz extensível em termos de funcionalidades, respetivamente, em Aluno c/Estatuto e Coordenador e Diretor de Curso. Assim, a título de exemplo, vemos que no caso do Professor, o mesmo pode "Ver a lista dos turnos"assim como "Marcar faltas", no entanto, se este mesmo docente em si também concentrar o papel de Coordenador, para além das previamente referidas operações, também poderá alte-

rar a composição do turno que rege. O mesmo se raciocínio se aplica aos restantes atores, relações entre os mesmos e entre os cenários de uso.

2.2.1 Especificação de Use Cases (ver Figura 5.3)

Diagramas que descrevem a maneira como os objetos interagem ao longo do tempo. Cada um diz respeito ao seu Use Case e especifica aqui as mensagens trocadas entre os objetos. Estes diagramas dão muito ênfase a ordenação temporal em que cada mensagem é enviada entre os objetos. A representação passa por ter o sistema, os actores envolvidos em cada interação com o sistema, das mensagens geradas pelos actores e pelas respostas do sistema. Diagramas importantes que nos permitem ter uma pequena noção do futuro sistema.

Alterar a composição de um turno

Permite ao coordenador efetuar alterações nos turnos que coordena. Assumindo o pré-povoamento do sistema, o estado de coordenador autenticado e os turnos já alocados. O coordenador começa por informar o numero do aluno e o sistema(SGT) vai verificar se esse numero existe. No caso de não existir o sistema notifica que tal numero não esta registado e cancela o processo (break). Se se verificar que o aluno esta registado então o coordenador vai ter de indicar a UC e o respetivo turno. Mais uma vez o sistema tem de confirmar os dados recebidos. Após a verificação podem acontecer duas coisas ou o processo é cancelado(break) devido aos dados incorretos ou o turno estar cheio ou o sistema efetua a alteração e informa o coordenador que o aluno foi movido para o novo turno.

2.3 Máquinas de Estado

Diagramas de maquina de(transição de) estados são utilizados para representar os estados pelos quais uma instância de um determinado objeto passa durante a sua existência, quer seja de um Use Case, de um subsistema ou sistema completo. Estes diagramas mostram de forma mais pormenorizada o comportamento de um determinado objeto permitindo visualizar todas as mudanças sofridas dos seus estados. Um estado representa a condição em que um objeto se encontra num determinado momento. Durante um processo cada objeto pode passar por diversos estados, estados esses que têm uma determinada sequencia

começando pelo seu estado inicial passando por algumas transições(evento[condição]/ação, para o próprio ou interna) e que eventualmente chegam ao seu estado final. É de referir também que há atividades que nao provocam qualquer tipo de alteração de estado como por exemplo do/ação que é uma ação executada enquanto um objeto estiver num determinado estado. Para haver alterações de estado tem de se verificar pelo menos um destes requisitos:

- ocorrência de um evento
- reação a um estimulo
- execução de uma atividade
- satisfação de uma condição

Representação de um sistema (ver Figura 5.4)

Os diagramas de maquinas de estado também podem ser usados para modelar o comportamento do sistema mostrando como interagem as diferentes janelas de um programa(interface).

Representação dos estados de um objeto (ver Figura 5.5)

Neste caso mostra apenas os estados pelos quais um objeto passa durante um processo especifico. No caso do Login, em anexo, podemos ver que passa por dois estados diferentes. Começa pelo estado sem permissões e de seguida de acordo com a satisfação da condição passa para o estado autenticado, caso esta condição não se verifique permanece no mesmo estado. Com o evento "logout"passa do estado Autenticado para o sem permissão novamente. Neste estado sem permissão pode também optar por cancelar o processo e atingir o estado final.

2.4 Diagramas de Sequência (*Use Case*) (ver Figura 5.6)

Os Diagramas de Sequência (*Use Case*) representam a visão de mais alto nível dentro desta tipologia de diagrama, uma vez que permitem começar a análise do futuro sistema, representando por cada *Use Case* o sistema como uma “caixa negra”, as interações dos diversos atores, os eventos por eles gerados e

as respostas por parte do sistema.

Um dos Diagramas de Sequência (*Use Case*) herdado da modelação de funcionalidades através do Diagrama de *Use Case* foi o "Marcar Falta". Como evidenciado pela Figura 5.6, vemos que apenas duas *lifelines* são representadas, nomeadamente o ator, neste caso o Coordenador, e o sistema enquanto "caixa"negra. Numa fase inicial, o coordenador comunica ao sistema o nº do aluno, que como indicado por *self message*, é verificado. Em caso negativo, o cenário vê o seu fim, com a comunicação do facto ao Coordenador. De seguida, a mesma lógica se aplica à indicação da UC e turno, culminando na efetivação da troca de turno solicitada pelo aluno, novamente em *self message*, e consequente resposta do sistema em direção ator, relatando o resultado.

2.5 Diagramas de Sequência de Subsistemas (ver Figura 5.7)

Os Diagramas de Sequência de Subsistemas resultam da introdução dos subsistemas do projeto nos diagramas de sequência dos vários use cases, passando deste modo a haver uma visão mais concreta relativamente à especificidade do sistema.

Deste modo, a plataforma foi dividida em 3 subsistemas, nomeadamente, Aulas, Pessoal e Trocas, os quais passaram a integrar a constituição dos diagramas do ponto anterior, substituindo a dita "caixa negra" representativa do sistema no seu todo, bem como a adequada adaptação das interações entre as várias *lifelines*.

2.6 Diagramas de Sequência de Implementação (ver Figura 5.8)

Os Diagramas de Sequência de Implementação cumprem o propósito da continuação da modelação de uma funcionalidade/processo/função do sistema, ao mais baixo nível dentro desta tipologia de diagramas, referindo-se já às expectáveis classes a existir aquando o começo da elaboração do código relativo ao projeto, estando assim representados os vários métodos através das mensagens que estabelecem linhas entre as *lifelines*, *lifelines* essas que representam instâncias das classes previamente referidas.

2.7 Diagrama de *Package* (ver Figura 5.10)

À medida que os sistemas de software se tornam mais complexos, torna-se difícil efetuar a gestão de um número crescente de classes. Assim, tendo em conta que a identificação de uma classe e o seu papel se encontram dependentes do próprio contexto, é determinante conseguir identificar as diferenças entre as diversas classes de um sistema.

De acordo com a interpretação por parte do grupo, dentro da camada de negócio, foi acordado diferenciar as diferentes classes em packages de aulas, pessoal, trocas, bem como do sistema SGT, package de *facade*, o qual é acessado por parte do package exterior da camada de apresentação. Todos os packages da camada de negócios dotados de visibilidade privada, por serem integrais ao funcionamento da plataforma, mostram necessidades de acesso à camada de dados, package exterior à *business layer*.

2.8 Diagrama de Classes (de especificação) (ver Figura 5.11)

O Diagrama de Classes de especificação, essencial a todo o projeto modelado no paradigma orientado a objetos, visa retratar a visão topológica do sistema, especificando todas as classes, juntamente com variáveis e métodos partilhados entre a totalidades das respetivas instâncias. São também particularizadas a visibilidade de cada método e variável, bem como as diferentes relações estabelecidas entre as várias classes.

Assim, de um ponto de vista prático, temos associações (neste caso qualificadas) representadas em Maps, como é o caso do Sistema de Gestão de Turnos que apresenta o Map listaUtilizadores, cujo *qualifier* mostra ser o numero, chave do Map previamente referido e variável de instância da classe Utilizador. Neste caso específico, considera-se que os Utilizadores somente existem no contexto do SGT, não tendo existência para além da existência do SGT. Por outro lado, o contrário se verifica com a associação entre as classes Turno e Aluno, uma vez que apesar dos Turnos fazerem parte da estrutura interna do aluno, os ditos Turnos podem existir para além do contexto do Aluno.

Aliada a este tipo de associação, encontra-se a relação entre Troca e PedidoTroca, sendo a 2ª uma extensão/especialização da 1ª, espelho do vínculo super-classe/sub-classe.

No caso das restantes relações, o raciocínio é homólogo.

2.9 Diagrama de Classes com DAOs (de implementação) (ver Figura 5.12)

O Diagrama de Classes de implementação cumpre o mesmo propósito do anterior, desta vez com inclusão de elementos de dados como é o caso dos DAOs, classes inerentes a uma futura camada de dados, de elevada importância para a atividade básica de cada classe que requer o uso de qualquer um dos seus métodos, métodos esses que garantem a persistência de dados e acesso aos mesmos. Estes elementos aproximam a fase de modelação à fase de elaboração efetiva do código, conjugando já 2 das 3 camadas aplicacionais.

2.10 Diagrama de Instalação (ver Figura 5.13)

O Diagrama de Instalação cumpre o propósito de especificação da arquitetura física do sistema, ou seja, a topologia de hardware sobre a qual são executados os componentes de software. É, deste modo, permitida a especificação da distribuição de componentes, bem com a identificação dos estrangulamentos de desempenho.

No contexto do projeto em desenvolvimento, considerou-se suficiente correr todo o sistema numa mesma máquina, a qual deverá suportar tanto o ambiente da própria aplicação como da base de dados PostgreSQL que suporta o seu *schema* e que com esta comunidade através da API JDBC.

Capítulo 3

Camadas Aplicacionais

3.0.1 Camada de Apresentação/Interface (*Presentation Layer*)

A camada de apresentação do programa é constituída por cinco janelas construídas em *Java Swing*. Esta linguagem é uma linguagem orientada ao evento, portanto a aplicação tem como objetivo dar resposta a uma serie de eventos que o utilizador faça. Estes eventos podem ser desde um clicar no botão, selecionar item's numa *comboBox*, inserir texto num campo de texto, etc.

Esta camada tem como principais funções, ser intuitiva e de fácil uso para o utilizador, mas mais importante ainda, ser uma camada que proteja possíveis exceções que a camada de negócios pudesse lançar face a um certo input que viesse da interface.

A camada de apresentação somente tem ligação com a camada de negócios, e esta é feita através da façade, que no nosso caso é a classe **SGT**, que controla e faz a gestão de toda a camada de negócios.

A titulo de exemplo, um aluno quando interage com a aplicação a camada de apresentação, esta é capaz de fornecer uma lista com as Unidades Curriculares e Turnos que o aluno está inscrito, a possibilidade de se inscrever numa lista de trocas a uma determinada UC, ou então efetuar um pedido de troca a outro aluno que esteja inscrito á mesma UC mas a um turno diferente. Quando a interação de utilizador/interface é feita por um aluno com estatuto, ou um professor, que pode ser regente, diretor de curso, ou simplesmente professor, esta camada apresenta um *layout* e funcionalidades exclusivas consoante o tipo de estatuto e o tipo de utilizador.

3.0.2 Camada de Negócio (*Business Logic Layer*)

A camada de negócios é a ponte entre a camada de apresentação e a camada de dados. Essa ponte é feita pela façade, que no nosso caso, é a classe **SGT**. Alem da façade é constituída pelo *package* de aulas que contem a classe **UnidadeCurricular** e a classe **Turno**, o *package* pessoal que contem a classe abstrata **Utilizador** e as classes **Professor** e **Aluno** e por fim, o *package* trocas que é constituído pela classe abstrata **Troca** e as classes **TrocaNormal** e **PedidoTroca**.

Esta camada implementa toda a lógica da aplicação, no entanto, sem as outras duas camadas não existia nenhum tipo de apresentação para o utilizador, e não havia persistência de dados, sendo que se perdia todas as alterações ao fechar o programa.

3.0.3 Camada de Dados (*Data Layer*)

3.0.4 Motivação e Objetivos

Sendo o projeto que tem vindo a ser relatado ao longo do presente projetado e concretizado numa implementação multicamada orientada aos objetos, com necessidade de manipulação póstuma de dados à sessão atual (onde serão gerados/alterados), prevê-se desde logo que a utilização da memória para a gestão dos mesmos será claramente insuficiente e incapaz de providenciar a funcionalidade pretendida. De forma similar, a escrita serializável em ficheiros, embora eventualmente capaz e simples (do ponto de vista de implementação), constitui uma abordagem incompleta, pouco consistente e não otimizada. Assim, tornou-se clara e imediata a utilização de uma base de dados, capaz de garantir a persistência dos dados gerados e manipulados ao longo das sessões de utilização de uma forma consistente, íntegra e otimizada.

3.0.5 Arquitetura

Do supra firmou-se a opção da utilização de uma base de dados como mecanismo de persistência dos dados gerados pela aplicação. Em particular, e tendo em conta a expectativa de crescimento dos mesmos, optou-se pela utilização de uma arquitetura relacional, modelada, normalizada e circunscrita às restrições de integridade de acordo com as necessidades específicas do projeto. Por forma a garantir

um elevado grau de disponibilidade e utilização contínua, foi criada uma instância de uma base de dados **PostgreSQL**, disponível no endpoint:

A escolha do *PostgreSQL* como sistema de base de dados adveio do seu elevado grau de fiabilidade, integridade dos dados e correção, com um suporte exímio e prestado pela comunidade *open-source* que o desenvolve, afirmando-se como uma das opções mais utilizadas à escala global.

3.0.6 Modelação, normalização e scripts SQL

Por forma a garantir a consistência e integridade dos dados gerados, a base de dados foi desenhada e implementada seguindo os tradicionais pressupostos de verificação e modelação das arquiteturas relacionais. Assim sendo, as relações criadas advieram de um processo de modelação com duas etapas (conceptual e lógica) e foram verificadas quanto ao cumprimento das restrições de integridade (identidade, entidade, referencial, domínio e específicas do projeto) e das formas de normalização 1,2 e 3, o que, numa última instância, levou à efetiva concretização no modelo físico da base de dados, cujos scripts foram disponibilizados para efeitos de observação e eventual análise/apreciação, em diretoria própria do repositório entregue.

3.0.7 Comunicação com a aplicação

Por forma a estabelecer a comunicação da camada de dados (devidamente estruturada e exposta em pacote aplicacional próprio) com as camadas superiores do projeto foram criadas múltiplas APIs de DAOs (Data Access Objects), que compreendem protocolarmente os métodos necessários à gestão dos dados da aplicação de uma forma expedita e íntegra, com recurso ao JDBC driver, responsável por providenciar a conexão à instância de base de dados criada (mencionada na secção anterior) e cuja dependência é automaticamente resolvida via Maven. As interfaces utilizadas encontram-se também no repositório entregue, devidamente documentadas em formato Javadoc.

3.0.8 Medidas de segurança

Com vista a proteger a instância de base de dados criada, algumas medidas de escopos administrativo e aplicacional foram implementadas, garantindo assim uma maior segurança no armazenamento dos dados, na comunicação entre camadas aplicacionais e maior capacidade de resposta em caso de falhas.

- A conexão estabelecida com a base de dados ao nível aplicacional é efetuada com recurso a um utilizador sem qualquer privilégio administrativo. De facto, o utilizador criado apenas tem acesso ao schema "sgt", tendo acesso completo às tabelas que o compõe e permissões de execução das funções armazenadas e criadas pelo administrador da base de dados (acedidas nas interfaces dos DAOs). A qualquer momento este utilizador pode ser eliminado e o schema restaurado para momentos temporais anteriores (conforme especificado no ponto V).

```
CREATE USER g6 WITH NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT NOREPLICATION
NOBYPASSRLS PASSWORD '                ';
```

- A comunicação feita com a base de dados implementada via interfaces dos Data Access Objects previne SQL Injection ou ataques similares, por executarem queries sob a forma de PreparedStatement e com chamada exclusiva a funções pré-armazenadas/definidas na base de dados, cujos argumentos são passados não sob a forma de concatenação de String, mas de métodos próprios e restritos do objecto PreparedStatement (prevenindo assim injeções maliciosas);

```
public void addStudent(
    final String email, final String name, final String password, boolean worker) {
  try {
    this.connection = PostgreSql.connect();
    String query = "SELECT sgt.add_student(?, ?, ?, ?)";
    PreparedStatement statement = this.connection.prepareStatement(query);
    statement.setString( parameterIndex: 1, email);
    statement.setString( parameterIndex: 2, name);
    statement.setString( parameterIndex: 3, password);
    statement.setBoolean( parameterIndex: 4, worker);
    statement.execute();
  } catch (Exception e) {
    logger.error(e.getMessage());
  } finally {
    PostgreSql.close(this.connection);
  }
}
```

- A verificação de login de um utilizador é feita com recurso a uma função da base de dados que

recebe os dados do formulário, valida os mesmos e retorna um booleano. Nenhuma password é carregada em memória a partir da base de dados. De facto, não existe sequer qualquer método nas APIs implementadas para os DAOs que permitam obter a password de um utilizador.

```
public boolean verifyLogin(final String email, final String password) {
  boolean valid = false;
  try {
    this.connection = PostgreSql.connect();
    String query = "SELECT sgt.verify_login(?, ?)";
    PreparedStatement statement = this.connection.prepareStatement(query);
    statement.setString( parameterIndex: 1, email);
    statement.setString( parameterIndex: 2, password);
    statement.execute();
    ResultSet result = statement.getResultSet();
    if (result.next()) {
      valid = result.getBoolean( columnIndex: 1);
    }
  } catch (Exception e) {
    logger.error(e.getMessage());
  } finally {
    PostgreSql.close(this.connection);
    return valid;
  }
}
```

- As passwords são armazenadas de forma segura, garantido assim que eventuais acessos indevidos revelem informação sensível. Esta codificação faz uso do algoritmo blowfish com uma salt gerada aleatoriamente com 8 iterações, com recurso à extensão pgcrypto, disponível no PostgreSQL.

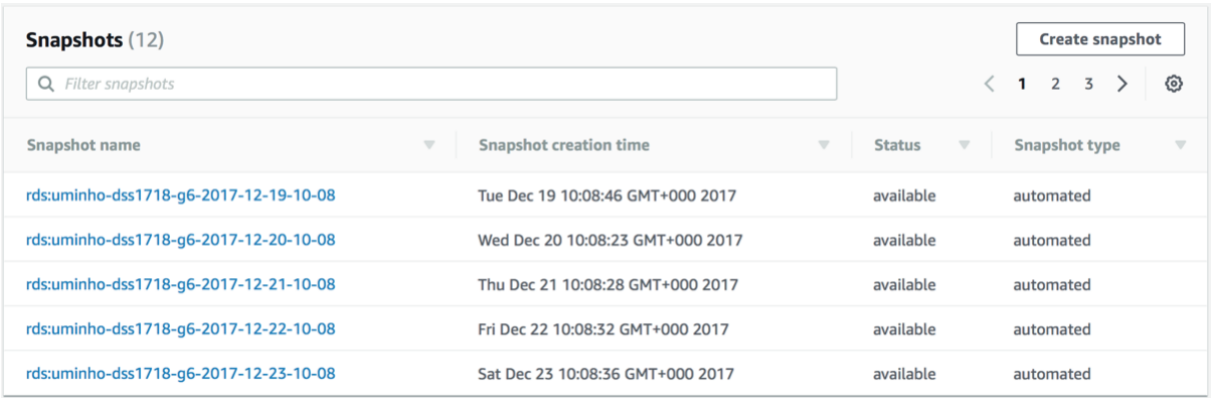
12	8@uminho.pt	José Creissac	$2a$08$JveDORcderqF8ablQJLAAeq4i67aJ...	teacher

- Snapshots diários do estado atual da base de dados são efetuados, por forma a permitir uma recuperação imediata e segura dos dados em caso de falha.

3.0.9 Requisitos de armazenamento

Para efeitos essencialmente informativos, segue-se uma tabela com o tamanho máximo que um registo pode ter, para cada relação criada na base de dados, bem como a extrapolação do valor para um milhão de registos.

Os limites do *PostgreSQL* encontram-se tablados de seguida, cuja informação foi retirada do sítio



Snapshots (12)

Create snapshot

Filter snapshots

1 2 3

Snapshot name	Snapshot creation time	Status	Snapshot type
rds:uminho-dss1718-g6-2017-12-19-10-08	Tue Dec 19 10:08:46 GMT+000 2017	available	automated
rds:uminho-dss1718-g6-2017-12-20-10-08	Wed Dec 20 10:08:23 GMT+000 2017	available	automated
rds:uminho-dss1718-g6-2017-12-21-10-08	Thu Dec 21 10:08:28 GMT+000 2017	available	automated
rds:uminho-dss1718-g6-2017-12-22-10-08	Fri Dec 22 10:08:32 GMT+000 2017	available	automated
rds:uminho-dss1718-g6-2017-12-23-10-08	Sat Dec 23 10:08:36 GMT+000 2017	available	automated

Relação	Armazenamento máximo/ registo (bytes)	Armazenamento máximo para um milhão de registos (megabytes, valor aproximado)
sgt.user	180	172
sgt.subject	150	143
sgt.shift	92	88
sgt.student_shift	50	48
sgt.direct_trade	52	50
sgt.trade_by_request	76	72
Total	**600 bytes**	**573 megabytes**

oficial do sistema de bases de dados (http://postgresql.org/about/)

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

Capítulo 4

Conclusão

Com o término da presente fase, é admissível afirmar que todos os objetivos requeridos pelo docente, no enunciado cedido, foram alcançados.

O grupo passou numa 1ª fase pela modelação do sistema a desenvolver, elaborando um modelo de domínio e diagrama de use cases, com direito às respetivas especificações, primeiros passos essenciais para uma estruturada continuação da modelação do sistema, assim como, posterior desenvolvimento.

No contexto da 2ª fase, o grupo prosseguiu com a modelação da plataforma, tendo à partida, com os diagramas de sistema de use case, detetado uma significativa quantidade de falhas em especificações elaboradas na fase anterior, aspeto ilustrativo da importância da fase de modelação de um projeto, tamanha é a clareza do incremento de conhecimento da estrutura e funcionamento da plataforma a desenvolver à medida que o grupo foi elaborando os vários diagramas.

Do ponto de vista prático de escrita de código, com modelação a um nível relativamente baixo, tal foi o caso dos Diagramas de Sequência de Implementação e Diagramas de Classes, o grupo encontrou uma benvinda fluidez neste processo, sem qualquer obstáculo significativo. O caminho ficou claramente delineado pela modelação em linguagem UML, resultando numa necessidade limitada de atenção para o desenvolvimento dos DAOs e implementação da GUI em SwingFX.

O modelo por camadas foi também bastante útil na elaboração do sistema, uma vez que com compartimentalização por ele garantida, qualquer aspeto a alterar numa das camadas, em nada interferia

com as restantes, garantindo assim robustez adicional a toda a plataforma.

A qualidade do trabalho apresentado agrada ao grupo, tendo havido tempo suficiente para o desenvolvimento das soluções necessárias aos exercícios propostos, sem precipitações e com espaço de maneio suficiente para que todos os membros pudessem inteirar-se do trabalho realizado.

Capítulo 5

Anexos

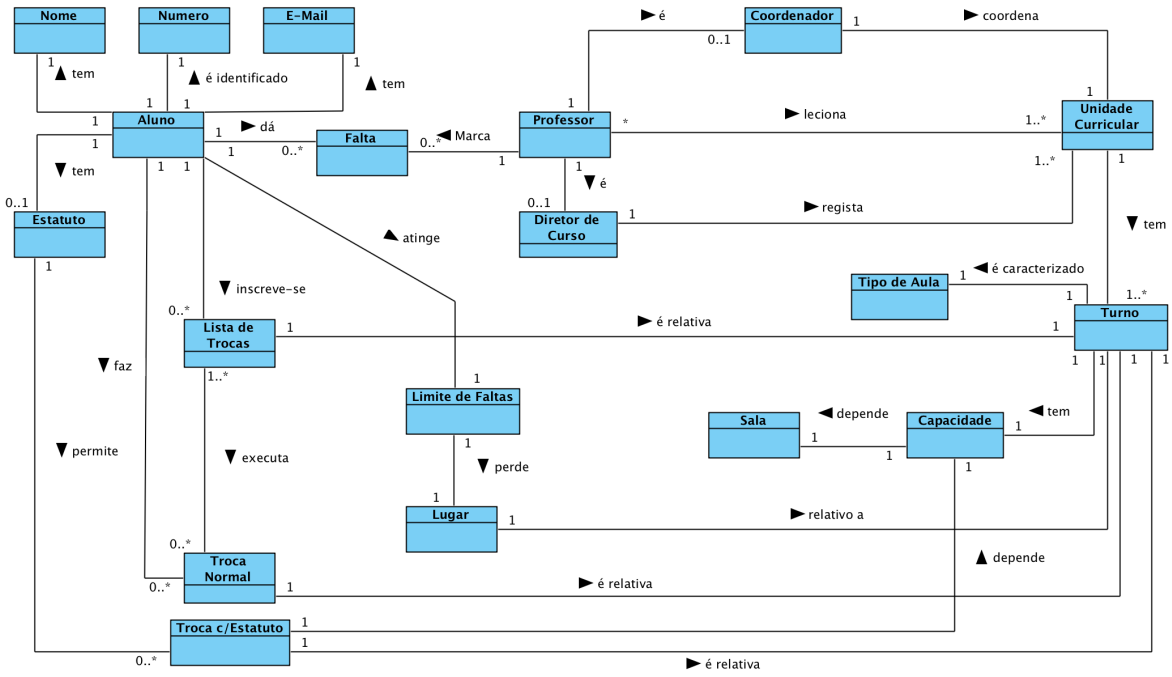


Figura 5.1: Modelo de Domínio

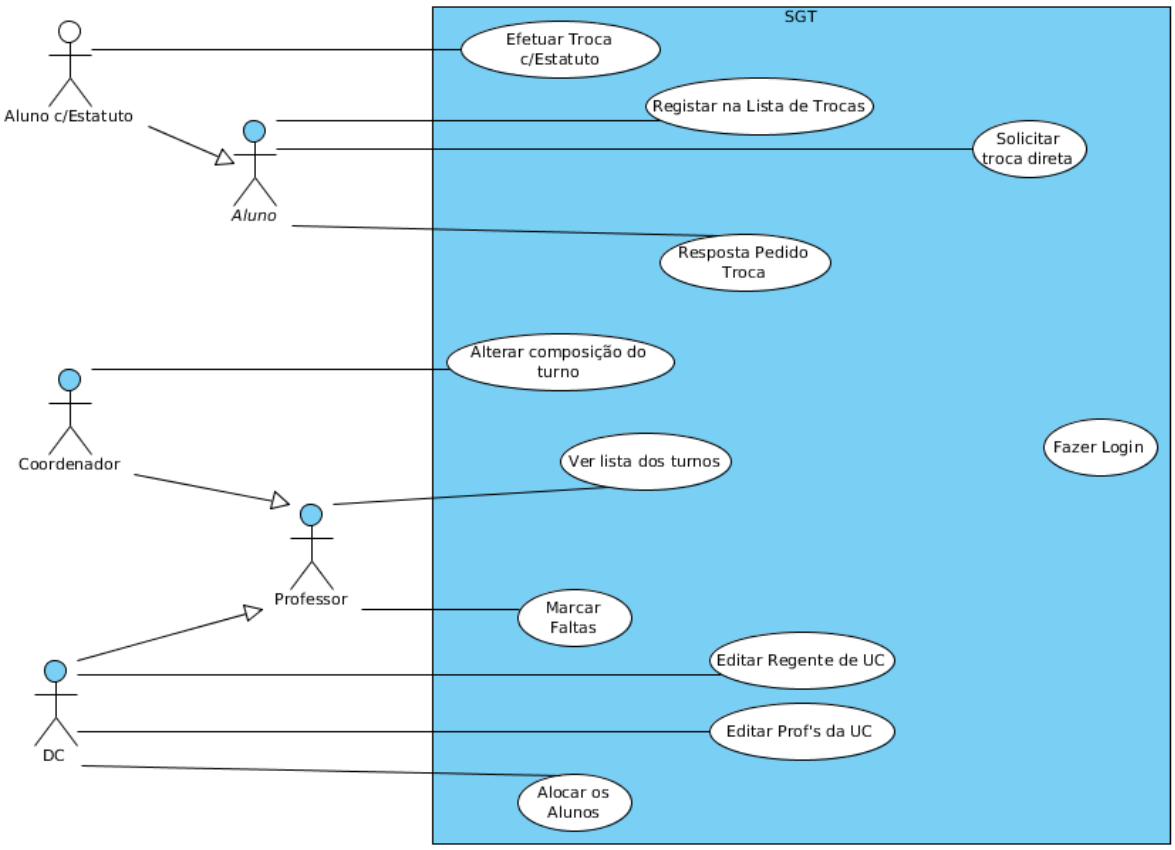


Figura 5.2: Diagrama de Use Cases

Use Case:	**Alterar a composição de um turno.**	
Descrição:	O Coordenador solicita o sistema para fazer alterações nos turnos que pertencem à UC que coordena.	
Pré-Condição:	Sistema está povoado, os turnos ja foram alocados e o professor fez o login.	
Pós-Condição:	O Coordenador faz as trocas que pretende.	
Comportamento Normal	Ator(Professor Coordenador)	Sistema
	1.Notifica o sistema com o numero de aluno que pretende mover.	
		2.O sistema recebe a informação.
	3.Indica o numero do turno para onde pretende mudar o aluno.	
		4. O sistema recebe a informação.
		5. O sistema efectua a alteração.
Excepção [O aluno nao existe]. (passo2)		2.1 - O sistema recebe a informação, no entanto o aluno nao existe no sistema.
Excepção. [O turno, onde pretende colocar o aluno, está cheio]. (passo 4)		4.1 O sistema recebe a informação e notifica o professor que esse turno está lotado.

Figura 5.3: Especificação de Use Case - Alterar a composição de um turno

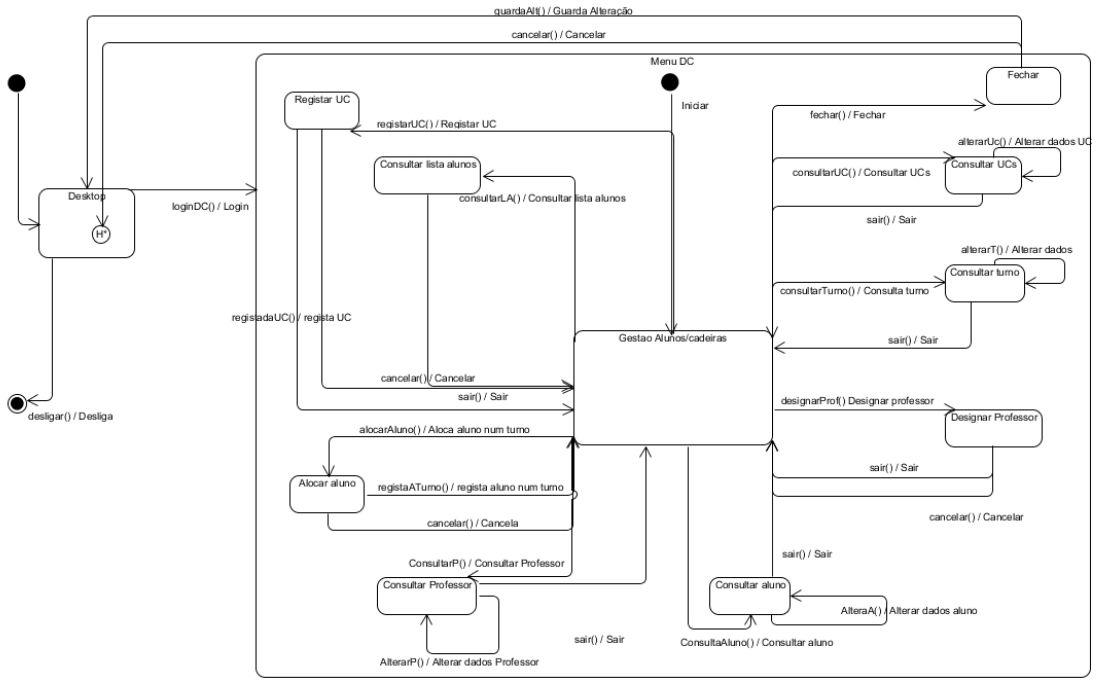


Figura 5.4: Maquina de estado - DC

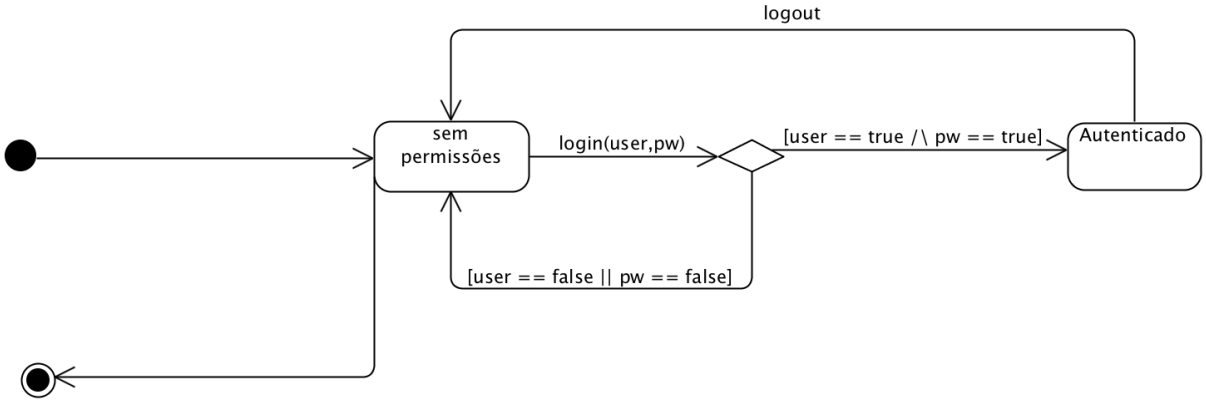


Figura 5.5: Maquina de estado

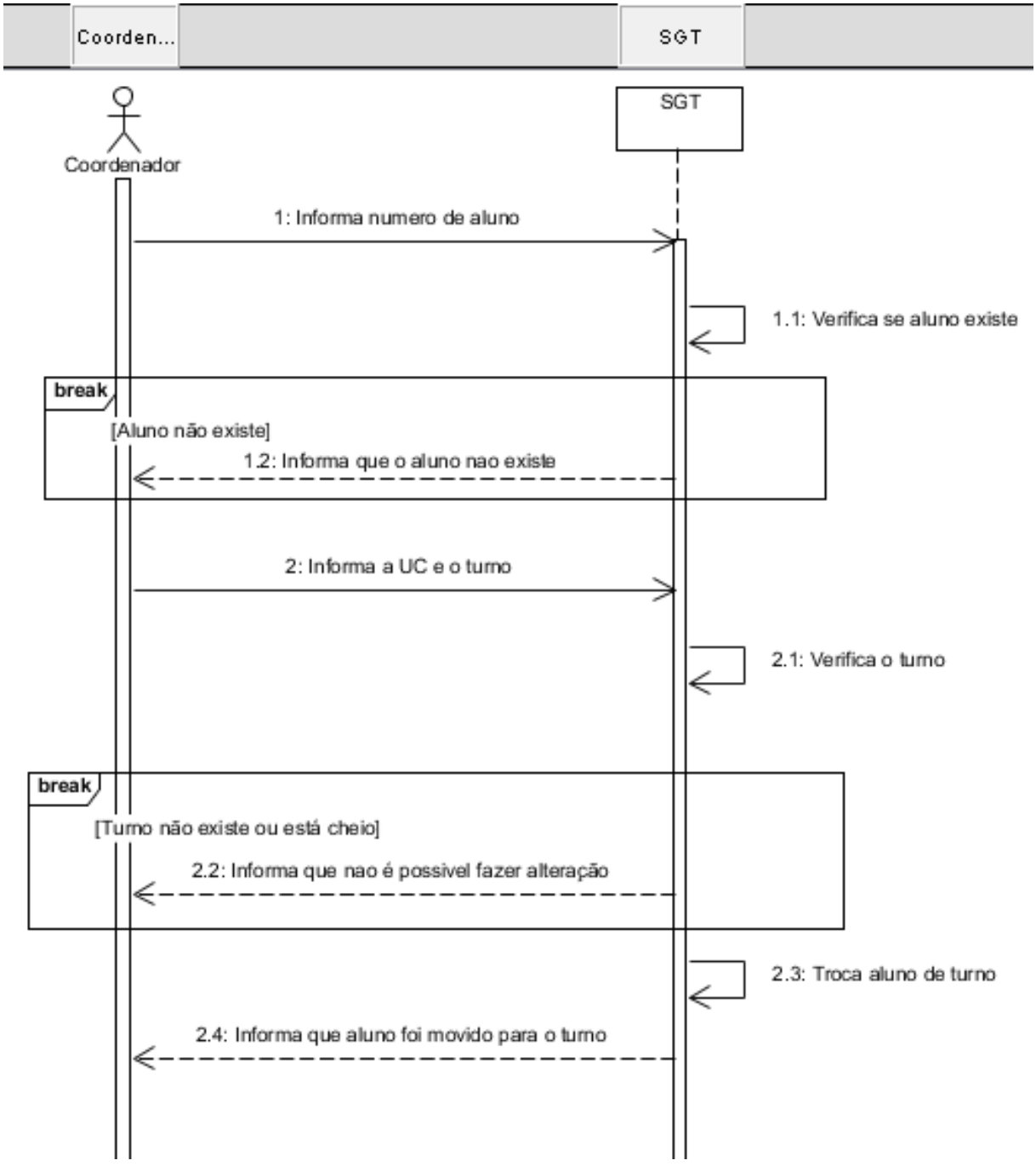


Figura 5.6: Diagrama de Sequência de Use Case

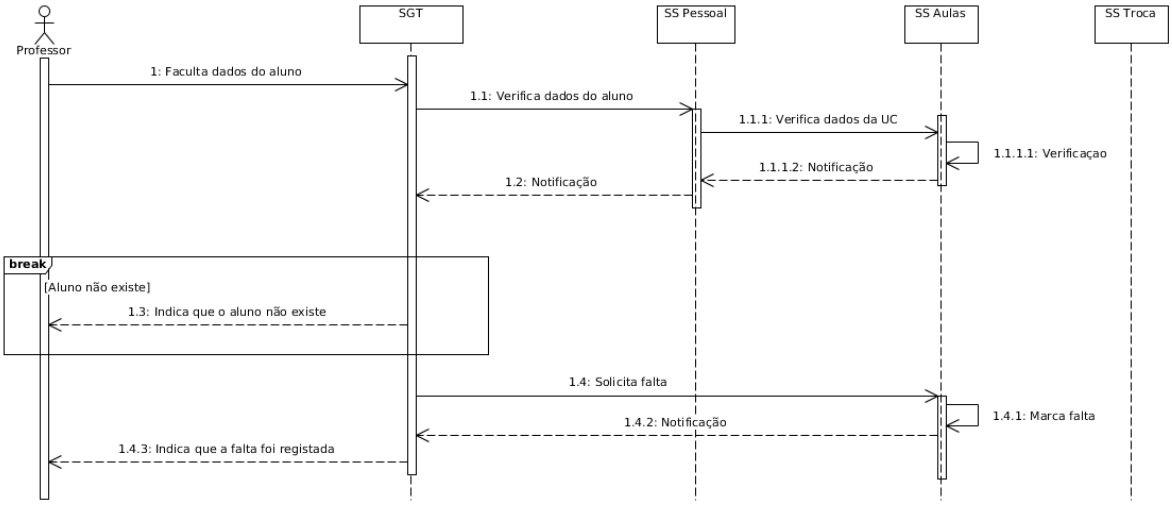


Figura 5.7: Diagrama de Sequência de Subsistema - Marcar Falta

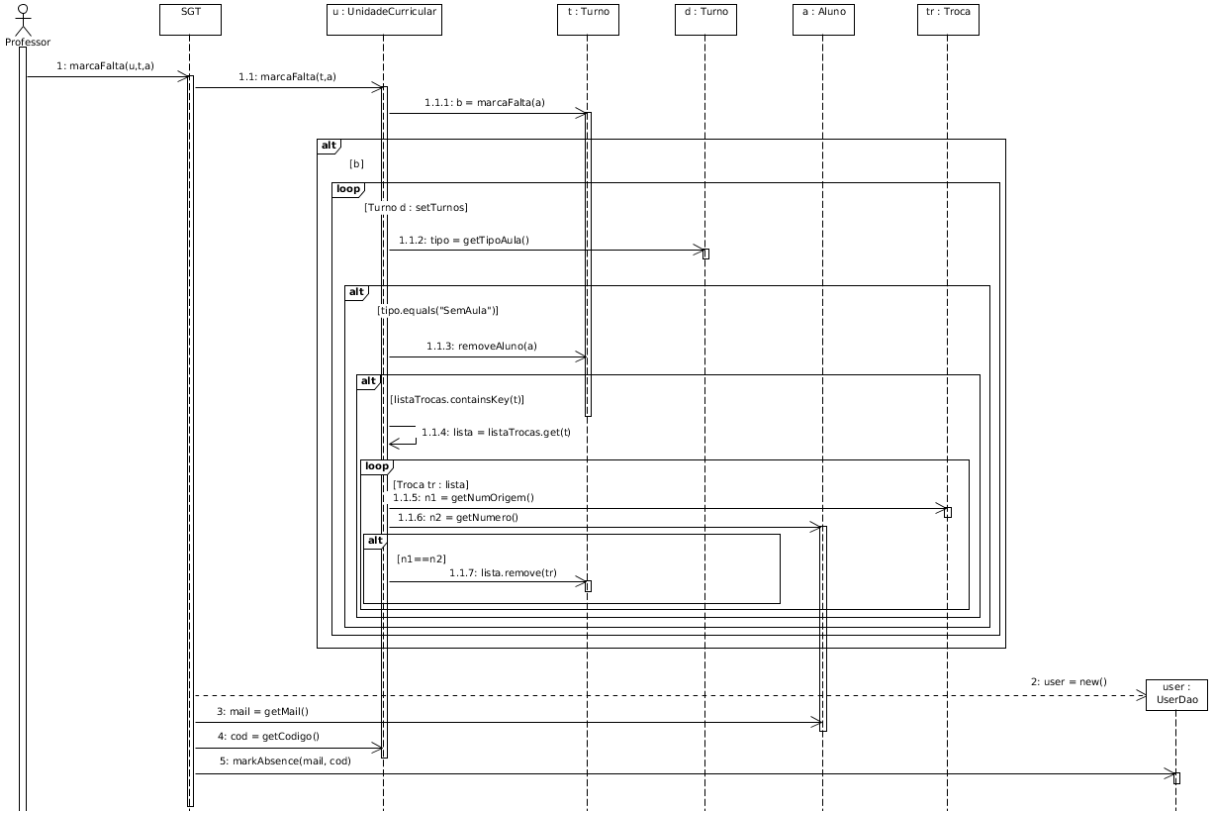


Figura 5.8: Diagrama de Sequência de Implementação - Marcar Falta

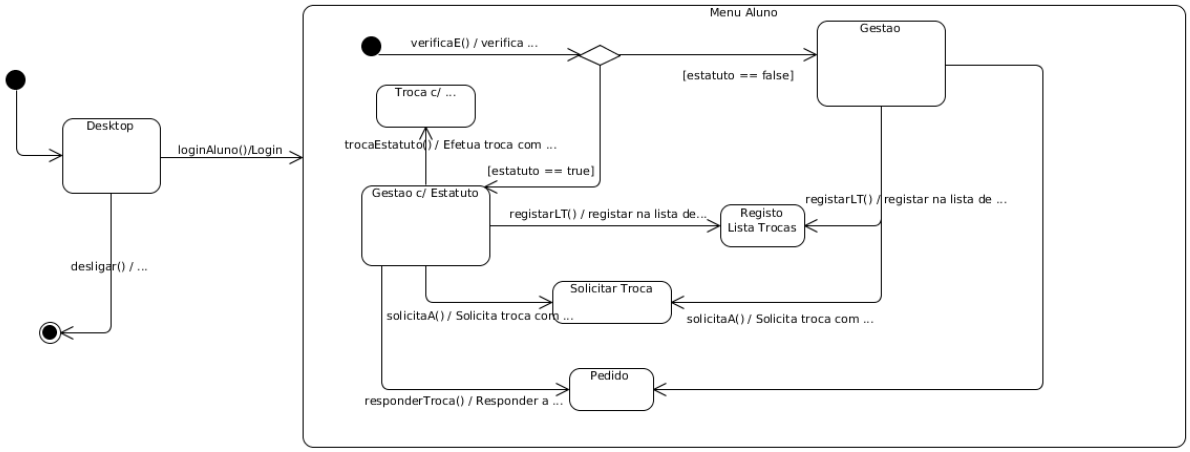


Figura 5.9: Diagrama de Máquina de Estado - Aluno

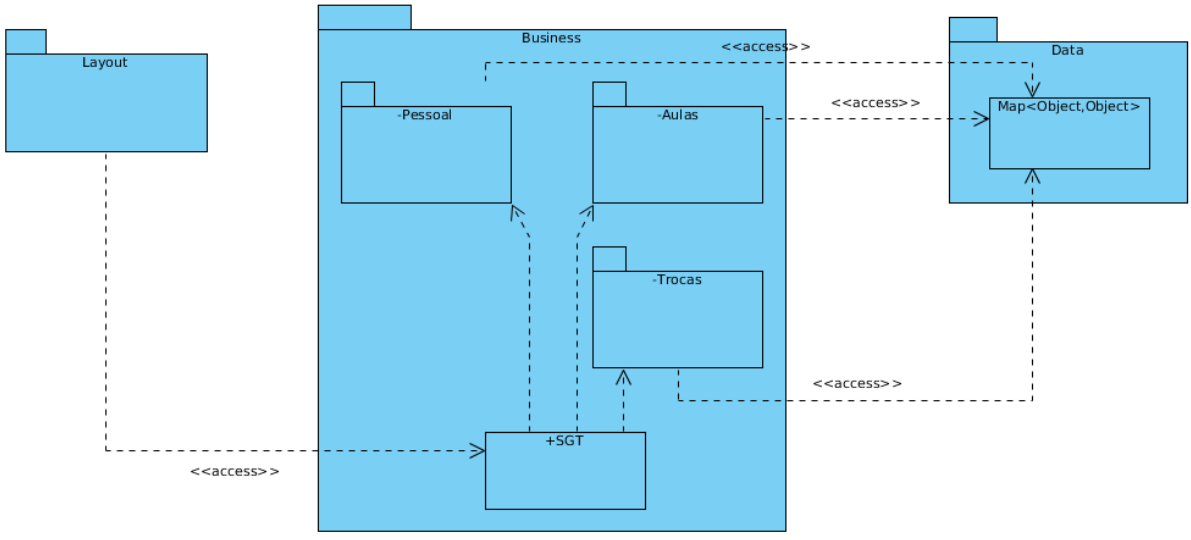


Figura 5.10: Diagrama de Packages

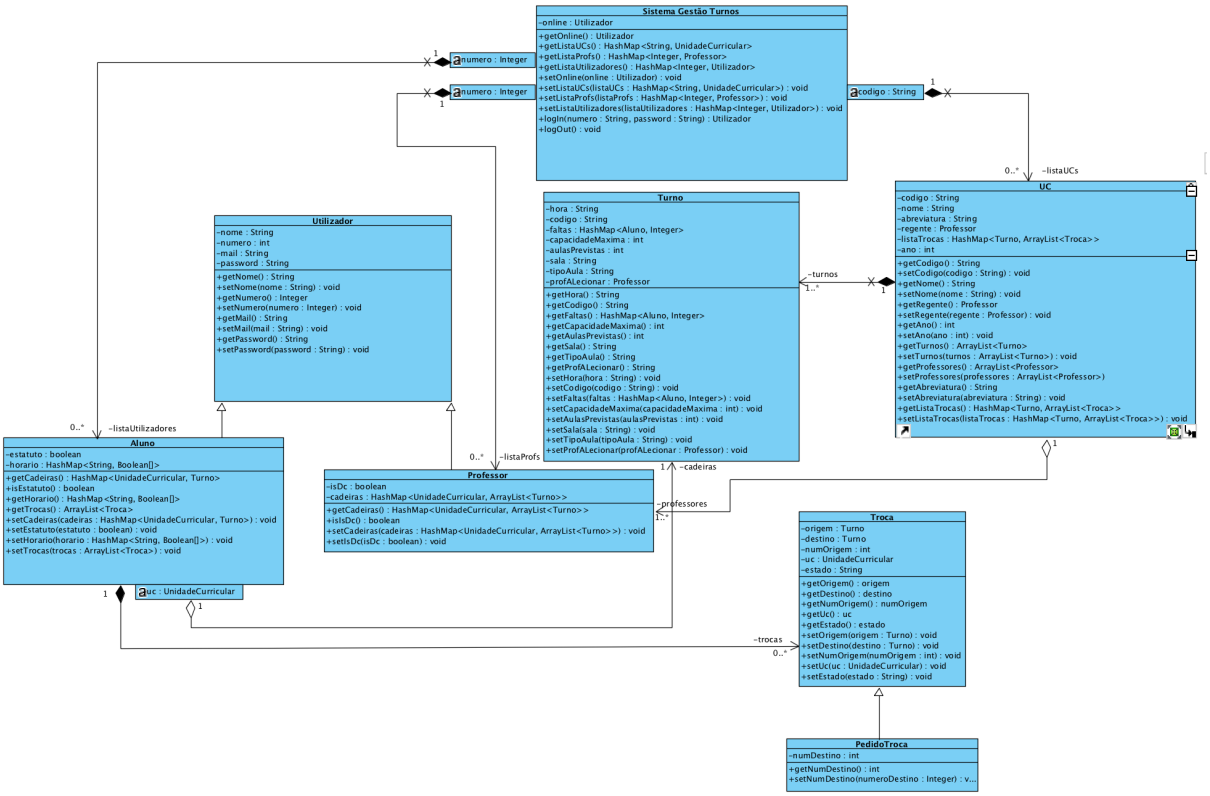


Figura 5.11: Diagrama de Classes (de especificação)

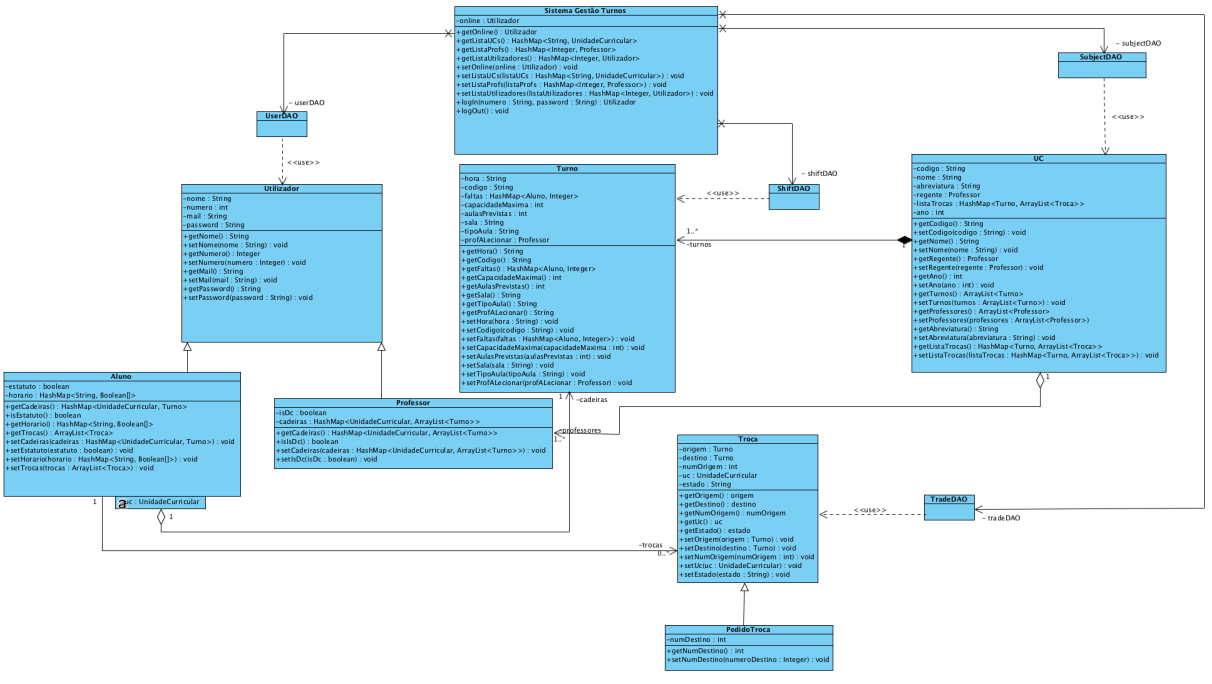


Figura 5.12: Diagrama de Classes com DAOs (de especificação)

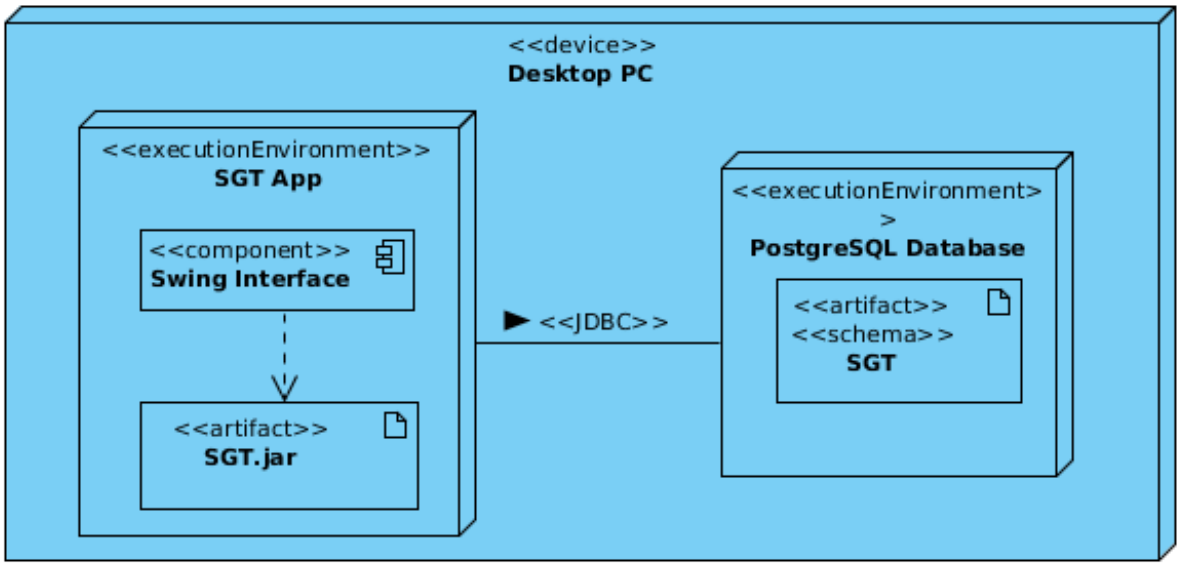


Figura 5.13: Diagrama de Instalação