Licenciatura em Engenharia Informática e de Computadores

Relatório Técnico

Tiago Pereira: nº 43592
Denise Rodrigues: nº 44881

Projeto de Laboratório de Software
2019/2020 verão

25/04/20

Introdução

O projeto do Laboratório de Software é composto pela análise, design e implementação de um sistema de informação para gerenciar salas e a reserva das mesmas.

O seu desenvolvimento é dividido em 4 fases, nesta primeira fase foi definido tanto o domínio como a funcionalidade e a interação com o sistema de informações é feita através da execução de comandos em consola.

Se o aplicativo for executado sem argumentos entra em modo interativo, lendo as linhas de entrada e executando os comandos correspondentes, esta ação é interrompida através do comando EXIT.

Caso seja chamado com argumentos estes serão interpretados como um comando, após a execução desse comando a aplicação finaliza.

Cada comando é definido usando a seguinte estrutura genérica:

{method} {path} {parameters}

Onde *method* define o tipo de ação a realizar, podendo ser GET ou POST;

Path define o local em que o comando é executado;

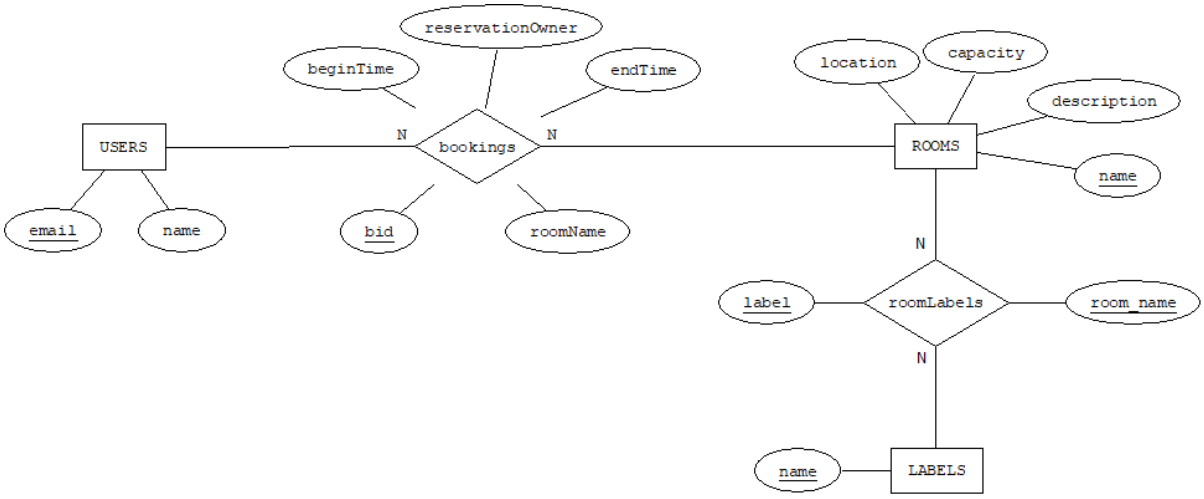
Parameters definem uma sequência de pares nome-valor, separados por &;

Esta fase tem como principais requisitos a possibilidade de podermos tanto criar novas instâncias de ROOM, BOOKING, LABEL e USER, bem como obter informação relativa a essas entidades.

Modelação da base de dados

Modelação conceptual

O seguinte diagrama apresenta o modelo entidade-associação para a informação gerida pelo sistema.



Destacam-se os seguintes aspetos deste modelo:

É composto pelas entidades e relações apresentadas abaixo, cada uma com pelo menos uma chave primária (atributo sublinhado).
ROOMS (name, location, capacity, description)
bookings (bid, reservationOwner, roomName, beginTime, endTime)
LABELS (name)
USERS (email, username)
roomLabels (roomName, label)

Achou-se relevante criar a relação roomLabels, que representasse a ligação entre room e as labels associadas, sendo assim temos roomName como chave estrangeira de ROOMS (name) e label como chave estrangeira de LABELS (name);
Adicionou-se também a bookings o atributo bid, que representa o identificador de cada booking e tendo por isso sido escolhido para chave primária.

O modelo conceptual apresenta ainda as seguintes restrições:

Relativamente a **ROOMS**, temos room como um espaço físico que uma ou mais pessoas podem usar, assim sendo a cardinalidade entre ROOMS e USERS é de N-N;

Relativamente a **bookings**, temos booking como reserva de um room por um determinado intervalo de tempo. Tanto o valor do tempo de início como de fim da reserva devem ser múltiplos de 10 e a duração mínima da reserva deve ser de 10 minutos;

Relativamente a **LABELS**, cada label representa uma característica que pode ser associada aos rooms, a mesma label pode estar associada a 0 ou mais rooms e cada room pode ter 0 ou mais labels, representando uma cardinalidade N-N;

Relativamente a **USERS**, um user é uma pessoa que pode reservar rooms (um ou mais).

Modelação física

Após a realização do modelo EA seguiu-se a implementação do modelo físico do sistema, contemplando todas as restrições possíveis de garantir.

O modelo físico da base de dados está presente em …

Destacam-se os seguintes aspetos deste modelo:

Foram desenvolvidos os scripts CREATETABLES e INSERTS, o primeiro contendo instruções CREATE TABLE e adicionalmente instruções DROP TABLE para criação e remoção do modelo respetivamente.

O comando DROP TABLE é composto pela cláusula IF EXISTS, o que implica remoção da tabela apenas caso esta exista, bem como dos seus dados.

Depois de termos as tabelas construídas foi possível passar a inserir dados na BD, para isso foi criado o script INSERTS.

Organização do software

Processamento de comandos

A execução de cada tipo de comando é tratada por um componente diferente.

(Um comando é composto por method, path e um conjunto de parameters)

É importante ter em conta que existe uma diferença entre path e path template, o path do comando é definido pelo path template, que nos fornece uma forma de extrair os valores do parâmetro de um caminho que satisfaça o predicado.

Temos a interface CommandHandler e consecutivamente uma classe de implementação por comando.

Esta interface, composta pelos métodos execute() e description() , é implementada por todas as classes Get e Post de user, room, label e booking.

É em execute() que recebemos o pedido do comando (CommandRequest), é também aqui que é estabelecida conexão e definida a instrução preparedStatement a ser executada no contexto dessa conexão.

Em description() é descrito o tipo de informação a ser retornada.

Encaminhamento dos comandos

É usado o Router para localizar o comando adequado de acordo com o método e path do pedido (findRoute).

Através do router é ainda possível a aplicação registar novos handlers e os métodos e paths associados (addRoute).

O método checkPathMatch(), é responsável por verificar se os methods e paths introduzidos pelo utilizador correspondem aos de Key, caso haja é retornada uma instância de routeResult composta pelo handler e uma lista de parameters.

Podemos então dizer que o router decide o handler, que pode ou não lidar diretamente com a base de dados.

Gestão de ligações

Este ponto tem como base 3 conceitos essenciais:

1) A **interface de conexão**, que representa uma conexão com o RDBMS;
2) A **interface PreparedStatement**, que representa uma instrução (query, inserção, remoção) a ser executada no contexto da conexão;
3) O **resultado da execução** da instância de PreparedStatement, uma instância ResulSet, que permite acesso programático a um conjunto de linhas da tabela.

Relativamente ao estabelecimento de conexão:

Havia duas formas de obter uma instância de Connection, neste projeto foi usada uma implementação concreta de DataSource.

Uma instância de DataSource permite-nos aceder ao método getConnection, usado para providenciar a conexão, sendo assim, para obter a instância DataSource foi criada diretamente uma instância de uma classe que implementasse DataSource, PGSimpleDataSource foi a classe escolhida.

De seguida, por meio do método setURL(), definiram-se as propriedades necessárias.

No fim é sempre fechada a conexão através da chamada ao método close(), todos os recursos (statements, etc) são também fechados.

Relativamente à interface PreparedStatement:

As instâncias de PreparedStatement foram obtidas por meio de um método de Connection, da seguinte forma:

PreparedStatement statement = *connection.preparedStatement (queryString);*

É fornecida uma queryString a cada PreparedStatement, essa query pode conter marcadores de parâmetros representados por "?".

Antes de ser executado o statement este deve receber o valor do parâmetro para os marcadores através dos métodos *setX(int parameterIndex, …),* que por sua vez contém parameterIndex como primeiro parâmetro, definindo a posição do parâmetro a ser atribuído.

Por fim, quando o statement estiver pronto para ser executado, é chamado o método executeQuery() e retornado um CommandResult com o resultado.

É importante referir que estes pontos formam tidos em conta e postos em prática na implementação de todas as classes GetX e PostX que implementassem CommandHandler.

Acesso a dados

Foram criadas as seguintes classes para ajudar no acesso aos dados:

	HANDLER	GET	POST	PUT	DELETE
BOOKING	BookingHandler	GetBooking GetBookingById GetBookingByOwner	PostBooking	PutBooking	DeleteBooking
LABEL	LabelHandler	GetLabel	PostLabel		
ROOM	RoomHandler	GetRoom GetRoomById GetRoomByLabel	PostRoom		
USER	UserHandler	GetUser GetUserById	PostUser		

Foi implementado nas classes GET e POST o método execute(), que estabelecida a conexão nos permite processar o comando e executar as queries. Aqui podemos aceder e ir buscar dados às tabelas (Get) , bem como inserir nova informação (Post).
Na fase 2 foram adicionados novos comados PUT e DELETE, que permitem atualizar dados já existentes na tabela ou remove-los.

Nos Handlers de cada classe temos ainda a implementação de alguns queries de verificação que achamos relevantes para o bom funcionamento do sistema.
O nome atribuído a estes métodos é bastante claro, não deixando dúvidas acerca daquilo que se quer validar, como por exemplo:

checkIfRoomIsAvailable(), checkIfLabelAlreadyExists(), checkIfEmailAlreadyExists()…

Dependendo do valor retornado a nova informação pode ser inserida na BD ou não, caso não seja possível, é então lançada uma exceção SQL com a mensagem do respetivo erro.

Em PostBooking é lançada exceção caso o room especificado não esteja disponível ou caso o tempo de duração da reserva não contemple os requisitos (mínimo de 10 minutos).
Caso não haja problemas é adicionado a commandResult informação da nova reserva dado o bid, através de *getNextBookingId()* ;

Em PostLabel pretendemos criar uma nova label, no entanto é lançada exceção cajo essa label já exista;

Quando criamos um novo room é adicionada informação a roomLabels, essa relação é estabelecida em RoomHandler, através dos métodos *getRoomLabels()* e *inserLabelsRoom().*
É adicionada a commandResult informação sobre o novo room, no entanto, caso label não seja válida é lançada exceção.

Quanto ao User, é lançada exceção caso o email especificado já exista, senão é apenas adicionada ao commandResult informação sobre a nova inserção.

Alterações a ter em conta na fase 2

Os packages pertencentes a handler na fase 1 passam agora a conter um novo package result, com a implementação dos métodos columns() e description().

Note-se que o resultado é sempre adicionado a uma lista.

Booking

A classe BookingHandler passa a conter 3 novos métodos *getBeginTime(), getDuration(), checkDuration(),* necessários à implementação do novo comando PUT Booking.

Em PutBooking é criada uma query de update que recebe begintime, endtime, reservationowner e atualiza o booking identificado por *bid* caso o room esteja disponível e a duração dentro dos requisitos.

É retornado um PostBookingResult que recebe a lista bookingResult contendo bookingId.

Já através em DeleteBooking é possível remover um booking dado o *bid* através da query de delete.

Rooms

Foi adicionado a GetRooms o método executeGetCommands() onde:

1) Começamos por verificar begin e duration, caso haja dados relativamente a esses parâmetros é retornado em result uma lista de rooms disponíveis a partir de beginTime e com a dada duração, senão é retornada a lista de todos os rooms com as respetivas labels;
2) Caso seja indicada a capacidade é retornada a lista de rooms que aceitam no mínimo o número de pessoas indicado em capacity, ou seja, rooms com capacity inferior são removidos da lista;
3) Dada uma label removemos da lista os rooms cuja label não corresponda a esta.

Para tal foram implementados em RoomHandler os métodos:

getAvailableRooms() e getAllRoomsWithLabels()

Time e Option

Foram adicionados novos comandos nesta fase;

GET/time que nos permite obter o tempo atual, OPTION/ que retorna uma lista dos comandos disponíveis dado getRoutes(), que nos apresenta os routes com as respetivas descrições.

Request

Foi adicionado ao CommandRequest o método getHeader() e ao enumerado Method os valores OPTION, DELETE e PUT;

Foram ainda adicionadas as seguintes classes:

1) Enumerado HeaderType com os valores ACCEPT e FILENAME;
2) HeaderValue, que retorna o valor de header;
3) Header, contendo um hashmap e um método addPair() que permite adicionar ao hashmap, isto porque a componente header é composta por uma sequência de pares nome-valor, onde cada par é separado por '|' e ':' entre o par. A analise do header é feita em App através de checkHeader();

Output

Podemos ter duas formas de representação output, textplain ou html, logo foi implementado:

1) Um enumerado com os valores PLAIN e HTML;
2) Uma interface OutputInterface com dois métodos printToConsole() e printToFile() a serem invocados em UserInterface;
3) Classe PlainTextOutput que permite devolver um formato igual ao apresentado na fase 1, contém implementação de printToConsole() e printToFile();
4) Classe HTMLOutput contém implementação de printToConsole() e printToFile(), é ainda definido o formato html através dos métodos headerFormat(), titleFormat(), footerFormat() e tableResultFormat() que chama ainda getRows() e getColumns().

UserInterface

Tendo em conta que:

- Todos os comandos Get devem suportar o header accept, caso não seja inserido deve ser usado texto/plain;
- Comandos Get devem ainda suportar file-name header, que define a localização do ficheiro para representação do output, caso o header não seja referido a representação é escrita no standard output;

Foram implementados os métodos show() e askForCommand() invocados mais tarde em App.

É feita a análise do header, para ver se contém accept e verificação de filename;

Caso haja filename é chamado printToFile(filename), senão é chamado printToConsole();

Se outputType for PLAIN é retornado PlainTextOutput, senão é retornado HTMLOutput.

Processamento de erros

Visto que há possibilidade de serem lançadas exceções durante o programa foi necessário garantir um correto processamento de erros.

A classe App é então responsável pelo tratamento das exceções.

Podemos estar, por exemplo, perante um pedido cujo método não corresponda a nenhum dos valores aceites (GET, POST, PUT, DELETE ou EXIT) e nesse caso é lançada exceção com a mensagem “Request not found”. Caso tal não aconteça prosseguimos então com o processamento do pedido, no entanto durante o processamento podem ainda ocorrer outras exceções.

Essas exceções são capturadas e tratadas de forma a ser retornada a mensagem de erro correspondente.

Nota Adicional

Foram criadas as classes RouterGetsTests e RouterPostTests com o objetivo de testar pedidos e comprovar o bom funcionamento do programa.