Databases Project
2021/2022

(2022/03/03: versão 1.0 do enunciado; pequenos ajustes podem ser realizados nos próximos dias)

Introduction

The goal of this project is to provide students with experience in developing database systems. The project is guided by the industry's best practices for software development; thus, students will experience all the main stages of a common software development project, from the very beginning to delivery.

Objectives

After completing this project, students should be able to:

- Understand how a database application development project is organized, planned and executed
- Master the creation of conceptual and physical data models for supporting and persisting application data
- Design, implement, test, and deploy a database system
- Install, configure, manage and tune a modern relational DBMS
- Understand client and server-side programming in SQL and PL/pgSQL (or similar).

Groups

The project is to be done by **groups of 2 or 3** students. Depending on the size of the group, different functionalities are to be developed (as defined later in this document).
IMPORTANT: the members of each group **must** be enrolled in PL classes of the same Professor.

Quality attributes for a good project

Your application must make use of:

(a) A transactional relational DBMS (the use of PostgreSQL is optional)
(b) A distributed database application architecture, providing a REST API
(c) SQL and PL/pgSQL, or similar
(d) Adequate and relevant triggers, functions and procedures running on the DBMS side
(e) Good strategies for managing transactions and concurrency conflicts, and database security
(f) Good error avoidance, detection and mitigation strategies
(g) Good documentation

Your application must also respect the functional requirements defined in the companion document (annex A) and execute without "visible problems" or "crashes". To fulfill the objectives of this assignment, you can be as creative as you want, provided you have implemented this list of defined features in your solution.

Milestones and deliverables

Midterm presentation (20% of the grade) – 23:55 of March 27th

The group **must present** their work in the PL classes (privately, to the professor of the class). The following artifacts should be developed and uploaded at *inforestudante* until the deadline (one member of the group uploads the artifact, but all submissions must clearly identify all students working in the project):

- **Presentation** (e.g., power point) with the following information:
  - Name of the project
  - Team members and contacts
  - Brief description of the project
  - Definition of the main database operations, transactions, and potential concurrency conflicts

    - Description of a potential solution (if you already have one)
    - Core technologies: programming language, DBMS, Libraries, etc. The group is free to select the technologies to be used
    - Development plan: planned tasks, initial work division per team member, timeline
- **ER diagram**
    - Description of entities, attributes, integrity rules, etc.
- **Relational data model**
    - The physical model of the database, i.e., the tables.

Final delivery (80% of the grade) – 23:55 of May 16th

The project outcomes must be submitted to *inforestudante* until the deadline. Each group must select a member for performing this task. All submissions must clearly identify the team and the students that compose the group working in the project. Upload into *inforestudante* the following materials:

- **Final report** with:
    - **User manual** describing how users can interact with the application
    - **Installation manual** describing how to deploy and run the software you developed
    - **Final ER and relational data models**
    - **Development plan:** make sure you specify which tasks were done by each team member and the effort involved (e.g., hours)
    - **All the information, details, and design decisions you consider relevant to understand how the application is built and how it satisfies the requirements of the project**
- **Source code and Scripts:**
    - Include the source code, scripts, executable files and libraries necessary for compiling and running the software (identify the DBMS used, do not upload its binaries)
    - DB creation scripts containing the definitions of tables, constraints, sequences, users, roles, permissions, triggers, functions, and procedures

Defense – May 17th to 31st

- Prepare a 5-minute live presentation of your software
- Prepare yourself **(individually)** to answer questions regarding all deliverables and implementation details
- Sign up for any available time slot for the defense in *inforestudante,* before the delivery due date
  The list of available slots will be released before the deadline for the final delivery

Assessment

- This project accounts for 8 points (out of 20) of total grade in the Databases course
- Midterm presentation corresponds to 20% of the grade of the project
- Final submission accounts for the remaining 80% of the grade of the project
- The minimum grade is 35%

Notes

- Do not start coding right away - take time to think about the problem and to structure your development plan and design
- Always implement the necessary code for **error detection and correction**
- Aspects related to **concurrency management**, **security**, and **best coding practices** will be valued
- Assure a **clean shutdown of your system** (no memory-leaks)
- **Plagiarism or any other kind of fraud will not be tolerated**

Anexo A: Plataforma de Vendas Online – Descrição Funcional

Este projeto consiste em desenvolver um sistema simplificado de venda de produtos eletrónicos online. O desenvolvimento de uma base de dados adequada às necessidades e restrições do modelo de negócio é essencial para garantir um armazenamento e processamento eficaz da informação.

A plataforma a desenvolver comercializa diferentes tipos de equipamentos eletrónicos (deve considerar pelo menos computadores, televisões e smartphones). Cada produto, vendido por uma empresa específica, possui diversos atributos comuns, entre eles um identificador único, descrição, preço e stock. No entanto, cada tipo de equipamento tem atributos específicos que o caracterizam (e.g., tamanho de uma televisão, processador de um computador). Por motivos de auditoria, sempre que há atualização dos detalhes de um produto (e.g., preço, especificações) deve ser mantido um histórico das versões anteriores.

Existem 3 tipos de utilizadores do sistema:

- **Administrador**: tem permissões de moderação e gestão da plataforma
- **Vendedor**: empresa que vende produtos, deve ter os dados necessários para faturação (e.g., NIF), morada de envio, (…)
- **Comprador**: deve conter todos os atributos necessários para realizar uma encomenda (e.g., morada)

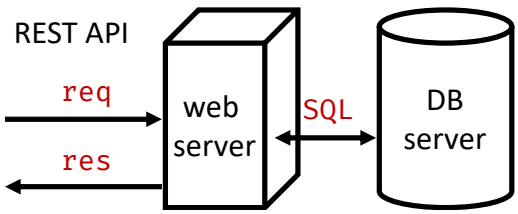
Cada encomenda pode ser composta por vários produtos (e quantidades), que por sua vez podem ser vendidos por empresas diferentes. Um produto só pode ser adquirido se existir em stock. Assuma, por uma questão de simplicidade, que a validação e atualização de stock é realizada no momento da compra.

A plataforma deve disponibilizar um sistema de *rating* de um produto, que contém uma classificação (0-5) e um comentário. Um utilizador apenas pode atribuir um *rating* por produto após aquisição. Deve também existir uma funcionalidade de perguntas/respostas por produto (qualquer utilizador pode perguntar/responder sobre qualquer produto, incluindo os que ainda não comprou). Deve ser possível responder especificamente a uma pergunta/resposta (i.e., *threads* com hierarquia).

Deverá existir também um sistema de notificações automatizado. Um vendedor deve receber notificações quando um dos seus produtos é adquirido (e os respetivos detalhes da encomenda) e quando alguém faz alguma pergunta num dos seus produtos. Um comprador deve receber todas as notificações expectáveis, tais como detalhes de uma encomenda efetuada. Qualquer utilizador deve receber notificações relativamente a respostas a perguntas feitas por ele em algum produto. O sistema de notificações deverá ser implementado com recurso a *triggers*.

(só para Grupos de 3) A plataforma terá um sistema de campanhas e cupões, gerido pelos administradores. Cada campanha decorre durante um período (data de início e fim), tem um número fixo de cupões, e a percentagem de desconto respetiva, que é igual em todos os cupões da campanha. Os utilizadores devem concorrer/subscrever cada campanha, sendo a atribuição dos cupões feita por ordem de subscrição (um por cliente). Cada cupão só pode ser usado uma vez, e só pode ser usado um cupão por compra (cada compra pode englobar vários produtos diferentes). Cada cupão tem um período de validade após atribuição, sendo este definido ao nível da campanha. Só pode existir uma campanha a decorrer em cada momento.

O sistema deve ser disponibilizado através de uma API REST que permita ao utilizador aceder ao sistema através de pedidos HTTP (quando conteúdo for necessário, deve ser usado `JSON`). A figura representa uma vista simplificada do sistema a desenvolver. Como podemos observar, o utilizador interage com o *web server* através da troca de *request*/*response* REST e por sua vez o *web server* interage com o servidor de base de dados através de uma interface SQL (e.g., JDBC no caso do Java, Psycopg no caso de Python).



Esta é uma das arquiteturas mais utilizadas atualmente e suporta muitas das aplicações web e mobile que usamos no dia a dia. Uma vez que o foco da disciplina está nos aspetos de bom desenho de uma base de dados e das funcionalidades associadas, o desenvolvimento de aplicações web ou mobile **está fora do âmbito deste trabalho**. Para usar ou testar a sua API REST, pode usar um cliente REST tal como o (e.g., `postman.com`). Nos casos em que o formato do pedido (`req`) e da resposta (`res`) estiver especificado nos parágrafos abaixo, estes devem **ser seguidos rigorosamente**. Nos casos em que não estiverem definidos, o grupo deve especificá-los e incluir a sua definição no relatório.

Importante: a lógica do sistema (e.g. pesquisas) deve ser implementada nas *queries* SQL e não no web server!

Funcionalidades a desenvolver

Para facilitar o desenvolvimento do projeto e manter o foco no âmbito da cadeira, os grupos poderão seguir/usar o código de base que será disponibilizado para o efeito. O nome dos *endpoints* da API REST deve **ser estritamente respeitado**. Quaisquer detalhes omissos devem ser identificados explicitamente no relatório.

As respostas da API REST devem seguir uma estrutura simples, mas rígida, **devolvendo sempre um *status code* adequado**:

- 200 – success
- 400 – error: bad request (erro no pedido)
- 500 – error: internal server error (erro na API)

Caso existam erros, estes devem ser devolvidos em *errors* e, caso existam dados a retornar, estes devem ser devolvidos em *results* (como se pode observar nos detalhes dos *endpoints* que se seguem).

Quando um utilizador começa a utilizar o sistema deve poder escolher entre registar uma nova conta e entrar com uma conta existente, tirando partido dos seguintes *endpoints*:

***NOTA**: Este é um cenário simplificado, num contexto real seria necessário utilizar uma ligação segura/encriptada o envio de credenciais (e.g., HTTPS).*

Registo de utilizadores. Criar um novo utilizador, inserindo os dados requeridos pelo modelo de dados. Qualquer pessoa se pode registar como comprador, mas apenas administradores podem criar novos administradores/vendedores (i.e., será necessário passar um token para validar o administrador).

req **POST** http://localhost:8080/dbproj/user
```
{"username": username, "email": email, "password": password, (…)}
```
res
```
{"status": status_code, "errors": errors (if any occurs), "results": user_id (if it succeeds)}
```

Autenticação de utilizadores. Login com *username* e *password*, recebendo um *token* (e.g., Json Web Token (JWT)) de autenticação em caso de sucesso. Este *token* deve ser incluído no *header* de todas as chamadas subsequentes.

req **PUT** http://localhost:8080/dbproj/user
```
{"username": username, "password": password}
```
res
```
{"status": status_code, "errors": errors (if any occurs), "token": auth_token (if it succeeds)}
```

Após autenticação, o utilizador poderá realizar as seguintes operações:

Criar novo produto – Cada vendedor deve poder criar novos produtos para comercializar.

req **POST** http://localhost:8080/dbproj/product
```
{"description": "product description", "type": "product type", "price": price, "stock": stock, (…)}
```
res
```
{"status": status_code, "errors": errors (if any occurs), "results": product_id (if it succeeds)}
```

Atualizar detalhes produto. Deve ser possível atualizar os detalhes de um produto. Para efeitos de auditoria é necessário manter as diferentes versões.

req **PUT** http://localhost:8080/dbproj/product/{product_id}
{"description": description, "price": price, (…)}

res {"status": status_code, "errors": errors (if any occurs)}

PROCEDURE

Efetuar compra – Deve ser possível registar uma nova compra, com todos os detalhes associados.

req **POST** http://localhost:8080/dbproj/order
{"cart": [(product_id_1, quantity), (product_id_2, quantity), (…)], "coupon": coupon_id (só para grupos de 3)}

res {"status": status_code, "errors": errors (if any occurs), "results": order_id (if it succeeds)}

Deixar rating/feedback – Deve ser possível deixar um rating a um produto comprado.

req **POST** http://localhost:8080/dbproj/rating/{product_id}
{"rating": rating, "comment": "feedback"}

res {"status": status_code, "errors": errors (if any occurs)}

Deixar comentário/pergunta – Deve ser possível fazer/responder a uma pergunta num produto.

req **POST** http://localhost:8080/dbproj/questions/{product_id}
POST http://localhost:8080/dbproj/questions/{product_id}/{parent_question_id} (if answering an existing question)

{"question": "question"}

res {"status": status_code, "errors": errors (if any occurs), "results": question_id (if it succeeds)}

Consultar informações genéricas de um produto – Deve ser possível obter os detalhes genéricos de um produto (e.g., título, stock), o histórico de preços, rating médio, e comentários (ter em conta que é possível que para um dado produto não haja rating ou comentários, devendo o mesmo ser devolvido). No servidor deve ser usada apenas **uma** *query* SQL para ir buscar esta informação.

req **GET** http://localhost:8080/dbproj/product/{product_id}

res {"status": status_code, "errors": errors (if any occurs)}, "results": {"description": "product_description", "prices": ["current_price_date - current_price", "prev_price_date - prev_price", (…)], "rating": average rating, "comments": ["comment 1", "comment 2", (…)]}

Obter estatísticas (por mês) nos últimos 12 meses. Deve ser possível obter os detalhes das vendas (e.g., número de vendas, valor) por mês nos últimos 12 meses. No servidor deve ser usada apenas **uma** *query* SQL para ir buscar esta informação.

req **GET** http://localhost:8080/proj/report/year

res {"status": status_code, "errors": errors (if any occurs), "results": [
{"month": month_0, "total_value": total_value_orders, "orders": orders_count},
(…)
]}

Não esquecer: deverá existir um sistema automatizado de notificações, implementado através de *triggers*, conforme descrito no enquadramento do projeto. Deve implementar o(s) endpoint(s) necessários para consultar estas notificações.

(só para Grupos de 3) **Criar nova campanha –** Um administrador deverá poder criar novas campanhas. Tenha em conta que não poderão existir campanhas ativas em simultâneo.

req **POST** http://localhost:8080/dbproj/campaign
{“description”: “campaign description”, “date_start”: “starting date”, “date_end”: “ending date”, “coupons”: number of coupons to be generated, “discount”: discount percentage, (…)}

res {“status”: status_code, “errors”: errors (if any occurs), “results”: campaign_id (if it succeeds)}

(só para Grupos de 3) **Subscrever campanha/cupões –** Deve ser possível um comprador subscrever uma campanha de atribuição de cupões. Assim que esgotarem os cupões, a campanha termina.

req **PUT** http://localhost:8080/dbproj/subscribe/{campaign_id}

res {“status”: status_code, “errors”: errors (if any occurs), “results”: {"coupon_id”: coupon_id, “expiration_date”: expiration_date} (if it succeeds)}

(só para Grupos de 3) **Obter estatísticas dos descontos aplicados por campanha –** Deve ser possível obter uma lista das diversas campanhas, número de cupões emitidos e utilizados, bem como o valor total dos descontos aplicados. Campanhas sem cupões utilizados/atribuídos também devem ser devolvidas. No servidor deve ser usada apenas **uma** *query* SQL para ir buscar esta informação.

req **GET** http://localhost:8080/dbproj/report/campaign

res {“status”: status_code, “errors”: errors (if any occurs), “results”: [
{"campaign_id”: campaign_id, “generated_coupons”: generated_coupons_count, “used_coupons”: used_coupons_count, “total_discount_value”: total_discount_value},
(…)

] (if it succeeds)}

NOTAS FINAIS

- Conforme explícito em alguns *endpoints*, a solução pretendida deve ser obtida através de uma única *query* SQL no lado do servidor. **No entanto**, caso não consigam resolver dessa forma é preferível utilizarem mais *queries* do que não implementarem o *endpoint*.
- O processamento dos dados (e.g., ordem, restrições) deve ser feito ao nível das *queries* sempre que possível
- O controlo de transações, concorrência, e segurança serão considerados para avaliação