Desafio - Plataforma de Cursos Online "AcademiaDev"

A startup de tecnologia educacional **AcademiaDev** está lançando sua nova plataforma de cursos online. Seu modelo de negócio é baseado em um sistema de assinaturas que dá aos alunos acesso a um catálogo de cursos de alta qualidade, focados no desenvolvimento de software.

Para validar sua proposta de negócio, a AcademiaDev contratou sua equipe para desenvolver um protótipo inicial da aplicação. Por um infortúnio do destino, parte de sua equipe foi hospitalizada após tentarem comemorar mais uma vitória do Corinthians sobre o Palmeiras subindo na mesa do bar, apenas para descobrir que ela era menos estável que um deploy em produção numa sexta-feira. Dessa forma, cabe a vocês, os(as) únicos(as) desenvolvedor(es/as) *geração saúde* da equipe, trabalhar na implementação desse protótipo inicial utilizando todos os conceitos que conhece sobre a linguagem Java. Nesse protótipo os requisitos são focados na implementação da lógica de negócio principal, utilizando um conjunto de dados já existente.

Para garantir que o protótipo seja manutenível e escalável (e para que você possa ser promovido quando sua equipe voltar do hospital), **a AcademiaDev exige que a implementação siga estritamente os princípios da Clean Architecture.**

Para focar na lógica principal da aplicação, **não será necessário implementar as funcionalidades de CRUD completas**. Em vez disso, sua aplicação deve iniciar com um conjunto de dados pré-cadastrado. Crie uma classe utilitária (ex: InitialData) na camada externa, que popule suas estruturas de dados em memória assim que a aplicação iniciar. Ou seja, não é necessário criar um CRUD completo de Courses ou Users - apenas o suficiente para validar a lógica de negócio.

Nesse sentido, o protótipo deverá implementar os seguintes Casos de Uso para:

- Gerenciamento do catálogo de Courses (cursos);
- Gerenciamento de Users (usuários) e seus respectivos planos de assinatura;
- Sistema de Enrollments (matrículas) e acompanhamento de progresso dos alunos;
- Um sistema de fila para atendimento de Support Tickets;
- Geração de relatórios e exportação de dados da plataforma.

A equipe de analistas da sua empresa, a partir de reuniões com os fundadores da AcademiaDev, já havia determinado os requisitos a seguir.

1. Requisitos Funcionais

1) Gerenciamento do Catálogo de Courses Os cursos da plataforma devem possuir as seguintes características:

- title e description. O title de cada curso deve ser único na plataforma.
- instructorName.
- durationInHours (carga horária).

- difficultyLevel, que pode ser BEGINNER, INTERMEDIATE ou ADVANCED.
- status, que pode ser ACTIVE ou INACTIVE. Um curso com status INACTIVE não pode receber novas matrículas.

2) Users e Subscription Plans Os usuários da plataforma podem pertencer a dois perfis:

- Admin: Possui name e email. Tem permissão para gerenciar o catálogo de cursos e usuários.
- Student: Possui name, email e um subscriptionPlan.
- O email de cada usuário (seja Student ou Admin) deve ser único.

Os planos de assinatura (SubscriptionPlan) disponíveis para alunos são:

- BasicPlan: Permite que o aluno esteja matriculado em, no máximo, **3 cursos ativos** simultaneamente.
- PremiumPlan: Permite que o aluno se matricule em um **número ilimitado** de cursos.

3) Sistema de Enrollments e Progress

- Um aluno só pode se matricular em um curso (Course) se o seu plano de assinatura permitir e se o curso estiver com o status ACTIVE.
- Ao se matricular em um curso, o progresso (progress) do aluno é iniciado em 0%.
- O sistema deve permitir que um aluno atualize o seu percentual de progresso (0 a 100) em qualquer curso no qual esteja matriculado.

4) Fila de Suporte ao User

- Qualquer usuário da plataforma pode abrir um SupportTicket, contendo um title e uma message.
- Os tickets devem ser armazenados em uma fila de atendimento para serem processados pela equipe de administradores. O atendimento deve seguir rigorosamente a **ordem de chegada** (FIFO - First-In, First-Out).

5) Relatórios e Análises da Plataforma O sistema deve ser capaz de gerar as seguintes informações analíticas:

- Uma lista de cursos pertencentes a um determinado difficultyLevel, **ordenada alfabeticamente** pelo title do curso.
- Uma relação de todos os **instrutores únicos** que ministram cursos ativos na plataforma, sem nomes repetidos.
- Um relatório que agrupe os alunos de acordo com seu subscriptionPlan.
- O cálculo da **média geral de progress**, considerando todas as matrículas de todos os alunos.

- A identificação do **aluno com o maior número de matrículas** ativas.
- **Exportação de Dados para CSV:** A plataforma precisa de uma funcionalidade de exportação que permita a um administrador gerar um arquivo CSV a partir de qualquer lista de dados (seja de Courses, Students, etc.). O administrador deve poder **escolher dinamicamente quais colunas (campos) quer no arquivo** no momento da exportação. **Nesse momento, não é necessário gerar um arquivo .csv físico: a função deve apenas retornar e exibir a estrutura do CSV formatada como uma String no console.**

2. Funcionalidades da Aplicação (Interface de Linha de Comando)

A aplicação deve ser desenvolvida como um sistema de linha de comando, com um menu que ofereça, no mínimo, as seguintes funcionalidades:

1. **Operações de Administrador (Admin)**
    - **Gerenciar Status de Cursos:** Ativar/inativar cursos existentes (não precisa implementar CRUD completo).
    - **Gerenciar Planos de Alunos:** Alterar o plano de assinatura de um aluno existente.
    - **Atender Tickets de Suporte:** Processar tickets da fila em ordem FIFO.
    - **Gerar Relatórios e Análises:** Acessar todos os relatórios da plataforma.
    - **Exportar Dados:** Gerar a String CSV com colunas selecionáveis dinamicamente.
1. **Operações do Aluno (Student)**
    - **Matricular-se em Curso:** Desde que o plano permita e o curso esteja ACTIVE.
    - **Consultar Matrículas:** Ver todos os cursos em que está matriculado e seu progresso.
    - **Atualizar Progresso:** Modificar o percentual de conclusão de um curso.
    - **Cancelar Matrícula:** Remover-se de um curso (libera vaga para planos básicos).
1. **Operações Gerais (Qualquer Usuário):**
    - **Consultar Catálogo de Cursos:** Listar cursos ativos disponíveis.
    - **Abrir Ticket de Suporte:** Criar um novo ticket para a fila de atendimento.
    - **Autenticação Simples:** Sistema básico de login por email para distinguir entre Admin e Student. Não é necessário que o usuário tenha senha, apenas o e-mail é suficiente para autenticação. Você poderá elaborar duas telas distintas ou simplesmente atribuir papeis aos usuários e fazer a verificação de permissão.

3. Lógica de Negócio e Regras

A partir dos requisitos acima, foram destacadas as seguintes regras de negócio que devem ser consideradas no processo de implementação.

Sistema de Matrículas (Enrollments)

- Um Student só pode se matricular em um Course se:
    - Seu plano de assinatura permitir (BasicPlan: máximo 3 matrículas ativas).
    - O curso estiver com status ACTIVE.
    - Não estiver já matriculado no mesmo curso.
- O progresso inicia em 0% e pode ser atualizado de 0 a 100%.

Fila de Suporte

- Tickets são processados em ordem FIFO (First-In, First-Out).
- Qualquer usuário pode abrir tickets, mas apenas Admin pode processá-los.

Relatórios e Análises O sistema deve gerar as seguintes informações utilizando **Stream API**:

- Lista de cursos por difficultyLevel, ordenada alfabeticamente.
- Relação de instrutores únicos que ministram cursos ACTIVE.
- Agrupamento de alunos por subscriptionPlan.
- Média geral de progresso de todas as matrículas.
- Aluno com maior número de matrículas ativas (retorno: Optional<Student>).

4. Requisitos de Arquitetura (A Estrutura de Camadas)

O projeto **deve** ser estruturado nos seguintes pacotes, respeitando rigorosamente a **Regra da Dependência** (setas apontam para dentro): infrastructure -> application -> domain.

1. Camada Domain

Esta é a camada mais interna. Não pode depender de NINGUÉM.

- **Pacotes Sugeridos:** domain.entities, domain.enums, domain.exceptions
- **Conteúdo:**
    - **Entidades:** Course, User, Admin, Student, SubscriptionPlan (pode ser uma interface/abstrata), Enrollment, SupportTicket.
    - **Enums:** DifficultyLevel, CourseStatus.
    - **Exceções:** EnrollmentException.java, BusinessException.java (exceções de negócio customizadas).

- **Regra:** Classes aqui não podem ter import de nada das camadas application ou infrastructure. Elas contêm a lógica de negócio mais pura (ex: a lógica de um Student para saber se pode se matricular em mais um curso, student.canEnroll()).

2. Camada Application

Esta camada orquestra as entidades para executar os Requisitos Funcionais (Casos de Uso).

- **Pacotes Sugeridos:** application.usecases, application.repositories
- **Conteúdo (UseCases):**
    - Classes que implementam *um* caso de uso específico.
    - Ex: MatricularAlunoUseCase.java, AtenderTicketUseCase.java, GerarRelatorioCursosPorNivelUseCase.java, AtualizarProgressoUseCase.java, AbrirTicketUseCase.java.
- **Conteúdo (Repositories - ABSTRAÇÕES):**
    - As **INTERFACES** que definem o contrato de persistência.
    - CourseRepository.java (ex: save, findByTitle, findByDifficultyLevel)
    - UserRepository.java (ex: save, findByEmail)
    - EnrollmentRepository.java (ex: save, findByStudent, countActiveByStudent)
    - SupportTicketQueue.java (ex: addTicket, nextTicket, isEmpty)
- **Regra:** Depende do domain, mas **NUNCA** do infrastructure. Os UseCases recebem as interfaces de Repositório por **injeção de dependência** (via construtor).

3. Camada Infrastructure (Interface Adapters)

Implementa os detalhes. É a camada que conhece o mundo exterior.

- **Pacotes Sugeridos:** infrastructure.persistence, infrastructure.ui, infrastructure.utils
- **Conteúdo (Persistence):**
    - As **implementações concretas** das interfaces de application.repositories.
    - CourseRepositoryEmMemoria.java (usa um Map<String, Course> para garantir title único).
    - UserRepositoryEmMemoria.java (usa um Map<String, User>).
    - SupportTicketQueueEmMemoria.java (usa um ArrayDeque para garantir FIFO).
- **Conteúdo (UI):**
    - ConsoleController.java: Recebe a entrada do usuário, chama o UseCase apropriado e trata as exceções (ex: EnrollmentException).

    - ConsoleView.java: Apenas imprime menus e resultados no console.

- **Conteúdo (Utils - "Drivers"):**
    - GenericCsvExporter.java: A classe que usa **Reflection**. Este é um detalhe de framework e deve ficar aqui, bem longe da lógica de negócio.

4. Camada main (Frameworks & Drivers)

O ponto de entrada da aplicação e a "Raiz de Composição".

Conteúdo:

- Main.java: O único lugar que "conhece todo mundo". É aqui que a **Injeção de Dependência manual** acontece.
- InitialData.java: A classe que popula os repositórios em memória no início.

5. Requisitos de Implementação e Ferramentas

Para este protótipo, as seguintes ferramentas e abordagens devem ser utilizadas:

- **Persistência em Memória:** Toda a persistência de dados deve ser simulada em memória utilizando as **Collections do Java**. Não é necessário usar um banco de dados real.
- **Estruturas de Dados Específicas:**
    - Para garantir a unicidade e a busca eficiente de Courses por title e Users por email, utilize a interface Map.
    - Para a listagem de instructors únicos no relatório, utilize a interface Set.
    - Para a fila de Support Tickets, utilize uma implementação da interface Queue (como LinkedList ou ArrayDeque) para garantir o comportamento FIFO.
- **Programação Funcional com Java 8+:**
    - A lógica dos relatórios (Streams) deve ser implementada **dentro dos UseCases** (application.usecases). O UseCase solicita os dados brutos ao repositório (ex: repository.findAll()) e então aplica a lógica de Stream.
- **Reflection (CSV Exporter):**
    - A classe GenericCsvExporter deve residir em infrastructure.utils.
    - A camada application (UseCases) **NÃO PODE** saber da existência dessa classe.
    - O ConsoleController (infrastructure.ui) é quem deve:
    - Chamar um UseCase (ex: GerarRelatorioCursosUseCase) para obter a lista de dados (ex: List<Course>).
    - Pegar essa lista e passá-la para o GenericCsvExporter para formatação.

- **Tratamento de Exceções:**
    - Exceções customizadas (ex: EnrollmentException) devem ser definidas no domain ou application. Os UseCases (application) as lançam, e a UI (infrastructure.ui) as captura e exibe mensagens amigáveis.

6. Modelagem da Solução

Para implementar a aplicação, utilize conceitos da Programação Orientada a Objetos (POO) com Java. **Lembre-se que a aplicação deve implementar a Clean Architecture.**

Além da implementação do código, elabore um **Diagrama de Classes UML** que represente a estrutura do sistema, demonstrando as relações entre as classes. Entregue juntamente com o diagrama uma **justificativa para suas escolhas de design** do protótipo explicando como a Regra da Dependência foi seguida, com foco em:

- Como a Injeção de Dependência foi feita no Main.java.
- Como a camada domain foi mantida pura.
- Como os detalhes (CSV, Persistência em Memória) foram isolados na camada infrastructure.

Esse exercício poderá ser feito em duplas.