TC02 - Dinamiques: conhecimento ao alcance de todos

1. Problema

Hoje, professores(as) da rede pública de educação muitas vezes não têm ferramentas para postarem suas aulas e transmitirem conhecimento para os alunos e alunas de forma prática, centralizada e tecnológica.

---

2. Solução proposta

📌 **OBSERVAÇÕES!**

1. Nessa sessão há apenas um breve relato do problema e com um esboço da solução proposta, porém, detalhes completos como requisitos, jornada atual e da solução se encontram na sessão requisitos nessa documentação.
2. Alguns papéis ou funcionalidades não serão implementados inicialmente, porém estão mapeados para serem desenvolvidos futuramente, por exemplo: como não há o perfil administrador na API, o papel de gerir categorias também ficará com o docente e não com o administrador.

Afim de possibilitar a **comunicação** e **integração** mais efetiva entre o docente e o aluno, assim como possibilitar o consumo de conteúdos adicionais e ou complementares de um determinado tema pelos alunos, será desenvolvido um sistema de **blogging dinâmico**.

O sistema contará com dois segmentos principais:

2.1. Painel administrativo do blog (CMS)

2.1.1. Administradores

Usuários com o perfil *“operacional”* irão poder por meio de uma **interface simples** e **intuitiva**, se autenticar e gerenciar:

▼ **administradores**

- criar (1)
- listar (2)
- atualizar (3)
- excluir (4)

CRITÉRIOS!

1. Não é permitido criar administradores de nível superior, exemplo: *“gestor”* e quando um administrador é criado, é obrigatório em se criar um usuário para ele.
2. Lista todos os administradores com o perfil *“operacional”* ou lista filtrados por nome ou lista apenas um por id.
3. Não é permitido atualizar administradores de nível superior, exemplo: *“gestor”*.
4. Não é permitido excluir administradores de nível superior, exemplo: *“gestor”* e quando um administrador é excluído *(soft delete)* o seu usuário é automaticamente inativado.

- Nesse primeiro momento não será demonstrado o fluxo de um administrador com perfil *“gestor”*, porém, futuramente poderá ser implementado e este além de ter acesso completo no sistema e atribuir níveis de acesso, também poderá gerenciar a tabela de perfil de usuários.

▼ **docentes**

- criar (1)

- listar (2)
- atualizar
- excluir (3)

CRITÉRIOS!

1. Quando um docente é criado, é obrigatório em se criar um usuário para ele.
2. Lista todos os docentes ou lista filtrados por nome ou lista apenas um por id.
3. Quando um docente é excluído *(soft delete)* o seu usuário é automaticamente inativado.

▼ **categorias**

- criar
- listar (1)
- atualizar
- excluir (2)

CRITÉRIOS!

1. Lista todas as categorias ou lista filtradas por nome ou lista apenas uma por id.
2. Não é permitido excluir uma categoria que esteja associada a um artigo.

2.1.2. Docentes

Usuários com o perfil *"docente"* irão poder por meio de uma **interface simples** e **intuitiva**, se autenticar e gerenciar:

▼ **artigos**

- criar (1)
- listar (2)
- atualizar (3)
- excluir (4)
- publicar ou despublicar

📣 **CRITÉRIOS!**

1. Ao se criar um artigo, o docente é automaticamente associado como autor do mesmo e é obrigatório associar o artigo a uma categoria já existente.
2. Lista todos os artigos de sua autoria ou lista filtrados por título ou lista apenas um por id.
3. Não é permitido atualizar artigos que não sejam de sua autoria.
4. Não é permitido excluir artigos que não sejam de sua autoria.

- O docente irá acessar o url do CMS do blog e poderá se autenticar ao receber os dados por um administrador da escola. Nesse primeiro momento, o docente não poderá alterar seus próprios dados de cadastro e quem poderá fazer esse papel é o administrador.

2.2. Website

Poderá ser acessado por **qualquer usuário não autenticado** e **não** necessariamente um **aluno** matriculado na instituição de ensino. É por isso que o website aberto ao público deverá ser otimizado para mecanismos de **SEO**, **acessível**, **responsivo** e com uma **interface amigável**, para que esteja disponível para qualquer pessoa que tenha o interesse em aprender.

2.2.1. Página inicial do blog

2.2.1.1. Arquitetura da informação

- **Header**
  - Logotipo
  - Navegação principal
    - [Link] - página inicial do blog
- **Sessão de destaques (três últimos artigos recém publicados. Será implementado posteriormente)**
  - Destaques 1, 2 e 3
    - Título
    - Breve descrição
    - Imagem de fundo
    - Tempo de leitura
    - [Link] - página de detalhes do artigo
- **Sessão de artigos**
  - Filtros
    - Categorias ativas
    - Barra de pesquisa por título do artigo
    - Paginação
  - Listagem de todos os artigos ativos ou apenas filtrados
    - Título
    - Descrição resumida
    - Categoria
    - Autor
    - Data de publicação
    - Tempo de leitura
    - Imagem principal
- **Footer**
  - Direitos autorais

2.2.2. Página com detalhes do artigo

2.2.2.1. Arquitetura da informação

- **Header**
  - Logotipo
  - Navegação principal
    - [Link] - página inicial do blog
- **Introdução do artigo**
  - Título
  - Categoria
    - [Link] - página inicial do blog com filtro na categoria
  - Autor
  - Data de publicação
  - Tempo de leitura
  - Imagem principal de fundo
  - Breadcrumb
    - [Link] - página inicial do blog
    - [Link] - página inicial do blog com filtro na categoria (será implementado posteriormente)
    - Título do artigo atual
- **Conteúdo do artigo**
  - Conteúdo renderizado a partir de um markdown
- **Artigos relacionados**
  - Sugestão de três artigos de mesma categoria do atual
- **Footer**
  - Direitos autorais

3. Domain driven design

3.1. Contextos

Partindo de uma visão macro, uma escola é formada por diversas áreas de interesse ou departamentos e considerando o domínio: **ambiente educacional** de uma escola, foi possível identificar alguns dos possíveis subdomínios, conforme imagem abaixo.

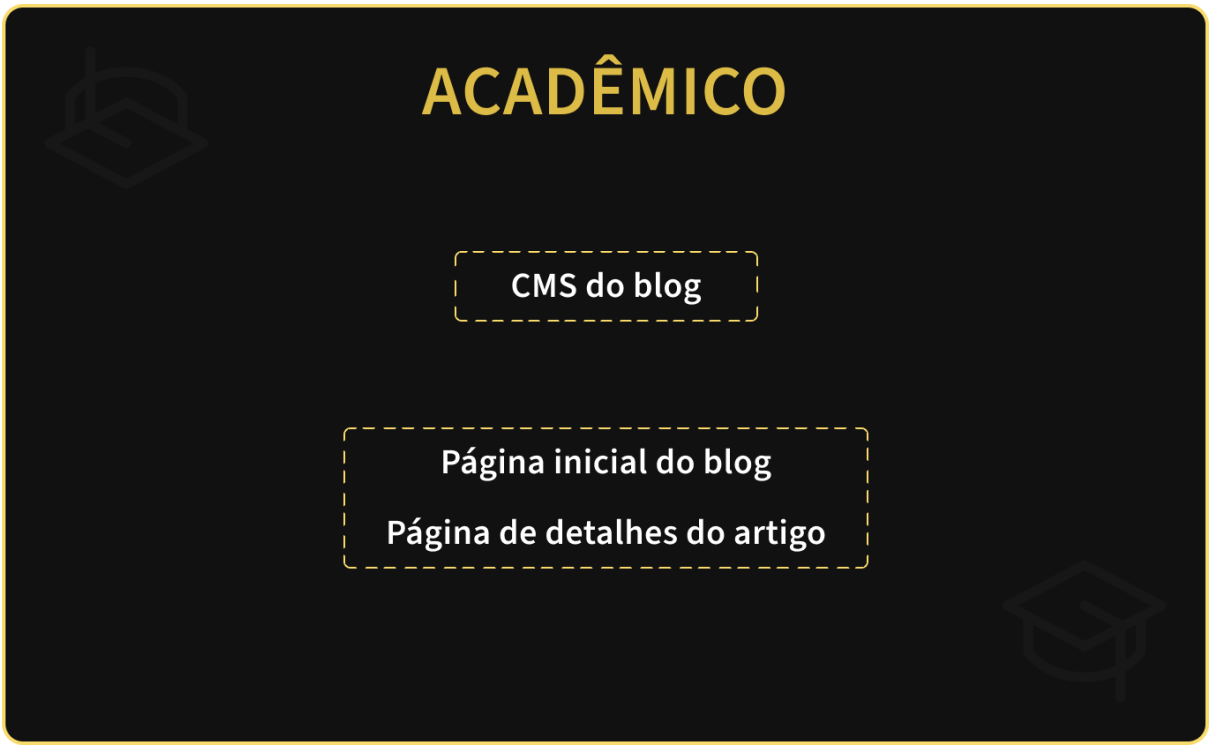
[Contextos do ambiente educacional]

Após a compreensão geral do **domínio**, podemos focar no levantamento de atividades e responsabilidades do subdomínio: **acadêmico**, que são:

- CMS do blog
- Página inicial do blog e página de detalhes do artigo

📣 CONSIDERAÇÕES!

1. É importante ressaltar que embora a área de interesse seja o **acadêmico** e o escopo ficaria em torno de artigos, ainda assim, será considerada e implementada a possibilidade de se gerenciar **administradores (futuramente)**, **docentes** e **categorias**.
2. Em uma situação real, a gestão de administradores e docentes poderiam ficar em um escopo diferente do **acadêmico** e o CMS do blog a ser desenvolvido, além de gerir artigos e categorias, se comunicaria com uma outra plataforma e base de dados existente para se obter dados de administradores e docentes, relacionando-os com os artigos quando necessário. **Mas para fins didáticos**, como serão **implementados itens adicionais**, tais como gestão de administradores **(futuramente)** e docentes, logo essas gestões estarão disponibilizadas no mesmo CMS do blog **(futuramente)** e não sendo representada a comunicação com um outro **subdomínio** e outra plataforma ou base de dados que presumidamente em um contexto real, já seria existente. **É por isso que para representar de forma correta o cenário nessa simulação em uma situação didática**, será apresentado apenas o subdomínio acadêmico e no CMS do blog será possível gerir: administradores **(futuramente)**, docentes, categorias e artigos.



[Subdomínio acadêmico]

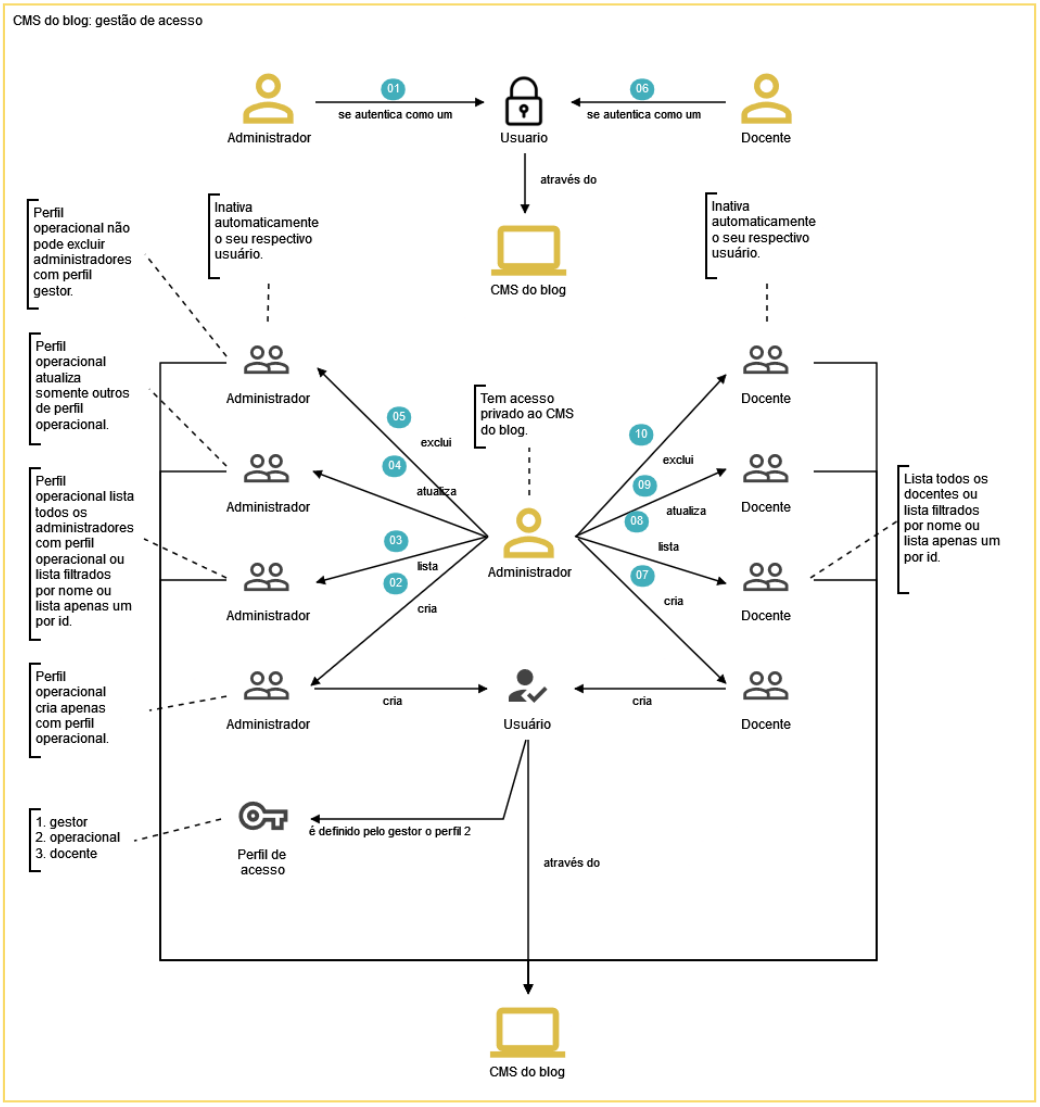
3.2. Storytelling

A seguir será demonstrado os principais modelos extraídos e desenvolvidos após diversas reuniões com os ***domain experts***.

CONSIDERAÇÕES!

1. É importante ressaltar que alguns atores ou ações não serão implementadas na API inicialmente, porém, já foram mapeados justamente para que futuramente sejam desenvolvidos, assim como quando houver sido iniciado e finalizado o front-end da aplicação.
2. Para simular um ambiente real, nem todas as opções de filtragem ou ordenação de dados foi mencionada na etapa de construção do *storytelling*, já que não necessariamente seriam relatadas, porém, opções de filtrar por status, ordenar por nome, título, etc, irão constar como um requisito mais adiante nessa documentação, assim como sua implementação.

3.2.1. Gestão de acessos



[Storytelling: gestão de acessos]

Ator	Ações
Administrador	1 - Se autentica como um usuário 2 - Cria administrador 3 - Lista administrador 4 - Atualiza administrador 5 - Exclui administrador 7 - Cria docente 8 - Lista docente 9 - Atualiza docente 10 - Exclui docente

Docente	6 - Se autentica como um usuário
CMS do blog	No diagrama é representado como um ator para ser possível representar a interação entre o objeto de trabalho Usuário e ele, no software *egon.io*

📣 **COMENTÁRIOS NAS AÇÕES ACIMA!**

2 - Administrador com perfil *operacional* cria um administrador e um usuário preparado somente para o perfil *operacional* e um administrador com perfil *gestor* irá atribuir o perfil *operacional* a esse usuário criado.

3 - Administrador com perfil *operacional* lista todos os administradores com perfil *operacional* ou lista filtrados por nome ou lista apenas um por id.

4 - Administrador com perfil *operacional* atualiza somente outros administradores com perfil *operacional*.

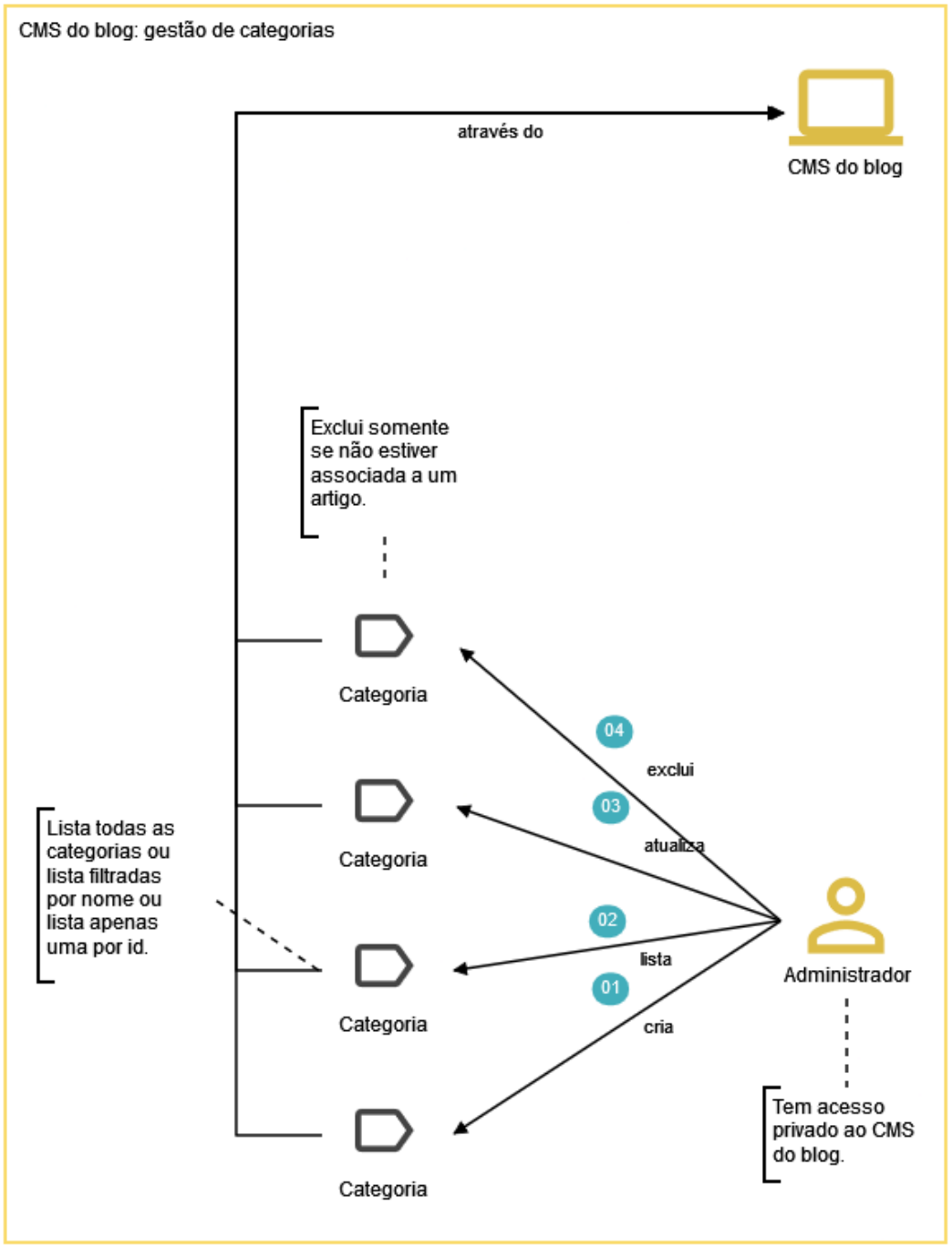
5 - Administrador com perfil *operacional* exclui (soft delete) apenas administradores com perfil *operacional* e quando essa ação é feita, o usuário deste é inativado automaticamente.

7 - Quando um docente é criado, um administrador com perfil *gestor* irá atribuir o perfil *docente* a esse usuário criado.

8 - Lista todos os docentes ou lista filtrados por nome ou lista apenas um por id.

10 - Quando um docente é excluído (soft delete), o usuário deste é inativado automaticamente.

3.2.2. Gestão de categorias



[Storytelling: gestão de categorias]

Ator	Ações
Administrador	1 - Cria categoria 2 - Lista categoria

	3 - Atualiza categoria 4 - Exclui categoria
CMS do blog	No diagrama é representado como um ator para ser possível representar a interação entre o objeto de trabalho Categoria e ele, no software *egon.io*

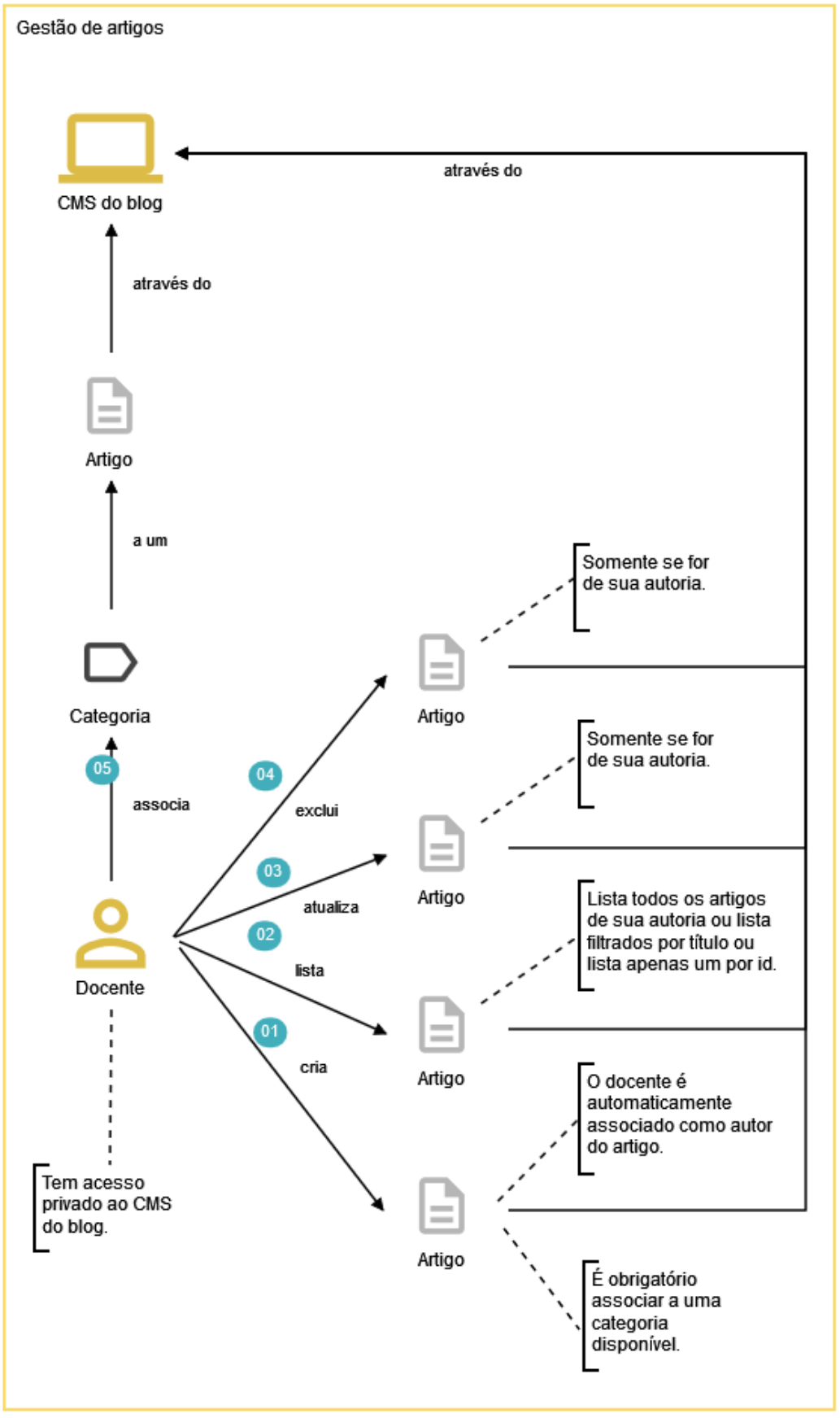
📣 **COMENTÁRIOS NAS AÇÕES ACIMA!**

- É importante ressaltar que na API desenvolvida, inicialmente quem fará a gestão de categorias será o próprio docente e não um administrador.
- Todas as operações abaixo requerem que o administrador esteja previamente logado no CMS do blog.

2 - Lista todas as categorias ou lista filtradas por nome ou lista apenas uma por id.

4 - Exclui categoria somente quando ela não estiver associada a um artigo.

3.2.3. Gestão de artigos


Gestão de artigos
CMS do blog
através do
através do
Artigo
a um
Categoria
05
associa
Docente
Tem acesso privado ao CMS do blog.
04
exclui
03
atualiza
02
lista
01
cria
Artigo
Artigo
Artigo
Artigo
Somente se for de sua autoria.
Somente se for de sua autoria.
Lista todos os artigos de sua autoria ou lista filtrados por título ou lista apenas um por id.
O docente é automaticamente associado como autor do artigo.
É obrigatório associar a uma categoria disponível.

[Storytelling: gestão de artigos]

Ator	Ações
Docente	1 - Cria artigo 2 - Lista artigo 3 - Atualiza artigo 4 - Exclui artigo 5 - Associa categoria a um artigo
CMS do blog	No diagrama é representado como um ator para ser possível representar a interação entre o objeto de trabalho Artigo e ele, no software *egon.io*

📣 **COMENTÁRIOS NAS AÇÕES ACIMA!**

- A restrição de listar ou deletar somente de sua autoria não foi implementada inicialmente.
- Todas as operações abaixo requerem que o docente esteja previamente logado no CMS do blog.

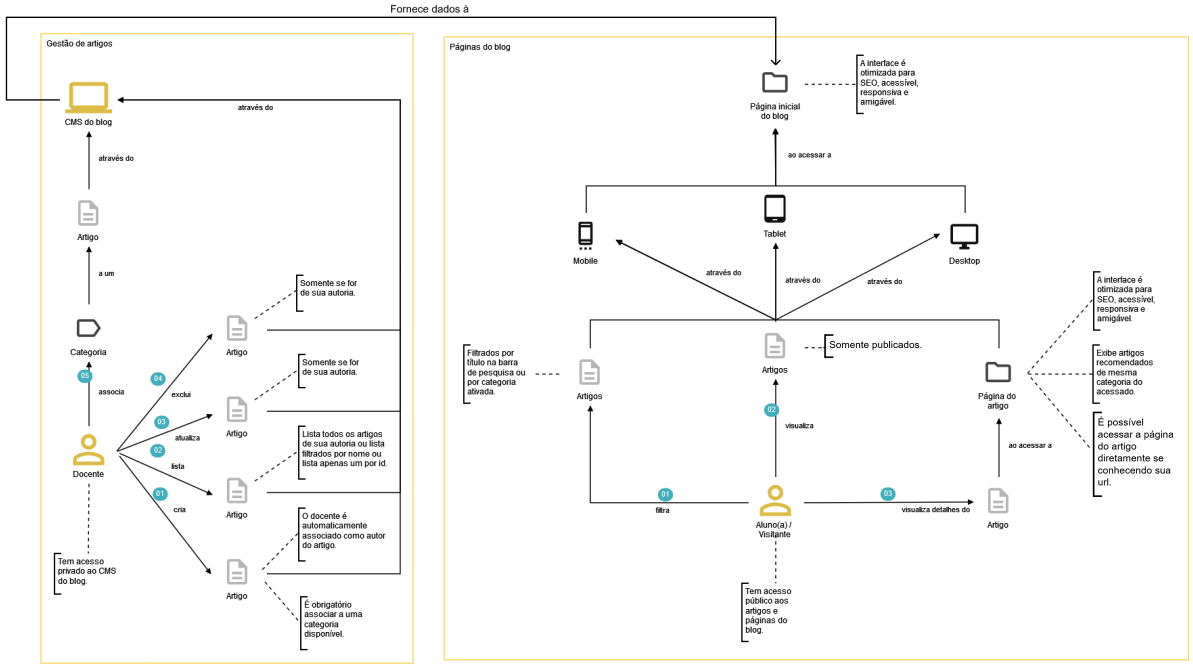
1 - O docente é automaticamente associado como autor do artigo e é obrigatório associar o artigo a uma categoria disponível.

2 - Lista todos os artigos de sua autoria ou lista filtrados por título ou lista apenas um por id.

3 - Atualiza artigo somente se for de sua autoria.

4 - Exclui (soft delete) artigo somente se for de sua autoria.

3.2.4. Integração entre gestão de artigos e páginas do blog



[Storytelling: integração entre gestão de artigos e páginas do blog]

> **IMPORTANTE!**
>
> Não será descrito novamente o módulo de gestão de artigos, pois o mesmo já foi documentado logo acima e a intenção de representar tanto ele, quanto o módulo de páginas do blog abaixo é devido a estarem relacionados.

Ator	Ações
Aluno(a) / Visitante	1 - Filtra artigo 2 - Visualiza artigo 3 - Visualiza detalhes do artigo

📣 **COMENTÁRIOS NAS AÇÕES ACIMA!**

- A parte do *front-end* ainda não foi implementada, mas consta nessa documentação para fins de relação e histórico com a API.
- O módulo de gestão de artigos descrito anteriormente fornece dados ao módulo de páginas do blog.
- O Aluno(a) / Visitante possuem acesso público aos artigos e páginas do blog (a interface é otimizada para SEO, acessível, responsiva e amigável) e este acesso pode ocorrer por meio de dispositivos *mobile*, *tablet* e *desktop*.

1 - Filtra por título na barra de pesquisa ou por categoria ativada na página inicial do blog.

2 - Visualiza artigos somente ativos (com o status publicado) na página inicial do blog.

3 - Visualiza detalhes na página do artigo. Recebe recomendações de artigos de mesma categoria do acessado e há a possibilidade em acessar a página do artigo diretamente a partir de sua url conhecida.

3.3. Requisitos

📣 **CONSIDERAÇÕES!**

- Nem todas as personas abaixo ou fluxo da solução foram implementados inicialmente, porém, trata-se de mapeamento para histórico e implementação futura.
- O docente é que inicialmente está encarregado de gerenciar categorias.

3.3.1. Persona

- Administrador(a)

    - Gestor
    - Operacional
- Docente
- Aluno(a)
- Visitante externo

3.3.2. Problema

Atualmente, docentes não possuem ferramentas capazes de possibilitar a transmissão de conhecimentos aos aluno(a)s de forma prática, centralizada e tecnológica.

3.3.3. Jornada atual

1. Um docente deseja compartilhar informações adicionais a respeito de um determinado assunto além do que se é ministrado em aula, ou então, complementar o assunto da aula com novas referências, ou então, explicar um assunto de uma dúvida muito recorrente e realizada por vários alunos.
2. Ao final da aula ministrada em sala, o docente solicita aos aluno(a)s que se puderem, permaneçam em sala para que ele possa discursar.
3. Nem sempre todos os aluno(a)s podem estender o seu horário e permanecer em sala.
4. Somente alguns aluno(a)s puderam assistir o conteúdo adicional ministrado pelo docente e consequentemente gerando uma defasagem de conhecimento entre a turma.

3.3.4. Objetivo com a solução

Possibilitar a comunicação e integração centralizada e efetiva entre os docentes e aluno(a)s, para que estes possam consumir conteúdos adicionais e ou complementares de um determinado assunto, sem serem impactados devido a necessidade de estender a permanência em sala ao final da aula. Além disso, os docentes também serão beneficiados, uma vez que poderão

atender a uma dúvida comum de tal forma que será atingido um maior número de aluno(a)s de uma única vez.

Para isso, será desenvolvido um sistema de *blogging* dinâmico, possibilitando que docentes possam gerenciar artigos que serão consumidos pelos aluno(a)s a qualquer momento. Além de que o acesso às páginas do blog serão públicas e assim, qualquer visitante que deseja aprender, poderá acessar o conteúdo da plataforma e consequentemente favorecendo os índices e métricas de ensino no país.

3.3.5. Jornada da solução

▼ **Administrador**

- Se autentica como *“operacional”* na área administrativa do blog
  - Gerencia administradores
    - Criar (com usuário preparado para receber o perfil *“operacional”*)
    - Listar (todos com o perfil *“operacional”*, filtrados por nome, status ou por id)
    - Atualizar (somente com o perfil *“operacional”*)
    - Excluir (somente com o perfil *“operacional”*)
  - Gerencia docentes
    - Criar (com usuário preparado para receber o perfil *“docente”*)
    - Listar (todos, filtrados por nome, status ou por id)
    - Atualizar
    - Excluir
  - Gerencia categorias
    - Criar
    - Listar (todas, filtradas por nome, status ou por id)
    - Atualizar
    - Excluir (somente se não estiver associada a um artigo)

- Se autentica como *“gestor”* na área administrativa do blog com o perfil gestor para gerenciar perfis e todas as demais ações existentes.
    - Atribui os perfis *“operacional”* ou *“docente”* para os usuários existentes e requisitados pelos administradores operacionais

▼ **Docente**

- Se autentica como *“docente”* na área administrativa do blog
    - Gerencia artigos
        - Criar (automaticamente se vincula como autor e é obrigatório associar a uma categoria já existente)
        - Listar (todos de sua autoria, filtrados por título, status de publicação ou por id)
        - Atualizar (somente de sua autoria)
        - Excluir (somente de sua autoria)
        - Publicar ou despublicar

▼ **Aluno(a) / Visitante**

- Página do blog
    - Filtrar artigos por
        - Título na barra de pesquisa
        - Categoria ativa
    - Navega para página com detalhes do artigo
- Página do artigo
    - Visualizar o conteúdo do artigo
    - Visualizar recomendação de outros artigos de mesma categoria para a leitura

3.4. Requisitos da solução

CONSIDERAÇÕES!

- Nem todos os requisitos ou parte do sistema foram implementados, porém estão mapeados para serem desenvolvidos futuramente.
- As páginas inicialmente não fazem parte da implementação, mas sim somente a API.

3.4.1. Funcionais

▼ **CMS do blog**

- RF001 - Autenticar-se na plataforma e implementar níveis de acesso (administrador ou docente)
- RF002 - Criar administrador
- RF003 - Listar todos os administradores com perfil *“operacional”*
- RF004 - Filtrar e listar administradores por status
- RF005 - Filtrar e listar administrador com perfil *“operacional”* por id
- RF006 - Filtrar e listar administradores com perfil *“operacional”* por nome
- RF007 - Filtrar e listar administradores por página
- RF008 - Ordenar e listar administradores por id, nome, status ou criado em
- RF009 - Atualizar administrador com perfil *“operacional”*
- RF010 - Excluir administrador com perfil *“operacional”* e automaticamente o seu usuário (soft delete)
- RF011 - Criar docente
- RF012 - Listar todos os docentes
- RF013 - Filtrar e listar docentes por status
- RF014 - Filtrar e listar docente por id
- RF015 - Filtrar e listar docentes por nome

- RF016 - Filtrar e listar docentes por página
- RF017 - Ordenar e listar docentes por id, nome, status ou criado em
- RF018 - Atualizar docente
- RF019 - Excluir docente e automaticamente o seu usuário (soft delete)
- RF020 - Criar usuário ao se criar um novo administrador ou docente com exceção do perfil *“gestor”*
- RF021 - Filtrar e listar usuário após consultar um administrador ou docente por id com exceção do perfil *“gestor”*
- RF022 - Atualizar usuário após consultar um administrador ou docente por id com exceção do perfil *“gestor”*
- RF023 - Excluir usuário após consultar para deleção um administrador ou docente com exceção do perfil *“gestor”* (soft delete)
- RF024 - Criar categoria
- RF025 - Listar todas as categorias
- RF026 - Filtrar e listar categorias por status
- RF027 - Filtrar e listar categoria por id
- RF028 - Filtrar e listar categorias por nome
- RF029 - Filtrar e listar categorias por página
- RF030 - Ordenar e listar categorias por id, nome, status ou criado em
- RF031 - Atualizar categoria
- RF032 - Excluir categoria
- RF033 - Associar categoria a um artigo
- RF034 - Criar artigo
- RF035 - Associar docente automaticamente ao artigo criado
- RF036 - Listar todos os artigos
- RF037 - Filtrar e listar artigos por status de publicação
- RF038 - Filtrar e listar artigo por id
- RF039 - Filtrar e listar artigos por título
- RF040 - Filtrar e listar artigos por página

- RF041 - Ordenar e listar artigos por id, título, categoria, status ou criado em
- RF042 - Atualizar artigo
- RF043 - Excluir artigo (soft delete)
- RF044 - Implementar interface gráfica para cada página (principal, administradores, docentes, categorias e artigos)

▼ **Página inicial do blog**

- RF045 - Filtrar e listar artigos por título
- RF046 - Filtrar e listar artigos por categoria ativa
- RF047 - Filtrar e listar artigos por página
- RF048 - Listar todos os artigos ativos
- RF049 - Implementar interface gráfica com componentes essenciais (header, menu, categorias, artigos, footer)

▼ **Página do artigo**

- RF050 - Renderizar conteúdo interno do artigo (markdown)
- RF051 - Recomendar artigos de mesma categoria para leitura
- RF052 - Implementar interface gráfica com componentes essenciais (header, menu, hero, detalhes do artigo, artigos recomendados, footer)

3.4.2. Não funcionais

▼ **CMS do blog**

- RNF001 - Segurança dos dados (não permitir que senhas sejam expostas)
- RNF002 - Segurança dos dados (criptografar e descriptografar senhas)

- RNF003 - Segurança dos dados (armazenar informações para auditoria: status, criado por, criado em, atualizado por, atualizado em)
- RNF004 - Não permitir que um administrador de mesmo nome seja criado
- RNF005 - Não permitir que um docente de mesmo nome seja criado
- RNF006 - Não permitir que um docente liste artigos de outra autoria
- RNF007 - Não permitir que um docente atualize artigo de outra autoria
- RNF008 - Não permitir que um docente exclua artigo de outra autoria
- RNF009 - Não permitir que uma categoria inativada seja associada a um artigo
- RNF010 - Não permitir que uma categoria seja excluída se estiver associada a um artigo
- RNF011 - Não permitir que uma categoria de mesmo nome seja criada
- RNF012 - Não permitir que um artigo de mesmo título seja criado
- RNF013 - Interface acessível, responsiva e amigável
- RNF014 - Exibir datas no formato brasileiro

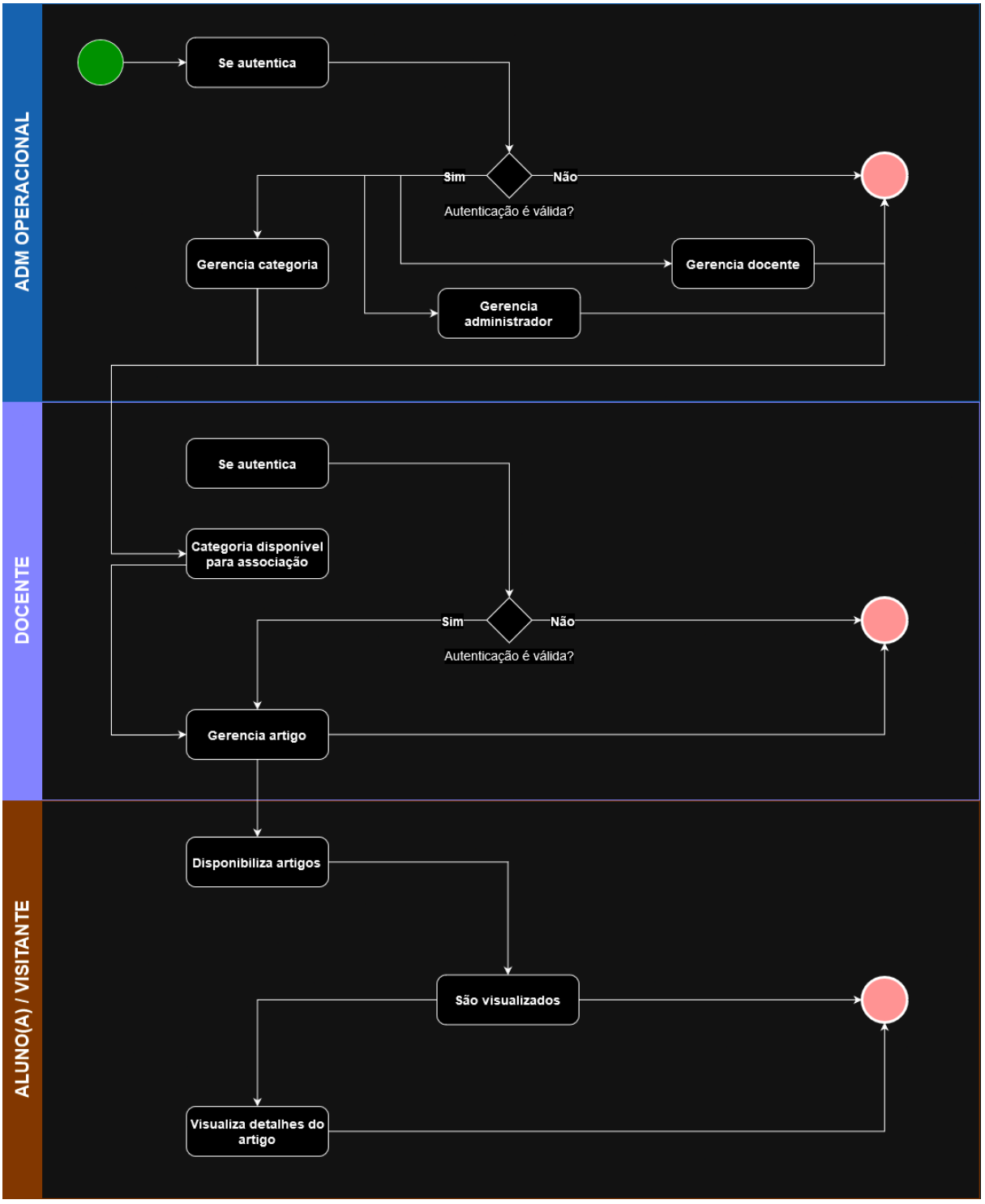
▼ **Páginas públicas do blog**

- RNF015 - Interface acessível, responsiva e amigável
- RNF016 - Otimização para SEO
- RNF017 - Boa performance na entrega das páginas
- RNF018 - Converter e exibir minutos em horas se necessário em um artigo
- RNF019 - Exibir datas no formato brasileiro
- RNF020 - i18n (possibilidade de alterar o idioma do webiste)

3.5. Esboço da solução

📣 CONSIDERAÇÕES!

1. O fluxo da gestão de administradores, perfil de usuário e docentes não foram demonstrados abaixo, mas serão parte integrantes da implementação.



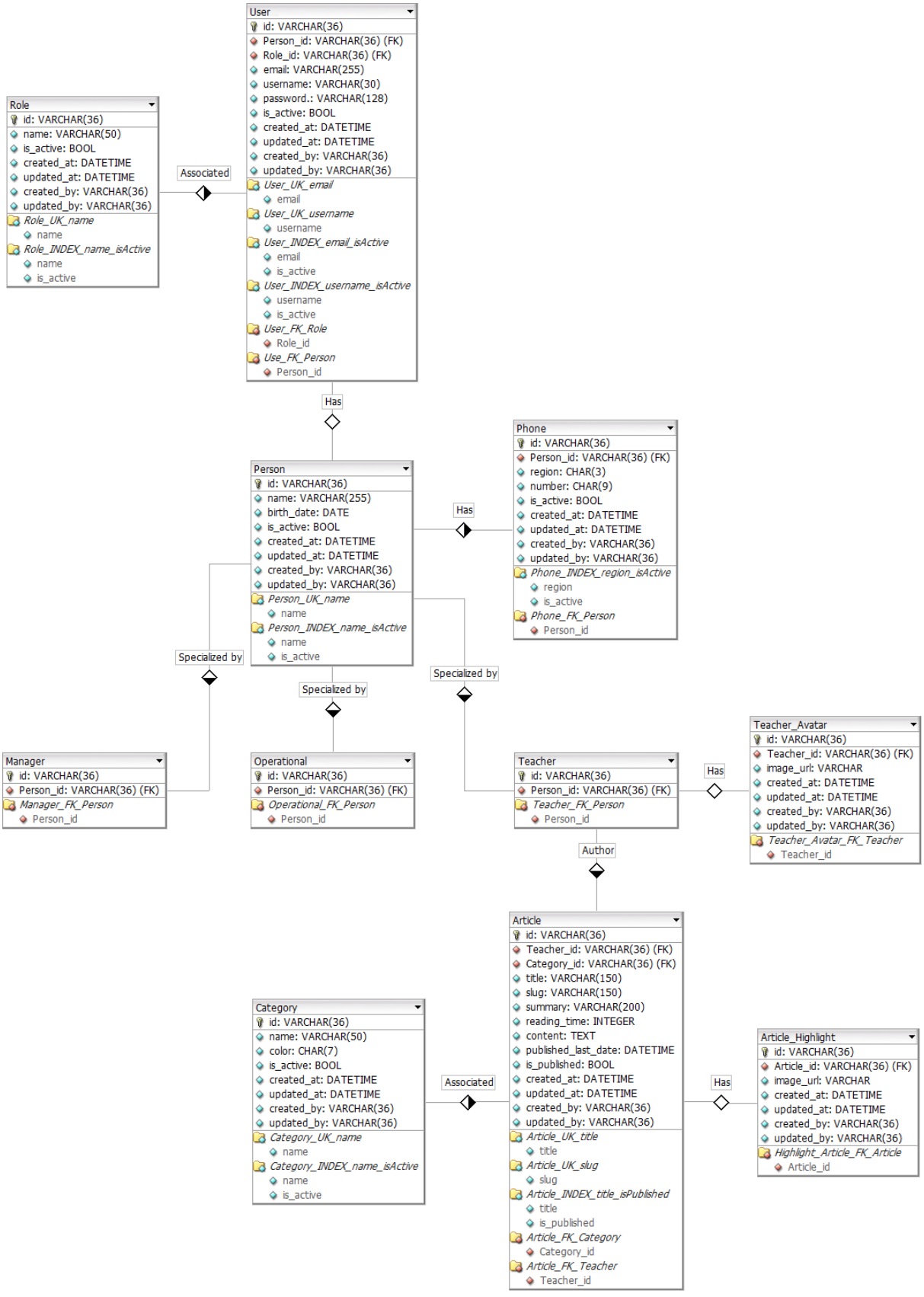
[Esboço da solução]

4. Banco de dados

4.1. Modelagem lógica

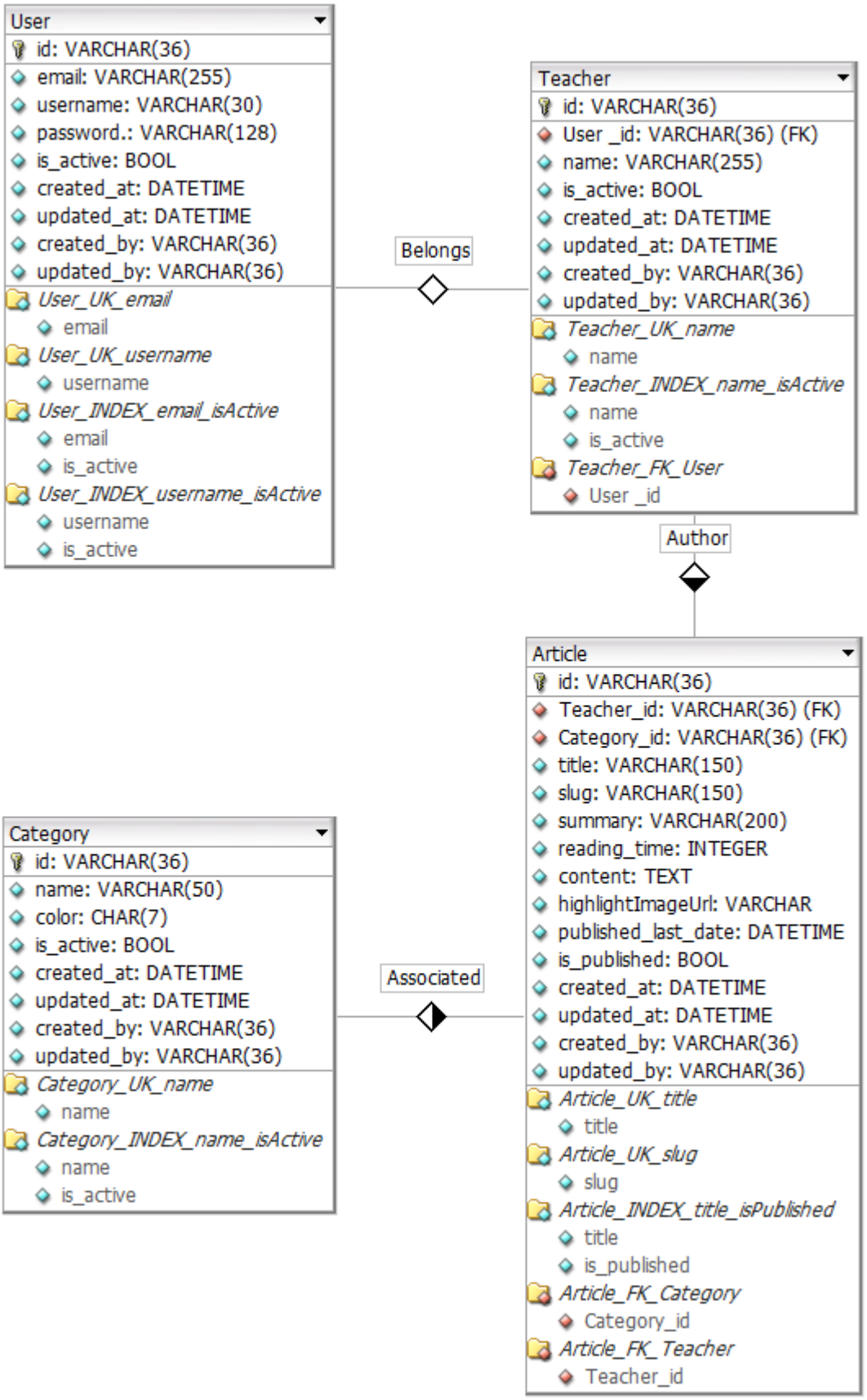
A seguir é representado o **DER (Diagrama Entidade Relacionamento)** com as principais entidades envolvidas na modelagem do banco normalizado, para um modelo previsto completo a ser implementado futuramente.

Conforme ilustração, haverá definição de *roles* para os usuários que se autenticam na aplicação e assim, cada um terá uma ação específica e permitida a desempenhar na plataforma.



[DER: modelo previsto]

A seguir é representado outro **DER**, porém em uma versão simplificada e adotada na modelagem para a API implementada nesse módulo.



[DER: modelo simplificado]

📣 **CONSIDERAÇÕES!**

1. Em todas as entidades foram implementados os atributos de auditoria para manter a segurança e integridade dos dados (`created_at` , `updated_at` , `created_by` , `updated_by`).
2. Foi implementado índices nos principais campos de todas as entidades para garantir o desempenho e performance no tempo de resposta.
3. Em todas as entidades com o atributo `is_active` foi definido o `default value` igual a `TRUE` , assim como no `is_published` .
4. Nos campos `created_at` e `updated_at` foi definido o `default value` igual a `CURRENT()` ,

4.2. Relacionamentos

4.2.1. Usuário x Docente

- **1 : 1** um usuário pertence a **um único** docente, assim como um docente possui um único usuário.

4.2.2. Docente x Artigo

- **1 : N** um docente é autor de **muitos** artigos, assim como um artigo é da autoria de **um único** docente.

4.2.3. Categoria x Artigo

- **1 : N** uma categoria é associada em **muitos** artigos, assim como um artigo é definido por **uma única** categoria.

4.3. Restrições

Para garantir a integridade dos dados, foram definidas restrições nas seguintes entidades.

4.3.1. Usuário

- User_UK_email: chave única no campo `email` o que impede a criação de e-mails idênticos.
- User_UK_username: chave única no campo `username` o que impede a criação de nomes de usuários idênticos.

4.3.2. Docente

- Teacher_UK_name: chave única no campo `name` o que impede a criação de nomes idênticos.

4.3.3. Categoria

- Category_UK_name: chave única no campo `name` o que impede a criação de nomes idênticos.

4.3.4. Artigo

- Article_UK_title: chave única no campo `title` o que impede a criação de títulos idênticos.
- Article_UK_slug: chave única no campo `slug` o que impede a criação de *slugs* idênticos.

4.4. Índices

Para fornecer uma melhor performance na pesquisa de dados com filtro comum, foram definidos os seguintes índices nas entidades.

4.4.1. Usuário

- User_INDEX_email_isActive: `email` e `is_active`
- User_INDEX_username_isActive: `username` e `is_active`

4.4.2. Docente

- Teacher_INDEX_name_isActive: `name` e `is_active`

4.4.3. Categoria

- Category_INDEX_name_isActive: `name` e `is_active`

4.4.4. Artigo

- Article_INDEX_title_isPublished: `title` e `is_published`

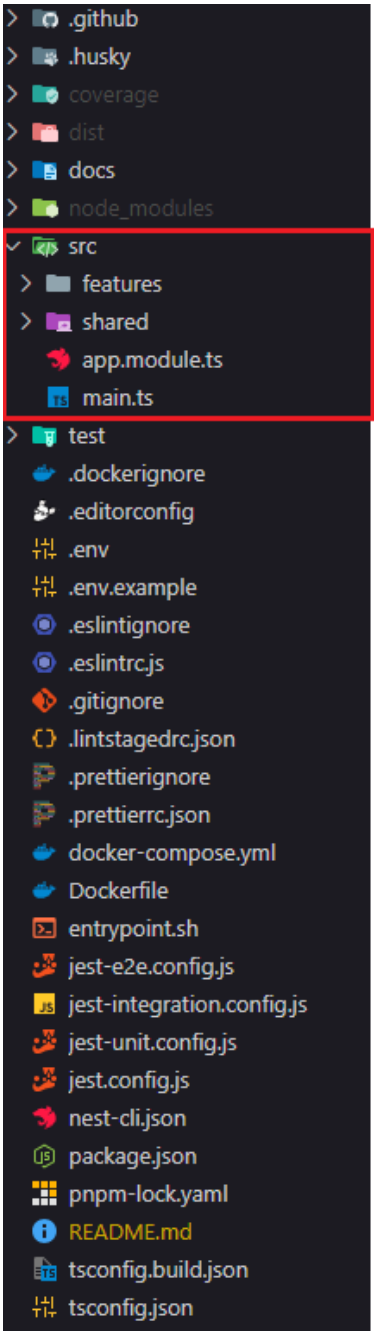
---

5. API

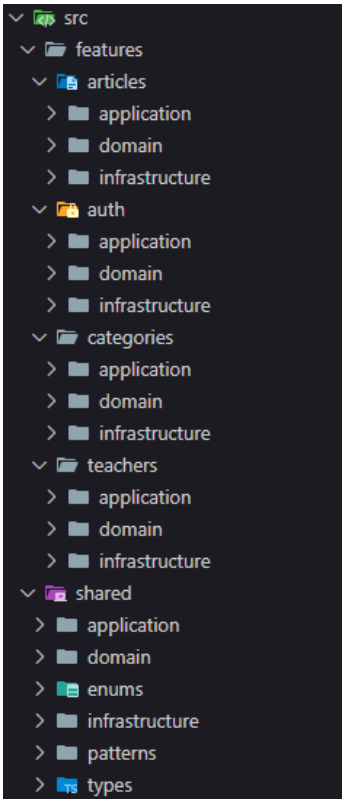
5.1. Arquitetura

A estrutura de pastas e separação de responsabilidades entre os arquivos da aplicação, estão estruturados no modelo DDD (domain driven design).

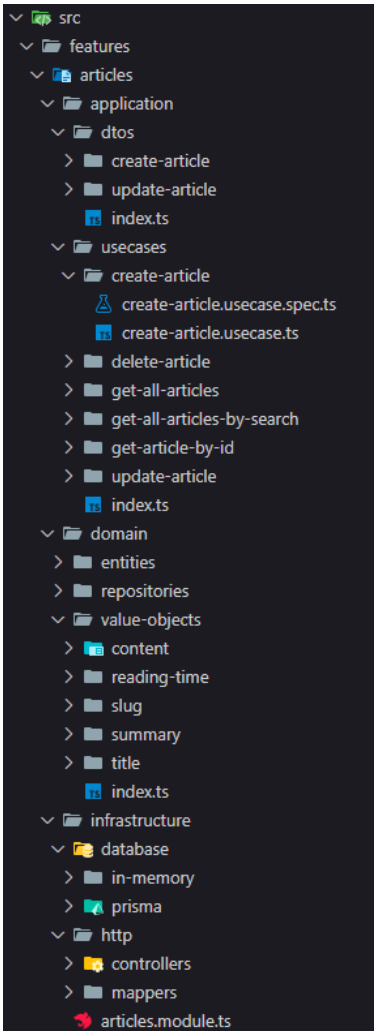
- Cada recurso irá conter pelo menos parte ou todas as pastas essenciais, tais como:
  - **/application**
  - **/domain**
  - **/infrastructure**
- Na pasta **/application**, poderá conter:
  - /dtos
  - /usecases
- Na pasta **/domain**, poderá conter:
  - /entities
  - /repositories
  - /value-objects
- Na pasta **/infrastructure**, poderá conter:
  - /database (*in-memory* e *prisma*)
  - /http (*controllers* e *mappers*)



[API: visão geral de pastas com DDD]



[API: visão de pastas específicas com DDD]

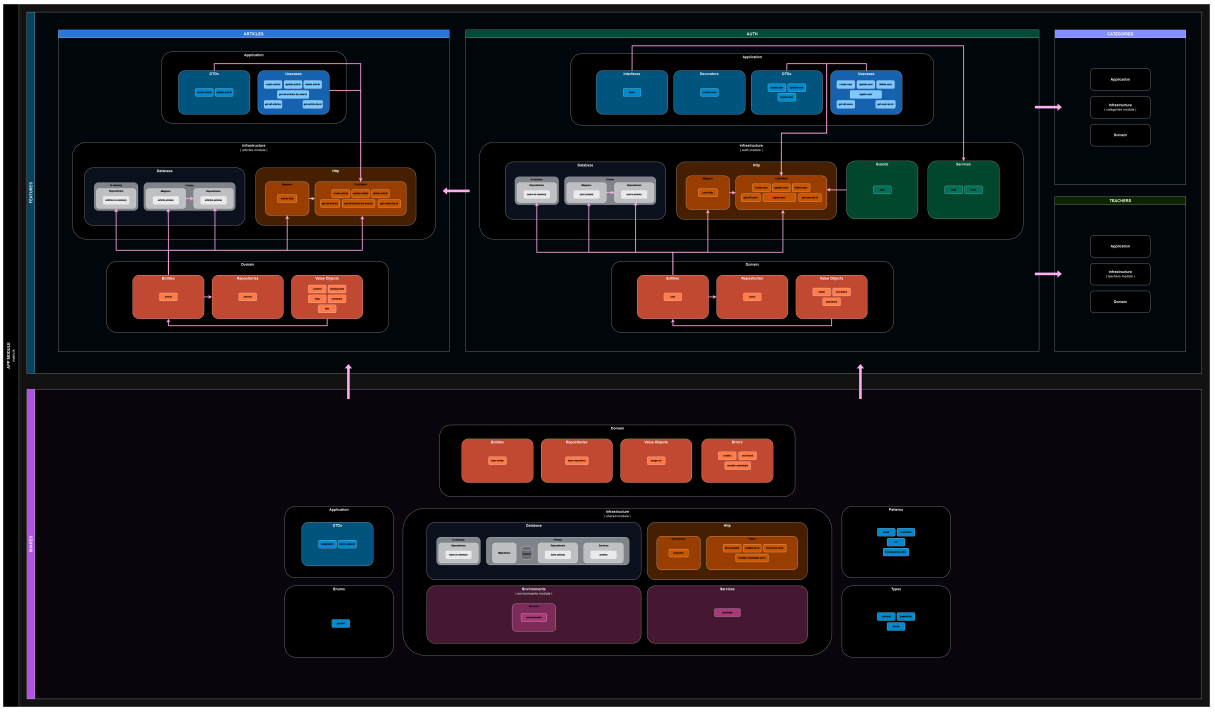


[API: visão de pastas de articles]

A seguir é representada a arquitetura da aplicação onde o módulo `shared` fornece recursos reusáveis e compartilhados com todos os demais módulos internos de `features`.

- O módulo `auth` além de ser responsável pelo gerenciamento de `users` concentra todas as **regras**, **serviços, decoradores** e **guarda de rotas** que são utilizados por outros módulos de `features`, como por exemplo, `articles`, `categories`, `teachers`.

- O módulo `shared` além de fornecer recursos que são reutilizáveis por outros módulos, também concentra arquivos que servem como um modelo a ser estendido por outros arquivos, por exemplo:
  - `base-entity` (propriedades base que toda entidade irá ter, tais como: `isActive`, `createdAt`, `updatedAt`, `createdBy`, `updatedBy`)
  - `base-repository` (métodos de contrato base que outros repositórios irão estender, tais como: `create`, `getAll`, `getById`, `update`, `delete`, `softDelete`)
  - `errors`
  - `interceptors`
  - `filters`
  - etc



[Arquitetura da api]

📣 **CONSIDERAÇÕES!**

1. Baixe a imagem da arquitetura acima para uma melhor visualização.
2. Não foi representado os recursos internos de `categories` e `teachers`, uma vez que são idênticos ao `articles` e que possuem as mesmas pastas e arquivos do DDD.

5.2. Padrões de projeto e boas práticas

5.2.1. Padrões de projeto

- DDD (domain driven design)
- SOLID
- Clean code
- DTO
- Repository in memory
- Repository pattern
- Adapters
- DRY (dont repeat yourself)

5.2.2. Boas práticas

- Tipagem estática com `typescript` .
- Nomenclatura padronizada e uso de sufixos nos arquivos para facilitar a busca.
- `Commit` semântico e padronizado com o `git-commit-msg-linter`.
- `ESLint` (com `airbnb`), `Prettier` e `editorcode` para padronizar a formatação do código.

- Ordenação automática ao salvar e separação em grupos nas importações dos arquivos de `libs`, `features`, `shared`, etc com o `eslint-plugin-import-helpers` .
- Validação de testes e formatação dos arquivos antes de chegarem ao repositório do `Github` com `husky` e `lintstaged`.
- `Alias paths` para evitar escrita longa nas importações dos arquivos.
- Versionamento da API.
- Resposta de retorno padronizada com `statusCode`, `method`, `path`, `message`, `error` e `data` .
- Remoção da propriedade `X-powered-by` no `header`, para evitar exposição de qual tecnologia foi utilizada.
- Adição da propriedade `Access-Control-Allow-Methods` no `header` de quais métodos são permitidos na API.
- Utilização de variáveis de ambiente e gerenciadas na aplicação por módulo denominado `environments`.
- Utilização de imagens do `docker` certificadas e reduzidas.
- Separação em etapas com `Builder` e posteriormente `Runner` no `Dockerfile`.
- Criação de usuário **não** `root` para não fornecer acessos além do previsto definido no `Dockerfile` .
- Descrição das etapas do workflow de `CI / CD`.
- Definição de `cache` nas etapas do workflow de `CI / CD`.
- Definição de variáveis para a reutilização e utilização de `secrets` no workflow de `CI / CD`.

5.3. Segurança

Foram aplicados conceitos de segurança para garantir a integridade da aplicação em seu funcionamento pleno.

5.3.1. Autenticação

- Autenticação de usuário por meio da obtenção de `token` **(JWT)**.
- Política de expiração de `token` para evitar utilização indevida.

- Guarda de rota para extrair `token` do `header` e verificar sua validade para liberar ou bloquear o acesso a determinado recurso ou ação.
- Futuramente será implementado o conceito de `roles` para restringir e definir acessos aos usuários da aplicação.

5.3.2. Proteção aos dados sigilosos

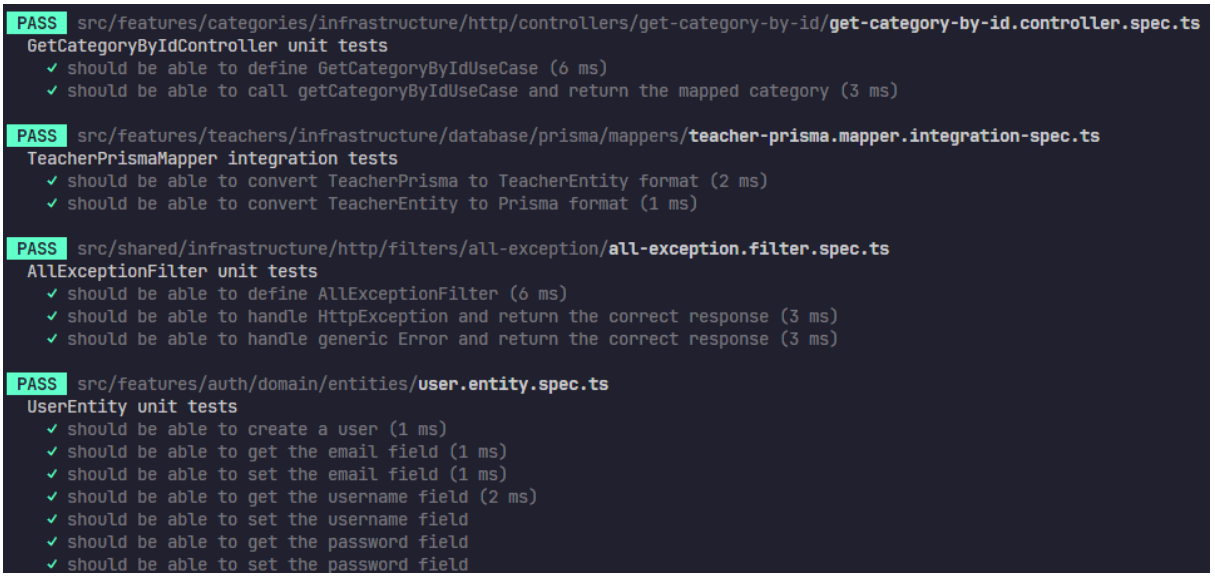
- **Criptografia** e **descriptografia** de senha do usuário.
- Armazenamento de senha criptografada no banco de dados.
- **Não** exposição do campo `password` em consultas do tipo `GET` em `users` , mesmo que criptografada.
- Implementação de `interceptor` para garantir a devolução de reposta correta e tratada para cada tipo de erro ou exceção, garantindo a não exposição de detalhes de erros internos.
- **Não** exposição de especificação de campo fornecido incorretamente de `email` ou `password`.
- Remoção da propriedade `X-powered-by` no `header`, para evitar exposição de qual tecnologia foi utilizada
- Adição da propriedade `Access-Control-Allow-Methods` no `header` de quais métodos são permitidos na API.

5.3.3. Campos de auditoria

- Implementação de campos para rastreio e histórico de modificações e modificadores, tais como: `isActive` , `createdAt` , `createdBy` , `updatedAt` e `updatedBy` .

5.4. Testes

- Foi utilizado o framework `Jest` para se efetuar testes **unitários** e de **integração**, assim como o `Fakerjs` para a geração de dados fictícios e `mocks`.
- Foi adotada a padronização do nome e descrição dos testes para facilitar a estruturação e identificação dos itens.



[API: exemplo da padronização em testes]

5.4.1. Unitários

Todas as `controllers`, `dtos`, `mappers`, `usecases`, `entities`, `value-objects`, `repositories` de recursos, `interceptors`, `filters`, `patterns` e `services` estão `100%` testadas conforme relatório de cobertura.

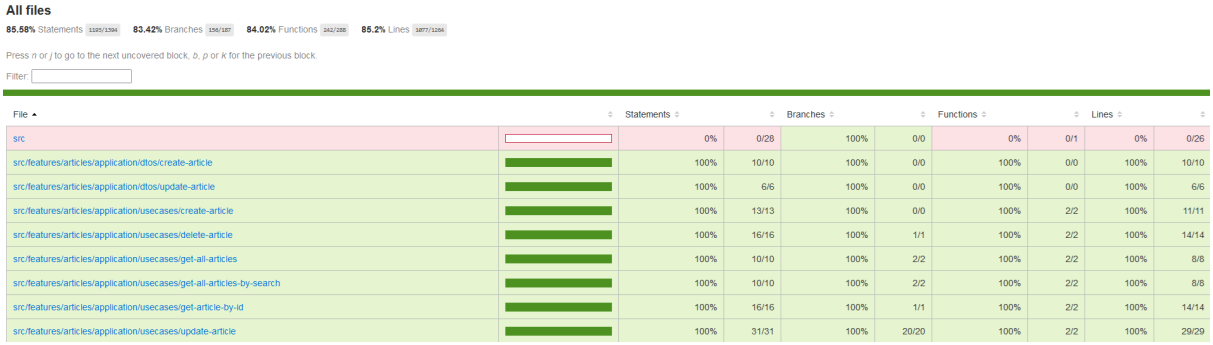
5.4.2. Integração e e2e

Foram aplicados testes de **integração** nos arquivos de mapeamento do prisma.

Demais testes de **integração** e **e2e** estão sendo aplicados e serão atualizados futuramente aqui, aumentando-se assim ainda mais a cobertura e integridade da aplicação.

5.4.3. Cobertura

Atualmente chegou-se a mais de `85%` de cobertura de testes, sendo que demais estão sendo feitos e será atualizado posteriormente aqui.



All files

85.58% Statements 1193/1394 **83.42%** Branches 156/187 **84.02%** Functions 242/288 **85.2%** Lines 1077/1264

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
src		0%	0/28	100%	0/0	0%	0/1	0%	0/26
src/features/articles/application/dtos/create-article		100%	10/10	100%	0/0	100%	0/0	100%	10/10
src/features/articles/application/dtos/update-article		100%	6/6	100%	0/0	100%	0/0	100%	6/6
src/features/articles/application/usecases/create-article		100%	13/13	100%	0/0	100%	2/2	100%	11/11
src/features/articles/application/usecases/delete-article		100%	16/16	100%	1/1	100%	2/2	100%	14/14
src/features/articles/application/usecases/get-all-articles		100%	10/10	100%	2/2	100%	2/2	100%	8/8
src/features/articles/application/usecases/get-all-articles-by-search		100%	10/10	100%	2/2	100%	2/2	100%	8/8
src/features/articles/application/usecases/get-article-by-id		100%	16/16	100%	1/1	100%	2/2	100%	14/14
src/features/articles/application/usecases/update-article		100%	31/31	100%	20/20	100%	2/2	100%	29/29

[API: cobertura de testes]

CONSIDERAÇÕES!

- Para a verificação do relatório de testes na íntegra ou para a geração de um novo, basta acessar o repositório da API indicada na apresentação e seguir o passo a passo para a sua obtenção.

5.5. Docker

Foi utilizado o `docker` e `docker-compose` para facilitar o processo de `build` da aplicação e evitar problemas entre ambientes diferentes ou incompatíveis.

A seguir, está a implementação dos arquivos do `Dockerfile` , `docker-compose.yml` e `entrypoint.sh` e que também seguem boas práticas, tais como:

`Dockerfile`

- Utilização de imagens certificadas e reduzidas.
- Separação das etapas em `builder` inicialmente e posteriormente `runner` .
- Criação de usuário **não** `root` .

`docker-compose.yml`

- Utilização de imagens certificadas.
- Utilização de variáveis de ambiente.

`entrypoint.sh`

- Oferece um intervalo de tempo para que as `migrations` e `seeds` sejam executadas apenas quando o banco de dados `Postgresql` estiver disponível.

▼ Arquivo `Dockerfile`

```
FROM node:18-alpine AS builder

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install --legacy-peer-deps

COPY . .

COPY src/shared/infrastructure/database/prisma/schema.pris
COPY src/shared/infrastructure/database/prisma/migrations
COPY src/shared/infrastructure/database/prisma/seeds/seed.

RUN npx prisma generate --schema=./prisma/schema.prisma

RUN npm run build

FROM node:18-alpine AS runner

RUN addgroup -S appgroup && adduser -S appuser -G appgroup

WORKDIR /usr/src/app

COPY --from=builder /usr/src/app/dist ./dist
COPY --from=builder /usr/src/app/node_modules ./node_modul
COPY --from=builder /usr/src/app/prisma ./prisma
COPY entrypoint.sh .

ENV NODE_ENV=production

EXPOSE 3000
```

```
USER appuser

CMD ["sh", "-c", "./entrypoint.sh && node dist/main.js"]
```

▼ Arquivo `docker-compose.yml`

```
services:
  api:
    container_name: dinamiques-nest-api
    build:
      context: .
      dockerfile: Dockerfile
    image: felipedr7/dinamiques-nest-api:latest
    ports:
      - '3000:3000'
    env_file:
      - .env
    environment:
      DATABASE_URL: ${DATABASE_URL}
    restart: unless-stopped
    depends_on:
      - postgres
    networks:
      - dinamiques-network

  postgres:
    container_name: dinamiques-nest-postgres
    image: postgres:14
    ports:
      - '5432:5432'
    env_file:
      - .env
    environment:
      POSTGRES_USER: ${DATABASE_USER}
      POSTGRES_PASSWORD: ${DATABASE_PASSWORD}
      POSTGRES_DB: ${DATABASE_NAME}
    restart: always
    volumes:
```

```
      - pgdata:/var/lib/postgresql/data
    networks:
      - dinamiques-network

networks:
  dinamiques-network:
    driver: bridge

volumes:
  pgdata:
```

▼ Arquivo `entrypoint.sh`

```
#!/bin/sh

set -e

PROTOCOL=${DATABASE_PROTOCOL:-postgres}
HOST=${DATABASE_HOST:-postgres}
PORT=5432
TIMEOUT=60
WAIT_INTERVAL=1

echo "Waiting for the database at ${HOST}:${PORT} to becom

COUNTER=0

while ! nc -z "$HOST" "$PORT"; do
  if [ "$COUNTER" -ge "$TIMEOUT" ]; then
    echo "Timeout reached: database did not become availab
    exit 1
  fi
  echo "Database is not available yet. Retrying in ${WAIT_
  sleep "$WAIT_INTERVAL"
  COUNTER=$((COUNTER + WAIT_INTERVAL))
done

echo "Database is available."
```

```
echo "Deploying prisma migrations..."
npx prisma migrate deploy --schema=./prisma/schema.prisma
echo "Migrations deployed successfully."

echo "Seeding the database if necessary..."
node ./prisma/seeds/seed.mjs
echo "Database seeded successfully."
```

5.6. CI / CD

Embora esteja sendo utilizado o `husky` com o `lintstaged` para já validar o arquivo tanto em sua estrutura de código, quanto na validação dos testes após cada `push`. Há ainda um `workflow` configurado com o `github actions` responsável por efetuar os fluxos abaixo, assim como executar os devidos `scripts` definidos em `package.json` com o sufixo `ci`.

- **Integração contínua**
  - Validação de testes **unitários** a cada `push` e em qualquer `branch` .
  - Validação de testes de **integração** e de **e2e** em cada `pull request` .
- **Entrega contínua**
  - Após ser efetuado um `push` na `branch main` e já ter sido executado todo o fluxo de integração contínua, será feito o `build` prévio da aplicação e disponibilizada a imagem no `dockerhub`.

A seguir, está a implementação dos arquivos de cada etapa do fluxo e que também seguem boas práticas, tais como descrição das etapas e utilização de `cache` , definição e reutilização de variáveis e utilização de `secrets`.

▼ Workflow: `run-unit-tests.yml`

```
name: CI - unit tests
```

```yaml
on: [push]

jobs:
  run-unit-tests:
    name: Run unit tests
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install pnpm
        run: npm install -g pnpm

      - name: Cache pnpm modules
        uses: actions/cache@v4
        with:
          path: ~/.pnpm-store
          key: ${{ runner.os }}-pnpm-${{ hashFiles('**/pnp
          restore-keys: |
            ${{ runner.os }}-pnpm-

      - name: Install dependencies
        run: pnpm install --frozen-lockfile

      - name: Run unit tests
        run: pnpm run test:unit:ci
        env:
          DATABASE_IN_MEMORY: true
```

▼ Workflow: `run-integration-e2e-tests.yml`

```yaml
name: CI - integration and e2e tests

on: [pull_request]

jobs:
  run-integration-and-e2e-tests:
    name: Run integration and e2e tests
    runs-on: ubuntu-latest

    services:
      postgres:
        image: bitnami/postgresql
        ports:
          - 5432:5432
        env:
          POSTGRES_USER: ${{ secrets.DATABASE_USER }}
          POSTGRES_PASSWORD: ${{ secrets.DATABASE_PASSWORD
          POSTGRES_DB: ${{ secrets.DATABASE_NAME }}

    env:
      DATABASE_URL: '${{ secrets.DATABASE_PROTOCOL }}://${

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install pnpm
        run: npm install -g pnpm

      - name: Cache pnpm modules
        uses: actions/cache@v4
        with:
          path: ~/.pnpm-store
```

```
          key: ${{ runner.os }}-pnpm-${{ hashFiles('**/pnp
          restore-keys: |
            ${{ runner.os }}-pnpm-

      - name: Install dependencies
        run: pnpm install --frozen-lockfile

      - name: Generate prisma client
        run: npx prisma generate --schema ./src/shared/inf
        env:
          DATABASE_URL: ${{ env.DATABASE_URL }}

      - name: Run integration tests
        run: pnpm run test:integration:ci
        env:
          DATABASE_IN_MEMORY: false
          DATABASE_URL: ${{ env.DATABASE_URL }}

      - name: Run e2e tests
        run: pnpm run test:e2e:ci
        env:
          DATABASE_IN_MEMORY: false
          DATABASE_URL: ${{ env.DATABASE_URL }}
```

▼ Workflow: `run-build-and-deploy.yml`

```
name: CD - build and deploy

on:
  push:
    branches:
      - main

jobs:
  run-build-and-deploy:
    name: Run build and deploy
    runs-on: ubuntu-latest
```

```
    env:
      DATABASE_URL: '${{ secrets.DATABASE_PROTOCOL }}://${
      DATABASE_USER: '${{ secrets.DATABASE_USER }}'
      DATABASE_PASSWORD: '${{ secrets.DATABASE_PASSWORD }}
      DATABASE_NAME: '${{ secrets.DATABASE_NAME }}'
      DOCKER_HUB_IMAGE_NAME: 'felipedr7/dinamiques-nest-ap

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up docker
        uses: docker/setup-buildx-action@v3

      - name: Install docker compose
        run: |
          sudo apt-get update
          sudo apt-get install -y docker-compose

      - name: Log in to docker hub
        run: |
          echo "${{ secrets.DOCKER_HUB_TOKEN }}" | docker

      - name: Build docker images
        run: |
          echo "DATABASE_URL=${DATABASE_URL}" > .env
          echo "DATABASE_USER=${DATABASE_USER}" >> .env
          echo "DATABASE_PASSWORD=${DATABASE_PASSWORD}" >>
          echo "DATABASE_NAME=${DATABASE_NAME}" >> .env
          docker-compose -f docker-compose.yml build

      - name: Check if the build was successful
        run: |
          IMAGE_ID=$(docker images -q $DOCKER_HUB_IMAGE_NA
          if [ -z "$IMAGE_ID" ]; then
            echo "Error: image '$DOCKER_HUB_IMAGE_NAME' no
            echo "Build may have failed or the image may n
            exit 1
```

```
          fi
          echo "Image '$DOCKER_HUB_IMAGE_NAME' built succe

      - name: Push docker image to docker hub
        run: |
          docker push $DOCKER_HUB_IMAGE_NAME
```

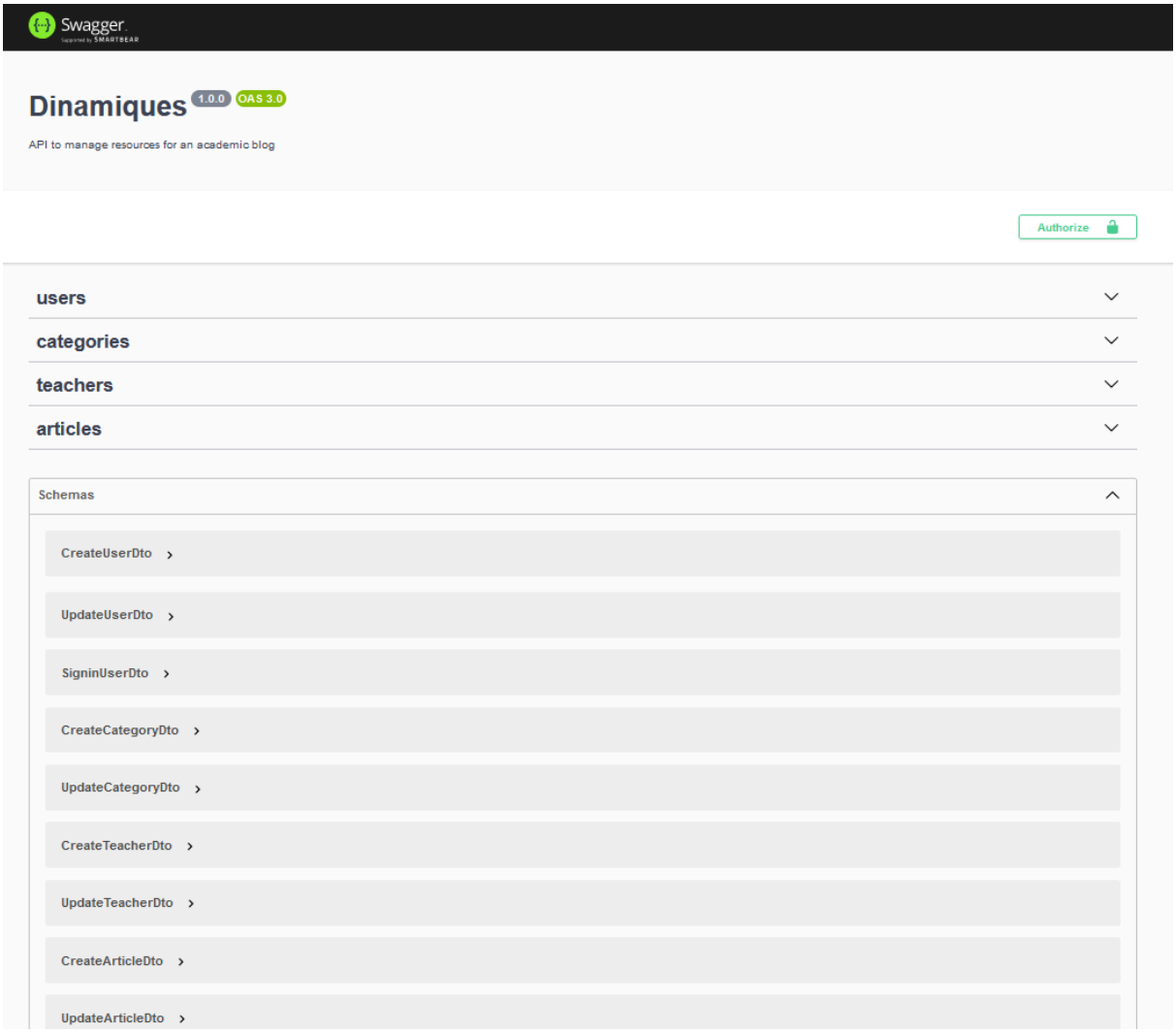
5.7. Swagger

Ao executar a aplicação a documentação da API estará disponibilizada na rota: `http://localhost:3000/api/v1/docs`

Os recursos existentes até o momento são: `users`, `categories`, `teachers` e `articles`, sendo que algumas rotas são restritas com autenticação e outras são públicas.

CONSIDERAÇÕES!

- No repositório do projeto apresentado, há um usuário pré-cadastrado (inserido por meio da ação de `seed` no `prisma`) com o seu respectivo acesso, para ser possível efetuar a autenticação e acessar os demais recursos existentes.



[API: documentação com o swagger]

5.8. ORM

Foi utilizado o `Prisma` como o **ORM (object relational mapper)** da aplicação e todas as suas definições e implementações se encontram na pasta `/shared/infrastructure/database/prisma/` . Cada parte específica está separada em pasta própria, tais como: `migrations`, `repositories`, `seeds` e `services`.

Conforme exemplo abaixo, cada entidade possui campos mapeados e de acordo com as especificações definidas na modelagem lógica do banco de dados.

▼ Implementação do `scheme.prisma`

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id        String    @id @default(uuid())
  email     String    @unique @db.VarChar(255)
  username  String    @unique @db.VarChar(30)
  password  String    @db.VarChar(128)
  isActive  Boolean   @default(true) @map("is_active")
  createdAt DateTime  @default(now()) @map("created_at")
  updatedAt DateTime? @updatedAt @map("updated_at")
  createdBy String    @map("created_by")
  updatedBy String?   @map("updated_by")
  teacher   Teacher[]

  @@index([email, isActive])
  @@index([username, isActive])
}

model Teacher {
  id        String    @id @default(uuid())
  userId    String?   @map("user_id")
  name      String    @unique @db.VarChar(255)
  isActive  Boolean   @default(true) @map("is_active")
  createdAt DateTime  @default(now()) @map("created_at")
  updatedAt DateTime? @updatedAt @map("updated_at")
  createdBy String    @map("created_by")
  updatedBy String?   @map("updated_by")
  articles  Article[]
  User      User?     @relation(fields: [userId], referenc

  @@index([name, isActive])
```

```
}

model Category {
  id        String    @id @default(uuid())
  name      String    @unique @db.VarChar(50)
  color     String    @default("#000000") @db.Char(7)
  isActive  Boolean   @default(true) @map("is_active")
  createdAt DateTime  @default(now()) @map("created_at")
  updatedAt DateTime? @updatedAt @map("updated_at")
  createdBy String    @map("created_by")
  updatedBy String?   @map("updated_by")
  articles  Article[]

  @@index([name, isActive])
}

model Article {
  id                String    @id @default(uuid())
  teacherId         String    @map("teacher_id")
  categoryId        String    @map("category_id")
  title             String    @unique @db.VarChar(150)
  slug              String    @unique @db.VarChar(150)
  summary           String    @db.VarChar(200)
  readingTime       Int       @map("reading_time")
  content           String    @db.Text
  highlightImageUrl String?   @map("highlight_image_url")
  publishedLastDate DateTime  @map("published_last_date")
  isPublished       Boolean   @default(true) @map("is_publ
  createdAt         DateTime  @default(now()) @map("create
  updatedAt         DateTime? @updatedAt @map("updated_at"
  createdBy         String    @map("created_by")
  updatedBy         String?   @map("updated_by")
  Teacher           Teacher   @relation(fields: [teacherId
  Category          Category  @relation(fields: [categoryI

  @@index([title, isPublished])
}
```

5.9. Ferramentas de monitoramento e logs

Futuramente será implementado um módulo para esse fim.

---

6. Ferramentas e tecnologias utilizadas

6.1. Storytelling

- Egon.io

6.2. Modelagem de banco de dados

- DB Designer

6.3. Arquitetura e fluxograma de processos

- Draw.io

6.4. Documentação

- Notion

6.5. API

- Docker
- CI/CD: GitHubActions
- Nest.js
- Typescript
- JWT
- Bcryptjs
- UUID

- ClassTransformer
- ClassValidator
- ESLint
- Prettier
- TSConfigPaths
- GitCommitMsgLinter
- LintStaged
- Husky
- Jest
- Faker.js
- Prisma
- Swagger