Imagem de desenho animado

Descrição gerada automaticamente com confiança baixa

**DOCUMENTAÇÃ DO**

**PROJETO GREENSEED**

[ESTRUTURA DO PROJETO 4](#_Toc180886220)

[Solution 'GreenSeed' (3 of 3 projects) 4](#_Toc180886221)

[AZURE SQL DATABASE 5](#_Toc180886222)

[Area de BackOffice : **Os utilizadores** 6](#_Toc180886223)

[SeedData.cs 7](#_Toc180886224)

[Program.cs 8](#_Toc180886225)

[Area de BackOffice : **As Categorias** 9](#_Toc180886226)

[CategoryController.cs 9](#_Toc180886227)

[Repository.cs 10](#_Toc180886228)

[IRepository.cs 10](#_Toc180886229)

[Como o Código Funciona Junto? (CategoryController.cs e Repository.cs) 10](#_Toc180886230)

[Area de BackOffice : **Os Produtos** 11](#_Toc180886231)

[ProductController.cs 11](#_Toc180886232)

[Area de BackOffice : **As Encomendas** 13](#_Toc180886233)

[OrderController.cs 13](#_Toc180886234)

[SessionExtensions.cs 15](#_Toc180886235)

[Resumo Geral 15](#_Toc180886236)

[Area de BackOffice : As Encomendas (**TransportadoraApp**) 16](#_Toc180886237)

[TransportadoraApp 16](#_Toc180886238)

[OrderStatusMessage.cs 16](#_Toc180886239)

[QueueProcessor.cs 17](#_Toc180886240)

[Integração com a Entidade Order 18](#_Toc180886241)

[Diagrama Simplificado do Fluxo 18](#_Toc180886242)

[Área para upload de foto livre 19](#_Toc180886243)

[CommunityPhotoUploadController.cs 19](#_Toc180886244)

[Área de Desafios 22](#_Toc180886245)

[Admin - ChallengesController.cs 22](#_Toc180886246)

[User - ChallengesController.cs 24](#_Toc180886247)

[AZURE SQL DataBase 25](#_Toc180886248)

[TransportadoraApp 25](#_Toc180886249)

[GreenSeed.Tests : Testes 26](#_Toc180886250)

[Visão Geral do GreenSeed.Tests 26](#_Toc180886251)

[Tecnologias Utilizadas 26](#_Toc180886252)

[Explicação dos Tipos de Teste 26](#_Toc180886253)

[Testes Unitarios 27](#_Toc180886254)

[Testes de Integração 27](#_Toc180886255)

[Criar e Implantar um Azure Web App 28](#_Toc180886256)

[Criar um Azure Web App 28](#_Toc180886257)

[A Pipeline (Que está no repositório) 31](#_Toc180886258)

[Tecnologias utilizadas 32](#_Toc180886259)

[Tecnologias a ser implementadas com as Melhorias, e Boas Praticas das paginas seguintes 32](#_Toc180886260)

[Boas Praticas Que Faltou Implementar 33](#_Toc180886261)

[Gestão de Utilizadores 33](#_Toc180886262)

[SeedData.cs 33](#_Toc180886263)

[Gestão de Categorias e Gestão de Produtos 33](#_Toc180886264)

[Gestão de Produtos 33](#_Toc180886265)

[Credênciais 33](#_Toc180886266)

[AZURE SQL DataBase 33](#_Toc180886267)

[Melhorias 34](#_Toc180886268)

[Implementação de Containerização 34](#_Toc180886269)

[Como seria feito? 34](#_Toc180886270)

[Recursos Utilizados 35](#_Toc180886271)

# ESTRUTURA DO PROJETO

# Solution 'GreenSeed' (3 of 3 projects)

├── GitHub Actions

├── GreenSeed

│ ├── Connected Services

│ ├── Dependencies

│ ├── Properties

│ ├── wwwroot

│ ├── Areas

│ ├── Controllers

│ ├── Data

│ ├── Models

│ ├── Services

│ ├── Views

│ ├── appsettings.json

│ ├── Dockerfile

│ ├── Program.cs

│ └── ScaffoldingReadMe.txt

├── GreenSeed.Tests

│ ├── Controllers

│ ├── Integration

│ ├── Services

│ └── xunit.runner.json

├── TransportadoraApp

│ ├── Dependencies

│ ├── Data

│ ├── Models

│ ├── appsettings.json

│ └── Program.cs

AZURE SQL DATABASE

Diagrama

Descrição gerada automaticamente

# Area de BackOffice : **Os utilizadores**

**ApplicationUser.cs**

Este é o modelo que representa o usuário na aplicação. Ele estende a classe IdentityUser do ASP.NET Identity, o que significa que herda todas as propriedades padrão (como UserName, Email, PasswordHash, etc.) e adiciona propriedades específicas da aplicação.

**ICollection<Order> Orders**: Uma coleção de pedidos associados ao usuário, indicando que cada usuário pode ter vários pedidos.Texto

Descrição gerada automaticamente

## SeedData.cs

Este arquivo é responsável por inicializar dados básicos no banco de dados quando a aplicação é executada pela primeira vez. Ele cria os papéis (roles) "Admin" e "User" e também cria um usuário administrador padrão.

Texto

Descrição gerada automaticamenteNOTA: Tanto o email e a palavra pass aparecem aqui hardcoded, mas apenas por se tratar de um exercicio. O correto seria as credenciais estarem guardas num Azure KeyValt por exemplo, ou variaveis de ambiente.  
  
UsersController.cs

Este controlador está na área "Admin" e gerencia as operações relacionadas aos usuários, como listar, editar e excluir usuários. Somente usuários com o papel "Admin" têm acesso a este controlador.

Tela preta com letras brancas

Descrição gerada automaticamente

**Index()**: Lista todos os usuários cadastrados.

**Edit()**: Permite editar o email do usuário e Roles atribuídos a ele.

**Delete()**: Permite excluir um usuário após confirmação.

## Program.cs

Este arquivo configura os serviços necessários para a aplicação funcionar, incluindo a identidade, papéis e inicialização dos dados seed.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**Configuração da Identidade**: Define as regras de senha e requer confirmação de email para login.

**Adição de Roles**: Inclui suporte a papéis (roles) na identidade.

**SeedData**: Chama o método para inicializar os dados seed, criando os papéis e o usuário administrador.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Resumo Geral (Gestão de Utilizadores)**

* **Gestão de Usuários**: O sistema utiliza o ASP.NET Identity para gerenciamento de usuários, incluindo registro, login, autenticação e autorização.
* **Papéis (Roles)**: São definidos papéis para controlar o acesso a diferentes partes da aplicação. O papel "Admin" tem acesso a funcionalidades de gerenciamento de usuários.
* **Usuário Administrador Padrão**: Ao iniciar a aplicação, é criado um usuário administrador padrão para facilitar o acesso às áreas restritas.
* **Controlador de Usuários**: Permite que administradores visualizem, editem (incluindo a atribuição de papéis) e excluam usuários.
* **Segurança**: O controlador está protegido para que somente usuários autenticados com o papel "Admin" possam acessá-lo.

# Area de BackOffice : **As Categorias**

## CategoryController.cs

Este controlador gerencia as operações CRUD (Criar, Ler, Atualizar, Excluir) para as categorias. Ele utiliza um repositório genérico para interagir com o banco de dados.

**Dependências**: O controlador usa o Repository<Category> para interagir com o banco de dados.



**Construtor**: Inicializa o repositório de categorias usando o contexto do banco de dados.

Texto

Descrição gerada automaticamente

**Métodos CRUD**:

* **Index()**: Lista todas as categorias.
* **Details(int id)**: Exibe detalhes de uma categoria específica, incluindo os produtos relacionados.
* **Create()**: Exibe e processa o formulário de criação de categoria.
* **Edit()**: Exibe e processa o formulário de edição de categoria.
* **Delete()**: Exibe a confirmação e processa a exclusão de uma categoria, garantindo que não existam produtos associados.

## Repository.cs

Este é um repositório genérico que fornece métodos comuns para operações CRUD. Ele permite que você trabalhe com qualquer tipo de entidade, tornando o código mais reutilizável e organizado.

**\_context**: Referência ao contexto do banco de dados.

**\_dbSet**: Representa o conjunto de entidades do tipo T.

**Métodos CRUD**:

* **AddAsync(T entity)**: Adiciona uma nova entidade.
* **DeleteAsync(int id)**: Remove uma entidade pelo ID.
* **GetByIdAsync(int id, QueryOptions<T> options)**: Obtém uma entidade pelo ID, permitindo incluir relações e aplicar filtros.
* **UpdateAsync(T entity)**: Atualiza uma entidade existente.
* **GetAllAsync(QueryOptions<T> options)**: Obtém todas as entidades, com possibilidade de aplicar filtros, ordenação e incluir relações.

## IRepository.cs

Interface que define os métodos que o repositório deve implementar. Isso permite que diferentes implementações sejam usadas sem alterar o código que depende da interface.

Texto

Descrição gerada automaticamente**Definições de Métodos**:

**GetAllAsync()**: Obtém todas as entidades.

**GetByIdAsync(int id)**: Obtém uma entidade pelo ID.

**AddAsync(T entity)**: Adiciona uma nova entidade.

**UpdateAsync(T entity)**: Atualiza uma entidade.

**DeleteAsync(int id)**: Exclui uma entidade pelo ID.

## Como o Código Funciona Junto? (CategoryController.cs e Repository.cs)

1. **Controlador**: O CategoryController recebe solicitações do usuário e utiliza o repositório para manipular os dados das categorias.
2. **Repositório**: O Repository<T> fornece métodos genéricos para operações no banco de dados. Isso evita a repetição de código e facilita a manutenção.

# Area de BackOffice : **Os Produtos**

***Nota:*** *Tanto o controlador de* ***categorias*** *quanto o de* ***produtos*** *utilizam o* ***Repository.cs****, que depende da interface* ***IRepository****. Para evitar repetição e excesso de prints de código, os detalhes serão mostrados apenas na primeira ocorrência de cada interação entre múltiplos componentes ao longo da documentação.*

## ProductController.cs

Este controlador é responsável por gerenciar as operações CRUD (Criar, Ler, Atualizar, Excluir) para os produtos. Ele utiliza o repositório genérico para interagir com o banco de dados e também lida com o upload de imagens dos produtos.

**Dependências**:

* Repository<Product>: Repositório para operações com produtos.
* Repository<Category>: Repositório para obter as categorias disponíveis.
* IWebHostEnvironment: Utilizado para obter o caminho físico do servidor para salvar as imagens.

**Construtor**:

* Inicializa os repositórios de produtos e categorias usando o contexto do banco de dados.
* Recebe IWebHostEnvironment para manipular arquivos no servidor.

**Integração com o Repositório**

* O ProductController utiliza o repositório genérico Repository<Product> para interagir com o banco de dados.

**Associação com Categorias**

* Cada produto está associado a uma categoria.
* O controlador carrega todas as categorias disponíveis para permitir que o usuário selecione uma ao criar ou editar um produto.
* Utiliza o repositório Repository<Category> para obter as categorias.

Diferente do **CRUD** no **CategoryController**, aqui utilizei uma forma diferente, juntando os metodos **Create** e **Edit** em um só, para mostrar mais uma forma alternativa de fazer. Entao neste caso temos o metodo **AddEdit**.

**AddEdit(int id) [HttpGet]**:

**Funcionamento**:

* Carrega todas as categorias para preencher um dropdown.
* Se o **id** for 0, prepara a visão para adicionar um novo produto.
* Se o **id** for diferente de 0, busca o produto existente pelo id, incluindo sua categoria, e prepara a visão para edição.

**AddEdit(Product product) [HttpPost]**:

**Descrição**: Processa o formulário de adição ou edição de produto.

**Funcionamento**:

* Recarrega as categorias para a visão em caso de erro de validação.
* Verifica se o ModelState é válido.
* Se uma imagem foi enviada, processa o upload:

Gera um nome de arquivo único.

Salva o arquivo na pasta wwwroot/images.

* + Se o ProductId for 0:
    - Configura a ImageUrl do produto (nome da imagem ou imagem padrão).
    - Adiciona o novo produto ao banco de dados usando AddAsync().
  + Se o ProductId não for 0:
    - Busca o produto existente.
    - Atualiza suas propriedades com os novos valores.
    - Se uma nova imagem foi enviada:
      * Remove a imagem antiga do servidor, se não for a imagem padrão.
      * Atualiza a ImageUrl com o novo nome de arquivo.
    - Atualiza o produto no banco de dados usando UpdateAsync().
  + Redireciona para a lista de produtos após a operação.

# Area de BackOffice : **As Encomendas**

NOTA: Para que a parte da encomenda funcione do inicio ao fim (sendo o fim “o processamento”) foi criado um serviço de back-end que processa mensagens de uma fila do Azure Storage, e utiliza o metodo FiFo, first-in-first-out. Essa aplicação sera mostrada no fim.

## OrderController.cs

Este controlador é responsável por gerenciar as operações relacionadas às encomendas, incluindo a criação de novas encomendas, a adição de itens ao carrinho, visualização do carrinho e listagem de encomendas do usuário. Ele utiliza o repositório genérico para interagir com o banco de dados e utiliza sessões para armazenar temporariamente os dados do carrinho de compras.

**Dependências e Construtor**

* **Dependências:**
  + \_context: Contexto do banco de dados.
  + \_products: Repositório de produtos.
  + \_orders: Repositório de encomendas.
  + \_userManager: Gerenciador de usuários (para obter informações do usuário autenticado).
  + \_queueService: Serviço para processamento assíncrono das encomendas.
* **Construtor:**
  + Inicializa as dependências, recebendo o contexto, o UserManager e o OrderQueueService via injeção de dependência.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Métodos do Controller**

**Create() [HttpGet]**:

* Busca o OrderViewModel da sessão. Se não existir, cria um novo.
* Retorna a view com o modelo para exibir os produtos e o carrinho atual.

**AddItem(int prodId, int prodQty) [HttpPost]**:

* Busca o produto pelo prodId. Se não encontrar, retorna NotFound().
* Recupera o OrderViewModel da sessão ou cria um novo se não existir.
* Verifica se o produto já está no carrinho:
  + Se já existir, incrementa a quantidade.
  + Se não existir, adiciona um novo OrderItemViewModel à lista de itens.
* Atualiza o TotalAmount somando o total dos itens no carrinho.
* Armazena o OrderViewModel atualizado na sessão.

**Cart() [HttpGet]**:

* Busca o OrderViewModel da sessão.
* Se não existir ou não tiver itens, redireciona para Create.
* Retorna a view com o modelo para exibir os itens no carrinho.

**PlaceOrder() [HttpPost]**:

* Busca o OrderViewModel da sessão.
* Se não existir ou estiver vazio, redireciona para Create.
* Cria uma nova instância de Order, definindo a data, o total, o UserId do usuário autenticado e o status inicial como "Pending".
* Adiciona os itens do carrinho à encomenda como OrderItem.
* Salva a encomenda no banco de dados usando o repositório.
* Envia a encomenda para processamento assíncrono através do \_queueService.
* Remove o OrderViewModel da sessão (limpa o carrinho).
* Redireciona para a ação ViewOrders para que o usuário veja suas encomendas.

**ViewOrders() [HttpGet]**:

* Obtém o UserId do usuário autenticado.
* Utiliza o repositório para obter todas as encomendas do usuário, incluindo os itens e produtos relacionados.
* Ordena as encomendas por data em ordem decrescente.
* Retorna a visão com a lista de encomendas para o usuário.

## SessionExtensions.cs

Esta classe estática fornece métodos de extensão para facilitar o armazenamento e recuperação de objetos complexos na sessão usando serialização JSON.

**Métodos**:

* **Set<T>()**:
  + Serializa o objeto value em JSON e o armazena na sessão com a chave key.
* **Get<T>()**:
  + Recupera a string JSON da sessão com a chave key e a desserializa de volta para um objeto do tipo T.
  + Se não existir ou for vazio, retorna o valor padrão de T.

**Uso no OrderController**:

* Utilizado para armazenar o OrderViewModel (carrinho de compras) na sessão.
* Permite que o estado do carrinho seja mantido entre as requisições HTTP.

## Resumo Geral

* **Gestão de Encomendas**:
  + O OrderController permite que os usuários autenticados adicionem produtos ao carrinho, visualizem o carrinho, finalizem a encomenda e visualizem suas encomendas passadas.
* **Carrinho de Compras usando Sessão**:
  + O carrinho é implementado usando a sessão HTTP para armazenar temporariamente os itens que o usuário deseja comprar.
  + A classe SessionExtensions facilita o armazenamento de objetos complexos na sessão.
* **Processamento Assíncrono de Encomendas**:
  + Após o usuário finalizar a encomenda, uma mensagem é enviada para a fila através do OrderQueueService para processamento posterior (por exemplo, envio de emails, atualização de estoque).

# Area de BackOffice : As Encomendas (**TransportadoraApp**)

TransportadoraAppEste aplicativo é responsável por processar as encomendas que foram enviadas para uma fila de mensagens, atualizando o status das encomendas no banco de dados após o processamento.

## OrderStatusMessage.cs

Este modelo representa a estrutura da mensagem que é enviada para a fila de processamento. Ele contém informações essenciais sobre a encomenda que será processada.

Texto

Descrição gerada automaticamente

**OrderId**: O identificador único da encomenda que precisa ser processada.

**Status**: O status atual ou desejado da encomenda (por exemplo, "Pending", "Processing", "Entregue").

**Program.cs**

Ele configura os serviços necessários e inicia o processamento da fila.

**Configuração do Aplicativo**

* **Main()**:
  + Ponto de entrada do aplicativo que constrói e executa o host.
  + Chama o método CreateHostBuilder() para configurar os serviços e dependências.
* **CreateHostBuilder()**:
  + Utiliza o Host.CreateDefaultBuilder() para configurar um host genérico.
  + **ConfigureAppConfiguration**:
    - Adiciona o arquivo appsettings.json para carregar as configurações necessárias, como strings de conexão.
  + **ConfigureServices**:
    - **AddDbContext<ApplicationDbContext>**:
      * Configura o acesso ao banco de dados usando Entity Framework Core.
    - **QueueClient**:
      * Configura o cliente para acessar a fila do Azure Storage Queue.
      * Verifica se as configurações necessárias (AzureStorage e QueueName) estão presentes.
    - **AddHostedService<QueueProcessor>**:
      * Registra o serviço hospedado que processará as mensagens da fila.

## QueueProcessor.cs

Esta classe é um serviço hospedado que implementa IHostedService e é responsável por processar as mensagens da fila em intervalos regulares.

**Funcionamento do QueueProcessor**

* **Construtor**:
  + Recebe o QueueClient para interagir com a fila do Azure e o IServiceScopeFactory para criar escopos de serviço, permitindo o acesso ao DbContext.
* **StartAsync()**:
  + Inicia o serviço configurando um Timer que chama o método ProcessQueue a cada 10 segundos.
* **ProcessQueue(object state)**:
  + **Recebe Mensagens**:
    - Usa \_queueClient.ReceiveMessagesAsync() para obter mensagens da fila.
    - Define maxMessages: 1 para processar uma mensagem por vez.
    - Define um visibilityTimeout para evitar que outras instâncias processem a mesma mensagem simultaneamente.
  + **Processa a Mensagem**:
    - Desserializa o texto da mensagem em um objeto OrderStatusMessage.
    - Simula um processamento com Task.Delay de 5 segundos (representando, por exemplo, o tempo de entrega).
  + **Atualiza o Banco de Dados**:
    - Cria um escopo de serviço para obter o ApplicationDbContext.
    - Busca a encomenda pelo OrderId no banco de dados.
    - Se encontrada, atualiza o Status para "Entregue" e salva as alterações.
    - Exibe mensagens no console indicando o progresso.
  + **Remove a Mensagem da Fila**:
    - Após o processamento bem-sucedido, remove a mensagem da fila usando \_queueClient.DeleteMessageAsync().
* **StopAsync() e Dispose()**:
  + Métodos para parar o serviço e liberar recursos quando o aplicativo é encerrado.

## Integração com a Entidade Order

**Fluxo de Processamento**

1. **Envio da Encomenda para a Fila**:
   * Ver Metodo **PlaceOrder()** do **OrderController.cs**
2. **Processamento pela TransportadoraApp**:
   * O aplicativo TransportadoraApp está continuamente verificando a fila por novas mensagens.
   * Quando uma mensagem é recebida:
     + A mensagem é desserializada em um objeto OrderStatusMessage.
     + O aplicativo simula o processamento da encomenda
     + Atualiza o status da encomenda no banco de dados para "Entregue".
     + Remove a mensagem da fila para evitar reprocessamento.
3. **Atualização do Status da Encomenda**:

* Com o status atualizado para "Entregue", a webapp exibe essa informação.
* O usuário pode ver o status atualizado ao visualizar suas encomendas (ViewOrders()).

**Como o Código Funciona Junto**

1. **Finalização da Encomenda na Aplicação Principal**:

* O usuário faz uma encomenda na aplicação web.
* O OrderController salva a encomenda no banco de dados com o status inicial "Pending".
* Usa o OrderQueueService para enviar uma mensagem à fila contendo o OrderId e o Status.

1. **Processamento pela TransportadoraApp**:

* O aplicativo console está rodando continuamente.
* Verifica a fila a cada 10 segundos em busca de novas mensagens.
* Ao encontrar uma mensagem, desserializa para um OrderStatusMessage.
* Simula o processamento (exemplo: entrega da encomenda).
* Atualiza o status da encomenda no banco de dados para "Entregue".
  + Remove a mensagem da fila.

1. **Atualização do Status na Aplicação Principal**:

* Quando o usuário acessa a aplicação web e visualiza suas encomendas, o status atualizado é exibido.
* A aplicação consulta o banco de dados e encontra a encomenda com o status "Entregue".

## Diagrama Simplificado do Fluxo

Texto

Descrição gerada automaticamente

# Área para upload de foto livre

## CommunityPhotoUploadController.cs

Usuários autenticados façam upload de fotos para a comunidade, adicionem descrições, visualizem uploads, façam comentários e excluam seus próprios uploads ou comentários. Ele utiliza o Azure Blob Storage para armazenar as fotos.

**Métodos do Controlador**

**Create(CommunityPhotoUploadViewModel model) [HttpPost]**:

**Funcionamento**:

* Verifica se o modelo é válido.
* Obtém o usuário autenticado.
* Chama o método UploadPhotoAsync para fazer o upload da foto para o Azure Blob Storage e obter a URL da foto.
* Cria um novo objeto CommunityPhotoUpload com as informações fornecidas.
* Adiciona o upload ao repositório.
* Redireciona para a ação Index após o sucesso

**UploadPhotoAsync(IFormFile photo)**:

**Funcionamento**:

* Verifica se o arquivo de foto não é nulo e tem conteúdo.
* Gera um nome de arquivo único usando Guid e a extensão do arquivo original.
* Obtém uma referência ao blob no Azure Blob Storage.
* Faz o upload do arquivo para o blob, definindo o tipo de conteúdo.
* Retorna a URL pública do blob para ser armazenada no banco de dados.

**Delete(int id) [HttpPost]**:

**Funcionamento**:

* Obtém o usuário autenticado.
* Busca o upload pelo id, incluindo os comentários associados.
* Verifica se o upload existe.
* Verifica se o usuário é o proprietário do upload.
* Exclui todos os comentários associados ao upload.
* Exclui o upload do repositório.
* Chama DeletePhotoFromAzureAsync para remover a foto do Azure Blob Storage.
* Redireciona para a ação Index.

**DeletePhotoFromAzureAsync(string photoUrl)**:

**Funcionamento**:

* Verifica se a URL da foto não é nula ou vazia.
* Extrai o nome do blob a partir da URL.
* Obtém uma referência ao blob.
* Exclui o blob do armazenamento.

**AddComment(int id, string commentText) [HttpPost]**:

**Funcionamento**:

* Verifica se o texto do comentário não é nulo ou vazio.
* Obtém o usuário autenticado.
* Cria um novo objeto CommunityPhotoComment com as informações fornecidas.
* Adiciona o comentário ao repositório de comentários.
* Redireciona para a ação Index.

**DeleteComment(int id) [HttpPost]**:

**Funcionamento**:

* Obtém o usuário autenticado.
* Busca o comentário pelo id, incluindo a publicação associada.
* Verifica se o comentário existe.
* Verifica se o usuário é o autor do comentário ou o proprietário da publicação.
* Exclui o comentário do repositório.
* Redireciona para a ação Index.

**Como o Código Funciona Junto**

1. **Upload de Foto**:

* O usuário preenche um formulário com a descrição e seleciona uma foto.
* Ao enviar o formulário, o método Create é chamado.
* A foto é enviada para o Azure Blob Storage usando UploadPhotoAsync.
* Um novo registro é criado no banco de dados com as informações do upload.

1. **Visualização de Fotos da Comunidade**:

* A ação Index recupera todos os uploads com os usuários e comentários associados.
* A visão exibe a lista de uploads, fotos e permite interações como comentar ou excluir (se aplicável).

1. **Adicionar Comentário**:

* O usuário insere um comentário em uma foto e envia.
* O método AddComment cria um novo comentário associado ao upload e ao usuário.

1. **Excluir Upload ou Comentário**:

* O usuário solicita a exclusão de um upload ou comentário.
* Verificações de propriedade são realizadas.
* Se autorizado, o upload ou comentário é removido do banco de dados.
* Se um upload é excluído, a foto é também removida do Azure Blob Storage.

# Área de Desafios

## Admin - ChallengesController.cs

**Localização**: GreenSeed/**Areas/Admin**/Controllers/ChallengesController.cs

Este controlador permite que administradores gerenciem os desafios: criar novos, editar existentes, arquivar e visualizar o ranking dos usuários com base nos pontos acumulados.

**Métodos do Controlador**

**Index() [HttpGet]**:

* Obtém todos os desafios que não estão arquivados.
* Ordena por data de criação em ordem decrescente.
* Retorna a visão com a lista de desafios.

**Create(CreateChallengeViewModel model) [HttpPost]**:

* Verifica se o ModelState é válido.
* Cria uma nova instância de Challenge com as opções e a opção correta.
* Processa o upload da imagem:
  + Verifica se o arquivo de imagem é válido e tem extensão permitida.
  + Gera um nome de arquivo único e salva a imagem na pasta wwwroot/images/desafios.
  + Define o ImagePath do desafio.
* Adiciona o desafio ao repositório.
* Redireciona para a ação Index com uma mensagem de sucesso.

**Edit(int? id) [HttpGet]**:

* Verifica se o id é válido.
* Obtém o desafio pelo id.
* Preenche um EditChallengeViewModel com os dados existentes.
* Retorna a visão de edição.

**Edit(int id, EditChallengeViewModel model) [HttpPost]**:

* + - Verifica se o id corresponde ao model.Id.
    - Verifica se o ModelState é válido.
    - Obtém o desafio existente.
    - Atualiza as opções e a opção correta.
    - Se uma nova imagem foi enviada:
      * Processa o upload da nova imagem.
      * Deleta a imagem antiga do servidor.
      * Atualiza o ImagePath.
    - Atualiza o desafio no repositório.
    - Redireciona para a ação Index com uma mensagem de sucesso.

**Archive(int? id) [HttpGet]**:

* + - Verifica se o id é válido.
    - Obtém o desafio pelo id.
    - Define IsArchived como true.
    - Atualiza o desafio no repositório.
    - Redireciona para a ação Index com uma mensagem de sucesso.

**ArchivedChallenges() [HttpGet]**:

* + - Obtém todos os desafios onde IsArchived é true.
    - Ordena por data de criação.
    - Retorna a visão com a lista de desafios arquivados.

**Ranking() [HttpGet]**:

* + - Consulta as respostas corretas dos desafios.
    - Agrupa por UserId e soma os pontos atribuídos.
    - Ordena em ordem decrescente de pontuação.
    - Mapeia para um UserRankingViewModel para exibição.
    - Retorna a visão com o ranking.

## User - ChallengesController.cs

**Localização**: GreenSeed/Controllers/ChallengesController.cs

Este controlador permite que usuários participem dos desafios, enviem suas respostas e visualizem o ranking.

**Métodos do Controlador**

**Index() [HttpGet]**:

* + Obtém o desafio mais recente que não está arquivado.
  + Verifica se existe um desafio ativo.
  + Obtém o usuário autenticado.
  + Verifica se o usuário já respondeu ao desafio.
  + Cria um ChallengeViewModel com as informações do desafio e do usuário.
  + Retorna a visão com o modelo.

**Respond(int challengeId, int selectedOption) [HttpPost]**:

* + Obtém o desafio pelo challengeId.
  + Verifica se o desafio existe e não está arquivado.
  + Obtém o usuário autenticado.
  + Verifica se o usuário já respondeu ao desafio.
  + Determina se a resposta está correta comparando selectedOption com CorrectOption.
  + Calcula os pontos atribuídos com base na ordem das respostas corretas:
    - Primeiro a acertar: 7 pontos
    - Segundo: 5 pontos
    - Terceiro: 3 pontos
    - Demais: 1 ponto
  + Cria um novo ChallengeResponse com as informações.
  + Adiciona a resposta ao repositório.
  + Define mensagens de sucesso ou erro para exibir ao usuário.
  + Redireciona para a ação Index.

# AZURE SQL DataBase

**Passos:**

1. **Criar a Azure SQL Database** no grupo de recursos lucasmg-rg na região North Europe usando o Azure Portal.
2. **Configurar o firewall** para permitir todas as IPs (0.0.0.0 a 255.255.255.255).(Para efeitos de exercicio)
3. **Obter a string de conexão** ADO.NET da SQL Database.
4. **Configurar o appsettings.json** na sua aplicação para usar a string de conexão.

Texto

Descrição gerada automaticamente

1. **Adicionar e aplicar migrações** usando o EF Core CLI ou Package Manager Console, para criar o esquema e inserir dados seed.
2. **Verificar a base de dados** via Azure Portal e testar a aplicação localmente para garantir que tudo está funcionando.

## TransportadoraApp

Adicione a seção **"ConnectionStrings"** no AppSettings.json da TransportadoraApp:  
  
Texto

Descrição gerada automaticamente

# GreenSeed.Tests : Testes

## Visão Geral do GreenSeed.Tests

O projeto GreenSeed.Tests faz parte da mesma solução que a aplicação web e depende dela para funcionar. Ele contém uma série de testes projetados para garantir a correção e a confiabilidade dos componentes da sua aplicação. Os testes são escritos usando o framework de testes xUnit, e utilizam tecnologias como **Moq** para simular dependências e **Microsoft.AspNetCore.Mvc.Testing** para testes de integração.

## Tecnologias Utilizadas

* xUnit: Um framework de testes unitários gratuito e de código aberto para .NET.
* Moq: Uma biblioteca de mocking para .NET que permite simular dependências em seus testes.
* Microsoft.AspNetCore.Mvc.Testing: Fornece infraestrutura para testar aplicações MVC do ASP.NET Core.
* Microsoft.EntityFrameworkCore.InMemory: Um provedor de banco de dados em memória para o Entity Framework Core, usado para testes sem um banco de dados real.
* Selenium WebDriver: Uma ferramenta para automação de navegadores web, útil para testes de interface de usuário (UI).

### Explicação dos Tipos de Teste

* Testes Unitários: Focam em testar componentes ou métodos individuais isoladamente, usando mocks ou stubs para dependências.
* Testes de Integração: Testam as interações entre múltiplos componentes, incluindo middleware, roteamento e camadas de acesso a dados, para garantir que funcionem corretamente em conjunto.

Testes Unitarios

**1. HomeControllerTests**

* Objetivo: Testar o HomeController para garantir que suas ações retornem os resultados esperados.
  + Index\_Action\_Returns\_ViewResult: Verifica se a ação Index retorna um ViewResult.
  + Privacy\_Action\_Returns\_ViewResult: Confirma que a ação Privacy retorna um ViewResult.
  + Error\_Action\_Returns\_ViewResult\_With\_ErrorViewModel: Assegura que a ação Error retorna um ViewResult contendo um ErrorViewModel com o RequestId correto.

**2. SessionExtensionsTests**

* Objetivo: Testar os métodos de extensão de sessão (SessionExtensions) para garantir a serialização e desserialização corretas de objetos armazenados na sessão.
  + Set\_SerializesAndStoresObjectInSession: Verifica se o método Set serializa e armazena corretamente um objeto na sessão.
  + Get\_DeserializesAndRetrievesObjectFromSession: Confirma se o método Get desserializa e recupera corretamente um objeto da sessão.
  + Get\_ReturnsDefault\_WhenKeyDoesNotExist: Verifica se o método Get retorna o valor padrão quando a chave especificada não existe na sessão.

## Testes de Integração

**1. ProductControllerIntegrationTests**

* Objetivo: Testar o ProductController para garantir que ele funcione quando integrado com a pilha completa da aplicação, incluindo roteamento, middleware e interações com o banco de dados.
  + Index\_Returns\_Product\_List: Verifica se a ação Index retorna uma lista de produtos e se a resposta contém os nomes dos produtos esperados.
  + AddEdit\_Get\_Returns\_ViewResult\_For\_New\_Product: Confirma que a ação AddEdit com id=0 retorna a visão correta para adicionar um novo produto e que a resposta contém a operação esperada ("Adicionar").

**2. CustomWebApplicationFactory e BaseIntegrationTest**

* Objetivo: Configurar um ambiente de teste personalizado para testes de integração.
  + CustomWebApplicationFactory: Configura o host web para testes, usando um banco de dados em memória e um esquema de autenticação de teste.
  + BaseIntegrationTest: Fornece uma classe base para testes de integração herdarem, garantindo consistência e reutilização do código de configuração.

**3. TestAuthHandler**

* Objetivo: Fornecer um manipulador de autenticação para testes de integração, simulando um usuário autenticado.
  + Configura um ClaimsPrincipal com reivindicações (claims) predefinidas para imitar um usuário logado durante os testes.

# Criar e Implantar um Azure Web App

## Criar um Azure Web App

**Passo 1: Acessar o Portal Azure**

* Navegar até o Grupo de Recursos
* No menu lateral esquerdo, clique em "Resource groups".
* Selecione o grupo de recursos lucasmg-rg. Se ainda não existir, criar:
  + Clique em "Create".
  + Subscription: Selecione "Azure for Students".
  + Resource group: lucasmg-rg.
  + Region: Selecione "North Europe".
  + Clique em "Review + create" e depois em "Create".

**Passo 1.1: Adicionar um Novo Web App**

* Dentro do grupo de recursos lucasmg-rg, clique em "Create".
* Na barra de busca, digite "Web App" e selecione "Web App" na lista.
* Clique em "Create" para iniciar a criação do Web App.

**Passo 1.2: Configurar o Web App**

* Basics:
  + Subscription: Certifique-se de que está selecionada "Azure for Students".
  + Resource group: lucasmg-rg.
  + Name: Escolha um nome único para o seu Web App, por exemplo, greenseed-webapp.
  + Publish: Selecione "Code".
  + Runtime stack: Selecione ".NET 8" (versão usada no projeto).
  + Operating System: "Windows".
  + Region: "North Europe".
  + Clique em "Next: Deployment >".
* Deployment:
  + Deployment Center: Podemos configurar o **GitHub Actions** posteriormente.
  + Clique em "Review + create" e depois em "Create".

**2. Configurar as Configurações do Web App**

* Após a criação do Web App, precisamos configurar algumas definições, como a string de conexão com a **Azure SQL Database**.

**Passo 2.1: Navegar até o Web App**

* No **Azure Portal**, vá para **"Resource groups"** e selecione **lucasmg-rg**.
* Clique no seu Web App recém-criado, por exemplo, **greenseed-webapp**.

**Passo 2.2: Configurar as Configurações de Aplicação**

* No menu lateral do Web App, clique em **"Configuration"**.
* Na aba **"Application settings"**, clique em **"New connection string"**.
  + **Name**: DefaultConnection
  + **Value**: Cole a **Connection String** que você obteve anteriormente da sua Azure SQL Database. Deve ser semelhante a:
  + **Type**: **"SQLAzure"**
  + **Deployment slot setting**: Deixe desmarcado.
* Clique em **"OK"** para adicionar a string de conexão.
* **Salvar as Configurações**:
  + Após adicionar todas as strings de conexão necessárias, clique em **"Save"** no topo da página.
  + Confirme a operação clicando em **"Continue"**.

**3. Configurar o Repositório no GitHub para Implantação Contínua**

Vamos configurar o **GitHub Actions** para automatizar o processo de build e deploy da sua aplicação para o Azure Web App.

**Passo 3.1: Conectar o Repositório GitHub ao Azure Web App**

* No **Azure Portal**, dentro do seu Web App, no menu lateral, clique em **"Deployment Center"**.
* **Configuração Inicial**:
  + **Source Control**: Selecione **"GitHub"**.
  + **Repository**: Clique em **"Authorize"** para permitir que o Azure acesse seus repositórios do GitHub.
  + Após autorizar, selecione:
    - **Organization**: Sua organização ou usuário.
    - **Repository**: **GreenSeed**.
    - **Branch**: master.
* **Configuração de Build**:
  + **Build provider**: Selecione **"GitHub Actions"**.
  + O Azure irá detectar automaticamente que sua aplicação é **ASP.NET Core** e sugerirá um fluxo de trabalho adequado.
  + Clique em **"Continue"**.
* **Revisar e Finalizar**:
  + Revise as configurações e clique em **"Finish"**.
  + O Azure irá criar um arquivo de workflow do **GitHub Actions** no seu repositório, geralmente localizado em **.github/workflows/azure-webapp.yml**.

**Passo 3.2: Verificar e Editar o Workflow do GitHub Actions**

* **Acesse o Repositório no GitHub**:
  + Vá até o seu repositório no GitHub onde está o projeto **GreenSeed**.
* **Navegar até o Workflow**:
  + No repositório, clique em **"Actions"** no menu superior.
  + Você verá o workflow criado pelo Azure, geralmente chamado de **"Deploy ASP.NET Core app to Azure Web App"**.

**Passo 4.3: Confirmar a Configuração do GitHub Actions**

* **Commit e Push**:
  + Depois de fazer as alteraçoes necessarias(se precisar), fazer **commit**.
  + O GitHub Actions irá iniciar automaticamente o workflow quando você fizer **push** para a branch MASTER
* **Monitorar o Deploy**:
  + No **GitHub**, dentro do repositório, vá para **"Actions"** e selecione o workflow que está a correr.
  + Monitore o progresso do **Build** e **Deploy**.
  + Se tudo correr bem, o deploy será concluído com sucesso.
* **Verificar o Deploy no Azure**:
  + Volte ao **Azure Portal** e, dentro do seu Web App, clique em **"Browse"** para abrir a aplicação no navegador.
  + Verifique se a aplicação está funcionando conforme esperado e se os dados seed estão sendo exibidos.

# Texto Descrição gerada automaticamenteA Pipeline (Que está no repositório)

# Tecnologias utilizadas

ASP.NET Core MVC

Entity Framework Core

C# 8.0

Dependency Injection

DataAnnotations

IWebHostEnvironment

Azure Storage Queues SDK

Azure Blob Storage SDK

Logging

xUnit

Moq

Microsoft.AspNetCore.Mvc.Testing

Entity Framework Core InMemory Provider

Selenium WebDriver

.NET Core Console Application

GitHub Actions

Pipeline CI/CD

Tecnologias a ser implementadas com as Melhorias, e Boas Praticas das paginas seguintes

Azure Key Vault

Azure Container Registry ou DockerHub

Docker

Dokerfile

Docker Compose

.Net 8.0 SDK e Runtime

Boas Praticas Que Faltou Implementar

## Gestão de Utilizadores

### SeedData.cs

Tanto o email e a palavra pass aparecem aqui hardcoded, mas apenas por se tratar de um exercicio. O correto seria as credenciais estarem guardas num Azure KeyValt por exemplo, ou variaveis de ambiente.

## Gestão de Categorias e Gestão de Produtos

Ambos deveriam estar na Area de Admin assim como está o ChallengeController.cs para que houvesse uma maior coerência no projeto:

**Localização desejada**: GreenSeed/**Areas/Admin**/Controllers/

## Gestão de Produtos

Assim como a Area para upload de foto livre, ao criar um produto e adicionar uma imagem, deveria ser utilizado o azure blob storage para armazenamento das imagens.

## Credênciais

Todos os tipos de credências não deveriam aparecer no codigo, seja stringConnections, seed conta admin, entre outros. O correto é utilizar variaveis de ambiente ou serem armazenadas em Azure Key Vault por exemplo.

## AZURE SQL DataBase

“2. **Configurar o firewall** para permitir todas as IPs (0.0.0.0 a 255.255.255.255).”

* EstaConfiguração que foi feita no ponto 2, foi apenas para fins de exercicio e facilidade de acesso. Em um ambiente de produção não é seguro abrir todas as portas de IP como foi feito neste projeto

# Melhorias

## Implementação de Containerização

### Como seria feito?

* **Criar um Dockerfile**: O arquivo Dockerfile define como a imagem do contêiner será construída.
  + **Multi-Stage Build**: Ele compila a aplicação em uma etapa intermediária e copia os binários para a imagem final.
  + **Configurações de Ambiente**: Definir variáveis de ambiente necessárias e expor as portas utilizadas pela aplicação.
* **Configurar Variáveis de Ambiente e Segredos**:
  + **Segredos**: Utilizar variáveis de ambiente ou serviços como o **Azure Key Vault** para gerenciar segredos (como Connection Strings), evitando hardcode no código ou imagens.
* **Construir a Imagem do Docker**:
  + Executar o comando docker build -t greenseed . no diretório que contém o Dockerfile para construir a imagem.
* **Executar o Contêiner**:
  + Executar docker run -p 8080:8080 greenseed para iniciar o contêiner e mapear as portas conforme necessário.
* **Utilizar Docker Compose (se aplicável)**:
  + Se a aplicação depende de outros serviços (como um banco de dados SQL Server), usar o **Docker Compose** para orquestrar múltiplos contêineres.
* **Testar o Contêiner**:
  + Garantir que a aplicação funcione corretamente dentro do contêiner, incluindo todas as funcionalidades e acesso a recursos externos.
* **Publicar a Imagem**:
  + Opcionalmente, publicar a imagem em um Container Registry como o **Docker Hub** ou **Azure Container Registry**.

# Recursos Utilizados

* Documentação Microsoft
* Documentação GitHub Actions
* DockerDoc
* StackOverFlow
* ChatGpt
* [sergiocpxfontes/academiarumos81958](https://github.com/sergiocpxfontes/academiarumos81958)
* [ASP.NET Core MVC Tutorial – Full Course to Build YOUR Passion Project! - YouTube](https://www.youtube.com/watch?v=q9X3SDEZtpw)
* [EvanGudmestad/TequliasRestaurantMVC](https://github.com/EvanGudmestad/TequliasRestaurantMVC)