Manual do Programador - Projeto Litly

Projeto: Litly

Descrição: Este manual destina-se a programadores e integradores de sistemas que necessitam compreender a arquitetura, a estrutura do código e os procedimentos de instalação e configuração do projeto Litly. O Litly é uma aplicação desktop desenvolvida em C# com interface gráfica em Windows Forms, que permite a utilizadores partilhar e interagir com conteúdos literários.

Identificação do Aluno: Júlia Amaral de Souza – 2223248

Turma: TGPSI23S (23/26)

Índice

1. Introdução

Este manual tem como objetivo fornecer uma visão técnica aprofundada do projeto Litly, uma aplicação desktop desenvolvida em C# e Windows Forms. O Litly foi concebido para criar uma plataforma interativa onde os utilizadores podem partilhar e interagir com conteúdos literários, promovendo a leitura e a escrita. Este documento é direcionado a programadores, administradores de sistema e qualquer pessoa interessada em compreender a estrutura interna, o funcionamento e os requisitos

técnicos para a manutenção e evolução da aplicação. Serão abordados tópicos como a arquitetura do sistema, a organização do código-fonte, a configuração do ambiente de desenvolvimento, a estrutura da base de dados, a implementação das funcionalidades chave e as considerações de segurança. O objetivo é capacitar o leitor com o conhecimento necessário para modificar, estender ou depurar o código do Litly de forma eficiente e segura.

2. Arquitetura do Sistema

A aplicação Litly segue uma arquitetura em camadas, comum em aplicações desktop, que visa separar as diferentes responsabilidades do sistema, promovendo a modularidade, a manutenibilidade e a escalabilidade. As principais camadas são:

- **Camada de Apresentação (UI - User Interface):** Responsável pela interação com o utilizador. É implementada utilizando **Windows Forms** em C#. Esta camada lida com a exibição de dados, a captura de entradas do utilizador e a navegação entre as diferentes telas da aplicação. Os ficheiros `.cs` e `.Designer.cs` das páginas (e.g., `Login.cs`, `PaginaPrincipal.cs`, `AdicionarLivros.cs`) pertencem a esta camada.
- **Camada de Lógica de Negócio (BLL - Business Logic Layer):** Contém as regras de negócio e a lógica central da aplicação. Esta camada processa as requisições da camada de apresentação, valida os dados, executa as operações necessárias e coordena as interações com a camada de acesso a dados. Classes como `Utilizador.cs`, `Postagem.cs`, `Livro.cs` e `Sessao.cs` (se existirem classes de modelo ou de serviço para estas entidades) podem residir aqui, ou em uma camada de Domínio/Modelo separada.
- **Camada de Acesso a Dados (DAL - Data Access Layer):** Responsável pela comunicação com a base de dados. Esta camada abstrai os detalhes de como os dados são armazenados e recuperados, fornecendo métodos para realizar operações CRUD (Create, Read, Update, Delete) nas entidades do sistema. A interação é feita através de **ADO.NET** e o provedor de dados **Microsoft.Data.SqlClient**, conectando-se a uma base de dados **SQL Server**. A lógica de conexão e execução de queries SQL reside nesta camada.

Fluxo de Dados

O fluxo de dados típico na aplicação Litly ocorre da seguinte forma:

1. **Utilizador Interage:** O utilizador interage com a interface gráfica (Camada de Apresentação), por exemplo, clicando num botão para fazer login ou submeter uma nova postagem.
2. **Requisição para Lógica de Negócio:** A Camada de Apresentação envia a requisição e os dados relevantes para a Camada de Lógica de Negócio.
3. **Processamento da Lógica de Negócio:** A Camada de Lógica de Negócio valida os dados, aplica as regras de negócio e, se necessário, invoca a Camada de Acesso a Dados para persistir ou recuperar informações.
4. **Interação com a Base de Dados:** A Camada de Acesso a Dados executa as operações SQL necessárias na base de dados SQL Server.
5. **Retorno dos Dados:** Os resultados da operação são retornados da Camada de Acesso a Dados para a Camada de Lógica de Negócio.
6. **Atualização da Interface:** A Camada de Lógica de Negócio processa os resultados e os envia de volta para a Camada de Apresentação, que atualiza a interface do utilizador de acordo (e.g., exibe uma mensagem de sucesso, carrega uma nova tela, mostra dados atualizados).

Esta separação de preocupações facilita a manutenção do código, permite que as camadas sejam desenvolvidas e testadas de forma independente, e torna a aplicação mais resiliente a mudanças, como a eventual substituição da base de dados ou a migração para uma tecnologia de interface diferente.

3. Estrutura do Código

A estrutura do projeto Litly é organizada de forma a facilitar a navegação e a compreensão do código-fonte. O projeto principal, `Litly.02`, contém a maioria dos ficheiros da aplicação. Abaixo, é apresentada uma visão geral da estrutura de diretórios e dos principais ficheiros:

```
Litly_project/
├── Litly.02/
│   ├── Litly.02/
│   │   ├── Forms/ (ou similar, onde as páginas da UI estão)
│   │   │   ├── Login.cs
│   │   │   ├── Login.Designer.cs
│   │   │   ├── Login.resx
│   │   │   ├── PaginaPrincipal.cs
│   │   │   ├── PaginaPrincipal.Designer.cs
│   │   │   ├── PaginaPrincipal.resx
│   │   │   ├── AdicionarLivros.cs
│   │   │   ├── AdicionarLivros.Designer.cs
│   │   │   ├── AdicionarLivros.resx
│   │   │   ├── Biblioteca.cs
│   │   │   ├── Biblioteca.Designer.cs
│   │   │   ├── Biblioteca.resx
│   │   │   ├── DetalhesLivros.cs
│   │   │   ├── DetalhesLivros.Designer.cs
│   │   │   ├── DetalhesLivros.resx
│   │   │   ├── Form1.cs
│   │   │   ├── Form1.Designer.cs
│   │   │   ├── Form1.resx
│   │   │   ├── FormChat.cs
│   │   │   ├── FormChat.Designer.cs
│   │   │   ├── FormChat.resx
│   │   │   ├── PaginaPostagem.cs
│   │   │   ├── PaginaPostagem.Designer.cs
│   │   │   ├── PaginaPostagem.resx
│   │   │   ├── PerfilUtilizador.cs
│   │   │   ├── PerfilUtilizador.Designer.cs
│   │   │   ├── PerfilUtilizador.resx
│   │   │   ├── Resgistro.cs
│   │   │   ├── Resgistro.Designer.cs
│   │   │   ├── Resgistro.resx
│   │   │   ├── Utilizador.cs
│   │   │   ├── Utilizador.Designer.cs
│   │   │   ├── Utilizador.resx
│   │   │   ├── frmAmigos.cs
│   │   │   ├── frmAmigos.Designer.cs
│   │   │   └── frmAmigos.resx
│   │   ├── Properties/
│   │   │   ├── Resources.Designer.cs
│   │   │   ├── Resources.resx
│   │   │   ├── serviceDependencies.json
│   │   │   ├── serviceDependencies.local.json
│   │   │   └── serviceDependencies.local.json.user
│   │   ├── Resources/
│   │   │   └── (Imagens e outros recursos)
│   │   ├── Program.cs
│   │   ├── Sessao.cs
│   │   ├── Litly.02.csproj
│   │   └── (Outros ficheiros de código e configuração)
│   ├── scriptbd/
│   │   └── SQLQuery1.sql
│   ├── Litly.02.sln
│   └── README.md
└── (Outros ficheiros e diretórios gerados)
```

Descrição dos Principais Componentes:

- **`Litly.02.sln`**: O ficheiro de solução do Visual Studio. Contém a estrutura do projeto e permite abrir e gerir todo o projeto no IDE.
- **`Litly.02/Litly.02.csproj`**: O ficheiro de projeto C#. Define as propriedades do projeto, referências a pacotes NuGet (como `Guna.UI2.WinForms`, `Microsoft.Data.SqlClient`), ficheiros de código-fonte incluídos, e configurações de compilação. Este ficheiro indica que a aplicação é um executável Windows Forms (`<OutputType>WinExe</OutputType>`, `<UseWindowsForms>true</UseWindowsForms>`) e tem como alvo o .NET 8.0 (`<TargetFramework>net8.0-windows</TargetFramework>`).
- **`Program.cs`**: O ponto de entrada da aplicação. Contém o método `Main` que inicializa a aplicação Windows Forms e executa o formulário principal (geralmente o formulário de login ou a página principal).
- **Ficheiros de Formulário (`.cs`, `.Designer.cs`, `.resx`):** Cada formulário (tela) da aplicação é composto por três ficheiros:
  - **`.cs`**: Contém a lógica de negócio e os manipuladores de eventos para o formulário. É aqui que o código C# para a interação do utilizador e a lógica da aplicação é implementado.
  - **`.Designer.cs`**: Gerado automaticamente pelo Visual Studio. Contém o código que define a interface gráfica do formulário, incluindo a disposição dos controlos (botões, caixas de texto, etc.) e as suas propriedades. **Não deve ser editado manualmente.**
  - **`.resx`**: Ficheiro de recursos que armazena elementos como strings, imagens e outros recursos utilizados pelo formulário. Exemplos incluem `Login.cs`, `PaginaPrincipal.cs`, `AdicionarLivros.cs`, etc.
- **`Properties/`**: Contém ficheiros de configuração e recursos do projeto, como `Resources.Designer.cs` (para recursos globais da aplicação) e `serviceDependencies.json` (para configurações de dependências de serviço).
- **`Resources/`**: Diretório que armazena imagens e outros recursos estáticos utilizados na aplicação, como `Logo.png`, `Chat.png`, etc.
- **`Sessao.cs`**: Este ficheiro provavelmente contém uma classe para gerir a sessão do utilizador, armazenando informações como o ID do utilizador logado, o seu nome, etc., que podem ser acedidas globalmente pela aplicação.

- **`scriptbd/SQLQuery1.sql`**: Contém scripts SQL para a gestão da base de dados. (que foi analisado) é um exemplo de um script de O ficheiro `REFIZATABELA.txt` base de dados que define a estrutura das tabelas e as relações entre elas.

Organização do Código dentro dos Formulários:

Dentro de cada ficheiro `.cs` de formulário, a organização do código geralmente segue um padrão:

1. **Construtor:** Inicializa os componentes do formulário.
2. **Manipuladores de Eventos:** Métodos que respondem a ações do utilizador (e.g., `button_Click`, `textBox_TextChanged`).
3. **Métodos Auxiliares:** Funções privadas que realizam tarefas específicas, como carregar dados, limpar campos, etc.
4. **Lógica de Acesso a Dados:** Embora a arquitetura ideal separe a DAL, em projetos de menor dimensão ou para simplificar, a lógica de acesso a dados (conexão com SQL Server, execução de comandos) pode estar diretamente nos formulários ou em classes auxiliares dentro do projeto principal.

É fundamental que os programadores mantenham a consistência na nomenclatura e na organização do código para facilitar a colaboração e a manutenção futura do projeto.

4. Configuração do Ambiente de Desenvolvimento

Para desenvolver, compilar e executar o projeto Litly, é necessário configurar o ambiente de desenvolvimento com as ferramentas e dependências corretas. Siga os passos abaixo para preparar o seu ambiente:

4.1. Instalação do Visual Studio

O Visual Studio é o Ambiente de Desenvolvimento Integrado (IDE) recomendado para trabalhar com projetos C# e Windows Forms. Siga os passos para a instalação:

1. **Descarregar o Visual Studio:** Aceda ao site oficial do Visual Studio (https://visualstudio.microsoft.com/pt-br/downloads/) e descarregue a edição

Community (gratuita para estudantes, contribuidores de código aberto e equipas pequenas).

2. **Executar o Instalador:** Execute o ficheiro descarregado `vs_community.exe`.
3. **Selecionar Cargas de Trabalho:** No instalador do Visual Studio, certifique-se de selecionar as seguintes cargas de trabalho (Workloads):
    - **Desenvolvimento para desktop com .NET:** Essencial para aplicações Windows Forms.
    - **Desenvolvimento de área de trabalho do .NET:** (Se disponível, pode ser incluído na anterior ou ser uma opção separada).
    - **Desenvolvimento ASP.NET e web:** (Opcional, se pretender desenvolver aplicações web no futuro).
    - **Ferramentas de desenvolvimento de dados e armazenamento:** Essencial para ferramentas de base de dados e SQL Server.
4. **Instalar:** Prossiga com a instalação. Pode demorar algum tempo, dependendo da sua ligação à internet e das cargas de trabalho selecionadas.

4.2. Instalação do .NET SDK

O projeto Litly utiliza o .NET 8.0. Embora o Visual Studio geralmente instale as versões necessárias do .NET SDK, é boa prática verificar ou instalar manualmente, se necessário:

1. **Verificar Versão:** Abra o `Prompt de Comando` ou `PowerShell` e execute: `bash dotnet --list-sdks` Verifique se o .NET 8.0 SDK está listado. Se não estiver, ou se precisar de uma versão específica, continue para o próximo passo.
2. **Descarregar .NET SDK:** Aceda ao site oficial do .NET (https://dotnet.microsoft.com/download/dotnet/8.0) e descarregue o instalador do .NET 8.0 SDK para o seu sistema operativo.
3. **Instalar:** Execute o instalador e siga as instruções.

4.3. Instalação e Configuração do SQL Server

O Litly depende de uma base de dados SQL Server. Recomenda-se a instalação do SQL Server Express Edition para fins de desenvolvimento.

1. **Descarregar SQL Server Express:** Aceda ao site oficial do SQL Server Express (https://www.microsoft.com/pt-br/sql-server/sql-server-downloads) e descarregue a versão Express.
2. **Instalar SQL Server Express:** Execute o instalador. Escolha a opção de instalação **Básica** para uma configuração rápida, ou **Personalizada** para mais controlo. Certifique-se de que o serviço do SQL Server está a ser executado.
3. **Instalar SQL Server Management Studio (SSMS):** O SSMS é uma ferramenta gráfica para gerir as suas bases de dados SQL Server. Descarregue-o em (https://docs.microsoft.com/pt-br/sql/ssms/download-sql-server-management-studio-ssms) e instale-o.
4. **Criar a Base de Dados Litly:**
   - Abra o SSMS e conecte-se à sua instância do SQL Server (geralmente `(localdb)\MSSQLLocalDB` ou `localhost\SQLEXPRESS`).
   - No SSMS, clique com o botão direito do rato em `Databases` e selecione `New Database...`.
   - Nomeie a base de dados como `Litly` (ou o nome que preferir, mas lembre-se de atualizar a string de conexão na aplicação).
   - Execute o script `REFIZATABELA.txt` (localizado em `Litly_project/Litly.02/scriptbd/`) na base de dados `Litly` para criar as tabelas e as suas relações. Para fazer isso, abra o ficheiro no SSMS, selecione a base de dados `Litly` no menu suspenso e clique em `Execute`.

4.4. Abrir e Compilar o Projeto no Visual Studio

1. **Abrir a Solução:** No Visual Studio, vá a `File` > `Open` > `Project/Solution...` e navegue até o diretório `Litly_project/Litly.02/Litly.02.sln`. Selecione o ficheiro `.sln` e clique em `Open`.
2. **Restaurar Pacotes NuGet:** O Visual Studio deverá restaurar automaticamente os pacotes NuGet necessários (Guna.UI2.WinForms, Microsoft.Data.SqlClient, etc.). Se não o fizer, clique com o botão direito do rato na solução no `Solution Explorer` e selecione `Restore NuGet Packages`.
3. **Configurar String de Conexão:** A string de conexão com a base de dados é crucial. Ela geralmente é definida em ficheiros de configuração (como `App.config` ou `appsettings.json`) ou diretamente no código. Procure por

`SqlConnection` ou `ConnectionString` no código-fonte (por exemplo, nas classes de acesso a dados ou nas classes de formulário que interagem com a base de dados) e atualize-a para apontar para a sua instância do SQL Server e a base de dados `Litly`. Exemplo de string de conexão: `csharp string connectionString = "Data Source=localhost\\SQLEXPRESS;Initial Catalog=Litly;Integrated Security=True;TrustServerCertificate=True";` Ajuste `Data Source` para o nome do seu servidor SQL Server e `Initial Catalog` para o nome da sua base de dados.

4. **Compilar o Projeto:** No Visual Studio, vá a `Build` `Build Solution` (ou pressione `Ctrl+Shift+B`). Certifique-se de que não há erros de compilação.
5. **Executar a Aplicação:** Após a compilação bem-sucedida, pode executar a aplicação clicando em `Start` (ou pressionando `F5`).

Seguindo estes passos, o ambiente de desenvolvimento estará pronto para trabalhar com o projeto Litly.

5. Estrutura da Base de Dados

A base de dados do projeto Litly é implementada em SQL Server e foi concebida para suportar todas as funcionalidades da aplicação, gerindo informações de utilizadores, postagens, livros e interações sociais. A estrutura é definida por um conjunto de tabelas interligadas por chaves primárias e estrangeiras, garantindo a integridade referencial e a consistência dos dados.

O script `REFIZATABELA.txt` (localizado em `Litly_project/Litly.02/scriptbd/`) detalha a criação e reestruturação das tabelas. Abaixo, descrevemos as tabelas principais e as suas colunas:

Tabelas Principais:

1. **`Utilizadores`**
    - **`IdUtilizador`**: `INT PRIMARY KEY IDENTITY(1,1)` - Chave primária, auto-incremento. Identificador único do utilizador.
    - **`Nome`**: `VARCHAR(100) NOT NULL` - Nome completo do utilizador.

- **`Email`**: `VARCHAR(100) NOT NULL UNIQUE` - Endereço de email do utilizador, único para cada registo.
- **`PalavraPasse`**: `VARCHAR(200) NOT NULL` - Palavra-passe do utilizador (deve ser armazenada de forma segura, preferencialmente com hashing e salting).
- **`Bio`**: `VARCHAR(350) NULL` - Biografia ou descrição pessoal do utilizador.
- **`DataCriacao`**: `DATETIME DEFAULT GETDATE()` - Data e hora de criação do registo do utilizador, com valor padrão para a data atual.
- **`ImagemPerfil`**: `VARBINARY(MAX) NULL` - Imagem de perfil do utilizador, armazenada como dados binários.

2. **`Postagens`**
   - **`IdPostagem`**: `INT PRIMARY KEY IDENTITY(1,1)` - Chave primária, auto-incremento. Identificador único da postagem.
   - **`Titulo`**: `NVARCHAR(200) NOT NULL` - Título da postagem.
   - **`Autor`**: `NVARCHAR(100) NOT NULL` - Autor da postagem (pode ser o próprio utilizador ou o autor de um livro referenciado).
   - **`Conteudo`**: `NVARCHAR(MAX) NULL` - Conteúdo textual da postagem (anteriormente `Comentario`).
   - **`DataCriacao`**: `DATETIME DEFAULT GETDATE()` - Data e hora de criação da postagem (anteriormente `DataPostagem`).
   - **`Imagem`**: `VARBINARY(MAX) NULL` - Imagem associada à postagem.
   - **`IdUtilizador`**: `INT NOT NULL` - Chave estrangeira (`FK_Postagens_Utilizador`) que referencia `IdUtilizador` na tabela `Utilizadores`. Indica qual utilizador criou a postagem.
   - **`IdLivro`**: `INT NULL` - Chave estrangeira (`FK_Postagens_Livro`) que referencia `IdLivro` na tabela `Livros`. Opcional, para postagens relacionadas a livros específicos.
   - **`Nota`**: `INT NULL` - Uma nota ou classificação associada à postagem.

3. **`Livros`**

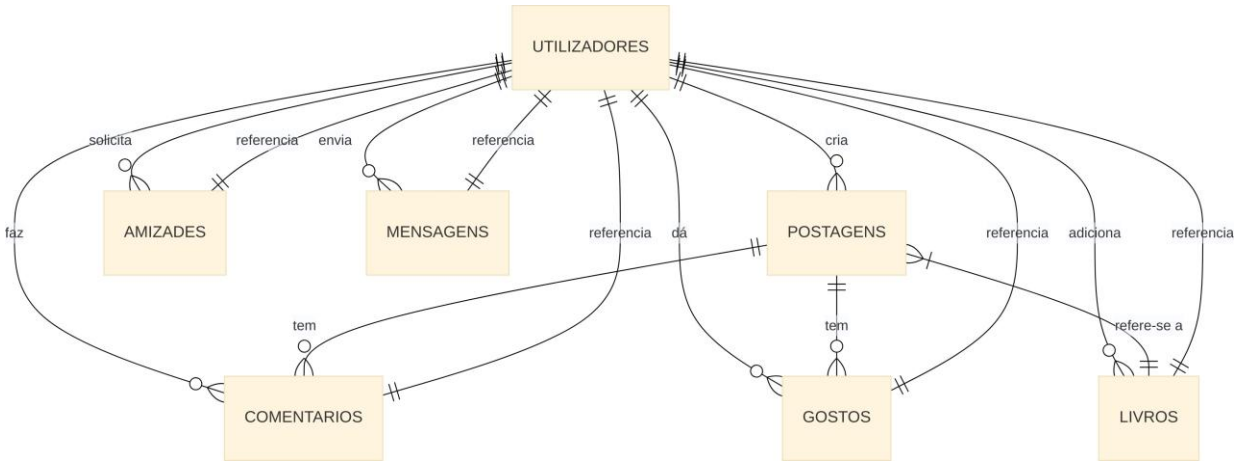
- (Assumindo colunas como `IdLivro`, `Titulo`, `Autor`, `Sinopse`, `Imagem` e `IdUtilizador` - o script `REFIZATABELA.txt` adiciona `IdUtilizador` e a FK para `Utilizadores`)

4. **`Comentarios`**
   - (Assumindo colunas como `IdComentario`, `Conteudo`, `DataComentario`, `IdUtilizador`, `IdPostagem`)
   - **`IdUtilizador`**: Chave estrangeira (`FK_Comentarios_Utilizador`) para `Utilizadores`.
   - **`IdPostagem`**: Chave estrangeira (`FK_Comentarios_Postagem`) para `Postagens`.
5. **`Gostos`**
   - (Assumindo colunas como `IdGosto`, `DataGosto`, `IdUtilizador`, `IdPostagem`)
   - **`IdUtilizador`**: Chave estrangeira (`FK_Gostos_Utilizador`) para `Utilizadores`.
   - **`IdPostagem`**: Chave estrangeira (`FK_Gostos_Postagem`) para `Postagens`.
6. **`Amizades`**
   - (Assumindo colunas como `IdAmizade`, `IdSolicitante`, `IdAceito`, `Status`, `DataAmizade`)
   - **`IdSolicitante`**: Chave estrangeira (`FK_Amizades_Solicitante_Utilizadores`) para `Utilizadores`.
   - **`IdAceito`**: Chave estrangeira (`FK_Amizades_Aceito_Utilizadores`) para `Utilizadores`.
7. **`Mensagens`**
   - (Assumindo colunas como `IdMensagem`, `Conteudo`, `DataEnvio`, `IdRemetente`, `IdDestinatario`)
   - **`IdRemetente`**: Chave estrangeira (`FK_Mensagens_Remetente`) para `Utilizadores`.

- **`IdDestinatario`**: Chave estrangeira (`FK_Mensagens_Destinatario`) para `Utilizadores`.

Diagrama de Entidade-Relação (ER)

O diagrama ER abaixo visualiza as relações entre as tabelas da base de dados, conforme a lógica do script `REFIZATABELA.txt` e a estrutura da aplicação:



Este esquema de base de dados foi projetado para ser flexível e extensível, permitindo a adição de novas funcionalidades e a gestão eficiente de um volume crescente de dados.

6. Funcionalidades Principais e Implementação

Esta secção descreve a implementação das funcionalidades chave do projeto Litly, detalhando como as interações do utilizador são processadas e como os dados são manipulados através das diferentes camadas da aplicação.

6.1. Login e Registo de Utilizador (REQ0001)

O processo de autenticação e registo é fundamental para a segurança e personalização da experiência do utilizador. A implementação envolve os formulários `Login.cs` e `Resgistro.cs` e a interação com a tabela `Utilizadores` na base de dados.

- **Registo:**

- O formulário `Resgistro.cs` recolhe o nome, email e palavra-passe do novo utilizador.
- Ao submeter, a aplicação valida os dados de entrada (e.g., formato do email, complexidade da palavra-passe).
- A palavra-passe é processada (idealmente, hashed e salted) antes de ser enviada para a camada de acesso a dados.
- A camada de acesso a dados insere um novo registo na tabela `Utilizadores`.
- Em caso de sucesso, o utilizador é redirecionado para a tela de login ou diretamente para a página principal.

- **Login:**
  - O formulário `Login.cs` recolhe o email e a palavra-passe do utilizador.
  - A aplicação consulta a tabela `Utilizadores` para verificar as credenciais.
  - A palavra-passe fornecida pelo utilizador é processada (hashed e salted, se aplicável) e comparada com a palavra-passe armazenada na base de dados.
  - Em caso de sucesso, uma sessão de utilizador é iniciada (geralmente através de uma classe `Sessao.cs` que armazena o `IdUtilizador` e outras informações relevantes) e o utilizador é redirecionado para a `PaginaPrincipal.cs`.
  - Em caso de falha (credenciais inválidas), uma mensagem de erro é exibida ao utilizador.

6.2. Publicação de Livros e Postagens (REQ0002, REQ0003, REQ0004)

As funcionalidades de publicação são o cerne da interação de conteúdo no Litly, permitindo aos utilizadores partilhar informações sobre livros e criar as suas próprias postagens. Os formulários `AdicionarLivros.cs` e `PaginaPostagem.cs` são centrais para estas operações, interagindo com as tabelas `Livros` e `Postagens`.

- **Adicionar Livros:**
  - O formulário `AdicionarLivros.cs` permite ao utilizador inserir detalhes de um livro (título, autor, sinopse, imagem).

    - A imagem do livro é convertida para um formato binário (`VARBINARY(MAX)`) antes de ser armazenada na base de dados.
    - Os dados são inseridos na tabela `Livros`.
- **Criar Postagens:**
    - O formulário `PaginaPostagem.cs` (ou um formulário similar) permite ao utilizador criar uma nova postagem.
    - Uma postagem pode ser sobre um livro existente (referenciando `IdLivro`) ou um texto autoral (`Conteudo`).
    - O `IdUtilizador` da sessão atual é associado à postagem.
    - Os dados são inseridos na tabela `Postagens`.
- **CRUD de Postagens:**
    - **Visualizar:** A `PaginaPrincipal.cs` exibe um feed de postagens, recuperando dados da tabela `Postagens` e, se aplicável, da tabela `Livros` e `Utilizadores` para exibir informações completas (autor, título, imagem, conteúdo).
    - **Atualizar/Apagar:** Funcionalidades para editar ou remover postagens existentes seriam implementadas através de botões ou menus contextuais nas postagens, que acionariam formulários de edição ou diálogos de confirmação, e subsequentemente atualizariam ou eliminariam os registos na tabela `Postagens`.

6.3. Sistema de Gostos e Comentários (REQ0005)

Estas funcionalidades promovem a interação social entre os utilizadores, permitindo-lhes expressar apreço e trocar ideias sobre as postagens. As tabelas `Gostos` e `Comentarios` são utilizadas.

- **Gostos:**
    - Quando um utilizador clica no botão 'Gostar' numa postagem, um novo registo é inserido na tabela `Gostos`, associando o `IdUtilizador` à `IdPostagem`.

- A aplicação deve verificar se o utilizador já gostou daquela postagem para evitar duplicados e permitir 'desgostar'.
- A contagem de gostos para uma postagem é obtida através de uma query `COUNT` na tabela `Gostos`.

- **Comentários:**
  - Quando um utilizador submete um comentário, o `Conteudo` do comentário, `IdUtilizador` e `IdPostagem` são inseridos na tabela `Comentarios`.
  - Os comentários são exibidos abaixo das postagens, recuperando os dados da tabela `Comentarios` e associando-os aos `Utilizadores` para exibir o nome do autor do comentário.

6.4. Pesquisa por Livro ou Utilizador (REQ0006)

A funcionalidade de pesquisa permite aos utilizadores encontrar rapidamente conteúdos ou outros utilizadores na plataforma. A implementação envolve a consulta às tabelas `Livros` e `Utilizadores`.

- A interface de pesquisa (provavelmente na `PaginaPrincipal.cs` ou um formulário dedicado) permite ao utilizador inserir termos de pesquisa.
- A aplicação executa queries SQL com `LIKE` ou `CONTAINS` nas colunas `Titulo`s `Autor` `Email` , (para livros) e `Nome` , (para utilizadores).
- Os resultados da pesquisa são exibidos numa lista ou grelha, permitindo ao utilizador navegar para os detalhes do livro ou perfil do utilizador.

6.5. Gestão de Perfil de Utilizador (REQ0007)

Os utilizadores podem personalizar e visualizar as suas informações através do formulário `PerfilUtilizador.cs`.

- **Visualização do Perfil:**
  - O `PerfilUtilizador.cs` exibe as informações do utilizador logado (nome, email, biografia, imagem de perfil) recuperadas da tabela `Utilizadores`.

    - Também exibe as postagens criadas pelo utilizador, recuperando dados da tabela `Postagens` filtrando por `IdUtilizador`.

- **Edição do Perfil:**
    - Permite ao utilizador atualizar a sua biografia, imagem de perfil ou outras informações pessoais.
    - As alterações são validadas e, em seguida, um comando `UPDATE` é executado na tabela `Utilizadores`.

Esta abordagem modular e a clara separação de responsabilidades facilitam a compreensão do fluxo de dados e a manutenção das funcionalidades da aplicação Litly.

7. Considerações de Segurança

A segurança é um aspeto crítico no desenvolvimento de qualquer aplicação, especialmente aquelas que lidam com dados de utilizadores. No projeto Litly, foram consideradas diversas práticas de segurança para proteger a aplicação e os dados armazenados. Esta secção aborda as principais considerações de segurança e as medidas implementadas ou recomendadas.

7.1. Validação de Entrada de Dados

Todas as entradas de dados fornecidas pelos utilizadores devem ser rigorosamente validadas antes de serem processadas ou armazenadas na base de dados. A validação ajuda a prevenir ataques comuns como Injeção de SQL, Cross-Site Scripting (XSS) e outros tipos de manipulação de dados. No Litly, a validação deve ser implementada tanto no lado do cliente (na interface Windows Forms, para feedback imediato ao utilizador) quanto no lado do servidor (na camada de lógica de negócio, para garantir a integridade dos dados).

- **Exemplos de Validação:**
    - **Emails:** Verificar o formato do email (`@`, `.com`, etc.).
    - **Palavras-passe:** Impor requisitos de complexidade (tamanho mínimo, caracteres especiais, números, letras maiúsculas/minúsculas).

- **Campos de Texto:** Limitar o comprimento máximo dos campos de texto para evitar overflows e truncamentos indesejados. Filtrar ou escapar caracteres especiais que possam ser usados em ataques de injeção.
- **Números:** Garantir que os campos numéricos contêm apenas dígitos e estão dentro de um intervalo esperado.

7.2. Prevenção de Injeção de SQL (SQL Injection)

A Injeção de SQL é uma das vulnerabilidades mais comuns e perigosas. O Litly utiliza **queries parametrizadas** para todas as interações com a base de dados, o que é a principal defesa contra este tipo de ataque. Em vez de concatenar strings para construir queries SQL, os valores de entrada são passados como parâmetros, garantindo que a base de dados os trate como dados e não como parte do código SQL.

- **Exemplo (Pseudocódigo C# com ADO.NET):** ```csharp // EVITAR: string query = "SELECT * FROM Utilizadores WHERE Email = '" + email + "' AND PalavraPasse = '"

  + password + "'";

  // CORRETO: string query = "SELECT IdUtilizador, Nome FROM Utilizadores WHERE Email = @Email AND PalavraPasse = @PalavraPasse"; using (SqlConnection connection = new SqlConnection(connectionString)) { using (SqlCommand command = new SqlCommand(query, connection)) { command.Parameters.AddWithValue("@Email", email); command.Parameters.AddWithValue("@PalavraPasse", hashedPassword); connection.Open(); // ... executar e ler resultados } } ```

7.3. Armazenamento Seguro de Palavras-Passe

As palavras-passe dos utilizadores **nunca** devem ser armazenadas em texto simples na base de dados. Em vez disso, devem ser armazenadas como hashes criptográficos, idealmente com um ***salt*** único para cada palavra-passe. O Litly deve implementar:

- **Hashing:** Utilizar algoritmos de hashing seguros (e.g., SHA256, SHA512, ou preferencialmente funções de derivação de chave como PBKDF2, bcrypt ou scrypt) para converter a palavra-passe em uma string de comprimento fixo e irreversível.
- **Salting:** Adicionar um valor aleatório único (salt) a cada palavra-passe antes de fazer o hash. Isso impede ataques de tabela arco-íris e garante que duas

palavras-passe idênticas resultem em hashes diferentes.

Ao autenticar, a palavra-passe fornecida pelo utilizador é hashed com o salt armazenado e o resultado é comparado com o hash armazenado na base de dados.

7.4. Tratamento de Erros e Logging

Um tratamento de erros adequado é crucial para a segurança. Mensagens de erro detalhadas não devem ser exibidas diretamente ao utilizador final, pois podem revelar informações sensíveis sobre a arquitetura interna da aplicação ou da base de dados. Em vez disso, erros devem ser registados em ficheiros de log seguros (logging) para análise por parte dos programadores, e uma mensagem de erro genérica e amigável deve ser apresentada ao utilizador.

7.5. Princípio do Menor Privilégio

As contas de utilizador da base de dados utilizadas pela aplicação devem ter apenas as permissões mínimas necessárias para executar as suas funções. Por exemplo, a conta que a aplicação usa para se conectar ao SQL Server não deve ter permissões de `DROP TABLE` ou `ALTER DATABASE`, apenas `SELECT`, `INSERT`, `UPDATE` e `DELETE` nas tabelas relevantes.

7.6. Proteção de Dados Sensíveis

Qualquer dado sensível (como imagens de perfil, se forem consideradas sensíveis) deve ser tratado com cuidado. Embora o Litly armazene imagens como `VARBINARY(MAX)`, é importante considerar a encriptação de dados em repouso (na base de dados) e em trânsito (durante a comunicação entre a aplicação e a base de dados) para ambientes de produção.

Ao seguir estas diretrizes de segurança, o projeto Litly pode oferecer uma experiência mais segura e confiável para os seus utilizadores e proteger a integridade dos seus dados.

8. Resolução de Problemas

Esta secção fornece orientações para diagnosticar e resolver problemas comuns que podem surgir durante o desenvolvimento, compilação ou execução do projeto Litly.

8.1. Problemas de Conexão com a Base de Dados

Sintoma: A aplicação não consegue conectar-se ao SQL Server, resultando em erros como "A network-related or instance-specific error occurred while establishing a connection to SQL Server" ou "Login failed for user '...' ".

Possíveis Causas e Soluções:

- **SQL Server não está a ser executado:**
  - **Verificação:** Abra o `SQL Server Configuration Manager` e certifique-se de que o serviço `SQL Server (MSSQLSERVER)` ou a sua instância nomeada (e.g., `SQLEXPRESS`) está em execução.
  - **Solução:** Inicie o serviço.
- **Nome da Instância Incorreto:**
  - **Verificação:** A string de conexão (`Data Source`) pode estar a apontar para uma instância de SQL Server incorreta. Verifique o nome da instância no `SQL Server Management Studio (SSMS)` ou no `SQL Server Configuration Manager`.
  - **Solução:** Corrija o `Data Source` na sua string de conexão (e.g., `localhost\SQLEXPRESS`, `(localdb)\MSSQLLocalDB`).
- **Firewall a Bloquear a Conexão:**
  - **Verificação:** O firewall do Windows pode estar a bloquear a porta utilizada pelo SQL Server (padrão é 1433).
  - **Solução:** Adicione uma exceção no firewall para a porta 1433 ou para o executável do SQL Server (`sqlservr.exe`).
- **Autenticação Incorreta:**
  - **Verificação:** Se estiver a usar autenticação SQL Server, verifique se o nome de utilizador e a palavra-passe estão corretos na string de conexão. Se estiver a usar autenticação Windows (`Integrated Security=True`), certifique-se de que o utilizador que executa a aplicação tem permissões para aceder à base de dados.
  - **Solução:** Corrija as credenciais ou as permissões do utilizador.
- **Base de Dados Inexistente ou Nome Incorreto:**

- **Verificação:** Certifique-se de que a base de dados `Litly` (ou o nome que usou) existe no seu SQL Server e que o `Initial Catalog` na string de conexão está correto.
- **Solução:** Crie a base de dados ou corrija o nome na string de conexão.

8.2. Erros de Compilação

Sintoma: O Visual Studio apresenta erros durante a compilação do projeto (linhas vermelhas no código, mensagens de erro no `Error List`).

Possíveis Causas e Soluções:

- **Pacotes NuGet em Falta:**
  - **Verificação:** Mensagens de erro como "The type or namespace name 'Guna' could not be found" indicam que os pacotes NuGet não foram restaurados corretamente.
  - **Solução:** No `Solution Explorer`, clique com o botão direito do rato na solução e selecione `Restore NuGet Packages`.
- **Versão do .NET Framework Incorreta:**
  - **Verificação:** O projeto está configurado para .NET 8.0. Se tiver uma versão diferente instalada ou selecionada, podem ocorrer erros de compatibilidade.
  - **Solução:** Verifique se o .NET 8.0 SDK está instalado e se o `TargetFramework` no `Litly.02.csproj` está definido para `net8.0-windows`.
- **Erros de Sintaxe no Código:**
  - **Verificação:** Erros de sintaxe (e.g., ponto e vírgula em falta, variáveis não declaradas) serão indicados pelo compilador.
  - **Solução:** Revise o código nas linhas indicadas pelo `Error List` e corrija os erros de sintaxe.
- **Ficheiros `.Designer.cs` Modificados Manualmente:**
  - **Verificação:** Modificar manualmente os ficheiros `.Designer.cs` pode causar erros de compilação ou comportamento inesperado.

- **Solução:** Reverter as alterações ou, em casos extremos, recriar o formulário (o que deve ser evitado).

8.3. Comportamento Inesperado da Aplicação

Sintoma: A aplicação compila e executa, mas as funcionalidades não se comportam como esperado (e.g., dados não são guardados, botões não funcionam).

Possíveis Causas e Soluções:

- **Lógica de Negócio Incorreta:**
  - **Verificação:** Utilize o depurador do Visual Studio (`F5` para iniciar a depuração, `F9` para definir breakpoints) para percorrer o código e verificar o fluxo de execução e os valores das variáveis.
  - **Solução:** Corrija a lógica nos métodos relevantes.
- **Problemas de Acesso a Dados:**
  - **Verificação:** Verifique as queries SQL e os comandos ADO.NET. Utilize o SSMS para executar as queries diretamente na base de dados e verificar se os resultados são os esperados.
  - **Solução:** Ajuste as queries SQL ou a forma como os dados são lidos/escritos.
- **Dados Inconsistentes na Base de Dados:**
  - **Verificação:** Verifique a integridade dos dados na base de dados. Registos em falta, duplicados ou com valores incorretos podem causar problemas.
  - **Solução:** Limpe ou corrija os dados na base de dados. O script `REFIZATABELA.txt` pode ser útil para recriar a estrutura da base de dados.
- **Eventos Não Atribuídos:**
  - **Verificação:** Certifique-se de que os eventos dos controlos (e.g., `Click` de um botão) estão corretamente atribuídos aos seus manipuladores de eventos no ficheiro `.Designer.cs` ou no código.
  - **Solução:** No `Properties` do controlo no Visual Studio, verifique a secção de eventos e atribua o método correto.

8.4. Problemas de Desempenho

Sintoma: A aplicação está lenta, especialmente ao carregar dados ou realizar operações na base de dados.

Possíveis Causas e Soluções:

- **Queries SQL Ineficientes:**
  - **Verificação:** Utilize o `SQL Server Profiler` ou as ferramentas de `Execution Plan` no SSMS para analisar o desempenho das queries SQL. Queries sem índices adequados ou com `SELECT *` em tabelas grandes podem ser lentas.
  - **Solução:** Otimize as queries, adicione índices às colunas frequentemente pesquisadas ou filtradas, e selecione apenas as colunas necessárias.
- **Carregamento Excessivo de Dados:**
  - **Verificação:** A aplicação pode estar a carregar uma quantidade excessiva de dados da base de dados de uma só vez.
  - **Solução:** Implemente paginação para carregar dados em blocos menores, ou utilize carregamento preguiçoso (lazy loading) para dados que não são imediatamente necessários.

Ao seguir estas diretrizes de resolução de problemas, os programadores podem identificar e corrigir eficientemente a maioria dos problemas que possam surgir no projeto Litly.

9. Conclusão

Este manual forneceu uma visão abrangente da arquitetura, estrutura de código, configuração do ambiente de desenvolvimento e considerações de segurança do projeto Litly. O objetivo foi capacitar programadores e integradores de sistemas com o conhecimento necessário para compreender, manter e estender esta aplicação desktop desenvolvida em C# e Windows Forms. A adesão às melhores práticas de desenvolvimento, como a arquitetura em camadas, a validação de dados e o armazenamento seguro de palavras-passe, é fundamental para garantir a robustez e a segurança da aplicação.

Com as informações aqui apresentadas, espera-se que os desenvolvedores possam navegar eficientemente pelo código-fonte, configurar o ambiente de desenvolvimento, interagir com a base de dados SQL Server e implementar novas funcionalidades de forma consistente com a arquitetura existente. O Litly, como uma plataforma interativa para entusiastas da leitura, tem um potencial significativo para futuras expansões e melhorias, e este manual serve como um guia essencial para qualquer intervenção técnica no projeto.

10. Referências

- Microsoft Docs – https://learn.microsoft.com
- C# Windows Forms GUI Tutorial – https://www.geeksforgeeks.org
- SQL Server Basics – https://www.w3schools.com/sql
- Artigo: "Como criar uma rede social com C#" – Medium
- Guna UI Framework: https://guna.net/