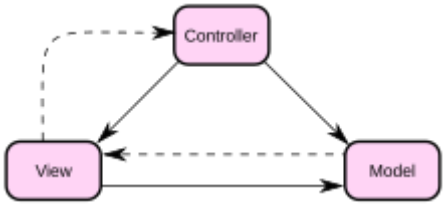
LumiTemp – Project Based Learning

O projeto do LumiTemp contém um modelo de construção de exibição de informações montado em uma modelagem MVC (*Model-View-Controller*), um padrão de design de software, ou padrão arquitetural, criado na década de 1970, com foco na reutilização de código e na separação de responsabilidades em três camadas interligadas. Nesse modelo, a camada responsável pela apresentação de dados e pela interação com o usuário (*front-end*) é mantida separada dos métodos que lidam com o banco de dados (*back-end*).

Comumente aplicado no desenvolvimento de interfaces de usuário, esse padrão divide uma aplicação em diferentes camadas ou componentes conectados. Essa abordagem permite que as representações internas da informação sejam isoladas das formas de apresentação e entrada de dados pelo usuário, promovendo o desenvolvimento paralelo de maneira mais eficiente.



Fonte: Wikipedia

Funcionalidade do modelo MVC (Model-View-Controller)

A arquitetura MVC (Model-View-Controller) é um padrão amplamente utilizado no desenvolvimento de software, especialmente em aplicações web, devido à sua estrutura modular que separa responsabilidades em três componentes principais: Model, View e Controller. Cada um desses componentes desempenha um papel fundamental na organização e manutenção do código, permitindo o desenvolvimento de sistemas robustos e escaláveis.

A Model é a camada que lida com a lógica de negócios e a manipulação de dados. Ela é responsável por representar os dados da aplicação e gerenciar as interações com o banco de dados. No contexto do projetom, compreende-se em três Models, uma para cadastro de funcionários autorizados a manipularem dados e informações dos sensores e das empresas clientes, uma Model para cadastro de empresas parceiras e uma última para cadastro de sensores. Models são seria as partes que definem os atributos de cada item ou contexto. Além de definir as propriedades dos objetos, a Model também valida os dados, garantindo que as informações armazenadas estejam corretas e dentro das regras de

negócio. A Model pode ainda encapsular cálculos, consultas complexas ao banco de dados e até mesmo realizar operações como filtragens e ordenações.

A View é a parte da aplicação que o usuário final vê e interage. Ela é responsável por exibir os dados de maneira organizada e interativa, fornecendo uma interface que facilita a navegação e o uso do sistema. No caso do exemplo anterior, a View pode exibir uma lista de pacientes cadastrados, formulários para inserção de novos dados ou gráficos que apresentam estatísticas sobre os atendimentos. A View é desenvolvida com tecnologias de apresentação como HTML, CSS e JavaScript, e deve focar exclusivamente na experiência visual, sem conhecimento das regras de negócios ou manipulação de dados diretamente. Além disso, ela recebe os dados processados pela Controller e os exibe em um formato amigável ao usuário.

A Controller desempenha o papel de intermediária entre a Model e a View, sendo a responsável por processar as interações do usuário e coordenar o fluxo de dados entre as outras duas camadas. Quando o usuário interage com a interface — por exemplo, ao enviar um formulário de cadastro de pacientes — a Controller intercepta essa ação, valida as informações, chama a Model para armazenar ou processar os dados, e depois define qual View será exibida em seguida. A Controller é a peça central que organiza o comportamento da aplicação, determinando como as diferentes partes do sistema respondem às solicitações dos usuários.

Essa separação de responsabilidades traz inúmeros benefícios para o desenvolvimento de software. Um dos principais é a manutenção e escalabilidade do código. Como cada camada tem uma função clara e bem definida, modificações podem ser feitas de maneira isolada, sem afetar o restante da aplicação. Se for necessário alterar a forma como os dados são exibidos, isso pode ser feito na View, sem tocar na Model ou na Controller. Da mesma forma, se houver mudanças na lógica de negócios, como novos campos a serem adicionados aos pacientes, isso pode ser tratado diretamente na Model, sem interferir na exibição.

Outro benefício significativo da arquitetura MVC é a facilidade no desenvolvimento paralelo. Equipes diferentes podem trabalhar simultaneamente nas três camadas sem conflito. Por exemplo, uma equipe de desenvolvedores de front-end pode focar na criação de interfaces de usuário, enquanto os desenvolvedores de back-end concentram-se nas regras de negócio e no banco de dados. A Controller age como um ponto de conexão entre essas equipes, integrando os dois mundos e garantindo que as interações funcionem corretamente.

Além disso, o MVC facilita a testabilidade da aplicação. Como as responsabilidades estão separadas, é possível testar cada componente individualmente. Testes unitários podem ser realizados na Model para garantir

que as regras de negócio estão corretas, enquanto testes de interface podem ser aplicados na View para verificar a experiência do usuário. A Controller também pode ser testada para assegurar que ela está direcionando corretamente as solicitações e manipulando os dados de maneira adequada.

Aplicação dos padrões MVC na construção do projeto

Banco de Dados

A utilização de um banco de dados em um projeto MVC (Model-View-Controller) é fundamental para o gerenciamento eficaz de dados, permitindo que as aplicações armazem, acessem e manipulem informações de maneira estruturada e eficiente. No padrão MVC, o modelo é responsável pela lógica de dados, o que significa que ele interage diretamente com o banco de dados para realizar operações como inserção, atualização, exclusão e consulta de dados. Isso garante que as informações sejam mantidas de forma consistente e integridade, além de facilitar o escalonamento da aplicação à medida que os requisitos de dados crescem.

Além disso, um banco de dados proporciona uma base sólida para a persistência de dados, permitindo que as informações sejam recuperadas e apresentadas na interface do usuário (view) de forma dinâmica e em tempo real. Isso é especialmente importante em aplicações que exigem a gestão de grandes volumes de dados, como sistemas de gerenciamento de clientes, e-commerce ou aplicações financeiras. A separação de responsabilidades que o padrão MVC oferece, combinada com a capacidade de um banco de dados, permite que desenvolvedores construam aplicações mais modulares, manuteníveis e escaláveis, promovendo uma melhor experiência para o usuário final.

Sendo assim, o software conta com um banco de dados construido seguindo os padrões internacionais de bussiness and analytics models, com três tabelas diferentes, uma com o nome de ‘cadr_func’ que é responsável por guardar informações dos funcionários:

Nome do campo	*Tipo do campo*	*Descrição do campo*	*Key*	*Foreing Key*
CD_FUNC	INT	Número único do usuário	Sim	Não
LOGIN	STRING	Login de cadastro do usuário	Sim	Não
SENHA	STRING	Senha de cadastro do usuário	Não	Não
DT_CADR	DATE	Data de cadastro do usuário	Não	Não

Uma segunda tabela com o nome de ‘cadr_empr_parc’ contém informações sobre as empresas parceiras que compram os sensores da empresa ficiticia

criada:

Nome do campo	*Tipo do campo*	*Descrição do campo*	*Key*	*Foreing Key*
CD_EMPR	INT	Código do motor que está sendo feita a leitura	Sim	Não
NM_EMPR	STRING	Nome da empresa parceira	Não	Não
CEP_EMPR	STRING	CEP da empresa parceira	Não	Não
CNPJ_EMPR	STRING	Número do CNPJ da empresa parceira	Não	Não
TELF_CONT_EMPR	STRING	Telefone da empresa parceira	Não	Não
CD_FUNC	INT	Número único do usuário	Não	Sim

E uma última tabela com o nome de 'cadr_sens' que armazena informações gerais sobre a venda dos sensores e o relacionamento dos mesmos com as empresas compradoras e os funcionários vendedores:

Nome do campo	*Tipo do campo*	*Descrição do campo*	*Key*	*Foreing Key*
CD_SENS	INT	Código do sensor	Sim	Não
DS_TIPO_SENS	STRING	Descrição do tipo de sensor	Não	Não
DT_HR_LEIT	DATE	Data e hora da leitura do sensor	Não	Não
VL_TEMP	DECIMAL	Valor lido da temperatura	Não	Não
VL_UMID	DECIMAL	Valor lido da umidade	Não	Não
VL_TEMP_ALVO	DECIMAL	Valor alvo de temperatura	Não	Não
VL_UMID_ALVO	DECIMAL	Valor alvo de umidade	Não	Não
CD_MOTOR	INT	Código do motor que está sendo feita a leitura	Não	Não
FK_CD_USUA	INT	Código do usuário que cadastrou o sensor	Não	Sim
FK_CD_EMPR	INT	Código da empresa da qual o sensor está alocado/vendido	Não	Sim

Models

O projeto conta com todas as completudes dos padrões MVC, ao inciar pelos modelos de completude da ideia de aplicação do projeto, temos a construção por padrão das Models, Views e por fim, as Controllers. O mesmo está construido em um modelo que faz o uso da linguagem C# (C sharp), e contém um WorksSpace único

Ao iniciar-se pelas Views, podemos observar três modelos montados de maneira distinta, um modelo contempla todos os atributos e definições de metadados das mesmas para cada. Iniciando a analise pelo modelo de View que coleta e define os atributos dados aos funcionários. Essa classe contém quatro propriedades principais. A propriedade cd_func é responsável por armazenar o número de identificação do funcionário (ID), funcionando como chave única para identificar cada funcionário no sistema. Já a propriedade login_func armazena o nome de usuário utilizado pelo funcionário para acessar o sistema, enquanto senha_func guarda a senha associada a esse login, garantindo a autenticação e segurança

do acesso. Além disso, a classe possui uma propriedade chamada dt_cadr, que armazena a data de cadastro do funcionário no sistema, utilizando o tipo DateTime da biblioteca padrão do C#. Essas propriedades são implementadas com os modificadores get e set, permitindo tanto a leitura quanto a escrita dos valores.

Em um segundo momento, compreende-se também na estrutura geral do projeto, uma segunda Model com a construção e informações das empresas parceiras que são compradoras dos sensores vendidos na ideia de concepção de atividades da empresa, A classe contém várias propriedades que armazenam dados essenciais sobre a empresa. A propriedade cd_empr é usada para armazenar o código identificador da empresa, que age como uma chave única (ID) para cada registro. Além disso, a propriedade nm_empr guarda o nome da empresa, enquanto cep_empr e cnpj_empr armazenam, respectivamente, o CEP e o CNPJ da empresa, garantindo que as informações de localização e identificação fiscal estejam corretas. Além desses campos, a classe possui a propriedade telf_cont_empr, que armazena o telefone de contato da empresa parceira, facilitando a comunicação. A última propriedade, fk_cd_func, é uma chave estrangeira que vincula o ID de um funcionário responsável ou associado à empresa parceira. Todas essas propriedades são implementadas com os modificadores get e set, permitindo que os dados sejam lidos e atualizados conforme necessário.

Ainda nesse contexto, o projeto apresenta uma terceira Model capaz de definir e registrar os atributos para os sensores que são vendidos pela empresa ficticia, A propriedade cd_sens armazena o código do sensor, que atua como identificador único (ID). Já a propriedade ds_tipo_sens armazena a descrição do tipo de sensor, permitindo distinguir os diferentes tipos de sensores utilizados na aplicação. Além disso, a propriedade dt_vend, que armazena a data de venda do sensor, é um campo do tipo DateTime?, o que significa que ele é nullable, permitindo que não tenha um valor definido se necessário. Além das informações básicas, a classe SensorViewModel também contém propriedades relacionadas à configuração do sensor. As propriedades vl_temp_alvo e vl_umid_alvo armazenam os valores de temperatura e umidade alvo configurados para o sensor, sendo ambos campos nullable do tipo decimal?, permitindo que essas propriedades possam ser deixadas em branco, se desejado. A propriedade cd_motor armazena o código do motor associado ao sensor, representando a conexão entre o sensor e o motor monitorado. Por fim, as propriedades fk_cd_func e fk_cd_empr representam, respectivamente, o código do funcionário e o código da empresa associados ao sensor, funcionando como chaves estrangeiras que indicam o vínculo entre o sensor, o usuário responsável e a empresa parceira.

Por fim, o projeto apresenta uma Model de erros que é utilizada para representar informações sobre erros que podem ocorrer no sistema. A classe possui dois

construtores: um que aceita uma mensagem de erro como parâmetro, permitindo inicializar o campo Erro, e outro construtor vazio que possibilita a criação de instâncias sem a necessidade de valores iniciais. Essa flexibilidade é útil em diferentes contextos onde a classe pode ser utilizada, como em tratamentos de exceção e relatórios de erro.

Além da propriedade Erro, que armazena a mensagem de erro, a classe também possui a propriedade RequestId, que é útil para rastrear a requisição que causou o erro em logs, facilitando o diagnóstico de problemas. A propriedade booleana ShowRequestId verifica se o RequestId foi preenchido, retornando verdadeiro se não estiver vazio ou nulo. Essa funcionalidade permite que a interface de usuário decida se deve exibir o ID da requisição, o que pode ajudar na identificação e resolução de erros em ambientes de produção.

DAOs

Fugindo um pouco do modelo padrão de construção, o projeto apresenta uma quarta classe de construção geral, as chamadas DAO (Data Access Object) é uma classe ou conjunto de classes responsáveis por realizar a comunicação entre a aplicação e o banco de dados. A função da DAO é encapsular todas as operações de acesso aos dados, como consultas, inserções, atualizações e exclusões, garantindo que a lógica de interação com o banco esteja separada da lógica de negócios. Isso promove a organização do código e facilita a manutenção, uma vez que qualquer modificação na forma como os dados são acessados pode ser feita diretamente na DAO, sem impactar outras partes do sistema. A DAO também contribui para a reutilização de código, permitindo que operações comuns com o banco de dados sejam centralizadas em métodos que podem ser chamados por diferentes partes da aplicação. No padrão MVC, a DAO trabalha em conjunto com a camada Model, manipulando os dados que o Model representa, enquanto a Controller usa a DAO para recuperar ou persistir informações no banco. Essa separação de responsabilidades torna o sistema mais modular e facilita a implementação de testes, além de permitir maior flexibilidade em mudanças futuras, como a troca de banco de dados ou de tecnologias de persistência de dados.

Nesse contexto, também se demonstrou necessário a construção de modelos DAO de maneira separada, para cada “item-chave” dentro do software, ou seja, dentro do projeto, foram concebidas três DAO’s, uma para funcionários cadastrados, uma segunda para empresas parceiras e uma última para sensores vendidos e devidamente cadastrados.

Seguindo um padrão de construção geral, todas as DAO contém os mesmos atributos e ações entre sí, alterando-se somente para qual View a DAO aponta de maneira direta e clara, porém, a fim de se simplificar a construção do software,

foi desenvolvida uma DAO com a função de auxiliar na velocidade de se declarar nos códigos de maneira separada, funções siginificativamente extensas em forma de simples chamdas dento do código, método ExecutaSQL é utilizado para executar comandos SQL que não retornam dados, como operações de inserção, atualização ou exclusão. Ele estabelece uma conexão com o banco de dados através do método GetConexao da classe ConexaoDB e cria um objeto SqlCommand para executar o comando SQL fornecido. Caso parâmetros sejam passados, eles são adicionados ao comando antes da execução, que é realizada com o método ExecuteNonQuery, fechando a conexão automaticamente após a conclusão da operação.

Além disso, a classe HelperDAO oferece o método ExecutaSelect, que permite a execução de consultas SQL que retornam dados, como instruções SELECT. Semelhante ao primeiro método, ele cria uma conexão com o banco de dados e utiliza um SqlDataAdapter para executar a consulta SQL. Se houver parâmetros, eles são anexados ao comando do adapter, e os resultados da consulta são armazenados em um DataTable, que é preenchido com os dados retornados. Por fim, o DataTable preenchido é retornado, permitindo que outras partes da aplicação utilizem os dados obtidos de forma estruturada e fácil. Essa organização de código favorece a reutilização e manutenção, centralizando a lógica de acesso a dados em um único local.

As demais classes DAO são responsáveis por realizar operações básicas de CRUD (Create, Read, Update, Delete) para registros, utilizando o padrão de design DAO que abstrai a lógica de acesso a dados do restante da aplicação. O código utiliza classes do namespace System.Data.SqlClient para executar comandos SQL e manipular os dados retornados, como SqlConnection, SqlCommand, e SqlParameter.

O método CriaParametros cria um array de parâmetros que são utilizados nas instruções SQL para evitar a injeção de SQL e garantir que os dados sejam passados de forma segura. Os métodos Inserir, Alterar, e Excluir realizam operações de inserção, atualização e exclusão de registros de funcionários no banco de dados, utilizando a classe auxiliar HelperDAO para executar as instruções SQL. O método Consulta permite buscar um funcionário específico pelo seu ID, enquanto ConsultaTodos e Listagem retornam listas de funcionários, permitindo a visualização de todos os registros.

Além disso, a classe inclui um método ProximoId, que busca o próximo ID disponível para um novo registro, garantindo a unicidade dos identificadores. O método MontaModel converte um DataRow retornado de uma consulta em um objeto.

Controllers

Tomando como base o uso geral dos mesmos métodos para diferentes referências contextualizadoras da construção da empresa (no caso, funcionários, empresas parceiras e sensores), as controllers mantém os mesmos padrões de construção para cada uma das referências contextualizadoras.

Os controladores herdam de Controller, permitindo que eles possam manipular requisições HTTP e retornar respostas apropriadas. O método principal das classes, Index, é responsável por exibir a lista de itens cadastrados para cada referência. Ao ser chamado, ele instancia o objeto DAO de cada referência para acessar os dados do banco de dados, busca a lista de cada referência e retorna uma view que exibe essas informações.

O método Create é responsável pela criação de um novo item. Ele inicia a construção de um novo objeto a partir da Model. Usando o DAO, o controlador obtém o próximo ID disponível para o novo registro e retorna a view do formulário, onde os dados da referência podem ser preenchidos. Caso ocorra algum erro durante esse processo, o controlador redireciona para uma página de erro, passando uma mensagem de erro apropriada.

O método Salvar serve para tanto inserir um novo item quanto atualizar um existente. Ele verifica se o item já existe no banco de dados usando o método Consulta do DAO. Se o item não for encontrado, o método Inserir é chamado para adicionar o novo registro. Se o item já existir, o método Alterar é utilizado para atualizar suas informações. Após a operação, o controlador redireciona para a página inicial, onde a lista de itens é exibida.

O método Edit permite editar os dados de um item existente. Ao ser chamado com um código único do item, ele consulta o banco de dados para obter os dados correspondentes. Se o item for encontrado, o controlador retorna a view do item com os dados preenchidos para edição. Se não for encontrado, um redirecionamento para a lista de itens é realizado. Como nos outros métodos, caso ocorra um erro, uma página de erro é apresentada.

Por fim, o método Delete é responsável pela exclusão de um item pelo código único. Ele utiliza o DAO para realizar a exclusão no banco de dados e, em seguida, redireciona o usuário de volta para a lista de funcionários. Assim como nos outros métodos, a classe lida com exceções, retornando uma view de erro caso algum problema ocorra durante a execução.

Views

Na completude final que compoem um projeto arquitetado em MVC, as Views possuem caracteristicas mais populares, pois como já explicado antes, as mesmas compoem a parte visual e de interação com o usuário do software. Sendo assim, projeto contém sete Views diferentes, uma para cada contexto.

Ao se iniciar pela View de home, a mesma faz a construção de uma tela que apresenta um design limpo e organizado, com um cabeçalho que contém uma barra de navegação no topo. Essa barra é composta por uma lista de links dispostos horizontalmente, permitindo que o usuário acesse diferentes seções do sistema, como cadastro de sensores, empresas, funcionários, um dashboard e uma página "Sobre". Logo abaixo do cabeçalho, há uma seção centralizada que apresenta um painel de login. Este painel é simples e direto, contendo um título "Login" destacado. Dentro do painel, o usuário encontra dois campos: um para inserir o nome de usuário e outro para a senha, ambos acompanhados por rótulos explicativos. Abaixo dos campos, há um botão grande e visível com a inscrição "Entrar", que convida o usuário a efetuar o login. O layout geral é amigável, com um esquema de cores e tipografia que favorece a legibilidade e a facilidade de uso.

A segunda view a ser citada na construção do projeto, é a de sobre, que contém informações sobre a construção da empresa criada para o conceito ficticio do projeto, além de seguir os padrões visuais usados para todas as Views, mantém, assim como todas as outras também, o cabeçalho de exibição initerrupta.

A View que exibe registros e ações a serem feitas com os dados dos funcionários, apresentam os mesmos padrões para as que referenciam as empresas e as empresas parceiras e os sensores. Todas Views apresentam dois modelos de exibição que se conectam. Os dois códigos apresentam interfaces para a gestão dos itens das referências dentro do sistema, cada uma com seu propósito específico. O primeiro código é voltado para o cadastro de um novo item da referência, apresentando um formulário que solicita informações. A tela é organizada, com rótulos claros e campos obrigatórios, permitindo ao usuário inserir as informações necessárias antes de clicar no botão "Salvar Dados". Além disso, há um link que redireciona o usuário de volta à página inicial dos funcionários, facilitando a navegação. O segundo código, por sua vez, exibe uma lista de itens da referêncoa já cadastrados no sistema, permitindo ações como editar ou excluir registros. A página é apresentada de forma clara, com uma tabela que lista os itens com todos os detalhes. Há também botões que facilitam a adição de novos registros e a extração de dados, além de permitir a exclusão de itens após uma confirmação. Juntas, essas duas telas oferecem uma experiência completa de gerenciamento de itens de cada referência, desde o cadastro até a visualização e edição dos dados já existentes, promovendo uma interação eficiente e organizada.

Por fim, contamos com a View do Dashboard de visualização de alterações de temperatura dos sensores dentro do sistema, essa tela, coleta dados dos sensores atraves da API Finware que será construida em um outros momento.