Introdução
O sistema permite realizar movimentações (créditos e débitos) em contas correntes e consultar o saldo das contas. A seguir, uma descrição detalhada do sistema:

Arquitetura:

- O sistema segue uma arquitetura monolítica de três camadas lógica: Apresentação, Aplicação e Infraestrutura.
- A camada de Apresentação contém o controlador `ContaCorrenteController`, responsável por receber as requisições HTTP e encaminhá-las para a camada de Aplicação.
- A camada de Aplicação contém os serviços (`ContaCorrenteService`, `MovimentoService` e `IdempotenciaService`), responsáveis por implementar as regras de negócio, validações e orquestrar as operações nos repositórios.
- A camada de Infraestrutura contém os repositórios (`ContaCorrenteRepository`, `MovimentoRepository` e `IdempotenciaRepository`), responsáveis por executar as operações de persistência de dados no banco de dados SQLite.

Entidades:

- `ContaCorrente`: Representa uma conta corrente bancária, contendo informações como número, nome do titular e status (ativa ou inativa).
- `Movimento`: Representa uma movimentação (crédito ou débito) em uma conta corrente, contendo informações como o valor, data e tipo de movimento (crédito ou débito).
- `Idempotencia`: Armazena informações sobre as requisições já processadas, a fim de garantir a idempotência das operações.

Funcionalidades:

- Movimentar Conta: Permite realizar uma movimentação (crédito ou débito) em uma conta corrente. A operação é idempotente, ou seja, se a mesma requisição for enviada novamente, a resposta será a mesma.
- Consultar Saldo: Permite consultar o saldo de uma conta corrente, retornando o número da conta, nome do titular, data/hora da resposta e o saldo atual.

Validações:

- O sistema implementa diversas validações, tanto para as contas correntes quanto para as movimentações, garantindo a integridade dos dados. Exemplos: verifica se a conta corrente existe e está ativa, se o valor da movimentação é válido, se o tipo de movimento é válido, entre outras validações.

Tratamento de Exceções:
- O sistema utiliza um `ExceptionFilter` para tratar exceções de forma centralizada, mapeando cada tipo de exceção para uma resposta HTTP adequada (Bad Request) com uma mensagem de erro correspondente.

Injeção de Dependências:
- O sistema utiliza a injeção de dependências para resolver as dependências entre os componentes, facilitando a manutenção e testabilidade do código.

Banco de Dados:
- O sistema utiliza um banco de dados SQLite para armazenar os dados das contas correntes, movimentações e informações de idempotência.
- O contexto do Entity Framework Core é configurado na classe **`SQLiteContext`**, onde também são aplicados os mapeamentos das entidades para as tabelas do banco de dados.

Configuração e Inicialização:
- O arquivo `program.cs` é responsável por configurar e inicializar o aplicativo, registrando os serviços necessários, configurando o contexto do banco de dados SQLite e inicializando o banco de dados (se necessário).