Levantamento de Requisitos para API de Gestão de Despesas com Compartilhamento de Despesas

Visão Geral do Sistema

A API de Gestão de Despesas permite o registro, a edição, a exclusão e o compartilhamento de despesas entre os usuários. O sistema também realiza o cálculo automático das porcentagens de divisão de valores entre os participantes, possibilitando um controle eficiente de finanças compartilhadas. A autenticação será feita por meio de token JWT, garantindo a segurança das operações. A plataforma será construída utilizando .NET Core 8, SQL Server, Redis, MongoDB, Docker, com práticas de testes automatizados e integração contínua.

Requisitos Funcionais

1. Autenticação e Autorização

1.1. A API deve exigir autenticação por **token JWT** para garantir que apenas usuários autorizados possam acessar os endpoints.

1.2. O token JWT será gerado após login com as credenciais do usuário e terá um tempo de expiração configurável.

1.3. O sistema deve fornecer a possibilidade de **renovação de token** sem a necessidade de nova autenticação, com base em uma política de refresh token.

1.4. O token JWT será transmitido em todas as requisições no cabeçalho HTTP como `Authorization: Bearer <token>`.

1.5. A API deve rejeitar requisições não autenticadas com **erro HTTP 401 Unauthorized**.

1.6. O sistema deve garantir que **apenas usuários com permissões adequadas** possam realizar operações específicas, como criação, edição e remoção de despesas.

1.7. A autenticação deverá ser realizada através de credenciais do usuário (usuário e senha), podendo haver integração com **provedores de autenticação externos** (OAuth2, OpenID, etc).

1.8. O sistema deve permitir a **revogação de tokens** quando o usuário fizer logout ou em situações de risco de segurança.

2. Gestão de Despesas

2.1. Criação e Edição de Despesas

2.1.1. O sistema deve permitir a **criação de despesas**, onde o usuário pode registrar os seguintes dados:

- Descrição da despesa (`descricao`)
- Valor total (`valor`)
- Categoria da despesa (`categoria`)
- Data de lançamento (`data`)
- Lista de usuários participantes (se for compartilhada)

2.1.2. O sistema deve permitir que as despesas sejam **editadas** por usuários com permissões adequadas, atualizando qualquer dado, incluindo a lista de usuários ou a divisão de valores.
2.1.3. O sistema deve validar se a **soma das porcentagens** de divisão de uma despesa compartilhada é **sempre 100%**, antes de permitir a edição.
2.1.4. Durante a edição de uma despesa compartilhada, o sistema deve **recalcular automaticamente** os valores devidos por cada usuário, conforme as novas porcentagens definidas.

2.2. Remoção de Despesas

2.2.1. A API deve permitir a **remoção de despesas** através do identificador único da despesa (`id`).

2.2.2. Caso a despesa seja compartilhada, o sistema deve garantir que a **removação dos usuários** seja consistente, ajustando as divisões de valor ou removendo a despesa completamente.

2.3. Compartilhamento de Despesas

2.3.1. O sistema deve permitir o **compartilhamento de despesas** entre dois ou mais usuários, com o cálculo proporcional das divisões.

2.3.2. O valor de uma despesa compartilhada será **dividido proporcionalmente** entre os usuários, de acordo com as porcentagens definidas por cada um.

2.3.3. O sistema deve permitir a **edição da porcentagem de divisão** de cada usuário, recalculando automaticamente os valores devidos.

2.3.4. O sistema deve garantir que a soma das porcentagens seja **sempre 100%** ou equivalente (1, se for representado como decimal).

2.3.5. O sistema deve permitir que um usuário seja **removido do compartilhamento**, redistribuindo as porcentagens entre os outros participantes.

2.3.6. O sistema deve garantir que a **divisão de despesas** seja recalculada automaticamente sempre que houver mudanças no número de usuários ou nas porcentagens de cada um.

2.4. Consulta de Despesas

2.4.1. A API deve permitir a **consulta de despesas por `id`**, retornando todos os dados relacionados à despesa, incluindo participantes e valores devidos.

2.4.2. O sistema deve permitir a **busca por descrição**, com suporte para correspondência exata ou parcial (filtro textual).

2.4.3. A API deve permitir a **busca por categoria**, como alimentação, transporte, etc.

2.4.4. O sistema deve permitir a **consulta por intervalo de datas** (ex: de `data_inicial` a `data_final`).

2.4.5. A consulta de despesas deve permitir a **paginação** de resultados em caso de grandes volumes de dados, para evitar sobrecarga no sistema.

2.4.6. A API deve permitir a **filtragem por valor** (ex: despesas com valores acima ou abaixo de um valor especificado).

2.5. Exportação e Importação de Dados

2.5.1. O sistema deve permitir a **exportação de despesas** para formatos **CSV** e **Excel**, incluindo todos os dados relevantes (descricao, valor, categoria, data, participantes).

2.5.2. A exportação de dados deve ser feita de forma otimizada, considerando grandes volumes de informações.

2.5.3. O sistema deve permitir a **importação de despesas** a partir de arquivos **CSV** ou **Excel**, validando os dados antes de inseri-los no sistema.

2.5.4. O sistema deve fornecer feedback sobre a **sucesso ou falha da importação**, destacando problemas de formatação ou inconsistências nos dados.

2.6. Histórico de Alterações

2.6.1. O sistema deve manter um **registro histórico completo** de todas as alterações realizadas em uma despesa, incluindo:

- Data e hora da alteração
- Usuário responsável pela alteração
- Campos alterados e seus valores antigos e novos.

2.6.2. O histórico de alterações deve ser **auditável** e acessível apenas por usuários autorizados (ex: administradores).

3. Relatórios e Dashboards

3.1. Geração de Relatórios

3.1.1. O sistema deve permitir a **geração de relatórios financeiros** que agreguem as despesas por:

- Categoria
- Período (ex: mês, trimestre, ano)
- Usuário (total de despesas por cada participante)

3.1.2. O sistema deve permitir a **exportação de relatórios** em formatos **Excel** ou **PDF** para análises financeiras.

3.1.3. Os relatórios financeiros devem ser **dinâmicos**, permitindo que o usuário selecione os filtros de período, categoria e outros parâmetros antes de gerar o relatório.

3.2. Dashboards Interativos

3.2.1. O sistema deve fornecer **dashboards dinâmicos**, com gráficos e tabelas, mostrando visualmente as despesas por:

- Categoria
- Período
- Usuário

3.2.2. O sistema deve permitir que o usuário **interaja com os dashboards**, aplicando filtros de data e categoria para visualizar dados específicos.

Requisitos Não Funcionais

4. Segurança

4.1. A API deve utilizar **HTTPS** para todas as comunicações, garantindo a segurança dos dados em trânsito.

4.2. O sistema deve proteger as **senhas dos usuários** usando **hashing seguro** (ex: bcrypt).

4.3. O sistema deve implementar **limitação de taxa (rate-limiting)** para proteger contra abusos e ataques de DDoS.

4.4. O sistema deve registrar **tentativas de acesso inválidas** e aplicar **bloqueio temporário** após um número configurável de falhas consecutivas.

4.5. O sistema deve garantir que **nenhum dado sensível** (como senhas ou informações financeiras) seja armazenado de forma insegura.

4.6. A API deve garantir **proteção contra ataques de injeção** (SQL Injection, XSS, CSRF).

5. Performance e Escalabilidade

5.1. A API deve ser projetada para **suportar alto tráfego de usuários**, sendo capaz de escalar horizontalmente conforme a demanda (via Kubernetes ou containers Docker).

5.2. O sistema deve ser capaz de processar grandes volumes de dados sem degradação significativa de performance.

5.3. O uso do **Redis** como cache deve garantir uma redução significativa no tempo de resposta para consultas frequentes (como a busca de despesas por categoria).

5.4. O sistema deve ter **respostas rápidas** (com tempo de resposta inferior a 1 segundo para as consultas mais comuns).

5.5. O banco de dados SQL Server deve ser otimizado para consultas complexas, utilizando índices adequados.

6. Manutenibilidade

6.1. O sistema deve ser projetado com uma **arquitetura modular**, facilitando a manutenção e a adição de novas funcionalidades.

6.2. O código da API deve ser bem documentado, utilizando comentários claros e descritivos.

6.3. A aplicação deve ter **testes unitários e de integração** cobertos por ferramentas de CI/CD.

6.4. O sistema deve suportar a **atualização sem interrupções** (via deploy contínuo).

7. Confiabilidade e Disponibilidade

7.1. O sistema deve ter uma **disponibilidade mínima de 99,9%** em operação, garantindo resiliência e tempo de inatividade mínimo.

7.2. O sistema deve ser projetado para **tolerância a falhas**, com backups automáticos dos dados críticos, para recuperação em caso de falha.