Para começar a implementar este software de gestão financeira, você pode seguir uma abordagem incremental e modular, dividindo o desenvolvimento em etapas. Isso ajudará a construir as partes fundamentais primeiro e, em seguida, expandir o sistema com funcionalidades mais complexas. Abaixo estão os passos iniciais sugeridos:

**1. Planejamento e Configuração do Ambiente**

* **Escolha da IDE**: Escolha uma IDE como Visual Studio, CLion, ou VS Code para facilitar o desenvolvimento em C++.
* **Instalação das Bibliotecas Necessárias**:
  + **SQLite**: Instale a biblioteca SQLite para C++ ou o MySQL Connector, se optar por MySQL.
  + **libcurl**: Para fazer requisições HTTP, baixe e configure libcurl na sua IDE.
  + **nlohmann/json**: Baixe o cabeçalho json.hpp do repositório GitHub oficial da biblioteca.
  + **SFML** ou **matplotlib-cpp**: Se planeja adicionar gráficos mais tarde, baixe e configure uma dessas bibliotecas.
  + **OpenSSL** (opcional): Se quiser adicionar criptografia, baixe e configure o OpenSSL.

**2. Configuração do Banco de Dados**

Comece projetando o banco de dados, pois ele será a base para o armazenamento dos dados financeiros.

* **Crie as Tabelas**:
  + Tabela usuarios: Para armazenar dados dos usuários (ex.: ID, nome, senha hash).
  + Tabela receitas e despesas: Para guardar entradas e saídas financeiras.
  + Tabela metas\_financeiras: Para armazenar as metas financeiras dos usuários.
  + Tabela investimentos: Para registrar os ativos e o valor atual de cada um.
* **Teste a Conexão ao Banco de Dados**: Escreva uma função simples para testar a conexão com o banco de dados, como criar uma tabela e inserir registros básicos.

**3. Implementação das Classes Básicas**

Crie as classes principais do sistema, com foco nas funcionalidades básicas de manipulação de dados.

* **Classe Usuario**: Inclua dados pessoais e autenticação. Implemente funções para criar, deletar e autenticar usuários.
* **Classe Transacao**: Crie uma classe base para receitas e despesas, com atributos como id, valor, data e categoria.
* **Classe MetaFinanceira**: Para definir metas com atributos como valorObjetivo, dataLimite e status.
* **Classe Investimento**: Para registrar ativos, com atributos como id, tipo, quantidade e valorAtual.

**4. Implementação de Operações no Banco de Dados**

Para cada classe, implemente funções para:

* Inserir, atualizar e deletar registros.
* Consultar dados específicos (ex.: somar receitas, listar despesas de um período).
* Atualizar os valores de investimentos a partir das APIs (funcionalidade a ser implementada na próxima etapa).

**5. Integração com APIs Financeiras**

Após configurar as operações básicas no banco de dados, adicione a funcionalidade de atualização automática dos preços dos ativos.

* **Escolha uma API e Crie uma Conta**: Escolha uma API de dados financeiros, como Alpha Vantage ou CoinGecko. Gere uma chave de API, se necessário.
* **Teste a Requisição HTTP**: Implemente uma função em C++ para realizar uma requisição GET à API, usando libcurl.
* **Parse de Dados JSON**: Utilize nlohmann/json para processar a resposta JSON e obter os valores dos investimentos.
* **Atualização do Banco de Dados**: Crie uma função para atualizar os preços dos investimentos no banco de dados com os valores retornados pela API.

**6. Implementação de Relatórios e Gráficos (Opcional)**

Quando a estrutura básica e a funcionalidade de atualização de investimentos estiverem funcionando, você pode adicionar relatórios para análise financeira:

* **Geração de Relatórios**: Implemente funções para gerar relatórios de despesas, receitas e metas.
* **Gráficos**: Usando SFML ou matplotlib-cpp, gere gráficos simples como gráficos de barras para visualizar despesas por categoria e gráficos de linha para o desempenho dos investimentos.

**7. Criptografia e Segurança (Opcional)**

Implemente a segurança para proteger dados sensíveis:

* **Hashing de Senhas**: Utilize OpenSSL para aplicar hash nas senhas dos usuários antes de armazená-las.
* **Armazenamento Seguro de Chaves de API**: Guarde a chave de API em um arquivo de configuração separado e encriptado.

**8. Testes e Depuração**

Realize testes unitários e de integração para verificar se as funcionalidades estão funcionando corretamente:

* Teste as operações CRUD no banco de dados para cada classe.
* Verifique a integração com a API, assegurando-se de que os preços de ativos são atualizados corretamente.
* Teste os relatórios e gráficos para assegurar a precisão dos dados exibidos.

**9. Documentação e Expansão**

Conforme o sistema fica funcional:

* Documente o código e as funcionalidades.
* Avalie novas funcionalidades, como um sistema de notificações para metas financeiras ou alertas de preços para investimentos.

Esses passos te darão uma estrutura sólida para implementar o sistema de gestão financeira em C++.