Acesse o link com todo o projeto em nuvem

http://34.70.125.41/ (http://34.70.125.41/)

Integrantes:

- Hugo Ferreira Marques
- Luiz Henrique Diniz Ferreira

Introdução

O início do trabalho foi pensado em garantir também os pontos extras e, para isso, precisou-se da escolha de uma web framework C++. Foram pensados em dois deles, o CWF (C++ Web Framework) e o Crow Web framework, sendo o último escolhido por parecer ser mais simples. O banco de dados escolhido foi o MySQL. Para unir todos esses programas e fazer de uma forma que fosse possível uma rápida edição entre os integrantes do grupo, foi usado o docker-compose em união com o GitHub; assim, criava-se um container Docker com as seguintes aplicações:

- Theia (IDE online)
- Crow (WebFramework c++ com todas suas bibliotecas integradas, principalmente a mysqlpp, uma API mysql para c++)
- Nginx (responsável pelo frontend e o proxy reverso (para comunicação com o crow))
- MySQL (Para o banco de dados)
- PHPMyAdmin (Para gerenciamento do banco de dados)

Desenvolvimento

Deploy da aplicação em nuvem

Para melhorar o deploy da aplicação foi utilizado um servidor do google cloud com uma instância ubuntu e containers docker. Ao iniciar a instalação e integração desses containers, enfrentou-se grandes dificuldades para o aprendizado do framework e para a integração do docker, o que tomou muito tempo dos integrantes. Foi criado um yaml file que facilita o deploy da aplicação, automatizando a implantação de contêineres de ide web, banco de dados, proxy reverso e backend.

GitHub

Como o desenvolvimento envolveu múltiplos integrantes optou-se por utilizar um sistema de controle de versão em nuvem, o git. Cada integrante desenvolvia uma parte do código e fazia push para um repositório remoto e depois entrávamos por ssh no servidor remoto e realizamos o pull da aplicação.
https://github.com/HugoFM2/sistemaBancarioWeb (https://github.com/HugoFM2/sistemaBancarioWeb)

Interface

Para a interface Web, optou-se utilizar a “game engine” Construct 3, uma vez que ela atendia todos os requisitos para o trabalho. Como o foco do trabalho em si não era a programação do frontend, foi utilizada essa ferramenta justamente por não requerer muito aprendizado e ser simples.

C++

Como foi utilizada uma interface Web, alguns erros foram tratados no backend e outros no frontend validando user inputs no navegador web, o frontend válida várias requests uma vez que ela somente permitia enviar dados e modificá-los com eles já sendo tratados. Como esse trabalho foi uma extensão do TP1, utilizou-se todas as classes criadas e foram implementadas novas classes com o intuito apenas de acrescentar/modificar classes já existentes. Uma das funções a ser adicionada, o do dia base da poupança não foi implementada pois não ficou claro segundo o roteiro como esta feature seria detalhadamente implementada. O uso de herança e polimorfismo foi essencial para essa parte e as classes que utilizaram esses mecanismos foram as classes:

ContaTP3(Conta)

Invés de criar 2 classes diferentes para Conta Corrente ou poupança acreditou-se ser melhor separa-las apenas através de um atributo protegido que definia 1 para conta corrente e 13 para poupança. Foi incluído também um atributo protegido que definia o `limiteCredito` ,assim quando a conta era do tipo Conta Corrente, existia um limite que podia ser definido pelo Usuário, caso fosse do tipo Poupança, esse limite era 0(ou seja, não havia limite). Assim, foi preciso usar polimorfismo apenas na função `DebitarValor` , uma vez que a mesma agora precisava do `limiteCredito` para seu funcionamento e com isso, o resto das funções necessárias foram herdadas da classe Conta.

ClienteTP3(Cliente)

A Classe cliente precisava apenas de um ID para o cliente, assim foi feita a herança de todas as funções da classe `Cliente` e no construtor da classe `ClienteTP3` foi adicionado esse recurso, adicionando também a função `getClienteID` para retornar qual era o ID do cliente.

BancoTP3(Banco)

Nessa Classe precisou-se principalmente modificar os vetores do tipo `Cliente` e `Conta` para suas versões modificadas `ClienteTP3` e `ContaTP3` respectivamente e modificar também todos os usos de funções que usava classes antigas e adapta-las para as novas Classes. Foram criadas também novas funções para verificar se a conta existe e se a conta pertence a certo cliente, o que facilitou para a remoção dos mesmos.

Integração Crow

Essa foi a parte mais importante do trabalho, ela fazia a integração frontend com backend, utilizando o WebFramework Crow e ainda tratava os erros (junto com o construct 3), caso os dados fossem inválidos ou diferentes. Como o construct 3 realizava o trabalho das requisições HTTP GET, não era necessário que o cliente digitasse os comandos no Browser, os exemplos a seguir mostram os endereços web só para tornar o exemplo mais claro. Com isso o sistema tem varias vulnerabilidades, porém não era o foco do trabalho tratar a segurança do sistema. Um exemplo de uma função que ao digitar no Browser `localhost/api/listarClientes` , retornava em `JSON` (o que tornava mais fácil o tratamento de

In [ ]:

```
CROW_ROUTE(app, "/listarContasCliente")
  ([&WebTeste](const crow::request& req){
    crow::json::wvalue x;
    if(req.url_params.get("idBanco") != nullptr) {
      std::cout << " Entrou listarClientes com algum id ";
      int idBanco = std::stoi(std::string(req.url_params.get("idBanco")));
      if(req.url_params.get("idCliente") != nullptr) {
        int idCliente = std::stoi(std::string(req.url_params.get("idCliente")));
        for(unsigned int i = 0; i < WebTeste.getBanco(idBanco)->getContas().size
(); i++){
          if(WebTeste.getBanco(idBanco)->getContas()[i]->getCliente()->getClient
eID() == idCliente){
            x["numConta"][i] = WebTeste.getBanco(idBanco)->getContas()[i]->getNu
mConta();
            x["tipoConta"][i] = WebTeste.getBanco(idBanco)->getContas()[i]->getT
ipoConta();
          }

      }
    }
  }
      return x;
  });
```

Outro exemplo que já tratava alguns erros é o exemplo de `RemoverCliente` . Para a remoção do cliente ele precisaria dos parâmetro `idCliente` , caso faltasse o parâmetro um erro era retornado na página. Como o cliente é selecionado através da interface, não foi necessário tratar o erro de colocar o `idCliente` errado. Essa função então verificava se existia alguma conta que pertencia a esse cliente, caso positivo, ele retornava "Não foi possivel remover o cliente, existem contas dependentes", caso negativo, ele deletava a conta e retornava "Cliente Deletado".

In [ ]:

```
CROW_ROUTE(app, "/removerCliente")
  ([&WebTeste](const crow::request& req){
        if(req.url_params.get("idCliente") != nullptr) {
          int idCliente = std::stoi(std::string(req.url_params.get("idCliente"
)));
          if(WebTeste.getBanco(0)->ExisteContaCliente(idCliente)){
              return "Não é possivel remover o cliente, existem contas dependent
es";
          } else {
            WebTeste.getBanco(0)->RemoverCliente(idCliente);
            return "Cliente Deletado";
          }
        }
        return "Faltou ID Cliente";
  });
```

Arquitetura do sistema

O sistema como um todo funciona da seguinte maneira, ao acessar o ip do servidor, o container com nginx verifica se a request deve ir para o backend ou se deve ser servidor um conteúdo estático, caso seja requisitado conteúdo estático, o nginx se encarrega de buscar os arquivos e entregar para o cliente, caso seja uma request de api, ela é repassada para o backend em c++ que se encarrega de acessar o banco de dados, tratar os resultados e enviar uma resposta para o frontend. No mesmo servidor docker, temos também uma ide web chamada theia por onde é possível editar o código da aplicação e modificar os arquivos estáticos. Além disso temos uma instância do phpmyadmin que é um container responsável por fazer acesso ao banco de dados mysql e administrar os bancos e tabelas presentes, facilitando o desenvolvimento da aplicação.

Vulnerabilidades de segurança previstas

Como se trata de um sistema bancário, para implantação em ambiente de produção seria necessário inspecionar algumas vulnerabilidades:

Página acessada por HTTP

Para que não houvesse necessidade de gerar um certificado RSA foi utilizado o protocolo HTTP para comunicação entre frontend e backend, isto é uma vulnerabilidade pois o ideal seria utilizar https que impede ataques man in the middle criptografando a comunicação. Isto permite que um invasor de uma rede possa modificar e visualizar o conteúdo trafegado entre o cliente web e o servidor.

Utilização de Requests Get para comunicação com o Backend

As requests Get que comunicam backend levam os parâmetros em formato raw data para o servidor, permitindo que eles sejam interceptados, o ideal seria utilizar requests POST criptografadas para aumentar a segurança do sistema.

Validação de user Inputs no Frontend

Algumas entradas de usuários são validadas no Frontend, por exemplo valor da transação, isso permite que uma request seja criada e force o backend a executar uma transação não prevista para um usuário comum do sistema

Não verificação de strings para querys sql no Backend

Isso permite que mesmo um sistema que tratasse todas as vulnerabilidades anteriores sofresse um tipo de ataque chamado injection em que um invasor poderia mandar uma string que ao invés de uma entrada de usuário fosse um código injetado executável pelo Backend

Conclusão

O desenvolvimento envolveu grandes desafios pois o grande foco do grupo foi desenvolver uma aplicação completamente voltada para um cenário de utilização real. Utilizando tecnologias consolidadas por grandes empresas do mercado desde o início do desenvolvimento até sua implantação Apesar das vulnerabilidades previstas, concluímos que o sistema pode inclusive um dia se tornar uma solução comercialmente viável desde que se empenhe tempo e esforço suficiente. Concluímos que foram aplicados e validados todos os conceitos aprendidos em sala e que este trabalho final cumpre o objetivo de demonstrar a união entre

linguagens de programação consolidadas (exemplo c++, mysql) e tecnologias recentes e inovadoras (exemplo docker, theia), de modo que os participantes conseguiram realizar a oportunidade de desenvolver um amplo conhecimento em uma enorme gama em se tratando de projetar um sistema e efetuar a sua implantação final.

Referências

1. Mysqlpp: https://github.com/rpetrich/mysqlpp (https://github.com/rpetrich/mysqlpp)
2. Crow: https://github.com/ipkn/crow (https://github.com/ipkn/crow)
3. Docker: https://www.docker.com/ (https://www.docker.com/)
4. MySQL: https://www.mysql.com/ (https://www.mysql.com/)
5. Theia: https://theia-ide.org/ (https://theia-ide.org/)
6. PHPMyAdmin: https://www.phpmyadmin.net/ (https://www.phpmyadmin.net/)