Relatório da Simulação de Sprint Scrum

Engenharia de Software

Gerardo Mikael do Carmo Pereira
Thiago Franke Melchiors
Yonathan Rabinovici Gherman

Professor: Rafael de Pinho

Monitor: Luís Henrique Domingues Bueno

RIO DE JANEIRO
2024

Conteúdo

1 Introdução

Neste relatório, abordaremos as decisões tomadas pelo nosso grupo em relação à arquitetura do site e à aplicação das práticas de Extreme Programming (XP) no contexto da disciplina de Engenharia de Software. O trabalho foi realizado como parte de uma simulação de sprint Scrum, na qual priorizamos características arquitetônicas essenciais para o sistema e buscamos adotar as práticas recomendadas de XP, equilibrando essas escolhas com as demandas do projeto e as peculiaridades da equipe.

2 Características de Arquitetura

No dia 9 de outubro, o grupo se reuniu para discutir e definir as características de arquitetura que seriam priorizadas no desenvolvimento do nosso projeto. Durante a reunião, tomamos como base as recomendações e requisitos fornecidos pelo stakeholder do projeto, o senhor Clebson Ronaldo da Costa, além de considerarmos nosso entendimento atual sobre as demandas futuras da aplicação. O objetivo foi garantir que o sistema estivesse alinhado com as expectativas e necessidades expressas por ele, priorizando os aspectos mais críticos para o sucesso da solução.

Após análise, decidimos focar em três principais características de arquitetura: **Segurança (Security)**, **Facilidade de Aprendizado (Learnability)** e **Estabilidade (Stability)**. A escolha dessas três características reflete tanto as preocupações apontadas pelo stakeholder quanto a importância dessas dimensões para a experiência dos usuários e o sucesso do aplicativo. A seguir, daremos justificativas mais detalhadas para cada atributo.

A `segurança` foi a primeira característica identificada como prioritária. Uma vez que o sistema irá lidar com dados sensíveis dos usuários, incluindo informações pessoais e, possivelmente no futuro, registros financeiros, a *proteção contra vazamento de dados* se torna um aspecto essencial. Com a crescente preocupação global em torno de privacidade e segurança digital, garantir que os dados dos usuários sejam armazenados e transmitidos de forma segura é crucial para construir confiança e credibilidade. Além disso, o nosso sistema precisa estar preparado para proteger-se contra ataques cibernéticos, acessos não autorizados e outras ameaças à segurança digital. Esta característica foi enfatizada pelo stakeholder, que destacou a necessidade de um sistema de segurança sólido como um fator decisivo para a aceitação do aplicativo, principalmente em um ambiente onde usuários estarão realizando transações e expondo dados sensíveis.

No entanto, até o momento, ainda não conseguimos nos aprofundar tanto nos aspectos avançados de segurança, pois priorizamos a entrega de um MVP funcional. Ainda assim, implementamos o hashing de senhas usando a biblioteca `bcrypt`, o que já oferece um nível básico de proteção. Com o uso de `bcrypt`, as senhas dos usuários são transformadas em um hash, o que significa que mesmo em casos de vazamento de dados, as senhas não são armazenadas em texto puro, garantindo assim que estejam menos vulneráveis a ataques diretos.

A qualidade de `fácil de aprender` se insere no contexto de que o aplicativo deve ser intuitivo o suficiente para que novos usuários possam utilizá-lo rapidamente, sem a necessidade de um manual extenso ou suporte constante. Isso se aplica tanto aos com-

pradores que utilizam as vending machines quanto aos vendedores que gerenciam seus produtos no sistema. A curva de aprendizado curta é essencial para garantir a adesão e permanência dos usuários no aplicativo. A experiência de compra e de gestão precisa ser rápida e fluida, de forma que o usuário não se sinta frustrado, entendiado ou confuso, o que impactaria negativamente a adoção do sistema. A usabilidade simples e direta é o segredo para o sucesso do negócio.

Nesse sentido, na implementação da interface de gerenciamento de produtos e máquinas de venda, buscamos garantir que as ações fossem claras e intuitivas. Por exemplo, ao visualizar uma máquina de venda, o sistema exibe uma lista simplificada de produtos disponíveis em formato de tabela, permitindo que o vendedor veja de forma clara e organizada as opções de produtos, preços e quantidades. O uso de inputs claros para adicionar, remover ou atualizar produtos foi projetado para ser direto e com mensagens de feedback adequadas, ajudando o usuário a entender o que está fazendo e, assim, minimizando a necessidade de um manual.

No contexto de um aplicativo que lida com transações e disponibiliza produtos em vending machines, é imperativo que o sistema tenha `estabilidade`: funcione de forma confiável e contínua. Falhas no sistema durante uma compra podem gerar insatisfação imediata dos usuários, além de causar perdas financeiras para os vendedores e frustração para os compradores, tornando-o indesejável para futuros usos. Garantir a estabilidade significa que o aplicativo estará preparado para suportar picos de uso, funcionar corretamente sem interrupções e apresentar o mínimo possível de erros ou falhas durante o uso. A confiabilidade do sistema é um fator crítico para garantir que os usuários sintam-se à vontade e seguros ao utilizá-lo repetidamente.

Realizamos testes com o framework `unittest` seguindo a filosofia de Test-Driven Development (TDD) da metodologia XP, garantindo que as funcionalidades fossem validadas desde o início e cobrissem boa parte do sistema. Implementamos transações atômicas no banco de dados para assegurar que operações de inserção e atualização fossem realizadas de maneira completa ou revertidas em caso de falha. No futuro, pretendemos implementar testes automatizados extensivos (Extensive Automated Testing, EAT) para aumentar a cobertura, detectar erros precocemente, garantir consistência e facilitar a evolução contínua do sistema sem comprometer funcionalidades já existentes.

Apesar de outras características também serem relevantes, elas foram classificadas como de menor prioridade em relação às três principais. Nesse caso, podemos citar a `escalabilidade`. Reconhecemos que a capacidade do sistema de lidar com um número considerável de transações simultâneas é importante, especialmente para um projeto que pode, no futuro, ser expandido para além do ambiente atual. No entanto, como o escopo inicial do aplicativo será limitado às instalações da FGV, o volume de transações e acessos simultâneos será relativamente baixo. Entendemos que o número de alunos e funcionários que estarão utilizando o sistema não justifica, neste primeiro momento, a dedicação de um grande esforço técnico para garantir alta escalabilidade.

Outra característica que foi considerada, mas não priorizada, é a `manutenibilidade`. Embora seja essencial que o sistema seja fácil de manter e atualizar, acreditamos que, quando o aplicativo estiver finalizado, em operação em um ambiente controlado e com um escopo restrito, a necessidade de frequentes atualizações e manutenções será

relativamente baixa no curto prazo. Ainda assim, sabemos que o atributo em questão é importante para garantir a longevidade e a evolução do projeto. Por isso, projetaremos o sistema de forma modular e com o código bem comentado para facilitar futuras melhorias e correções.

3 Extreme Programming

Durante o desenvolvimento do sistema, procuramos seguir os princípios de Extreme Programming (XP) tanto quanto possível, levando em consideração as limitações práticas enfrentadas pela equipe. No que tange às práticas adotadas, seguimos principalmente as abordagens de **Test-Driven Development (TDD), Integração Contínua, Small Releases e Refatoração**.

Adotamos o `TDD` em boa parte do projeto: antes de escrevermos o código para as funcionalidades, começamos definindo os testes que validariam os requisitos. Essa abordagem garantiu que o código estivesse sempre alinhado com os requisitos funcionais desde o início, além de proporcionar maior confiabilidade. Os testes que desenvolvemos ajudaram a evitar regressões, permitindo uma melhor integração entre as partes do sistema. Embora o TDD tenha sido aplicado de maneira significativa, observamos que em alguns momentos, devido à pressão do tempo, acabamos priorizando a implementação rápida de funcionalidades e voltamos a criar testes posteriormente para checar a corretude do código.

Em termos de `integração contínua`, utilizamos o *GitHub* como ferramenta de controle de versão, com os membros da equipe criando branches individuais para desenvolver suas funcionalidades. Realizamos commits frequentes e utilizamos pull requests para mesclar o código na branch principal. Isso possibilitou que o código fosse integrado e testado regularmente, evitando conflitos de fusão de código entre diferentes funcionalidades desenvolvidas em paralelo e facilitando a colaboração entre os componentes do grupo

No que se refere à prática de `small releases`, buscamos fazer entregas pequenas e frequentes sempre que possível. No entanto, olhando em retrospecto, identificamos que poderíamos ter aplicado essa prática de forma ainda mais eficiente. Em alguns momentos, poderíamos ter feito commits menores e mais frequentes, mas a metodologia de trabalho era nova para a equipe e, muitas vezes, acabamos pensando que poderíamos resolver "só mais um problema"antes de realizar o commit. Essa abordagem, apesar de prática em algumas ocasiões, não refletiu totalmente o princípio de small releases, e é algo que podemos melhorar em sprint scrum futuras.

Também aplicamos a `refatoração contínua`: durante o desenvolvimento, garantimos que o código fosse refatorado regularmente para manter a clareza e a manutenibilidade. Ajustes e melhorias foram feitos à medida que novas funcionalidades eram desenvolvidas, garantindo que o código seguisse boas práticas de programação e permanecesse eficiente e legível. Isso também nos ajudou a remover redundânciass de código e melhorar a estrutura geral do sistema, garantindo que ele fosse sustentável a longo prazo.

Por outro lado, uma prática importante que não conseguimos adotar foi o `pair programming`. A principal razão para isso foi a incompatibilidade de horários entre os

integrantes da equipe. Cada membro tinha uma rotina diferente e, devido aos compromissos diversos, não conseguimos organizar momentos em que pudéssemos trabalhar simultaneamente em uma mesma funcionalidade. Contudo, apesar de não termos utilizado essa metodologia de trabalho, buscamos garantir a qualidade do código através da prática de revisão de código assíncrona (code review), garantindo um nível aceitável de qualidade.

Adicionalmente, enfrentamos um imprevisto significativo logo no início do projeto, quando um dos membros da equipe teve um problema inesperado e precisou se afastar das tarefas. Isso alterou drasticamente a dinâmica do grupo, gerando a necessidade de reorganização e adaptação por parte dos dois membros restantes. Embora esse contratempo tenha impactado o ritmo inicial, com o tempo, conseguimos nos ajustar e estabelecer um ritmo de trabalho eficiente. Adotamos uma dinâmica de desenvolvimento em branches separadas, seguida de revisões de código mútuas, o que garantiu a integração adequada das funcionalidades e manteve a qualidade do código. Esse processo, embora diferente do pair programming, permitiu que a equipe/dupla se mantivesse produtiva e colaborativa, mesmo diante das adversidades.