UNIVERSIDADE FEDERAL DO CEARÁ

Documento de processos, ferramentas e tecnologias

Este documento visa detalhar os processos planejados e utilizados durante o desenvolvimento do sistema de gerência de eventos culturais da Casa de Saberes Cego Aderaldo, localizada em Quixadá, desenvolvido para a disciplina de Projeto Integrado em Engenharia de Software I. Além disso, este documento especifica quais ferramentas e tecnologias foram usadas ao longo do projeto, com seus prós e contras.

Equipe

- Francisco Paulino - 538451 - paulinofilho@alu.ufc.br
- Gustavo Henrique - 535735 - gustavohenriquefs.dev@gmail.com
- João Pedro - 539012 - joaopedroph@alu.ufc.br
- Robson Diógenes - 521437 - robsonad07@alu.ufc.br
- Jhordanna Oliveira - 536646 - jhordanna@alu.ufc.br

Processos

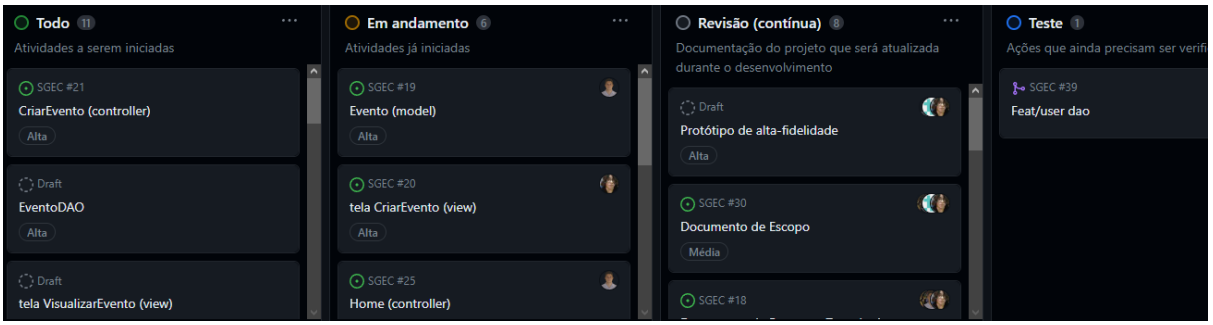
O processo de desenvolvimento da equipe é organizado usando uma Ideologia Kanban, que tem como característica apresentar um fluxo de atividades que devem ser realizadas. Inicialmente a equipe organizava essas tarefas pela ferramenta Trello, mas depois passamos a utilizar o Github Projects em conjunto com uma planilha para o backlog.

A divisão das tarefas é feita considerando a aptidão de cada membro da equipe e levando em conta as camadas da aplicação, dessa forma a equipe assegura que model, view, control e DAO serão desenvolvidos independentemente.

O quadro no Github projects conta com 6 grupos de atividades. “TODO" que são as tarefas ainda não iniciadas, e com um nível de especificação maior que na planilha do backlog, “Em andamento” contém tarefas já iniciadas por um membro da equipe. "Revisão contínua" inclui toda a documentação que deve ser atualizada ao longo do projeto. "Teste" contém tarefas concluídas, mas que ainda precisam ser verificadas. "Concluído" incluí as tarefas finalizadas, vale salientar que uma tarefa concluída pode voltar para a fase de testes caso ocorram erros não verificados. Por fim, “Tweaks” inclui atividades que agregam valor, seja facilitando o processo de desenvolvimento, seja melhorando a usabilidade para o usuário, mas não estão incluídas no escopo do projeto.

Além disso, também foi utilizada a integração contínua, com workflows do github actions. Onde o projeto sempre tinha sua build realizada a cada feature acrescentada, garantindo que a versão do programa a ser entregue sempre esteja sincronizada com a última versão desenvolvida.

Abaixo está um recorte do quadro kanban usado ao decorrer do processo:



Processo para adicionar uma Feature ao sistema:

Para o acréscimo de novas features ao sistema, nós optamos por um fluxo em que cada feature é desenvolvida em uma branch separada, e após a integração com nossa branch principal (development) deletada. As motivações para essa escolha são:

- Maior modularidade, visualizando as branches ativas sabemos exatamente qual parte independente do sistema está feita.
- Menos conflitos, ao implementar somente uma feature por branch, a possibilidade de um arquivo possuir conflitos diminui bastante, até essa etapa do desenvolvimento ocorreram três conflitos principais, dois deles no arquivo module-info.java e outro em LoginController que foram solucionados rapidamente.
- Proteção da branch development, evitando alterações não planejadas

Resumidamente, a sequência de ações realizadas ao longo da implementação de uma feature são:

1. acessar a branch development do repositório (git checkout development)
2. criar nova branch para feature escolhida (git checkout -b <nome_da_feature>)
3. implementar o código necessário
4. subir as mudanças em forma de pull request no repositório (git push origin <nome_da_feature>)

Processo para adicionar um teste ao sistema

O fluxo para adição de arquivos de teste não difere muito do fluxo de outras features, a exceção é que sempre é preciso verificar se os arquivos de testes são executados tanto na branch local como no workflow configurado pelo github actions.

Alguns detalhes importantes a serem considerados na adição de um teste são:

- Privacidade do construtor, caso for default os testes são visíveis apenas localmente
- Inicialização das variáveis, método @BeforeAll não é executado automaticamente
- Database utilizada, para as classes de teste existe uma instância específica
- Garantia de que novos registros no banco sejam removidos depois de um teste, usando o método `setUp`

Dado a especificidade dessa configuração, optamos por adicionar os testes na branch principal utilizando a opção de `squash and merge`, para não incluir múltiplos commits de relevância menor.

Tecnologias

Esta seção descreve as opções de tecnologias analisadas para o desenvolvimento do projeto, os motivos da escolha de uma tecnologia específica, vantagens e desvantagens de cada uma e também as consequências dessas escolhas em outros artefatos do sistema.

Versões

Esta seção lista as versões utilizadas de cada tecnologia.

- **Banco de dados**
  - Postgresql 15.4
  - Driver jdbc 42.5.1
- **Java**
  - aws sdk 1.12.565
  - jdk-20.0.2
  - javafx-media, javafx-controls e javafx-fxml,20.0.2
  - dotenv-java 3.0.0
  - maven 3.9.4
  - junit 5.6.0

Aplicação

Para definir a principal linguagem de programação a ser usada, os membros da equipe discutiram entre si para saber qual seria a melhor linguagem considerando as competências de cada um. A partir dessa discussão ficou acordado que a linguagem principal usada seria Java. Os fatores externos que influenciaram a decisão foram:

- Todos os membros da equipe tiveram contato com a linguagem;
- Para que o projeto seja utilizado na disciplina de Projeto Detalhado de Software ele precisa ser desenvolvido em Java, dessa forma podemos tirar dúvidas com mais de um professor acerca de certas características do projeto.

Já os fatores específicos da linguagem que motivaram essa decisão foram:

- Portabilidade entre diferentes sistemas operacionais, como Linux e Windows;
- Múltiplas escolhas de frameworks para cada finalidade, como JUnit e mockito para testes ou JavaFx e Swing para interfaces gráficas;
- A estrutura de um projeto permite que um arquivo jar executável seja facilmente exportado, o que nos beneficia na questão das entregas parciais, já que podemos fornecer aos usuários uma forma rápida de usarem alguma funcionalidade já implementada;
- Novas dependências podem ser acrescentadas facilmente ao projeto, utilizando maven.

Ainda assim, existem algumas desvantagens no uso da linguagem Java, consideramos como principais as seguintes:

- Caráter orientado a objetos da linguagem acarreta em uma arquitetura do sistema mais complexa, afetando as tarefas de modelagem e levando a um tempo de desenvolvimento maior;
- Verbosidade da linguagem leva a uma sintaxe complicada e, muitas vezes, prejudica o debugging
- Mensagens de erro extensas e pouco informativas

Banco de Dados

A escolha do sistema de gerenciamento de banco de dados (SGBD) foi influenciada pelas necessidades dos clientes, pelas restrições do projeto e também pela própria escolha da linguagem de programação. Inicialmente foram consideradas quatro alternativas: *cloud storage* da Google via *firebase*; *dynamodb* na Amazon Web Services; banco de dados gerenciado com *Postgresql* dentro da AWS e banco de dados local gerenciado com postgresql. A seguinte tabela mostra os principais pontos de cada uma das alternativas.

	Relacional	Limite de armazenamento gratuito	Preço por Gigabyte	Especializado em multimídia	Requer algum cadastro
Firebase Storage	Não	5 GB	0,026$/GB (plano blaze)	Sim	Sim
DynamoDB	Não	25 GB	0,375$/GB (região São Paulo)	Não	Sim
Postgres na AWS	Sim	20 GB (+20 GB para backup)	–	Não	Sim
Postgres com ElephantSQL[1]	Sim	20 mb	– (50$/mês por 50 gb em instância dedicada)	Não	Sim
Postgres local	Sim	Não há	Não há	Não	Não

A necessidade principal dos clientes é armazenar dados de multimídia e informações sobre eventos de uma forma centralizada, foi dito pela coordenadora Michelle em entrevista que seria preferível que este armazenamento fosse feito em nuvem.

Porém, considerando os prazos para entrega e a necessidade de testar o sistema com alguma conexão remota, nós optamos por utilizar durante o desenvolvimento da primeira entrega o ElephantSQL, a utilização dele será revisada após a definição sobre como será feito o armazenamento das mídias.

Armazenamento de mídia

A plataforma de armazenamento de mídia principal escolhida para a aplicação foi o serviço S3 da Amazon Web Services, utilizando uma instância de testes durante a fase alpha do projeto, as principais motivações para isso foram:

[1] Permite cinco conexões simultâneas

- o setor de desenvolvimento do IDM utiliza serviços da AWS para armazenamento de mídias
- sdk para linguagem Java fácil de instalar e utilizar
- limite de armazenamento razoável (5 gb) no plano gratuito

Dessa forma, as informações sobre arquivos que serão armazenadas no SGBD atuaram como uma espécie de cache, enquanto o conteúdo dos arquivos ficará armazenado nesse serviço.

Ferramentas

O desenvolvimento de projetos de software envolve uma série de etapas complexas, que vão desde a concepção da interface do usuário até a gestão eficiente de processos e colaboração em equipe. Nesse contexto, a combinação estratégica de ferramentas desempenha um papel fundamental para garantir o sucesso do projeto. A seguir, estão elencadas as ferramentas selecionadas, juntamente com as razões subjacentes à sua escolha:

- **JavaFX:**

O JavaFX é uma plataforma de software usada para criar aplicações gráficas de interface do usuário (GUI) em Java. Ele fornece uma maneira eficiente de criar interfaces de usuário atraentes e interativas para aplicativos de desktop, dispositivos móveis e outros contextos. A principal razão para usar o JavaFX é sua integração nativa com a linguagem Java, permitindo que você crie interfaces de usuário complexas e dinâmicas usando código Java padrão.

- **Maven:**

O Maven é uma ferramenta de automação de construção e gerenciamento de projetos em Java. Ele ajuda a simplificar o processo de compilação, teste, empacotamento e distribuição de projetos. Com o Maven, você pode definir as dependências do projeto em um arquivo de configuração (pom.xml) e deixar o Maven cuidar da resolução de dependências, compilação e criação de pacotes. Isso torna a colaboração mais fácil e mantém o projeto organizado.

- **GitHub:**

O GitHub é uma plataforma de hospedagem de código que facilita o versionamento e colaboração em projetos de software. Ele utiliza o sistema de controle de versão Git e oferece recursos para rastrear alterações, revisar código, gerenciar problemas e colaborar com outros desenvolvedores. Usar o GitHub permite que você mantenha um histórico completo de todas as alterações no código, facilite a colaboração entre membros da equipe e ofereça uma plataforma para compartilhar e distribuir seu projeto.

- **Figma:**

O Figma é uma ferramenta de design de interface do usuário baseada na web. Ele permite que equipes criem, colaborem e integrem designs de interface de usuário de maneira eficiente. O Figma é especialmente útil quando várias pessoas estão trabalhando juntas, pois suporta colaboração em tempo real e permite a criação de protótipos interativos. Ele ajuda a visualizar e refinar a aparência e a interação da interface antes de implementá-la no JavaFX.

- **Astah UML:**

Para realizarmos a modelagem de classes e, com menor foco, de sequências do sistema, utilizamos a ferramenta astah, sob licença de estudante. Para facilitar a edição por toda a equipe, mantivemos o arquivo .asta contendo os diagramas versionado no repositório, e para atualizarmos os diagramas utilizamos uma branch `modelagem`.

- **Trello (descontinuado);**

A equipe cessou a utilização do Trello, por não estar cumprindo adequadamente com sua função de gerenciar as tarefas, principalmente aquelas relacionadas a código, acreditamos que o Github Projects é uma alternativa mais eficiente para isso.