|  |  |
| --- | --- |
| Atividades | XP (Método Ágil) |
| Requisitos | - Fazer o que o cliente precisa! Atender suas necessidades!  Requisitos Funcionais (expressos através de Story Cards): são as tarefas, as funcionalidades do sistema.  - O cliente pode buscar tablaturas por nome da música e nome da banda  - O cliente pode visualizar a tablatura encontrada  - O Administrador do sistema pode adicionar uma tablatura  - O Administrador do sistema pode adicionar uma tablatura  - O Administrador do sistema pode editar uma tablatura  Requisitos Não Funcionais: são as qualidades do sistema.  - Separação de Interesses (Separation of Concerns - SoC): separar corretamente lógica, interação com o usuário e comportamentos na aplicação.  - Utilizar a Arquitetura de Software MVC (Model-View-Controller)  - Portabilidade: desenvolver para diferentes plataformas. A Arquitetura MVC, através de sua separação de interesse, permitirá a criação de diferentes Views para o mesmo Model.  - Plataforma Web  - App Android  - Usabilidade: oferecer uma experiência de uso simples para as pessoas considerando os padrões de interação das plataformas que elas estão usando.  - Usabilidade para plataforma Web  <http://www.w3.org/WAI/WCAG20/quickref/>  - Usabilidade para plataforma Android <http://www.nngroup.com/articles/ten-usability-heuristics/>  - Exemplos de outros requisitos não funcionais (não houve a necessidade de aplicá-los neste projeto, mas eles poderiam ser aplicados em outros)  - Escalabilidade: capacidade de aumentar o número de clientes utilizando o mesmo servidor, por exemplo.  - Por exemplo, alterar tecnologias de servidor relacionadas com o processamento das requisições dos clientes e envio das respostas . É importante notar que a arquitetura MVC ajuda muito nisso, pois mesmo alterando o servidor as Views continuarão as mesmas (Ex: ver pasta “Extras” do repositório).  - Desempenho: melhorar a performance da aplicação, por exemplo.  - Por exemplo, alterar o paradigma de BD (alguns BDs funcionam melhor para determinado tipo de aplicação. Ex: BD Orientado a Objetos ou BD NoSQL). Novamente, a arquitetura MVC vai permitir que se altere o BD sem alterar as Views (Ex: ver pasta “Extras” do repositório).  - Segurança: garantir que apenas os usuários habilitados acessarão a aplicação e que os dados serão transmitidos de forma segura, por exemplo.  - Por exemplo, ver https://github.com/pac4j/spark-pac4j |
| Projeto | - Projeto de Arquitetura de Software (é a arquitetura de software bem projetada que garantirá o requisito não funcional de “Separação de Interesses” (SoC) e, portanto, a “Portabilidade” para a execução da mesma lógica do sistema em diferentes plataformas. No nosso caso, plataformas Android e Web.  (Como a lógica da aplicação – ex: buscar tablaturas por nome da música e nome da banda – é a mesma em qualquer plataforma, porém a tecnologia de exibição de informações é diferente para cada plataforma – ex: Web uso HTML5, plataforma Android uso componentes em XML - , preciso de um padrão arquitetural que separe claramente a lógica da aplicação da visão (interface) da aplicação)  architecture.jpg  - Projeto de Interação com o Usuário (para atender o requisito não funcional de usabilidade)  mob1.png  8 -Estética e Design Minimalista (a interface apresenta apenas as informações essenciais)  mob2.png  6. Reconhecimento ao invés de lembrança (Ao invés de se lembrar do nome da banda, a interface oferece uma forma de reconhecimento)  5. Prevenção de erros (Ajuda o usuário a não entrar com um nome inválido de banda)  mob3.png  2. Conexão entre o sistema e o mundo real (O resultado da busca fala “a língua” do usuário com termos apropriados àquele usuário) |
| Implementação | <https://github.com/giulianobertoti/eXtremeProgramming>  classes+codigo.jpg |
| Testes | <https://github.com/giulianobertoti/eXtremeProgramming/tree/master/Tablatures/src>    1º Teste: Adicionei 2 objetos Music no atributo musics da classe MusicList e depois testei se de fato haviam 2.  2º Teste: Fiz uma busca passando um objeto Specification e conferi se o resultado era correto. |
| Manutenção | 1ª Testes Automatizados: em toda atualização, principalmente naquelas onde não fui eu que implementei a lógica (foi outro membro da equipe), eu tenho testes automatizados para checar se eu quebrei ou não o código.  2ª Encapsulamento: para acrescentar um novo requisito funcional do cliente, por exemplo comparação por ano da música, eu precisei alterar apenas a classe Specification, pois todas as comparações da busca por músicas estão lá (ou seja, meu projeto está bem encapsulado). A classe MusicList, que é a classe principal de lógica, não precisou ser alterada.  **public** **class** Specification {  **private** String band;  **private** String musicName;  **private** **int** year;  **public** Specification(String band, String musicName, **int** year){  **this**.band = band;  **this**.musicName = musicName;  **this**.year = year;  }    **public** **boolean** matches(Specification spec){  **if**(!band.equals(spec.band)) **return** **false**;  **if**(!musicName.equals(spec.musicName)) **return** **false**;  **if**(year!=spec.year) **return** **false**;  **return** **true**;  }  } |
|  | FIM |