**Gamelog [1.0]**

Alunos: João Marcus Cardoso e Igor Meurer.

Professor: Thiago Funk

Disciplina: Padrões de Projetos

**SUMÁRIO**

[**INTRODUÇÃO 2**](#_mz8h5omtbmcu)

[**Versionamento 3**](#_p8r96gkgmbs6)

[**REQUISITOS FUNCIONAIS 3**](#_xi6k5blp5efq)

[**REQUISITOS NÃO FUNCIONAIS 4**](#_5f5exc3imn4q)

[**REGRAS DE NEGÓCIO 5**](#_1xiyl77ph4z5)

[**RELATÓRIO 5**](#_jszylvlso0uw)

[Adapter](#_2gpay985kbey) [6](#_livwcxwh0nq6)

[State 7](#_tqsqz5ugvgjz)

[Visitor 7](#_p1af6sr0oyww)

[Composite 10](#_eyn167wexsnx)

[Decorator 14](#_9o4twhmf2vzb)

[**REFERÊNCIAS 14**](#_hm0mzdgcjnzx)

# **INTRODUÇÃO**

O projeto gamelog tem intuito de permitir que o usuário mantenha um catálogo de jogos e dlcs concluídos, permitindo assim que ele tenha um local onde ele possa organizar os seus jogos concluídos ou que esteja jogando. Podendo até mesmo criar reviews/notas sobre o que achou sobre o jogo e manter até mesmo as configurações como a plataforma em que ele jogou tal jogo.

Nesse software o usuário conseguirá criar conta e se autenticar, após isso será redirecionado para a página inicial, que seria sua “biblioteca” a onde ele poderá cadastrar seus jogos jogados, avaliá los e escrever reviews sobre eles. voltando ao seu perfil onde ele vai poder editar o mesmo, ele será redirecionado para uma tela a onde ele pode tanto mudar seu nickname, foto de perfil, bio e também adicionar seus jogos favoritos, que vão ser um total de 5 o qual o jogo que ele ditar como o seu top 1 favorito terá uma borda diferente do restante, voltando a página de perfil terá além dos jogos favoritos vai possuir também a contagem das horas e jogos adicionados na parte inferior do site, o site contará com um menu que possibilitará o redirecionamento para a página de exibição do catálogo e de jogos favoritos. Neste trabalho utilizaremos a linguagem Java com o Framework Spring Boot, juntamente do CSS, HTML5 e Javascript além do banco de dados Postgres.

# **Versionamento**

| Versão | Data | Alterações | Fonte | Autor |
| --- | --- | --- | --- | --- |
| 1.0 | 08/01/2023 | Criação do Escopo e Requisitos | João Marcus, Igor Meurer | João Marcus, Igor Meurer |

# **REQUISITOS FUNCIONAIS**

| ID | Descrição | Versão | Data | Fonte | Status |
| --- | --- | --- | --- | --- | --- |
| RF01 | Manter usuário | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF02 | Manter jogos | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF03 | Manter reviews | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF04 | Manter configs | 1.0 | 17/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF05 | O sistema deve permitir filtrar os jogos por meio do filtro escolhido pelo usuário | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF06 | O sistema deve pegar a imagem do jogo por meio de uma url | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF07 | Ao clicar em um jogo na biblioteca o sistema deve abrir um modal com as informações do jogo | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF08 | O usuário pode escolher os seus jogos favoritos em seu perfil | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF09 | O jogo favorito principal deve ter uma borda dourada em volta dele | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF10 | Os jogos na biblioteca devem mostrar a nota da review que o usuário deu | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RF11 | O sistema vai permitir que o usuário baixe seus jogos adicionados em formato de csv ou pdf | 1.0 | 17/01/2024 | Documento de requisição do aplicativo | Em construção |

# **REQUISITOS NÃO FUNCIONAIS**

| ID | Descrição | Versão | Data | Fonte | Status |
| --- | --- | --- | --- | --- | --- |
| RNF01 | O perfil de usuário deve contabilizar a quantidade de jogos cadastrado do usuário | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RNF02 | O sistema deve Fazer uma média das review do usuário | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RNF03 | O sistema deve mostrar uma prévia da imagem quando o usuário coloca a url dela | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RNF04 | O usuário pode ter uma bio e uma foto de perfil | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RNF05 | O Sistema deve mandar um e-mail para o usuário | 1.0 | 17/01/2024 | Documento de requisição do aplicativo | Em construção |

# **REGRAS DE NEGÓCIO**

| ID | Descrição | Versão | Data | Fonte | Status |
| --- | --- | --- | --- | --- | --- |
| RN01 | O sistema deve ser desenvolvido utilizando spring boot | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN02 | O sistema deve ser web | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN03 | O sistema deve utilizar postgresql | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN04 | O usuário pode ter até 5 jogos favoritos | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN05 | O usuário só pode adicionar ao seus jogos favoritos jogos que ele tenha cadastrado | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN06 | O sistema deve ter o e-mail como campo único. | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN07 | O sistema deve validar por meio de jwt | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |
| RN08 | O sistema deve validar se o e-mail existe | 1.0 | 09/01/2024 | Documento de requisição do aplicativo | Em construção |

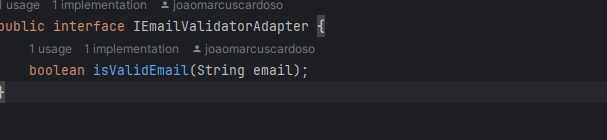
# **RELATÓRIO**

Neste trabalho aplicamos os padrões de projetos lesionados na aula, tendo como intuito entender melhor o funcionamento de cada padrão e suas respectivas estruturas, tendo isso em vista daremos uma explicação como foi aplicado e por que foi utilizado em cada módulo.

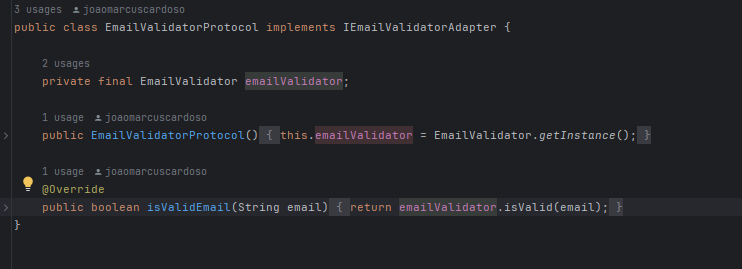
Por conta do curto tempo disponível para a execução do trabalho, focamos na parte do backend, onde aplicamos todos os padrões pedidos. E para o trabalho 2 foi decidido continuar o desenvolvimento da API.

## **Adapter**

## O padrão Adapter foi utilizado para a validação de emails usando uma biblioteca do Apache por conta da grande quantidade de funções de envio de email e sua versatilidade quando se trata de ser maleável quando é necessário trocar a biblioteca utilizada. Temos a interface que é uma assinatura do método:



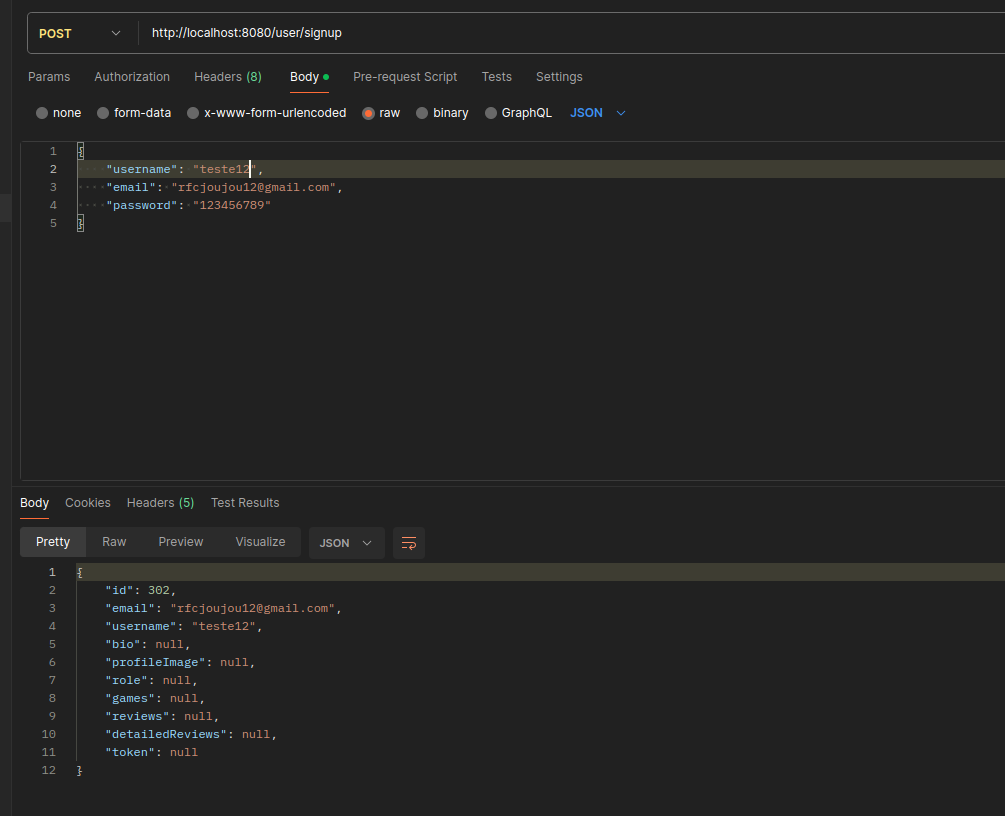
Em seguida temos o “EmailValidatorProtocol” que implementa a interface e o método “isValidEmail” o qual chamado “emailVaidator” que por sua vez recebe um e-mail e retorna se foi validado ou não.



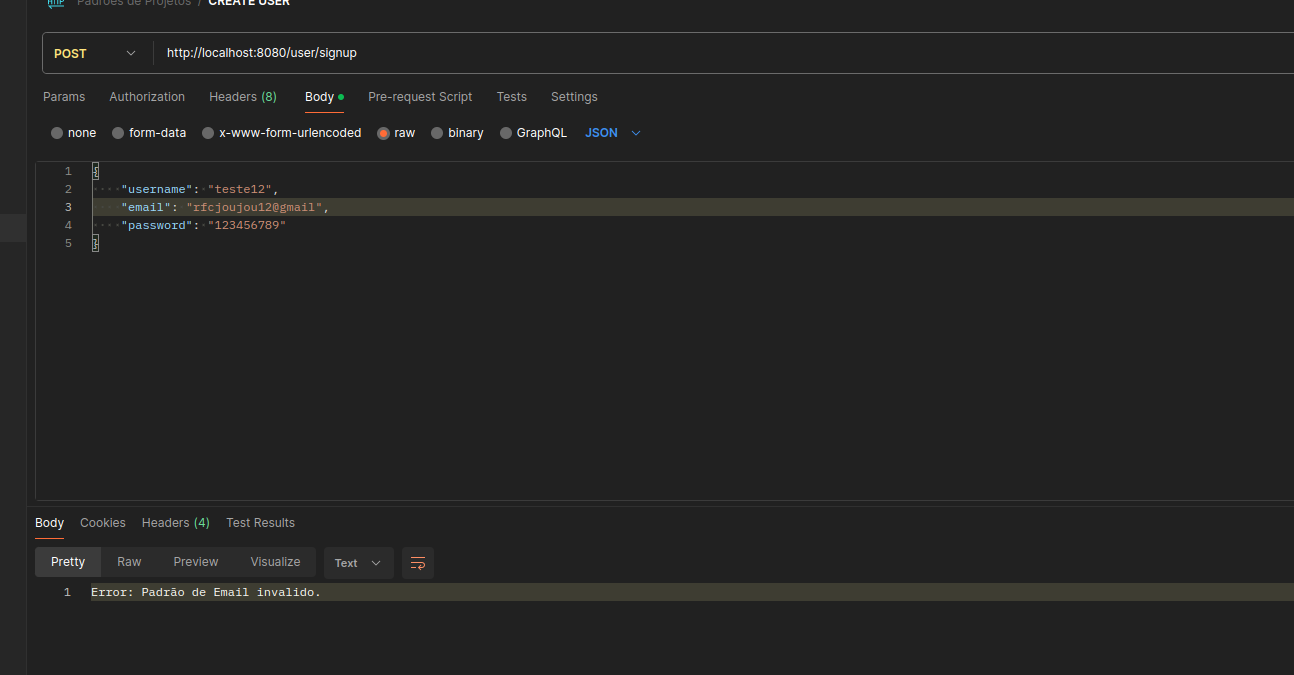
Então para finalizar utilizamos o adapter no controller de User para realizar o cadastro de usuário.



Exemplo de retornos:  
Sucesso:



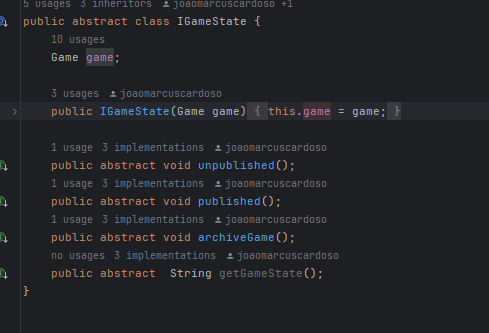
Exemplo de e-mail incorreto:



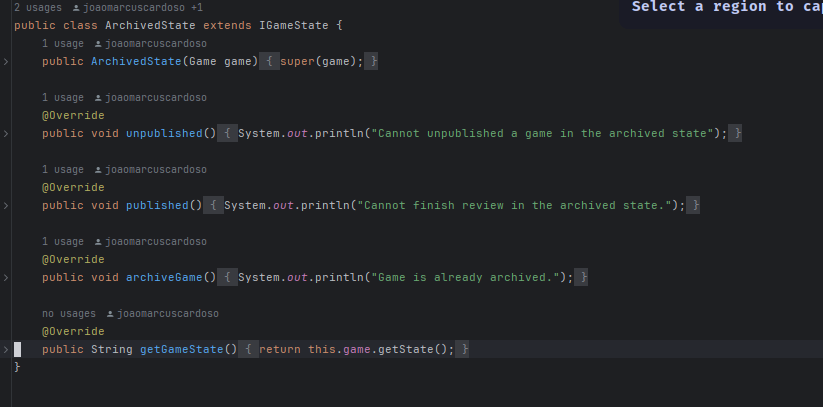
## **State**

O padrão State foi utilizado para atualizar o “status” do jogo de acordo com as ações do usuário. Quando usuário cadastra um jogo o “status” é “Rascunho”, esse “status” permite apenas a ação de publicar um “jogo”, e quando o jogo for publicado, a única ação possível será “arquivar” o jogo, para isso utilizamos a estrutura do State a fim de que cada classe implemente um modo de atualização de “status”.

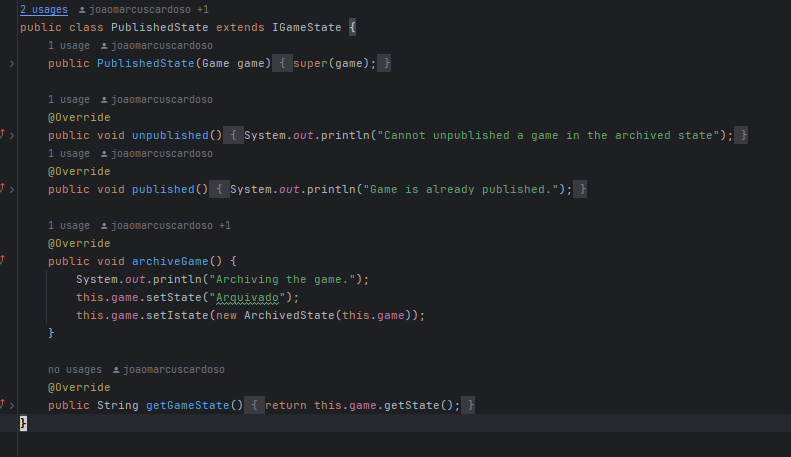
Temos uma classe abstrata que recebe o “game” e defini 4 métodos a ser implementados.



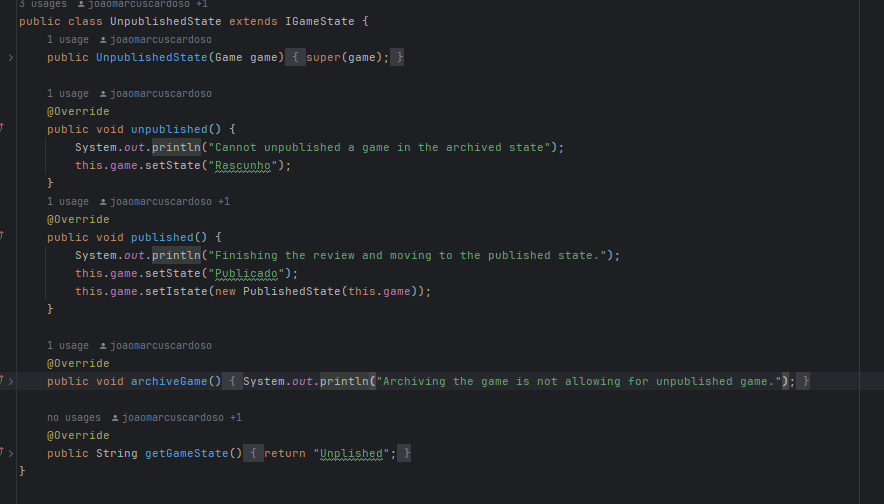
Podemos ver que a classe “ArchivedState” estende o “IGameState”, ela implementa validações dos status que não pode ser mudado como o unplished(Rascunho) e published(Publicado).



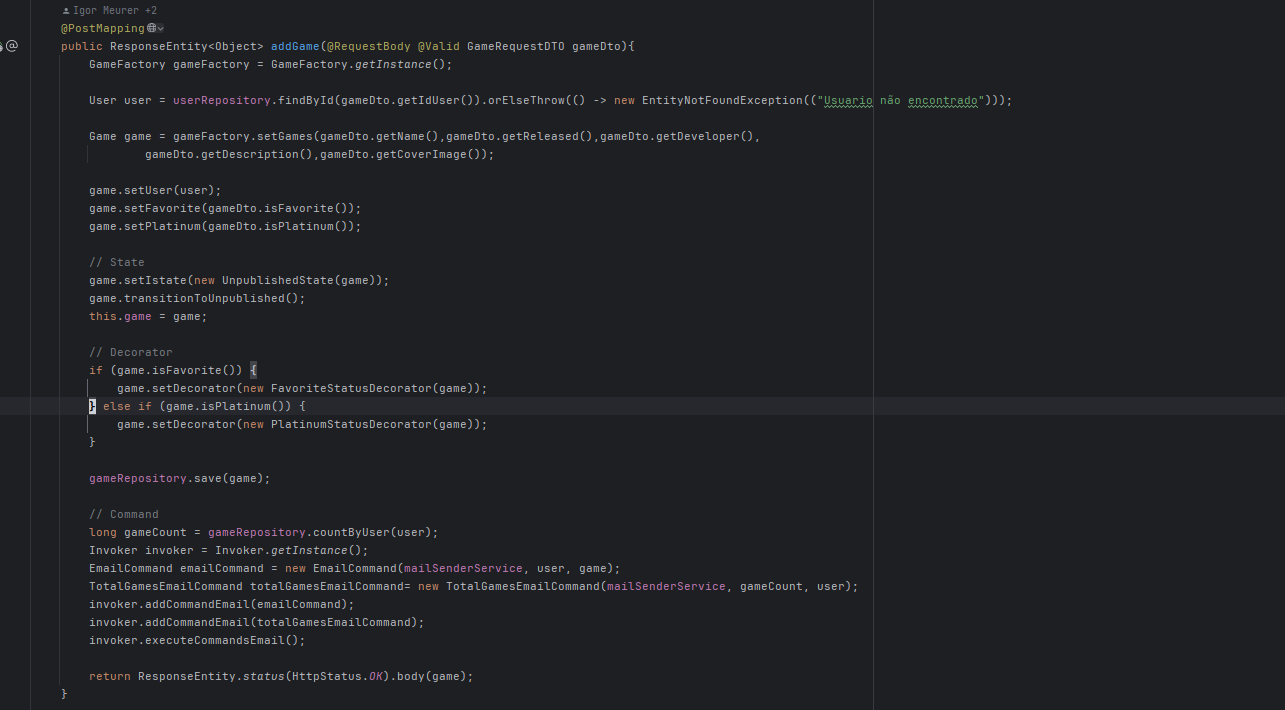
Temos a classe “PublishedState” que permite apenas atualizar o “status” para arquivado.



Além dela, temo a classes “UnpublishedState” que permite apenas atualizar o “status” para publicado.



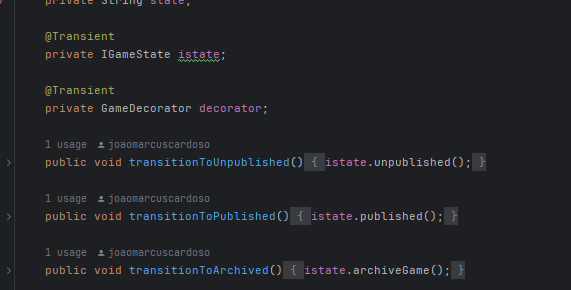
No endpoint para criar um jogo temos a lógica de setar o “state” com Unpubilshed, sendo assim o jogo fica com “status” de rascunho.



Nos endpoints de published e archive ele faz as chamadas na entidade para realizar a atualização do “state”.

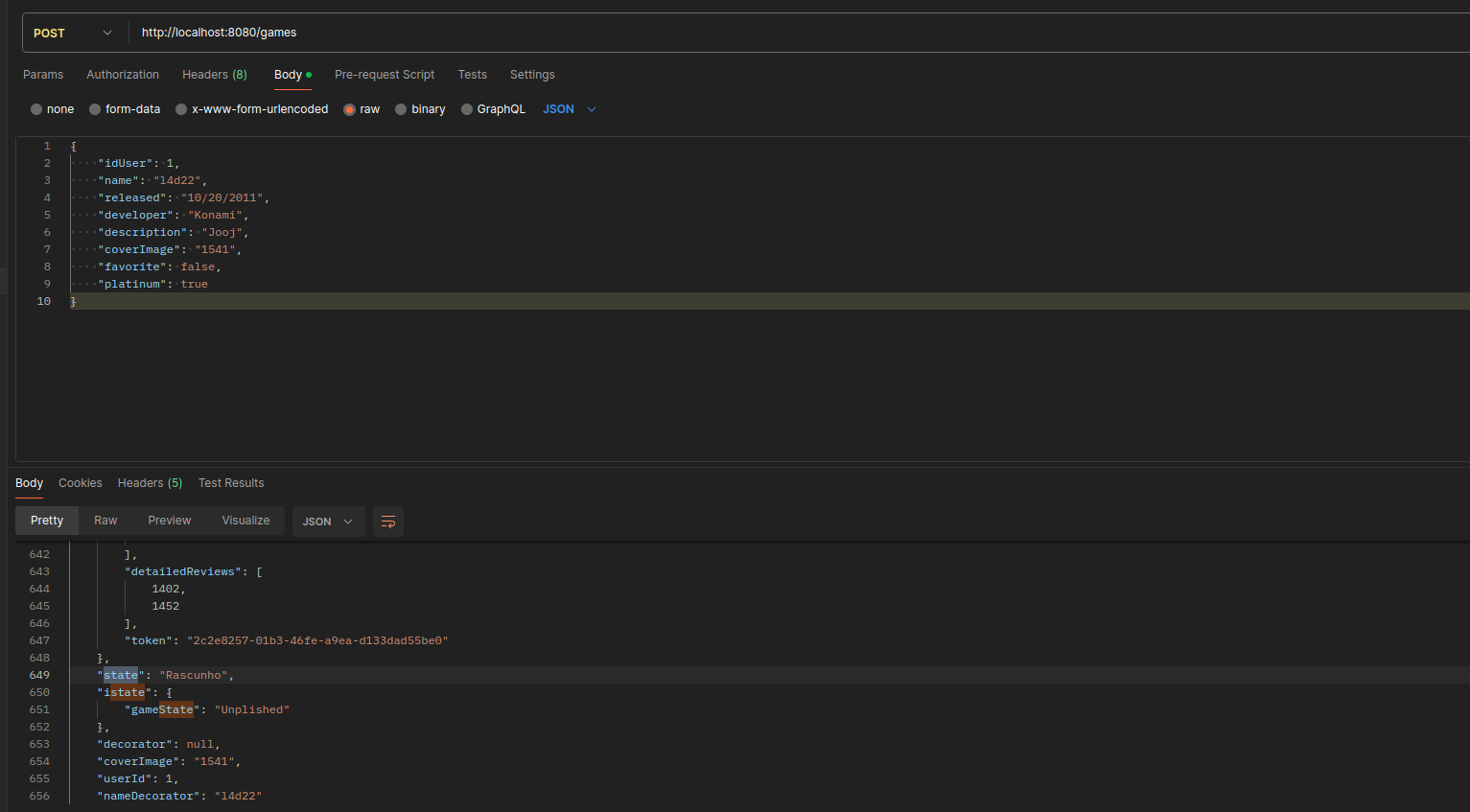


Abaixo temos o variável “istate”, que seria um “IGameState” que recebe o State atual do nosso objeto, sendo assim ele faz as chamadas dos métodos unpublished, published, archiveGame, vale ressaltar que dependendo do valor do “istate” temos ações diferentes como atualizar ou não o valor.

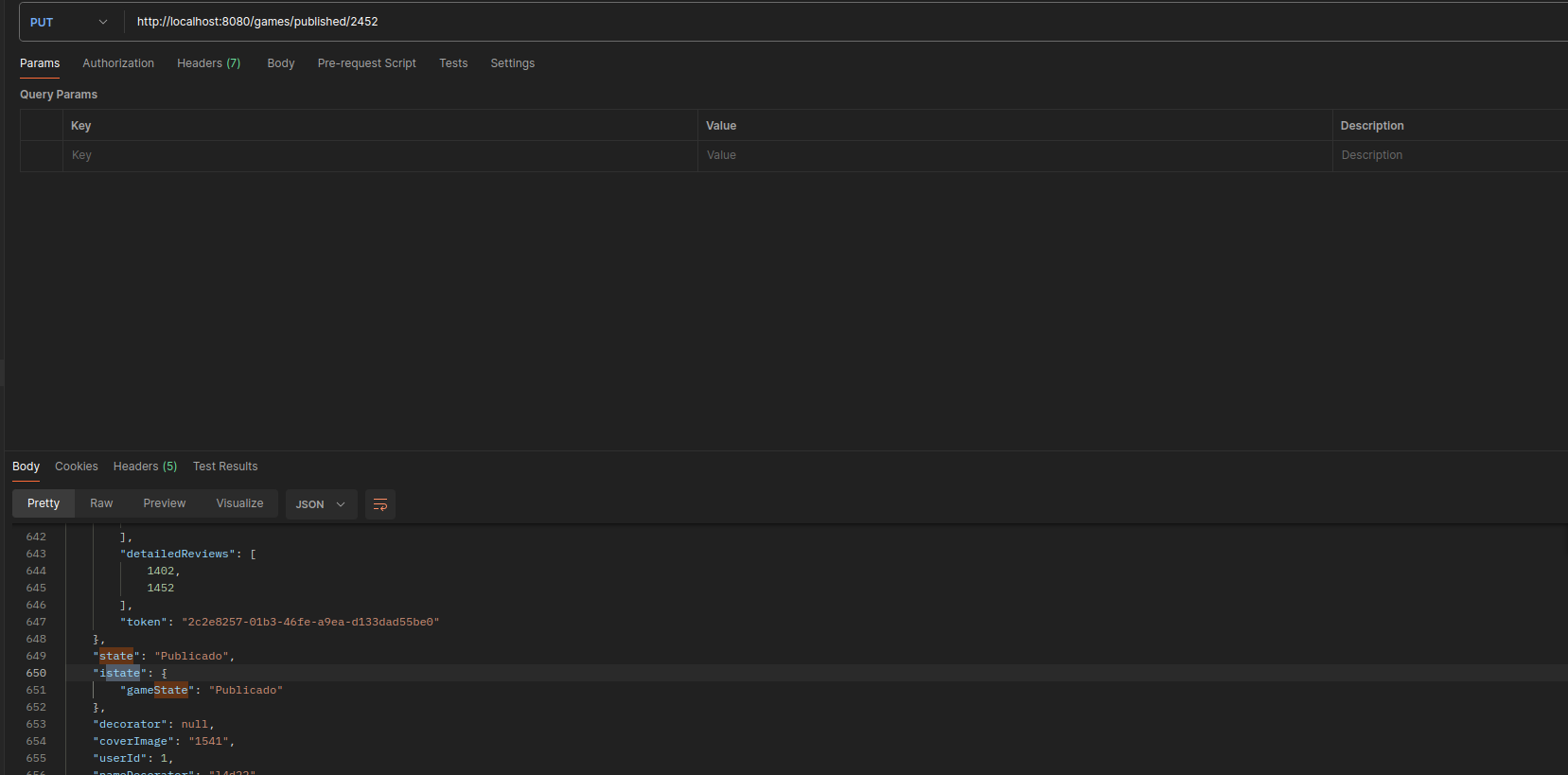


## Esse seriam os endpoints via postman:

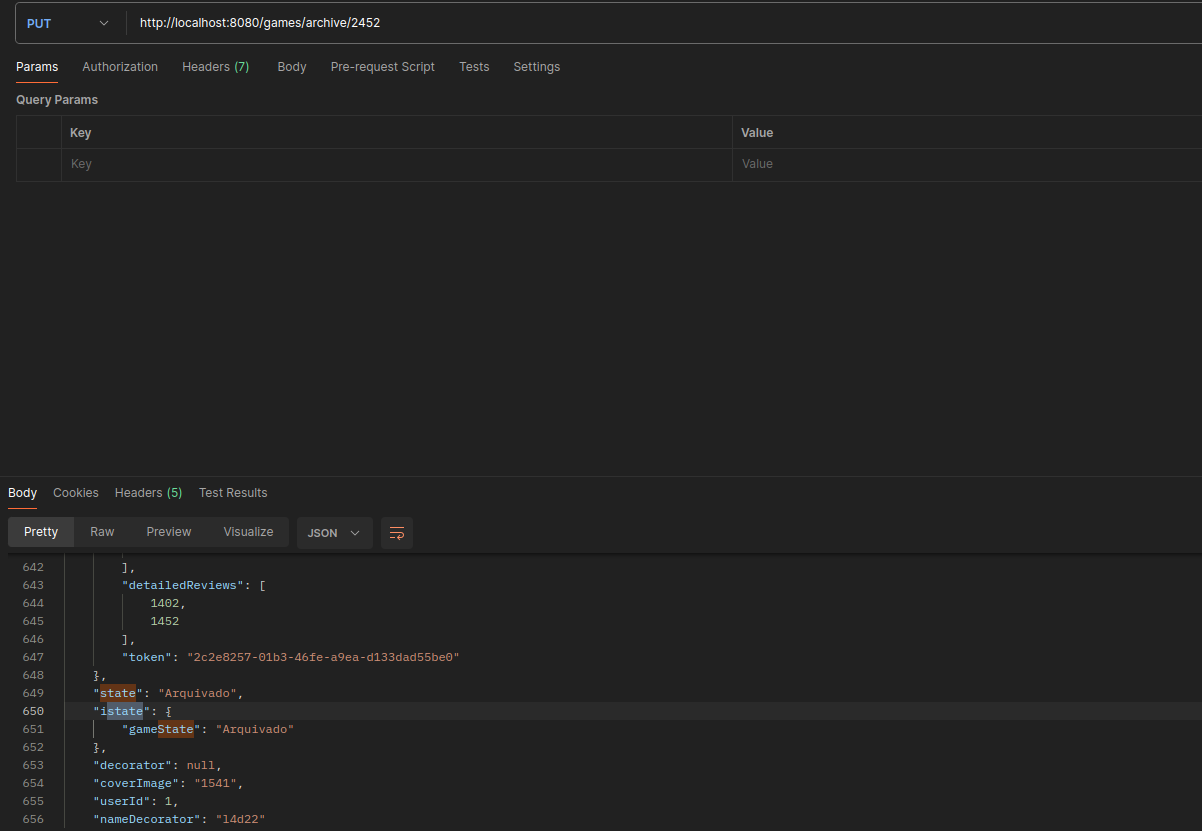
State em rascunho:



State publicado:



State arquivado:

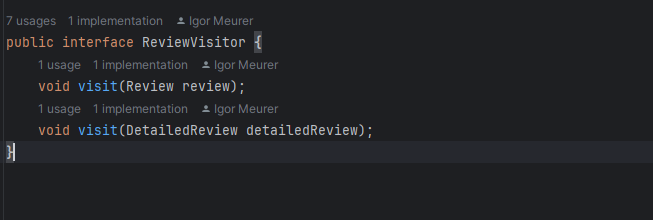


## **Visitor**

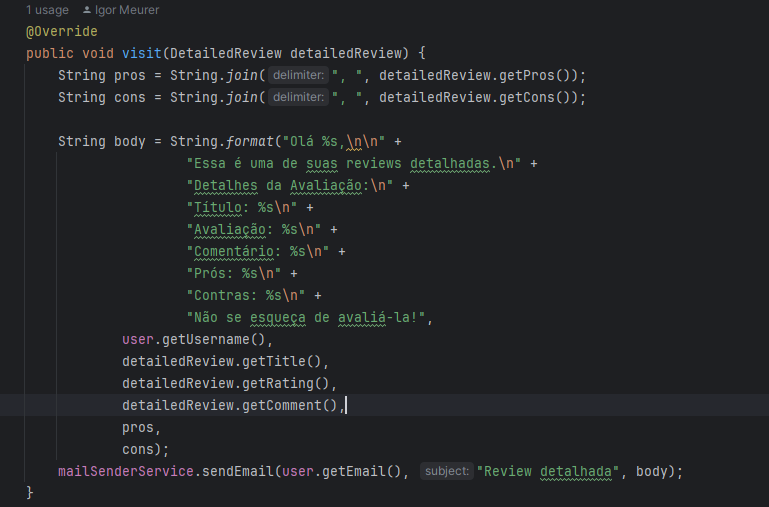
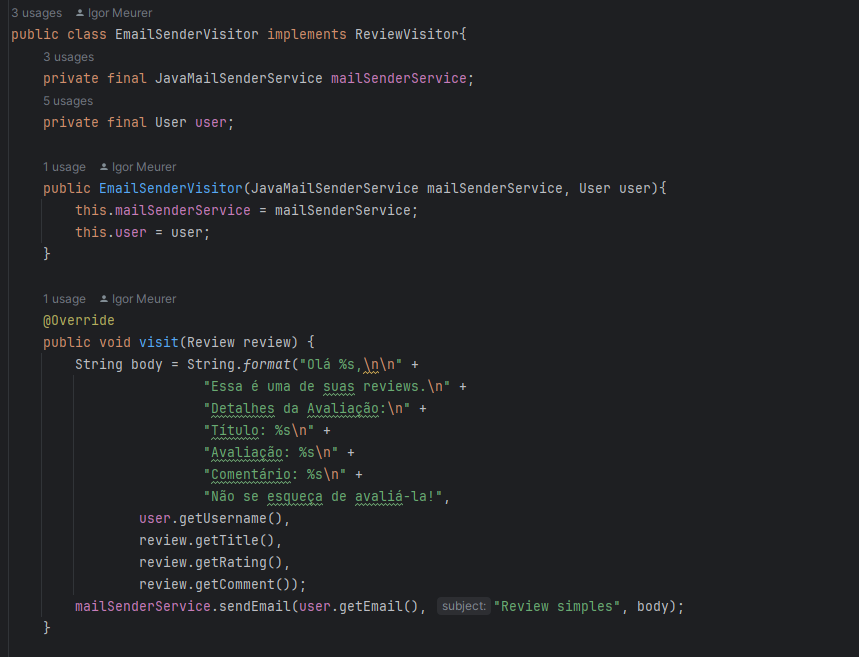
O padrão Visitor a estrutura é muito similar ao padrão Command, porém até mais poderoso que ele, e fácil de implementar, com essa ideia decidimos implementar ele com uma função parecida com a do Command anterior, pois na primeira ideia de implementar ele junto do Composite para utilizar a capacidade dele de modificar diversas partes de uma árvore acabou não funcionando como desejávamos.

Então ele foi implementado para enviar emails para o usuário de suas reviews caso ele quisesse.

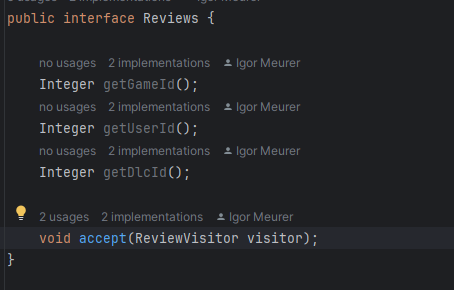
Possuímos a interface que é implementada pelo concrete visitor que por sua vez implementa os métodos que será enviado o e-mail das reviews.

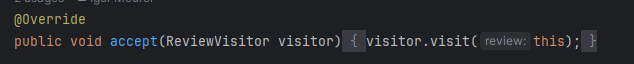


Temos o “EmailSenderVisitor”, que é o concrete visitor, onde ele implementa os métodos da interface e monta o body da mensagem com as informações das reviews e a envia para o e-mail do usuário. Nesse visitor ele é responsável tanto para enviar emails das reviews simples quanto para as reviews detalhadas. Vale ressaltar que caso fosse interessante executar alguma outra função em várias reviews seguidas, como por exemplo transformar elas em doc ou pdf, poderíamos criar outro concrete visitor e nele fazer a lógica para tal.

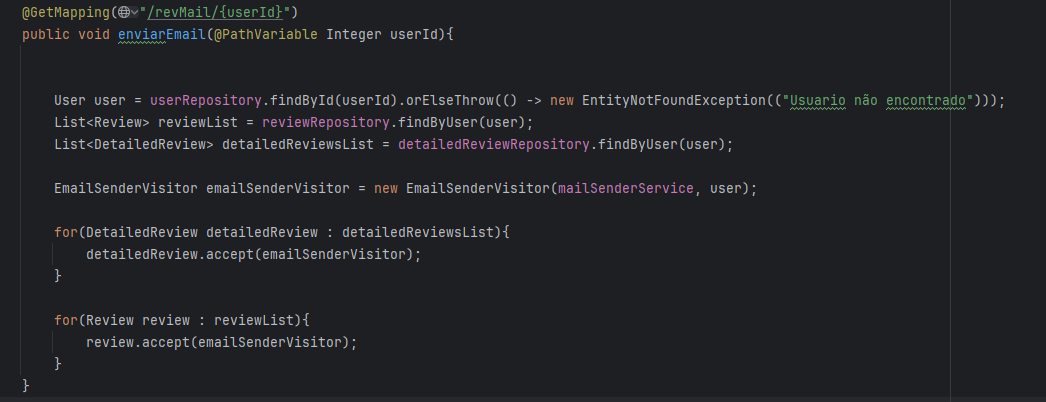


Também nas classes foi necessário colocar o accept para que o visitor funcionasse.

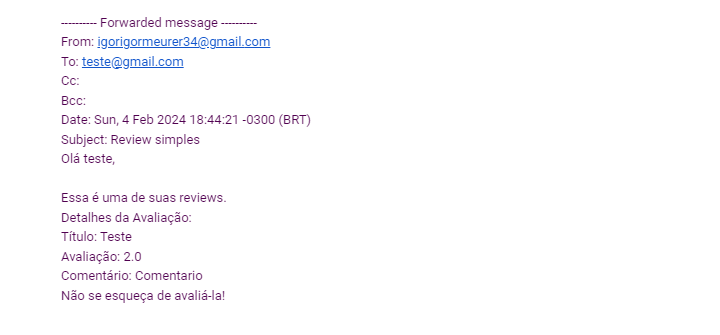
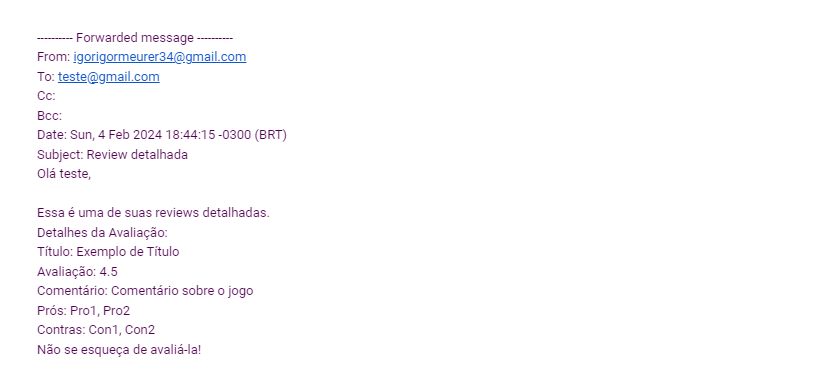




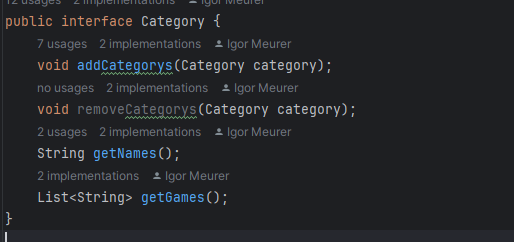
E no controller de review foi criado um método que retorna todas as reviews simples e detalhadas do usuário e a adiciona em uma lista, essa lista então é colocada em um for e o visitor é passado em todas as reviews fazendo com que ele mande um e-mail pra cada review na lista.

****

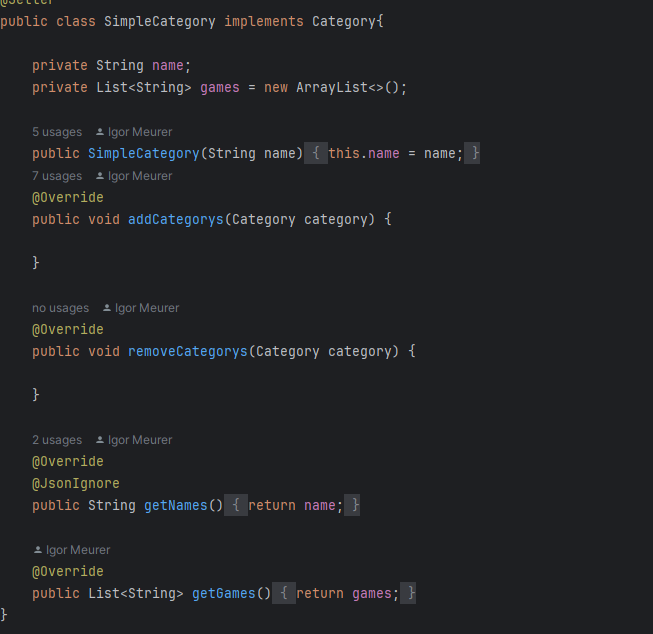
Um exemplo dos emails enviados:

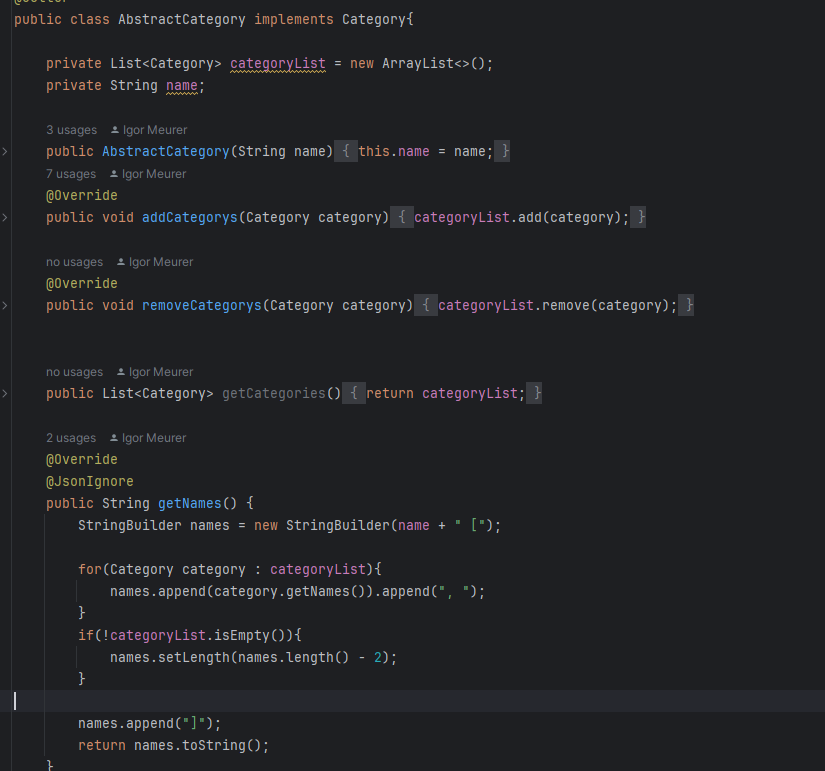


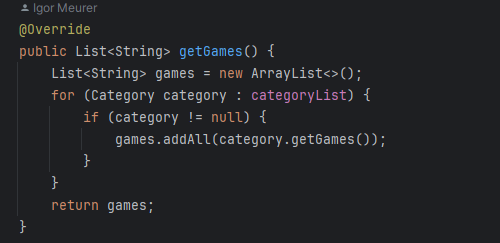
## **Composite**

Utilizando o Composite, foi decidido utilizar ele para fazer uma árvore de categorias de jogos, onde as categorias que podem ter filhas seriam “abstractCategory”, mesmo elas não sendo exatamente abstratas, e “simpleCategory” que seriam as folhas da árvore. Temos a interface que é implementada por ambas. 

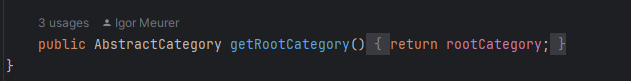
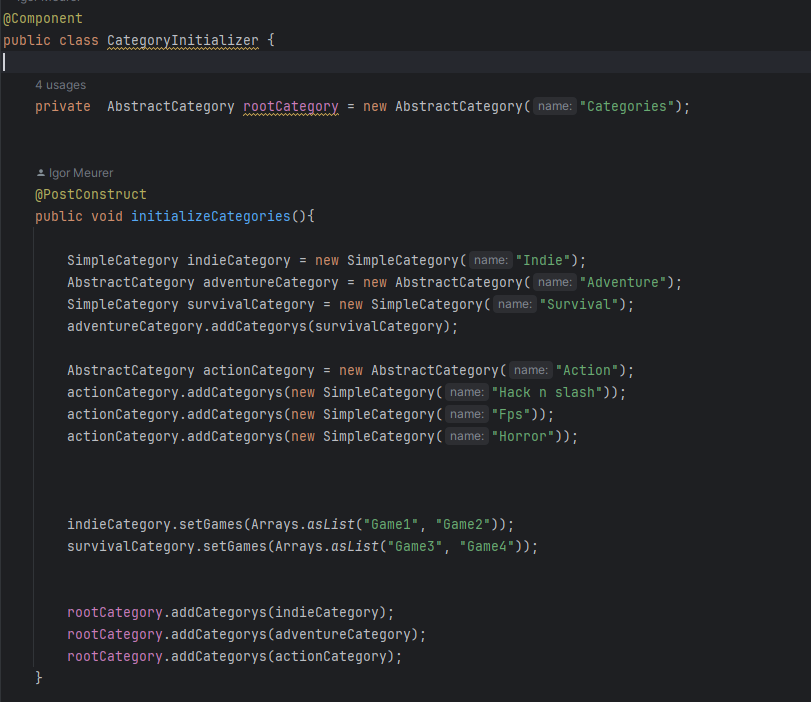
As duas “category”.



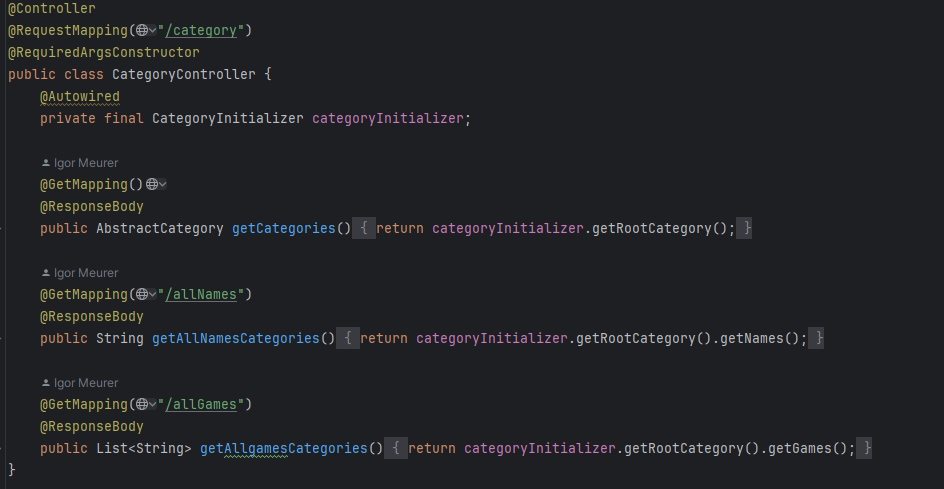
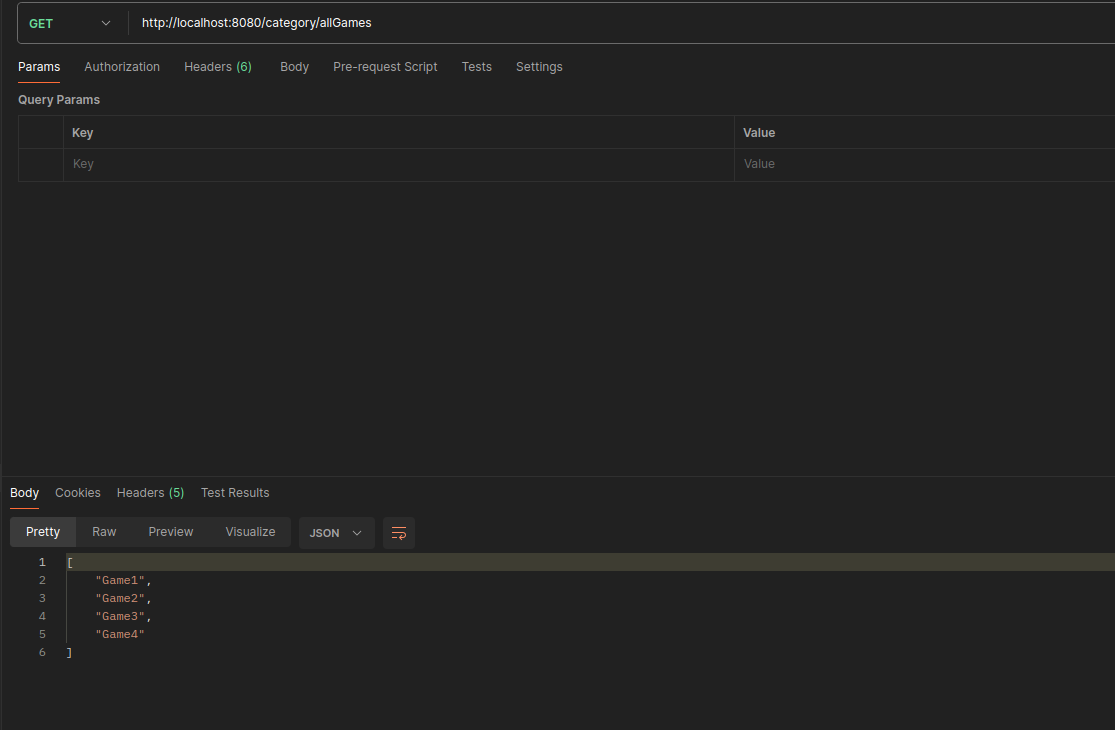


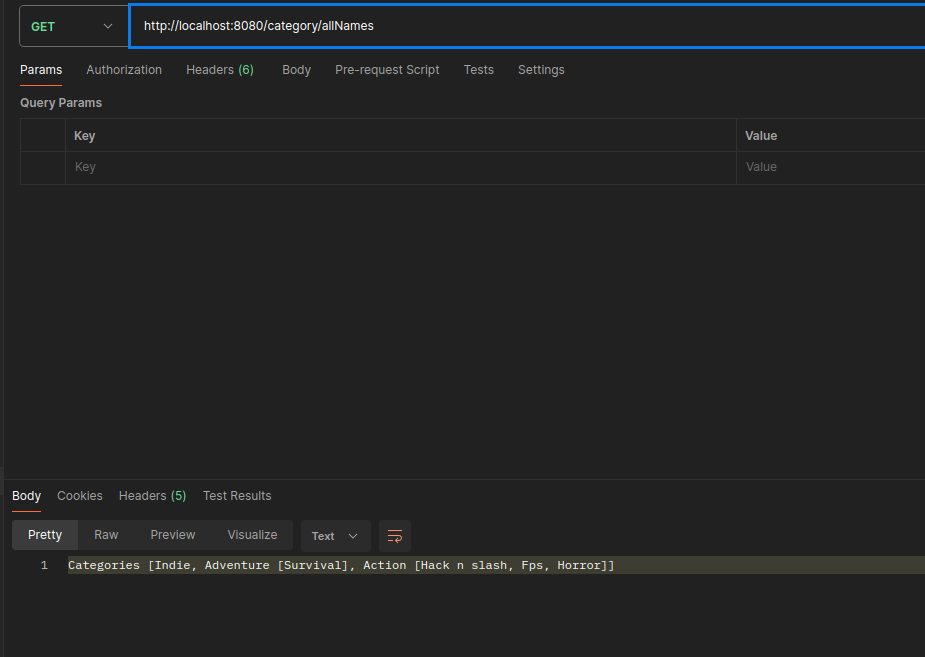


Ambas possuem nome, pois elas poderiam ser usadas para pegar as categorias disponíveis para classificar os jogos, como Ação para abstract que abrange várias categorias mais diretas e FPS para “simpleCategory” que é mais direta e não tem tanta possibilidade de ter categorias que derivam dela. Por conta de estar usando categoria como um composite, foi decidido instanciar as categorias e a árvore ao iniciar a API, pois para salvar o composite utilizando JPA se tornou algo bem complicado. Seria também utilizado uma lista de Games nas classes folhas, que serviram para as abstract também pois elas pegariam os jogos salvos nas classes filhas, porém foi deixado apenas como uma Lista de String para mostrar como ficaria ao retornar o resultado dos get no controller, pois estava ficando muito confuso o método para buscar todos os jogos que iriam pertencer a tal categoria no método que inicializa elas. Então com isso temos a classe que inicia e monta uma árvore relativamente simples, mas que mostra como funcionaria ela no sistema a “CategoryInitializer”.



Em seguida temos a “CategoryController” que interage diretamente com as categorias, onde colocamos apenas os métodos para pegar todos os nomes da árvore e todos os jogos da árvore para verificar se ambos os métodos estavam funcionando como deveriam e retornando corretamente.

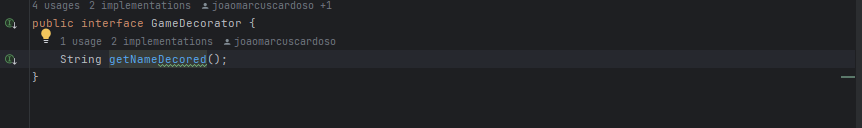
  
Endpoint retorno de categorias:  




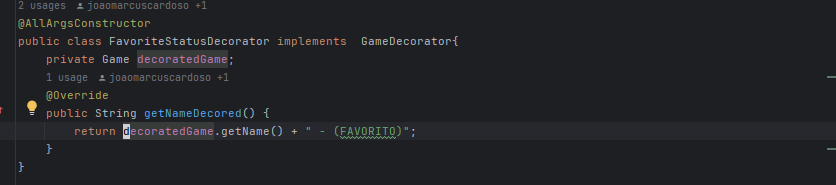
## **Decorator**

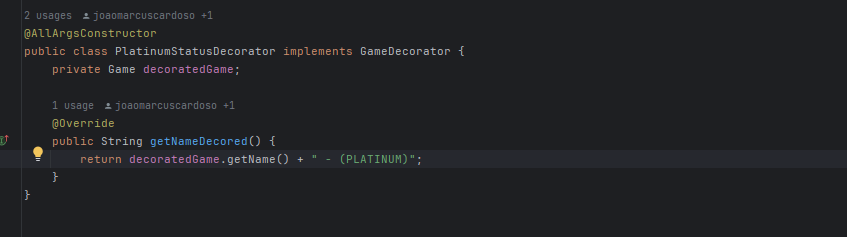
O decorator foi utilizado para setar quando um jogo foi platinado(que foi alcançadas todas as conquistas) ou favoritado(quando usuário favorita um jogo), sendo assim o jogo recebe diferente “nameDecorated”. Com esse padrão poderíamos personalizar as características dos jogos, pois quando marcamos uma verdade para “favorite” ou “platinum” podemos implementar os métodos de forma personalizada, como isso podemos adicionar campos que seriam alterados somente se tiver decorator ou alterar campos já existentes.

Temos uma interface que possui o método “getNameDecored”, que será implementado por “FavoriteStatusDecorated” e “PlatinumStatusDecored”.



Como podemos ver tanto o “FavoriteStatusDecorated” como o “PlatinumStatusDecorated” atualizam o campo “nameDecored”:

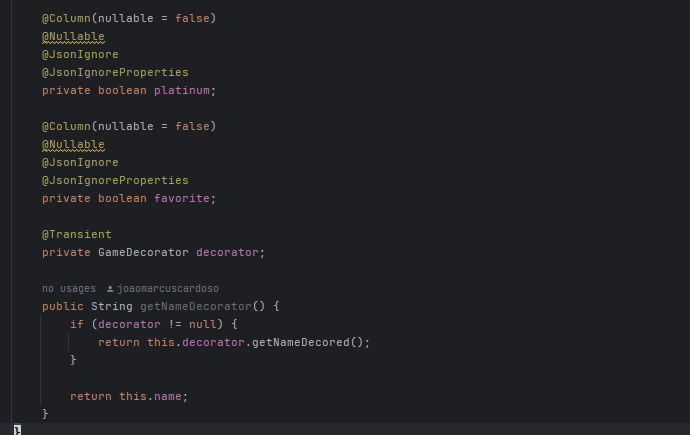




Foram criados dois endpoint para setar o decator como “favorite” ou “planitum” sendo assim o namedDecorated ia mudar de acordo como o valor do “decorator”:



Na entidade criamos os campos “platinum” e “favorite” que seriam um boolean, e temos um método “getNameDecorator” para retornar o nome quando a variável “decorator” for setada como “FavoriteStatusDecorator” ou “PlatinumStatusDecorator”



# 

# **REFERÊNCIAS**

<https://github.com/joaomarcuscardoso/gamelog>

<https://refactoring.guru/design-patterns/facade>

<https://refactoring.guru/design-patterns/observer>

<https://refactoring.guru/design-patterns/singleton>

<https://refactoring.guru/design-patterns/command>

<https://refactoring.guru/design-patterns/factory-method>

<https://refactoring.guru/design-patterns/abstract-factory>