Universidade Estadual da Paraíba
Bacharelado em Ciência da Computação

Alunos: Edilson do Nascimento Costa Júnior - 211080101,
Joyce Lima Avelino - 211080047,
José Vinícius Silva Alves - 211080390,
Caio Henrique Barbosa da Silva - 211080314,
Lucas Santos Andrade - 201080010

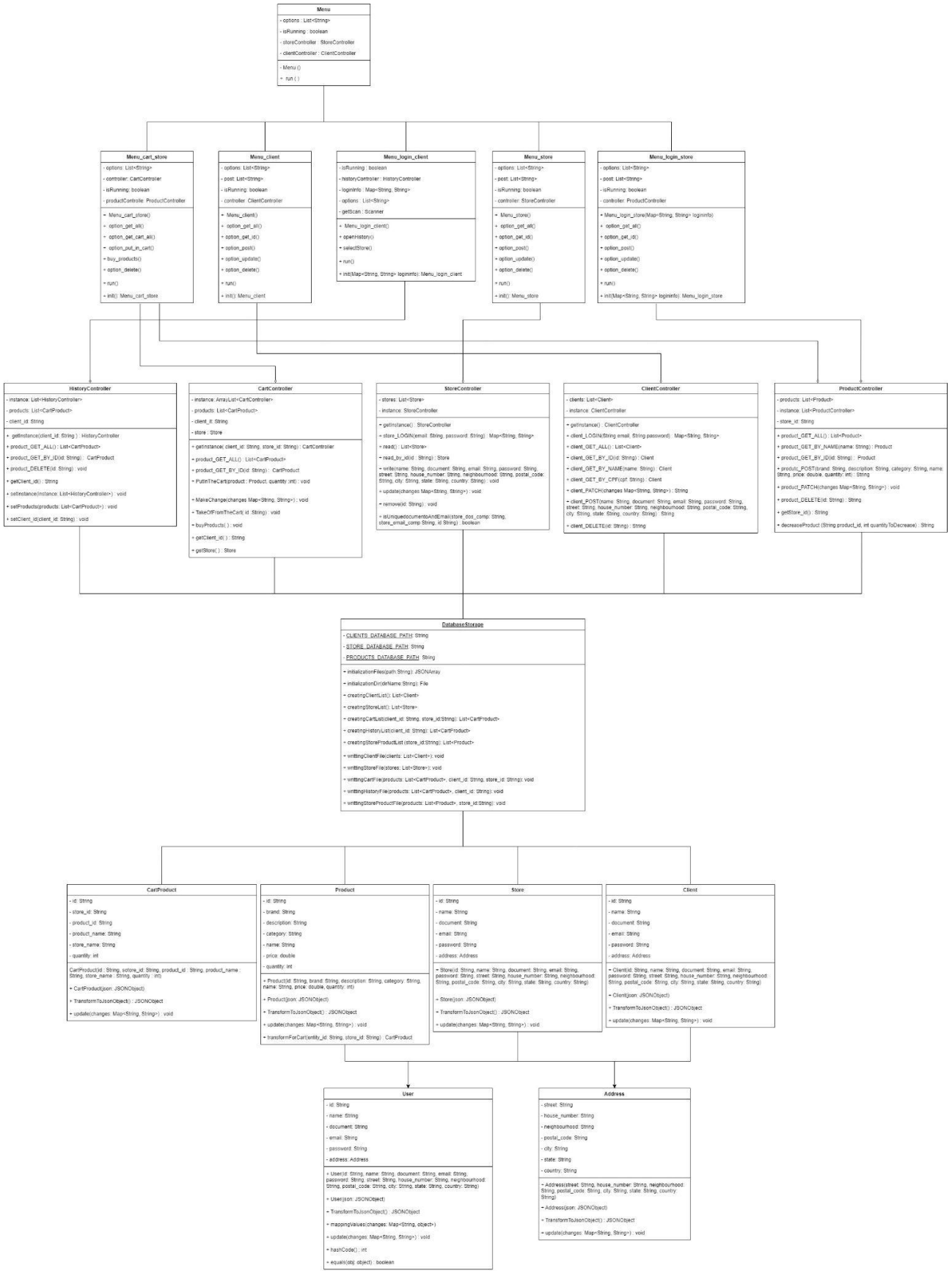
Professor(a): Sabrina de Figueirêdo Souto
Disciplina: Métodos Avançados em Programação

Projeto MAP - Marketplace

Campina Grande – Paraíba, 29 de junho de 2023

Sumário

1. Diagrama de classe



obs - link caso fique difícil a visualização: [Map atividade.drawio]

2. Padrões de projeto utilizados

No decorrer do desenvolvimento do projeto, foram analisadas alguns padrões para o compor, com isso, foram utilizados os seguinte padrões:

2.1. Facade

O padrão Facade é um padrão estrutural que fornece uma interface simplificada para um subsistema complexo. Ele encapsula a complexidade do subsistema, abstraindo os detalhes de implementação e oferecendo uma interface coesa para os clientes. O uso do Facade promove a simplicidade de uso, organiza o código, facilita a manutenção, promove boas práticas de design e melhora a legibilidade e a modularidade do código. É especialmente útil quando há um subsistema com várias classes e funcionalidades.

O facade no caso do projeto foi utilizado na criação das classes denominadas controllers, por fornecer interface simplificada e unificada para o subsistema de manipulação de solicitações e respostas em uma aplicação, assim facilitando com que nós lidamos com o banco de dados a partir dessas funções de forma prática e simples.

2.2. MVC

O padrão Model - View - Controller (MVC) é um padrão arquitetural que divide uma aplicação em três componentes principais: o Model (modelo), o View (visualização) e o Controller (controlador). Ele visa separar as preocupações e responsabilidades entre esses componentes para melhorar a manutenção, reutilização e testabilidade do código.

O Model representa a camada de dados da aplicação, sendo responsável pela manipulação e gerenciamento dos dados. Ele encapsula a lógica de negócio e fornece uma interface para acessar e modificar os dados.

A View é responsável pela apresentação da interface gráfica ao usuário. Ela exibe os dados do Model e permite a interação do usuário, enviando comandos para o Controller.

O Controller atua como intermediário entre o Model e a View. Ele recebe os comandos da View e coordena a lógica de negócio, manipulando o Model conforme necessário. Também atualiza a View para refletir as alterações nos dados.

No contexto do projeto, o padrão MVC foi aplicado para organizar a estrutura e o fluxo das classes. O Model corresponde à manipulação dos dados do banco de dados, enquanto o View representa a interface gráfica para exibir e interagir com esses dados. O Controller atua como um facilitador entre o Model e a View, coordenando as ações do usuário e atualizando o Model e a

View conforme necessário. Essa abordagem simplifica o desenvolvimento e a manutenção da aplicação, proporcionando uma separação clara entre as responsabilidades dos componentes.

2.3. Creator

O padrão Creator ajuda a separar a lógica de criação de objetos do código cliente, resultando em um código mais organizado, flexível e reutilizável. Ele permite que a responsabilidade de criação seja delegada a uma classe dedicada, como os controllers, concentrando a lógica de criação em um único lugar. Isso permite que o código cliente não precise conhecer os detalhes específicos da criação dos objetos, interagindo apenas com a interface fornecida pelo Creator.

A ideia central do padrão Creator é fornecer uma interface simples e consistente para a criação de objetos desejados. Ao utilizar os controllers como classes Creator, eles encapsulam a complexidade da criação de objetos, ocultando os detalhes específicos do processo de criação do código cliente. Dessa forma, o código cliente pode solicitar a criação dos objetos desejados por meio de uma interface clara e intuitiva, sem a necessidade de entender os detalhes internos da implementação. Isso simplifica a interação com a classe Creator e promove uma separação eficiente das responsabilidades.

2.4. Singleton

O padrão Singleton é um padrão de projeto de software que garante a existência de apenas uma instância de uma classe. Ele fornece um ponto de acesso global para essa instância, permitindo que outros objetos interajam com ela de forma consistente. O padrão Singleton é útil quando você precisa coordenar ações em vários pontos do código com uma única instância compartilhada, assim como foram feitos nos controllers do projeto. No entanto, é importante usar o Singleton com cautela, pois seu uso excessivo pode levar a problemas de acoplamento e dificuldades de teste.