# Entrega Final de Documentação


## Equipe: AutoPrime

Integrantes: Victor Miranda Florido, Felipe Mendes Salles, Guilherme, Pedro Arthur Nascimento Resende, Gabriel Tavares Gargur
Disciplina: Projeto de Software
Turma: A1
Professora: Vania de Oliveira Neves

Niterói-RJ, 2023

# 1- Escopo do projeto

O sistema da concessionária AutoPrime especializada na venda de carros é caracterizado como um sistema de software abrangente e eficiente que atende às necessidades tanto dos nossos clientes quanto da equipe de funcionários. Nosso objetivo principal é proporcionar uma experiência de compra de veículos simplificada e personalizada, onde os clientes possam pesquisar, comparar e configurar veículos de acordo com suas preferências. Além disso, o software permitirá aos clientes agendar test drives, obter cotações de preços, descontos em futuras compras e vantagens no pagamento.

Para a equipe de funcionários, o sistema fornecerá ferramentas de gerenciamento de estoque, vendas e atendimento ao cliente. Isso incluirá recursos para verificar o status dos veículos em estoque, gerar relatórios de vendas e monitorar o próprio desempenho. Também estamos focados em manter registros precisos de manutenção e histórico de veículos, garantindo um serviço pós-venda de alta qualidade. A nossa concessionária irá possuir um gerente responsável pela avaliação dos funcionários, soluções de problemas graves dos clientes e bonificação da equipe. Nosso projeto de software buscará acompanhar as demandas e necessidades do mercado automobilístico, oferecendo uma solução completa e tecnologicamente avançada para a nossa concessionária de carros, a AutoPrime.

# 2- Requisitos Arquiteturais

## 2.1- Objetivos da Arquitetura

- O sistema deve ser de uso fácil e rápido para usuários, sem exigir treinamento prévio (Usabilidade).
- O sistema deve permitir o cadastro de clientes.
- O sistema deve fornecer a um único usuário selecionado funções privilegiadas de gerência (Segurança).
- O sistema deve permitir o cadastro de funcionários.
- O sistema deve manter registro dos pedidos de cada cliente.
- O sistema deve fornecer e calcular descontos para clientes baseado em suas compras passadas.
- O sistema deve permitir o agendamento de *test drives*.
- O sistema deve gerenciar os modelos de veículos disponíveis e seus respectivos estoques.
- O sistema deve permitir que os usuários vejam as informações e modifiquem as partes de um modelo automobilístico que sejam personalizáveis.
- O sistema deve ser capaz de mostrar as especificações de mais de um veículo ao mesmo tempo para o cliente poder compará-los.

- O sistema deve permitir que clientes possam fazer uma compra segura (Segurança).
- O sistema deve calcular as devidas bonificações para um funcionário baseado nas suas vendas.
- O sistema deve possuir uma central de atendimento para solucionar problemas que os clientes vierem a ter (Confiabilidade).
- O sistema deve estar disponível 24/7 (Disponibilidade)

## 2.2- Restrições Arquiteturais

- O sistema deve manter seus registros atualizados e precisos em tempo real (Adequação Funcional).
- Os agendamentos de test drive devem ter ao menos x horas de antecedência.
- Não deve ser possível diferentes clientes marcarem um test drive com o mesmo carro simultaneamente (Segurança).
- Apenas gerentes poderão cadastrar funcionários (Segurança).
- O agendamento do test-drive deve ter a verificação do funcionário.
- O agendamento de manutenção poderá ser feito em até dois anos após a compra

## 2.3- Padrões Arquiteturais Adotados

O modelo arquitetural escolhido foi o MVC(Model-View-Controller) devido a maior afinidade da equipe com esse modelo, além de ter uma boa escalabilidade, contribuindo para a adição de possíveis novos automóveis e/ou incrementos e upgrades dos mesmos assim atendendo os requisitos de personalização e também conforme a quantidade de usuários for aumentando a arquitetura poderá contribuir para a adaptabilidade dos agendamentos e logins em larga escala, facilitar a divisão da equipe para que se concentrem em áreas específicas do projeto, somado ainda a facilidade de manutenção do sistema, devido a divisão dos componentes, ajudando a alterar um destes sem que seja necessário afetar algum outro. Além disso, por ser um padrão bastante adotado em projetos, isso facilita a busca por informações do grupo na hora do desenvolvimento para que o modelo atenda as necessidades do projeto com eficiência.

## 2.4- Justificativa das decisões arquiteturais adotadas

A escolha do padrão arquitetural MVC (*Model-View-Controller*) para o desenvolvimento do nosso sistema de concessionária chamado *AutoPrime* foi embasada em algumas considerações. Primeiramente, a afinidade da equipe com esse modelo proporciona um ganho de produtividade, pois uma equipe familiarizada com a arquitetura tende a produzir um código mais coeso e de qualidade superior. Além disso, o MVC por característica tem destaque por sua escalabilidade, então entendemos que isso pode ser um ganho considerável para o projeto, já que diferentes partes do sistema podem ser desenvolvidas e aprimoradas de forma independente. Com isso, podemos nos concentrar em áreas específicas do projeto, tornando o desenvolvimento mais eficiente e ágil.

Outro aspecto relevante que consideramos é a facilidade de manutenção proporcionada pelo modelo MVC. A divisão dos componentes em *Model*, *View* e *Controller* simplifica a identificação e correção de problemas, bem como a implementação de novos recursos, porque podemos realizar alterações em um componente sem que isso afete outros, reduzindo o risco de introduzir erros não intencionais no sistema. Outro fator menos relevante, mas que consideramos, foi que o MVC é amplamente adotado em projetos de software, então isso simplifica a obtenção de informações e soluções para desafios específicos que possam surgir durante o desenvolvimento do nosso sistema. Portanto, chegamos no consenso de adotar o padrão MVC por conta das considerações citadas e entendemos que seja uma boa escolha para atender às necessidades do sistema da *AutoPrime*.

# 3- Visões Arquiteturais

## 3.1 Atores do sistema

Usuários (Ator principal que realiza a maior parte das funções):

- Cliente (Ator para quem o serviço é prestado, realiza funções como se cadastrar, realizar agendamentos e comparar carros)
- Funcionário (Ator que presta serviço, realiza funções como gerenciar estoque e gerenciar serviços agendados)
- Gerente (Ator superior que coordena os prestadores de serviço)

Sistema (Funciona como uma ponte entre os usuários, realizando tarefas como notificar funcionário sobre novo agendamento por exemplo)

## 3.2 Casos de uso

**CS 01 (Realizar Cadastro Cliente):**

Cenário Típico:

1 - Cliente clica no botão Cadastre-se já
2 - Cliente digita suas informações (email, nome, cpf, data de nascimento, telefone)
3 - Cliente digita uma senha válida

Cenários Alternativos:

1ª - Email, CPF ou telefone já estão sendo usados
    1 - Servidor exibe mensagem de erro e solicita novas informações
2ª - A senha não segue os parâmetros válidos

1 - Servidor exibe mensagem de erro e solicita uma senha válida

**CS 02 (Realizar Cadastro Funcionário):**

Cenário Típico:

1 - Gerente clica no botão Cadastrar novo Funcionário
2 - Gerente digita informações do novo usuário (email, nome, cpf, data de nascimento, telefone, salário e função)
3 - Gerente cria uma senha

Cenários Alternativos:

1ª - Email, CPF ou telefone já estão sendo usados
1 - Servidor exibe mensagem de erro e solicita novas informações

Pré Condições:

1 - Gerente deve estar logado no sistema

**CS 03 (Fazer Login):**

Cenário Típico:

1 - O usuário clica no botão fazer login
2 - O usuário digita email/CPF e senha
3 - O usuário clica em Confirmar

Cenários Alternativos:

1ª - Email/CPF ou senha estão incorretos
1 - Servidor exibe mensagem de erro e solicita novas informações

**CS 04 (Atualizar Perfil do Usuário):**

Cenário Típico:

1 - O usuário clica no botão Meu Perfil
2 - O usuário altera o campo devido
3 - O usuário clica em Atualizar

Cenários Alternativos:

1ª - Dado a ser alterado não segue os parâmetros válidos
    1 - Servidor exibe mensagem de erro e solicita a informação válida

Pré Condições:

1 - Usuário deve estar logado no sistema

**CS 05 (Realizar Agendamento de Test-Drive):**

Cenário Típico:

1 - Cliente clica em Realizar Agendamento
2 - Cliente seleciona o tipo de Agendamento Test-Drive
3 - Cliente seleciona as informações do carro
4 - Cliente seleciona dentre as datas disponíveis a de sua preferência
5 - Cliente clica em Solicitar Agendamento

Pré Condições:

1 - Cliente deve estar logado no sistema.

**CS 06 (Realizar Compra de veículo):**

Pré Condições:
1 - O funcionário deve estar logado no sistema.
2 - O cliente deve ter um cadastro no sistema.
3 - O veículo a ser comprado deve estar disponível no estoque da concessionária.

Cenário Típico:
1 - O funcionário seleciona o veículo e as especificações desejadas pelo cliente.
2 - O funcionário seleciona na lista de clientes registrados, o cliente que está realizando a compra do veículo.
3 - O funcionário fornece data de compra, método de pagamento e acessórios.
4 - O sistema calcula o preço total da compra.
5 - O funcionário revisa as informações fornecidas pelo sistema.
6 - O funcionário clica em “Comprar veículo”.
7 - O sistema registra a compra, atualiza o estoque e emite o recibo do cliente.

Cenários Alternativos:

1 - veículo selecionado não está disponível em estoque:
    1.1 - Servidor exibe mensagem de erro e pede que o funcionário selecione um veículo disponível em estoque.

2 - Cliente não cadastrado no sistema:
    2.1 - Servidor exibe mensagem de erro e informa que o cliente precisa ser registrado para o funcionário poder prosseguir com a transação.

**CS 07 (Realizar Agendamento de Manutenção Pós Venda):**

Cenário Típico:

1 - Cliente clica em Realizar Agendamento
2 - Cliente seleciona o tipo de Agendamento Manutenção Pós Venda
3 - Cliente seleciona carro adquirido
4 - Cliente descreve o problema para uma avaliação mais rápida e eficiente
5 - Cliente seleciona dentre as datas disponíveis a de sua preferência
6 - Cliente clica em Solicitar Agendamento

Pré Condições:

1 - Cliente deve estar logado no sistema
1 - Cliente deve ter feito uma compra do carro em no máximo 2 anos

**CS 08 (Verificar Status dos Agendamentos/Compras):**

Cenário Típico:

1 - Cliente clica no seu Nome de Perfil
2 - Cliente clica em Meus Agendamentos/Compras
3 - Cliente procura o Agendamento/Compra que queira
4 - Cliente clica nele e verifica o status

Cenários Alternativos:

1ª - Cliente não realizou nenhum agendamento prévio
    1 - Servidor exibe mensagem de Nenhum Agendamento/Compra Realizado

Pré Condições:

1 - Cliente deve estar logado no sistema

**CS 09 (Gerenciar Agendamentos e Compras):**

Cenário Típico:

1 - Funcionário clica em Agendamentos e Compras
2 - Funcionário procura o Agendamento/Compra que queira
3 - Funcionário analisar Agendamento/Compra
    3.1 - Funcionário confirma Agendamento/Compra
    3.2 - Funcionário recusa Agendamento/Compra

Cenários Alternativos:

1ª - Clientes não realizaram nenhum agendamento prévio
    1 - Servidor exibe mensagem de Nenhum Agendamento/Compra Realizado

Pré Condições:

1 - Funcionário deve estar logado no sistema.

Gatilho:

1 - Clientes são notificados pelo servidor de que seus Agendamentos/Compras foram atualizados.

**CS 10 (Adicionar veículo no estoque):**

Cenário Típico:

1 - Funcionário clica em Adicionar veículo
2 - Funcionário preenche dados do veículo
3 - Funcionário clica em Confirmar

Cenários Alternativos:

1ª - Dados do veículo adicionado não segue os parâmetros válidos

1 - Servidor exibe mensagem de erro e solicita a informação válida.

2ª - veículo já cadastrado no sistema
1 - Servidor exibe mensagem de erro e informa que o veículo já está cadastrado.

Pré Condições:

1 - Funcionário deve estar logado no sistema.

**CS 11 (Dar baixa em veículo no estoque):**

Cenário Típico:

1 - Funcionário clica em Editar veículo
2 - Funcionário procura veículo
3 - Funcionário clica em Retirar do Sistema
4 - Funcionário confirma Solicitação

Pré Condições:

1 - Funcionário deve estar logado no sistema.

**CS 12 (Ver veículo):**

Cenário Típico:

1 - O usuário clica em “Ver Veículos”.
2 - O sistema apresenta uma interface com uma busca e listagem de veículos.
3 - O usuário seleciona/busca um veículo.
4 - O sistema exibe o veículo e suas especificações.

Cenários Alternativos:

1 - Carro buscado não foi encontrado no sistema:
1.1 - Servidor informa que o veículo não foi encontrado.

**CS 13 (Personalizar veículo):**

Pré Condições:
1 - O funcionário deve estar logado no sistema.
2 - O cliente deve ter um cadastro no sistema.

Cenário Típico:
1 - O funcionário seleciona o modelo de veículo desejado pelo cliente.
2 - O funcionário clica em “Personalizar veículo”.
3 - O sistema exibe opções de personalização (cores, acessórios e recursos adicionais).
4 - O funcionário seleciona as opções de acordo com as necessidades do cliente.
5 - O funcionário confirma a personalização.
6 - O sistema fornece as escolhas de personalização e o preço ajustado do veículo.
7 - O sistema emite um registro com a personalização solicitada pelo cliente para o funcionário.

Cenários Alternativos:

1 - Funcionário não selecionou todas as opções disponíveis:
1.1 - Sistema exibe mensagem de erro e informa que o funcionário precisa personalizar o veículo corretamente.

**CS 14 (Comparar veículos):**

Pré Condições:

1 - O cliente deve ter um cadastro no sistema.

Cenário Típico:

1 - O cliente clica em “Comparar 2 veículos”.
2 - O sistema apresenta uma interface para que o cliente selecione 2 veículos para compará-los.
3 - O cliente seleciona da lista apresentada pelo sistema, dois veículos.
4 - O sistema exibe uma tabela que mostra as especificações dos dois veículos selecionados, incluindo informações como marca, modelo, ano, preço.
5 - O cliente tem a opção de salvar a comparação ou realizar uma nova comparação.

**CS 15 (Verificar Descontos Pós Compra):**

Pré Condições:

1 - Gerente deve estar logado no sistema.
2 - Existem vendas com descontos concedidos por funcionários.

Cenário Típico:

1 - Gerente clica em “Verificar Descontos”.

2 - Sistema apresenta uma lista de transações que ofereceram desconto.
3 - Gerente seleciona uma transação.
4 - Sistema exibe detalhes da transação selecionada.
5 - Gerente verifica se o desconto concedido foi aplicado corretamente pelo funcionário que realizou a venda.

Cenários Alternativos:

1 - Não foram concedidos descontos nas compras da concessionária no mês vigente:
    1.1 - Servidor exibe mensagem informando que nenhum desconto foi registrado no sistema.

**CS 16 (Dar Bonificação para Funcionário):**

Pré Condições:

1 - Gerente deve estar logado no sistema.

Cenário Típico:

1 - Gerente clica em “Listar Funcionários”.
2 - Gerente clica no funcionário que deseja bonificar.
3 - Gerente insere o valor da bonificação.
4 - Gerente confirma bonificação ao funcionário selecionado.
5 - Sistema informa que a bonificação foi realizada com sucesso.

Cenários Alternativos:

1 - Valor de bonificação inserido pelo gerente não segue os parâmetros válidos.
    1.1 - Servidor exibe mensagem de erro e solicita novamente o valor da bonificação.
    1.2 - Servidor exibe mensagem de erro e informa que o funcionário selecionado já recebeu a bonificação.

**CS 17 (Fazer Logout):**

Cenário Típico:

1 - Usuário clica no seu nome.
2 - Usuário clica em Fazer Logout.

Pré Condições:

1 - Usuário deve estar logado no sistema.

**CS 18 (Editar veículo no sistema):**

Cenário Típico:

1 - Funcionário clica em Editar veículo.
2 - Funcionário procura veículo.
3 - Funcionário atualiza dados do veículo.
4 - Funcionário clica em Confirmar.

Cenários Alternativos:

1ª - Dados do veículo editado não segue os parâmetros válidos:
    1 - Servidor exibe mensagem de erro e solicita a informação válida.

Pré Condições:

1 - Funcionário deve estar logado no sistema.

## 3.3 Estilos Arquiteturais Utilizados e Justificativas de Uso

Em virtude do desenvolvimento de um sistema de concessionária, optamos por adotar o padrão MVC, visto que ele contribui para a separação de responsabilidades, já que os componentes são separados e isso facilita a manutenção e torna o sistema mais organizado e adaptável a mudanças. Além disso, temos a possibilidade de reutilizar os componentes, pois temos a possibilidade de reutilizar cada parte em diferentes partes do sistema ou interfaces de usuário que julgarmos necessárias ao projeto.

Outro fator importante é que sendo um estilo arquitetural independente, podemos trabalhar em paralelo no desenvolvimento de funcionalidades para o sistema sem que haja uma interdependência entre as atividades da equipe. Com isso, podemos adotar uma dinâmica mais eficaz para o desenvolvimento do projeto, além de oferecer a possibilidade de incluir novos recursos, atualizações na interface do cliente e novas mudanças com mais facilidade do que em outros tipos de padrão arquitetural.

## 3.4 Descrição de como a Arquitetura de Software contribui para os atributos de qualidade importantes do sistema

A Arquitetura de Software da AutoPrime contribui significativamente para a manutenibilidade e extensibilidade do sistema, pois a separação entre os

componentes permite que nosso time de desenvolvimento possa trabalhar em reparos, atualizações ou mudanças de forma independente. Além disso, outro atributo de qualidade do sistema que é aproveitado pelo uso da arquitetura da AutoPrime é o de escalabilidade, visto que a possibilidade de reutilização de componentes faz com que nosso sistema possa ser expandido, melhorado e possa se adaptar conforme as necessidades e mudanças do setor se caracterizando por ser um sistema de alta modificabilidade.

### 3.5 Modelo Conceitual e Diagramas de Casos de Uso, Sequência, Interação, Sistema e Classe

Todos os diagramas desenvolvidos para o projeto da concessionária AutoPrime foram enviados via anexo no classroom, links e no GitHub Classroom visto que a visualização dos diagramas no docs fica prejudicada.

# 4- Padrões de Projeto (SOLID e GRASP)

## 4.1 SOLID:

### 4.1.1 Single Responsibility - Arthur:

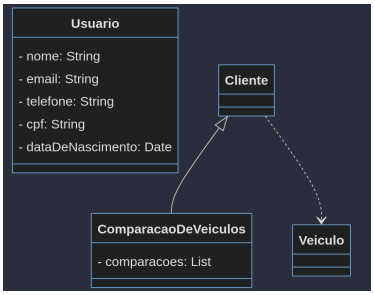
Apesar de algumas dependências que aumentam o acoplamento do sistema, o princípio de Responsabilidade Única é mantido. Classes do sistema ou operam sobre seus próprios dados, ou são gerenciadores com o propósito único de realizar operações sobre instâncias de outras classes.

### 4.1.2 Open-Closed - Gabriel:

A implementação original utilizava o princípio Open Closed, porém a partir de uma segunda análise, e seguindo o padrão de design decorator, a ideia da implementação das ações validadas pelo gerente foram mudadas para englobar diferentes dos estilos de gerente que a implementam, sendo assim, aberto para mudanças e fechado quando se trata de necessitar de uma implementação, por objetos externos.

### 4.1.3 Liskov Substitution - Felipe:

O princípio de **Substituição de Liskov (LSP)** define que uma subclasse deve ser substituível por sua superclasse em qualquer contexto. Na nossa entrega 1, a classe Cliente violou o princípio de Liskov porque ela não pode ser usada em qualquer lugar onde uma instância de Usuário seja esperada, isso ocorre porque na classe Cliente, temos um atributo adicional, o **ComparacoesSalvas**, que não está presente na classe **Usuario**. Com isso, podemos corrigir essa violação da seguinte forma: Podemos separar o atributo **ComparacoesSalvas** da classe Cliente e colocá-lo em uma classe própria. Essa nova classe, que chamaremos de **ComparacaoDeVeiculos**, será um **Decorator** de Usuário, o que significa que ela adiciona novos comportamentos a uma instância de Usuário. Dessa forma, uma instância de Cliente pode ser usada em qualquer lugar onde uma instância de Usuário seja esperada, sem violar o princípio da substituição de Liskov.




### 4.1.4 Interface Segregation - Guilherme:

Esse princípio foi um dos que nós estávamos mais conscientes ao fazer nosso projeto,apenas atribuímos métodos para classes se elas os forem utilizar.As interfaces eram apenas criadas quando vimos que duas ou mais classes eram parecidas o suficiente e usadas pelo mesmo controller,os métodos atribuídos a ela eram apenas aqueles que ambas as classes tinham em comum,o mesmo mindset foi usado para heranças.

### 4.1.5 Dependency Inversion - Victor:

O método cadastrar() em GerenciadorDeRegistro pode se aplicar tanto para funcionário como para cliente, dependendo da abstração, assim implementando esse padrão SOLID.

## 4.2 GRASP:

### 4.2.1 Especialista (Expert) - Arthur:

O padrão Especialista propositalmente não é priorizado no projeto. Em vez disso, como mencionado no padrão de Responsabilidade Única, a maior parte da lógica e de operações realizadas com dados ficam a cargo de gerenciadores especializados.

### 4.2.2 Criador (Creator) - Victor:

A classe GerenciadorDeRegistro cria instâncias das classes Cliente e Funcionário, uma vez que essa classe contém os valores iniciais das classes criadas respeitando o padrão Creator, da mesma forma SistemaDePersonalizacao cria uma instância de RegistroDePersonalizacao e usa essa classe, sendo também coerente com o padrão.

### 4.2.3 Coesão Alta (High Cohesion) - Gabriel:

O conceito da alta coesão é utilizado em todos os lugares ao longo do projeto. Todas as classes que estavam agregando funcionalidades demasiadas foram divididas em classes menores, sempre que possível.

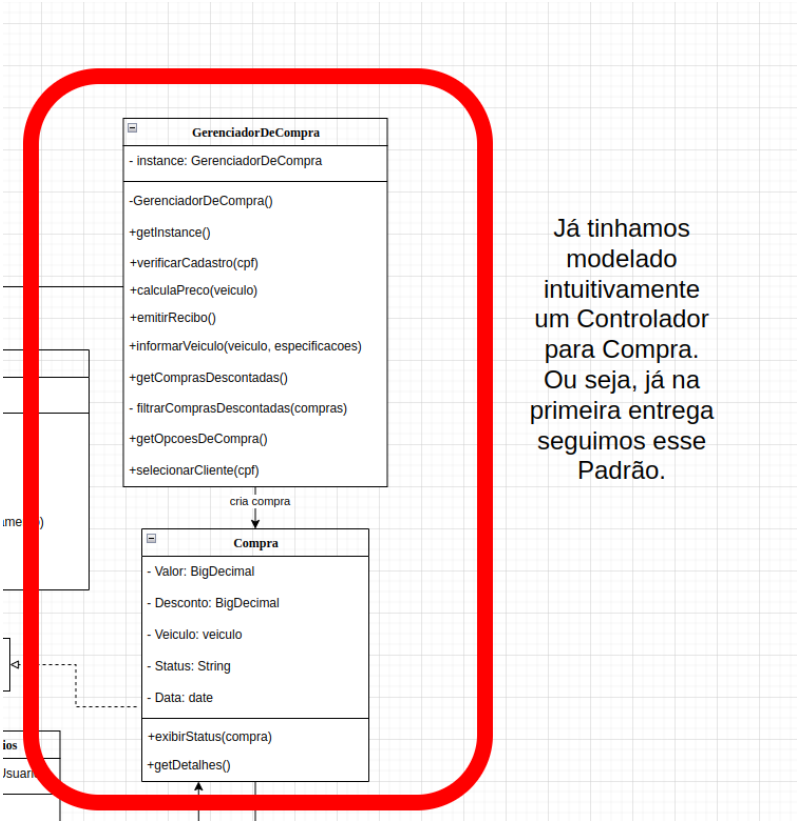
### 4.2.4 Acoplamento fraco (Low Coupling) - Guilherme:

Quando nós a criarmos os diagramas,pensamos muito em quem implementaria cada passo de cada caso de uso.O conceito do acoplamento fraco não é muito compatível com o padrão MVC usado em nosso projeto,as classes Controller(ex: GerenciadorDeRegistro,GerenciadorDeLogin,etc) acabam tendo muitas responsabilidades e fazem com que as outras classes sejam dependentes delas,porém criamos diversas classes Controller para serem responsáveis de apenas um aspecto do sistema(usuário,registro,login,compra,etc) para termos um controle e visibilidade melhor do código como um todo e facilitar a manutenção,refatoração.

### 4.2.5 Controlador (Controller) - Felipe:

A classe **GerenciadorDeCompra** pode ser considerada um Controller, pois é uma classe responsável por receber solicitações do usuário e processá-las. No sistema da AutoPrime, o **GerenciadorDeCompra** recebe solicitações de compra de veículos. Ele então verifica se o cliente está cadastrado, calcula o preço do veículo, emite o recibo e atualiza o estoque.

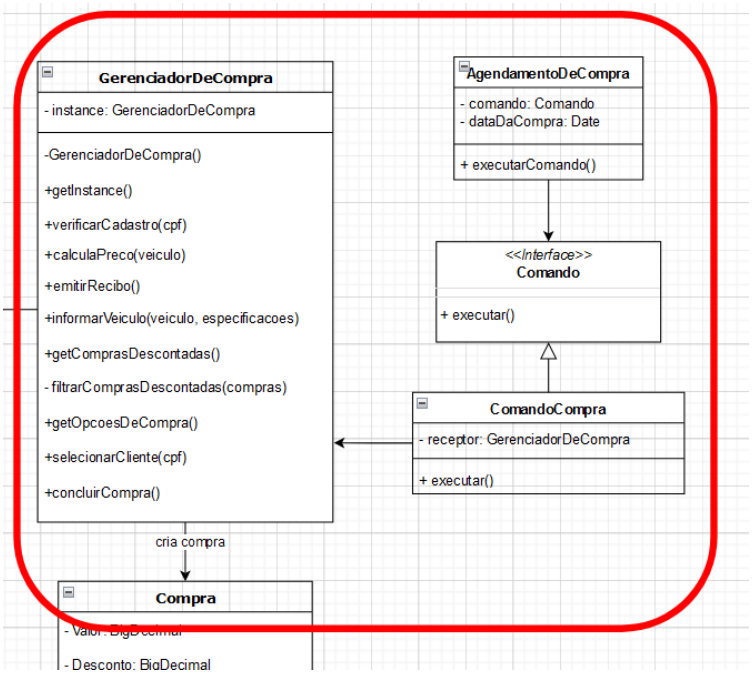
Os benefícios de usar um Controller nesse caso, são: Fornecer uma interface única para gerenciar todas as etapas do processo de compra, desde a verificação do cadastro do cliente até a emissão do recibo, além de melhorar a flexibilidade do sistema. O **GerenciadorDeCompra** pode ser facilmente adaptado para atender a novos requisitos de negócios futuramente.




# 5- Padrões de Projeto (GoF)

## 5.1 Command - Arthur:

O padrão Command pode ser utilizado para agendar ou adiar uma compra, caso um cliente tenha a intenção de deixar um veículo reservado e concluir a compra apenas em um momento posterior. Nesse caso, uma classe Comando deve ser adicionada, e a classe GerenciadorDeCompra teria de ser modificada para poder transformar uma chamada do método usado para concluir uma compra em um objeto serializável.




## 5.2 Decorator - Gabriel:

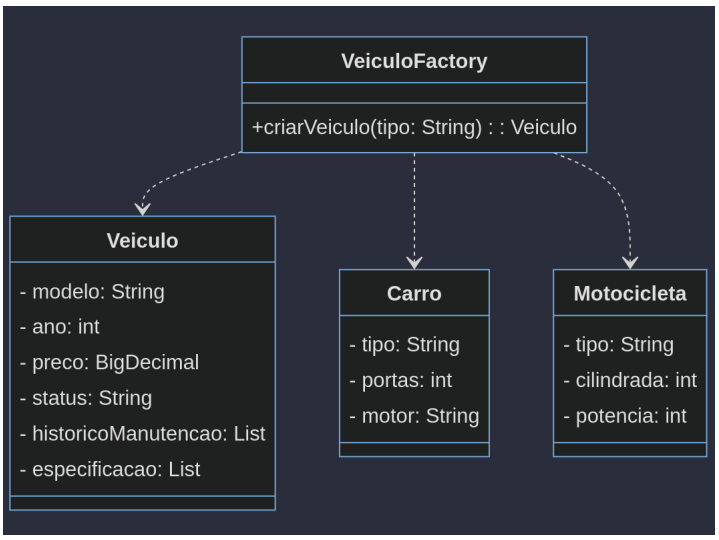
O decorator foi utilizado na classe Gerente como uma forma de adição de permissões para as possíveis extensões do que cada um dos gerentes pode fazer. Imaginando que nem todos os gerentes terão permissões para fazer cada uma das ações como adicionar novos usuários ou dar bonificações.

## 5.3 Singleton - Guilherme:

O padrão singleton foi utilizado nas classes Gerenciador(diversas) e SistemaDePersonalizacao(que controlam outras classes/dados) e as classes Estoque e Historico(que fazem o controle entre a aplicação e o banco de dados),pois essas classes não precisam ter mais de uma instância e elas controlam a criação/modificação de outras classes/tabelas do banco de dados,logo,houve um aumento no desempenho do aplicativo e o controle sobre as instâncias das classes dependentes a elas.

## 5.4 Factory Method - Felipe:

O padrão Factory Method pode ser usado para melhorar o design do nosso sistema de concessionária, tornando-o mais flexível e reutilizável. Uma maneira de aplicar o padrão Factory Method no nosso sistema é criar uma classe **VeiculoFactory** que seja responsável por criar instâncias de **Veiculo**. Essa classe teria um método **criarVeiculo** que receberia um parâmetro com o tipo de veículo que deve ser criado.




Com essa mudança, a criação de instâncias de **Veiculo** fica centralizada na classe **VeiculoFactory**. Isso torna o código mais flexível, pois permite que novos tipos de veículos sejam criados sem alterar o código de outras partes do sistema. Então, no caso de expansão do sistema da Concessionária, esse tipo de padrão beneficiaria os desenvolvedores do sistema.

### 5.5 Observer - Victor:

O padrão de projeto Observer será utilizado com o objetivo de notificar ao cliente interessado quando o carro que ele estava observando para comprar ficar disponível, para isso será necessário adicionar classes Publicadora e Assinante, com a Publicadora contendo uma lista desses assinantes (clientes interessados em receber a notificação) para enviar a notificação quando tiver a disponibilidade do carro e assim os clientes da AutoPrime serem melhor atendidos nas suas preferências.

## 6- Política de Colaboração do Grupo

- **Felipe Mendes Salles:** Descrição do Escopo do Sistema; Definição dos padrões arquiteturais adotados; Justificativa das decisões arquiteturais adotadas com base nos atributos de qualidade do sistema; Casos de Uso; Diagrama com a visão geral do sistema (organização), incluindo os estilos arquiteturais utilizados e justificativas de uso; Descrição de como a arquitetura de software contribui para os atributos de qualidade importantes do sistema; Diagrama de Classe; Padrão Liskov Substitution, Padrão Controller, Padrão GoF - Factory Method; Implementação de Cliente e Funcionário (Back-End e Front-End), LPS do Sistema.

- **Gabriel Tavares Gargur:** Diagramas de Interação; Requisitos Arquiteturais; Diagrama de Classe, Configuração do Projeto, Open-Closed; Coesão Alta (High Cohesion); Decorator; Implementação de Agendamentos (Back-End e Front-End), Resolução de Bugs de integrantes.

- **Guilherme França Moreira:** Diagramas de Sequência; Diagrama de Casos de Uso; Requisitos Arquiteturais; Diagrama de Classe, Interface Segregation; Acoplamento fraco (Low Coupling); Singleton; Implementação do Back-End e Front-End de Estoque e Veículos, além de aprimoramento do Back-End de Compras e o Front-End de Compras.

- **Pedro Arthur Nascimento Resende:** Diagramas de Sequência; Diagrama de Casos de Uso; Requisitos Arquiteturais; Diagrama de Classe; Modelo Conceitual; Single Responsibility; Especialista (Expert); Command; Implementação do Back-End de Compra.

- **Victor Miranda Florido:** Descrição do Escopo do Sistema; Definição dos padrões arquiteturais adotados; Justificativa das decisões arquiteturais adotadas com base nos atributos de qualidade do sistema; Casos de Uso; Descrição de como a arquitetura de software contribui para os atributos de qualidade importantes do sistema; Diagrama de Classe; Dependency Inversion; Criador (Creator); Observer; Implementação das funções de Funcionário, Main, Gerente e criação de classes.