Aplicações Web modulares e plugáveis

Ampliando o uso de eventos

QUEM É ESSE CARA?

Só um cara :)

Naylon Kessler de Aquino, Analista/Desenvolvedor a 17 anos.

Naylon Kessler de Aquino

Consultor em Web

www.naylonkessler.com

ANTES DE COMEÇARMOS

- Não acredite em nada do que eu te disser;
- Não se limite;
- Extrapole os conceitos;
- Não seja “radical”;
- Use a imaginação.

DESAFIOS DO DESENVOLVIMENTO WEB

- A própria Web;
- Sensibilidade a mudanças;
- Mudanças nos requisitos e regras de negócio;
- Manutenibilidade x escalabilidade x flexibilidade.

CONSIDERAÇÕES SOBRE ARQUITETURA

O PROBLEMA DA SUPERVALORIZAÇÃO

Arquitetura é uma palavra que usamos quando queremos falar sobre o desenho de um projeto mas queremos “inflá-lo” para fazer parecer importante.

(FOWLER, 2003, tradução nossa)

CERTO X ERRADO

Não existe arquitetura certa ou errada, existem arquiteturas adequadas ou não, quando analisados o contexto e os requisitos.

- Não use *;
- Não faça *;
- * está morto.

É EVOLUTIVA

Nenhuma arquitetura e fixa ou imutável.

Na maioria das vezes uma arquitetura é composta por outras menores.

DESIGN PATTERNS E ORIENTAÇÃO A OBJETOS

Design patterns nada mais são que a aplicação dos conceitos da orientação a objetos de maneira inteligente. Não são divindades :)

Quanto maior sua capacidade de abstração melhor será a aplicação da OO e o uso de design patterns.

CONCEITOS

APLICAÇÕES MODULARES X PLUGÁVEIS

Aplicações modulares e aplicações plugáveis não são as mesmas coisas, embora muito similares.

Por definição toda aplicação plugável é modular mas não o contrário.

MÓDULO

Módulo é como uma unidade de software implantável, gerenciável, nativamente reusável, combinável e sem estado que provê uma interface concisa aos consumidores.

(KNOERNSCHILD, 2012, tradução nossa)

MÓDULO PLUGÁVEL

É um módulo que pode ser instalado e desinstalado sem a necessidade de recompilar a aplicação, isto é, são “plug-and-play”, uma vez instalados podem ser utilizados de maneira transparente em conjunto com a aplicação.

MODULARIZAÇÃO

Modularizar uma aplicação significa quebrá-la em partes menores no sentido de melhor gerenciar suas complexidades e atribuições.

A ausência de um determinado módulo não deve comprometer em quaisquer aspectos o funcionamento da aplicação como um todo.

FROZEN SPOTS E HOT SPOTS

São dois conceitos originalmente utilizados na concepção de frameworks.

FROZEN SPOT

Parte de um arcabouço de software que não pode ser alterada. Ela contém as funcionalidades básicas e protocolos de comunicação de uma aplicação, é fixa. Pode ser visto como o núcleo da aplicação.

HOT SPOT

Partes de um arcabouço de software que podem ser estendidos e alterados.

Na prática “hot spots” se ligam a um “frozen spot” provendo pontos de extensão.

ESTRATÉGIAS

- Orientação a objetos;
- Arquitetura orientada a serviços;
- Microserviços;
- Arquitetura orientada a eventos.

ORIENTAÇÃO A OBJETOS

O **bom uso** dos conceitos da orientação a objetos pode reduzir os problemas de uma arquitetura monolítica, permitir uma melhor modularização, **elevar a coesão** e **diminuir o acoplamento**.

ORIENTAÇÃO A OBJETOS

No desenho de aplicações modulares “plugáveis” os conceitos de **abstração**, **encapsulamento** e **polimorfismo** são amplamente empregados através da utilização de **interfaces**, **talvez** os componentes mais importantes da modularização de sistemas.

ORIENTAÇÃO A OBJETOS

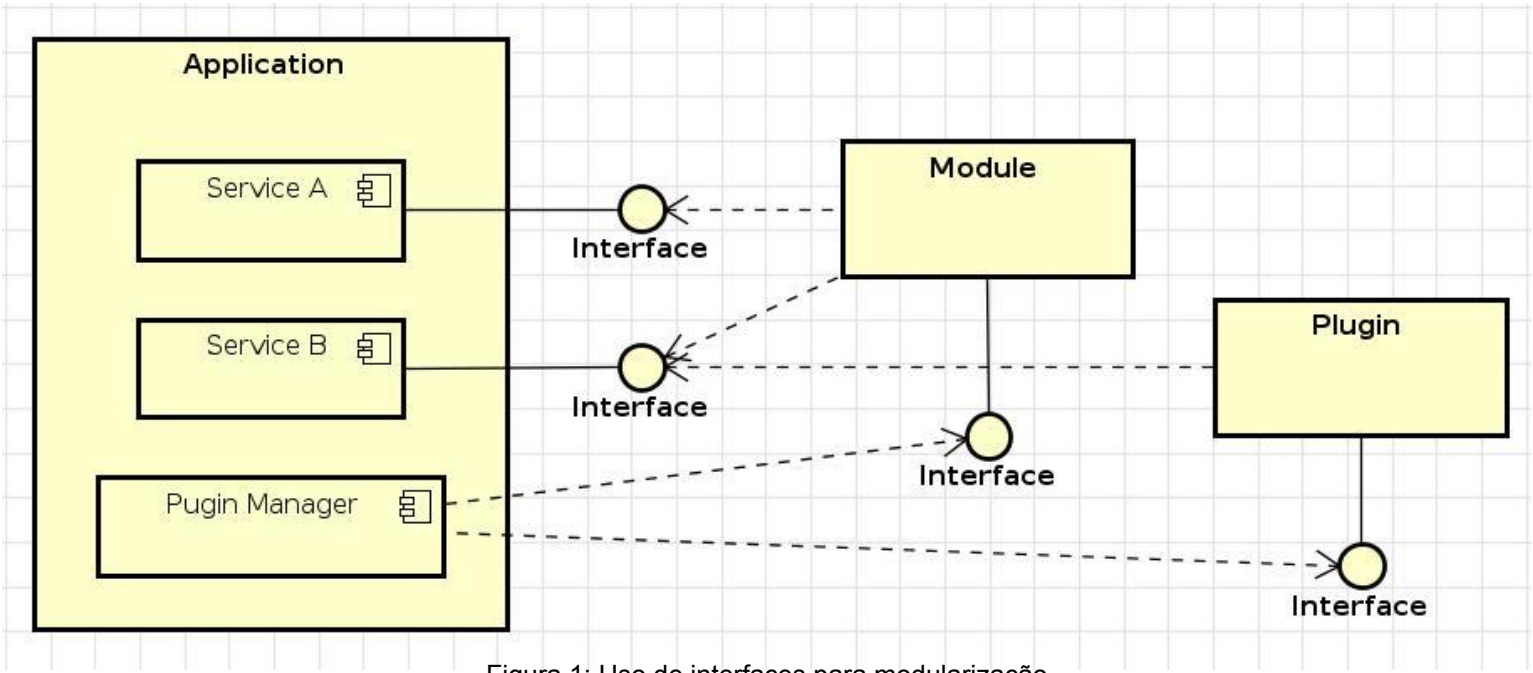

Figura 1: Uso de interfaces para modularização
Fonte: Elaborado pelo autor

ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

É uma disciplina e paradigma de design de sistemas que ajuda a TI a melhor atender demandas de negócios.

Focada em como diferentes funções (demandas) de negócios em um sistema interagem entre si.

ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

- Demandas são criadas, disponibilizadas e consumidas como serviços;
- Abrange como publicar, descobrir e consumir serviços;
- Utiliza mais comumente o protocolo SOAP;
- Foco na descentralização de funções.

MICROSERVIÇOS

Newman define microserviços como serviços pequenos e autônomos que trabalham juntos. (NEWMAN, 2015, tradução nossa)

MICROSERVIÇOS

- Seguem o “Single Responsability Principle” (S de SOLID);
- Focados em realizar funções pequenas, mas muito bem definidas e implementadas;
- Independentes entre si tanto do ponto de vista funcional quanto técnico.

ARQUITETURA ORIENTADA A SERVIÇOS

É um arcabouço de software que permite a produção, detecção e reação a eventos.

EVENTO

Mudança significativa de estado de um componente de software ou entidade da aplicação.

Uma sinalização de determinado estado ou ação em um componente de software ou entidade da aplicação.

OBSERVER

Tem por objetivo estabelecer uma ligação de um para muitos entre objetos de forma que ao mudar de estado um objeto notifica e atualiza todos os outros objetos dependentes automaticamente.

É um dos padrões mais utilizados com diferentes implementações.

SINGLE OBJECT

- Qualquer objeto pode disparar e escutar eventos especializando uma classe base ou implementando de uma interface adequada;
- Eventos identificados por strings;
- Certeza: eventos escutados serão disparados no target;
- Pode usar eventos arbitrários.

SINGLE OBJECT

```
$object->addEventListener('event.name', $handler);
$object->dispatchEvent('event.name', $param1, $param2);
$object->removeEventListener('event.name', $handler);
```

Figura 2: Uso da implementação single object
Fonte: Elaborado pelo autor

PUB/SUB

- Faz uso de um objeto centralizado para fazer o disparo e escuta de eventos;
- Qualquer objeto pode publicar e assinar para qualquer tipo de evento;
- Não existem relacionamentos entre a mensagem e o objeto target.

PUB/SUB

```
$bus->subscribe('event.name', $handler);
$bus->publish('event.name', $param1, $param2);
$bus->unsubscribe('event.name', $handler);
```

Figura 3: Uso da implementação pub/sub
Fonte: Elaborado pelo autor

SIGNALING

- Cada tipo de evento tem seu próprio controlador;
- Cada tipo de evento faz o disparo e a escuta de si;
- Não são identificados por seus nomes;
- Não podem publicar e assinar para eventos arbitrários, já que detêm comportamentos específicos.

SIGNALING

```php
$object->eventName->add($handler);
$object->eventName->dispatch($param1, $param2);
$object->eventName->remove($handler);
```

Figura 4: Uso da implementação signaling
Fonte: Elaborado pelo autor

PROTÓTIPO FUNCIONAL

Simples implementação realizada para prova de conceito.

EM LINHAS GERAIS

Foi experimentado como eventos poderiam ser explorados e expandidos para uso na modularização e “plugabilidade”.

Basicamente usar eventos de uma forma não trivial, como estados de leitura e estados de renderização.

CARACTERÍSTICAS

- Um núcleo: Base da arquitetura;
- Funcionalidades base: Controle de Contatos (frozen spot);
- Dois módulos plugáveis: Controle de Vendas e Produtos;

DESIGN

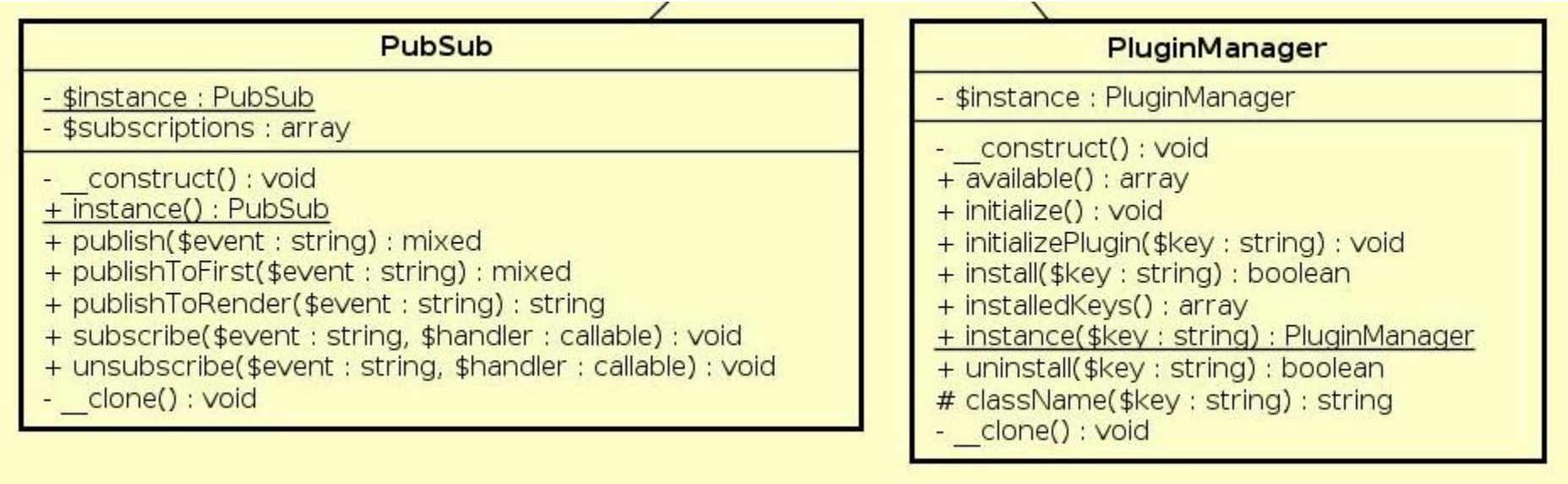

Figura 5: Principais peças do design - PubSub e PluginManager
Fonte: Elaborado pelo autor

APP DESIGN

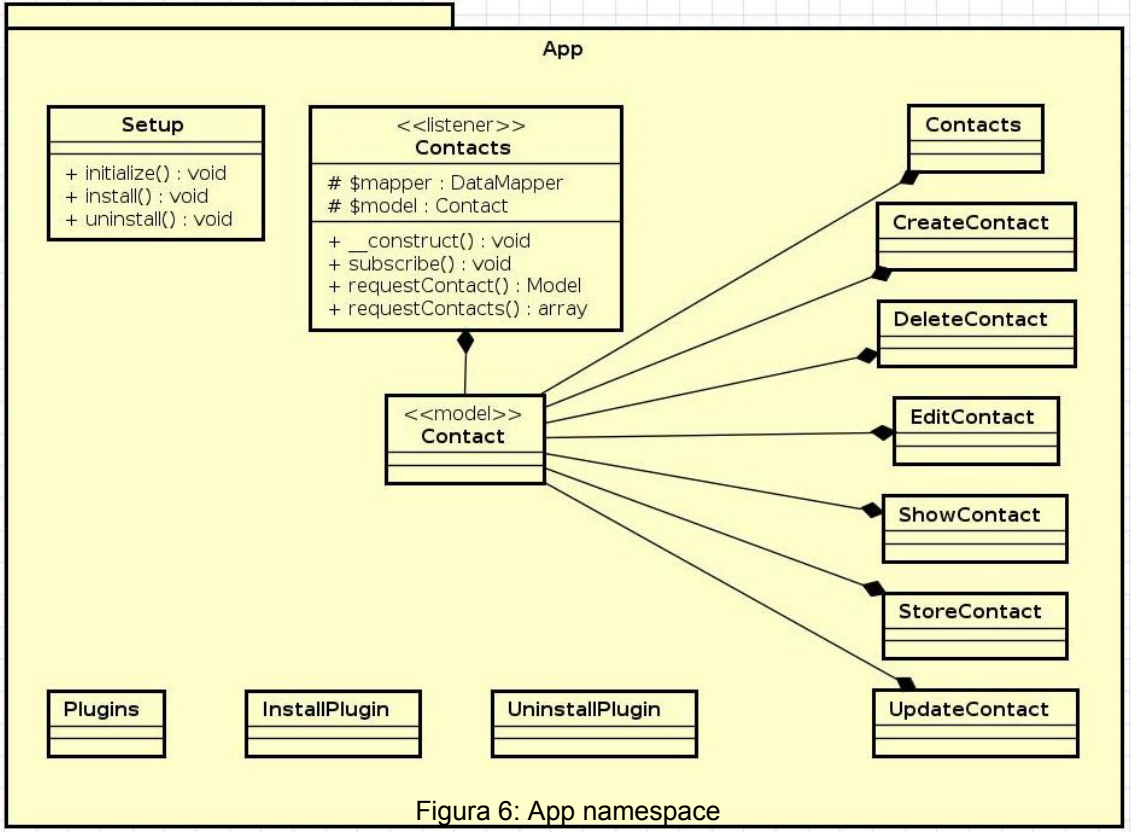

Figura 6: App namespace
Fonte: Elaborado pelo autor

SALES MODULE DESIGN

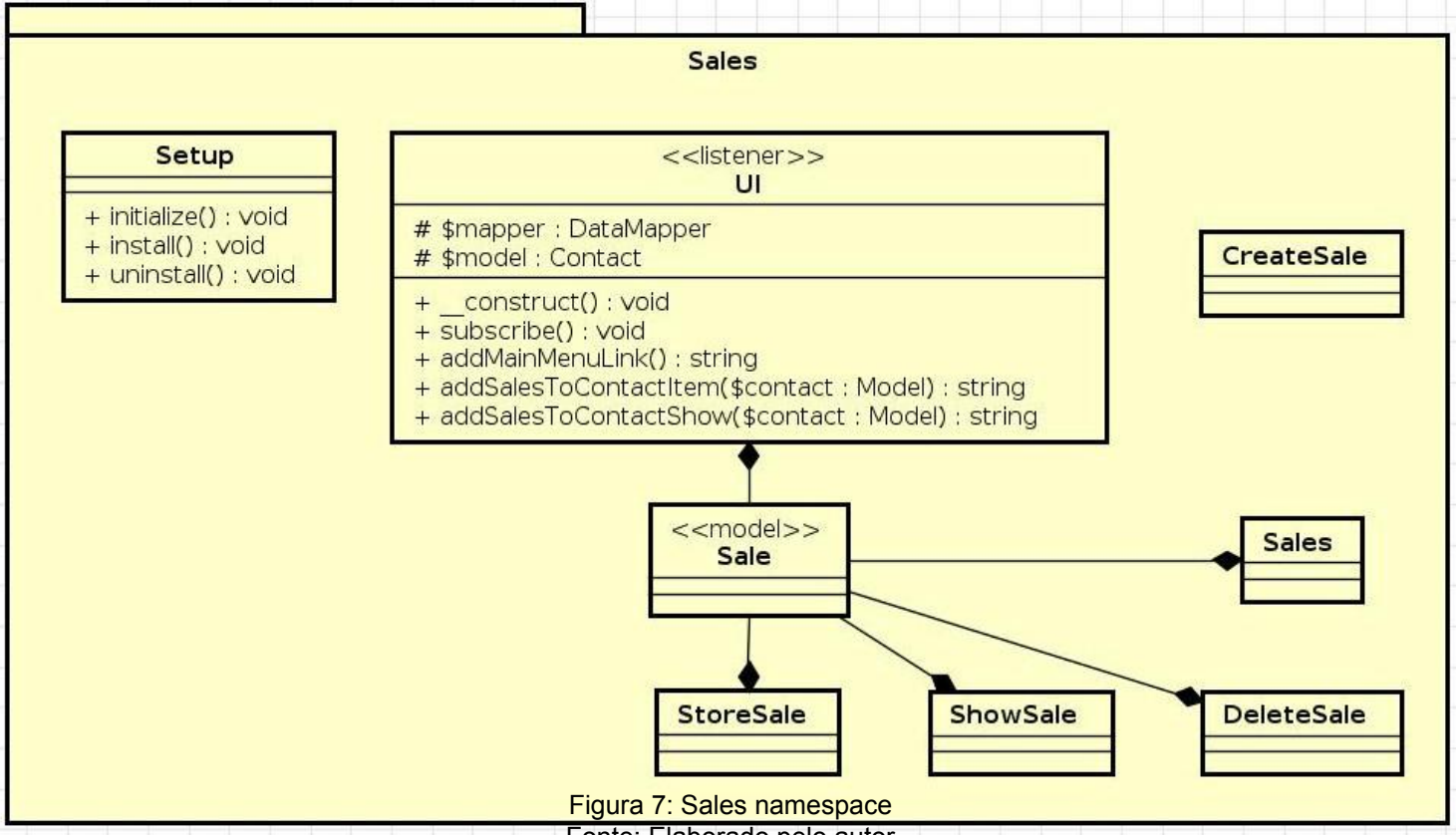

Figura 7: Sales namespace
Fonte: Elaborado pelo autor

PRODUCTS MODULE DESIGN

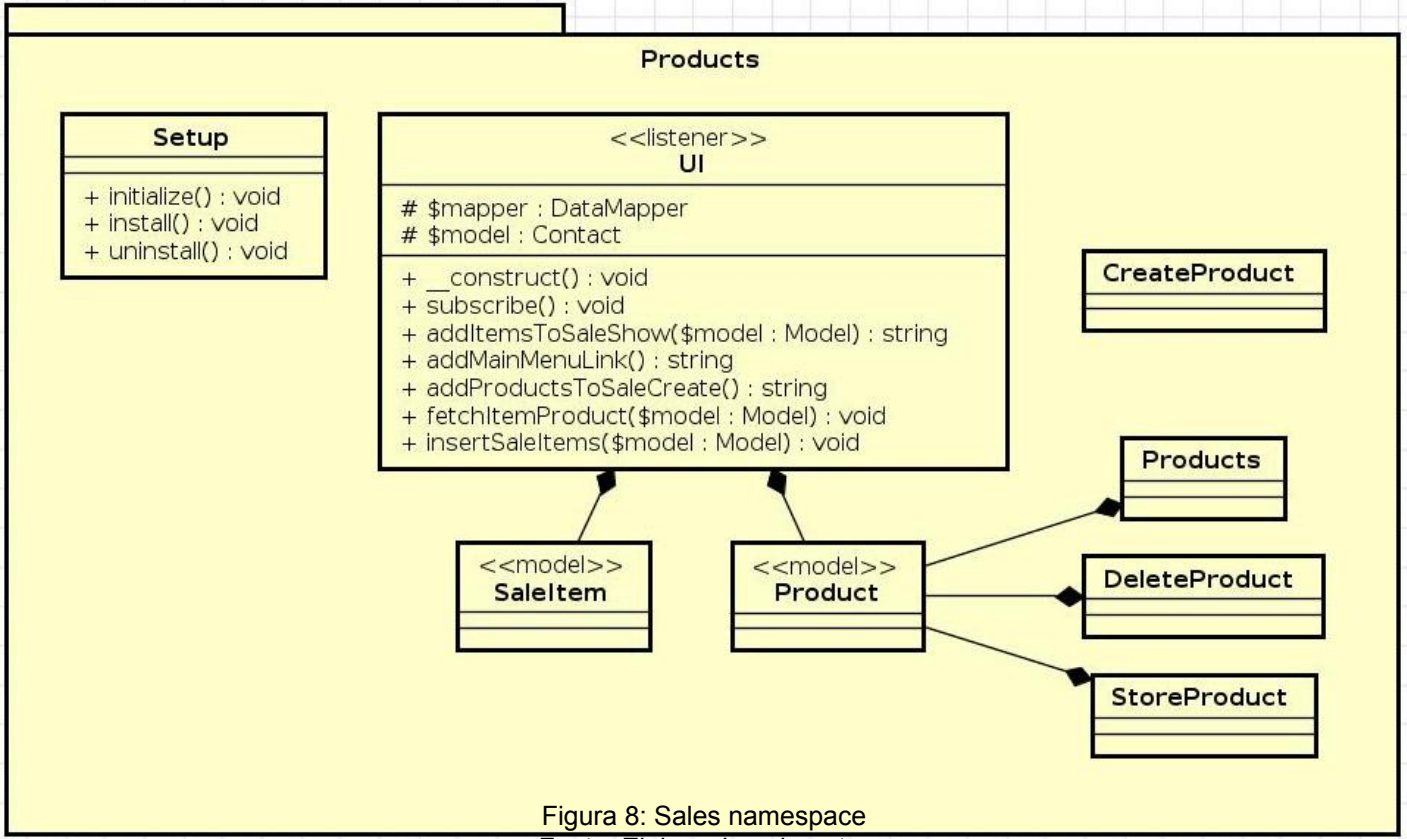

Figura 8: Sales namespace
Fonte: Elaborado pelo autor

EM FUNCIONAMENTO

REFERÊNCIAS

BAERBAK CHRISTENSEN, Henrik. Flexible, Reliable Software: Using Patterns and Agile Development. 1.ed. New York: Chapman and Hall/CRC, 2010

FOWLER, Martin. Who needs an architect?. IEEE Software, v.20, p.11-13, Sept.-Oct. 2003.

GAMMA, E.; JOHNSON, R.; HELM, R.; VLISSIDES, J. Padrões de Projetos: Soluções Reutilizáveis. Porto Alegre: Bookman, 2006.

KNOERNSCHILD, Kirk. Java Application Architecture: Modularity Patterns with Examples Using OSGi. 1.ed. Indiana: Prentice Hall, 2015

NEWMAN, Sam. Building Microservices. 1.ed. California: O'Reilly Media, 2015.

WOLFGANG, Pree. Design Patterns for Object-oriented Software Development. 1.ed. Addison-Wesley, 1994

OBRIGADO!

Naylon Kessler de Aquino

www.naylonkessler.com.br

naylon.kessler@gmail.com

(31) 98787-3384 Oi

(31) 97506-5665 Tim (whats)