

Arquitectura de Software

Mestrado em Engenharia Informática

Assignment 2

Report

Equipa

Carlos Santos	2009108991	carlosms@student.dei.uc.pt
Gonçalo Pereira	2009111643	gsp@student.dei.uc.pt
Marco Pereira	2009114979	macsp@student.dei.uc.pt

Departamento de Engenharia Informática

FACULDADE DE CIÊNCIAS E TECNOLOGIA

UNIVERSIDADE DE COIMBRA

Março de 2015

Conteúdo

1	Análise	3
1.1	Quais são os <i>drivers</i> arquitecturais do sistema e quais são as suas prioridades relativas?	3
1.1.1	Requisitos Funcionais	3
1.1.2	Atributos de Qualidade	4
1.1.3	Restrições	5
1.1.4	Prioridades	5
1.2	Quais os factores que motivaram as alterações do sistema? Quais os que influenciaram as suas decisões de <i>design</i> ?	5
1.2.1	Utilização de <i>Java EE</i>	6
1.3	Descrição da arquitectura do sistema modificado	6
1.3.1	Vistas	8
1.4	Estrutura geral do sistema modificado	8
1.5	Em que medida a estrutura geral do nosso sistema e as nossas escolhas de <i>design</i> suportam os objectivos do negócio do sistema, no que diz respeito aos drivers arquitecturais identificados na questão 1.1.	9
1.5.1	Comparação e escolha da arquitectura	9
1.5.2	Atributos de Qualidade	11
1.5.3	Requisitos Funcionais	11
1.6	<i>Tradeoffs</i> realizados na concepção do sistema. Que outras alternativas teriam? O que vos fez optar pelas vossas soluções em detrimento de outras?	12
1.6.1	RMI vs EJBs vs <i>Web Services</i>	12
1.6.2	Disponibilidade vs Segurança	12
2	Plano de Implementação	13
3	Conclusão	13
4	Tabelas	14
5	Diagramas e Vistas	20
	Bibliografia	32

Lista de Figuras

1	<i>Diagrama de casos de uso.</i>	3
2	<i>Arquitetura.</i>	7
3	<i>Vista de Módulos - Vista de Composição.</i>	20
4	<i>Vista de Componente e Conector - Shared Data.</i>	21
5	<i>Diagrama de alocação.</i>	22
6	<i>Diagrama de sequência - Login.</i>	23
7	<i>Diagrama de sequência - Orders.</i>	24
8	<i>Architecture Life Cycle.</i>	25
9	<i>Plain RMI vs RMI in ESB.¹</i>	26
10	<i>Layout do login.</i>	27
11	<i>Layout dos produtos em stock.</i>	28
12	<i>Layout do menu do Admin.</i>	29
13	<i>Exemplo de ficheiro de logs relativamente ao Login e Logout.</i>	30
14	<i>Data Model View.</i>	31

Lista de Tabelas

1	Caso de uso “Login”	14
2	Caso de uso “Adicionar encomendas no OrdersWebApp”	15
3	Caso de uso “Logging”	15
4	Caso de uso “ManagementWebApp”	16
5	Caso de uso “OrdersApp” (já existente)	16
6	Caso de uso “ShippingApp” (já existente)	16
7	Caso de uso “InventoryApp” (já existente)	17
8	Cenário de disponibilidade	17
9	Cenário de segurança	17
10	Cenário de consistência	18
11	Cenário de extensibilidade	18
12	Cenário de escalabilidade	19

1 Análise

1.1 Quais são os *drivers* arquitecturais do sistema e quais são as suas prioridades relativas?

Os *drivers* arquitecturais são todos os requisitos funcionais, atributos de qualidade e restrições que existem num projecto. Os **requisitos funcionais** descrevem de uma forma geral o que o sistema tem de fazer, obrigatoriamente; os **atributos de qualidade** são os requisitos não funcionais que o sistema tem de satisfazer; as **restrições** técnicas e/ou de negócio são todas as decisões ou condições já existentes que de alguma maneira irão restringir as opções de arquitectura.

1.1.1 Requisitos Funcionais

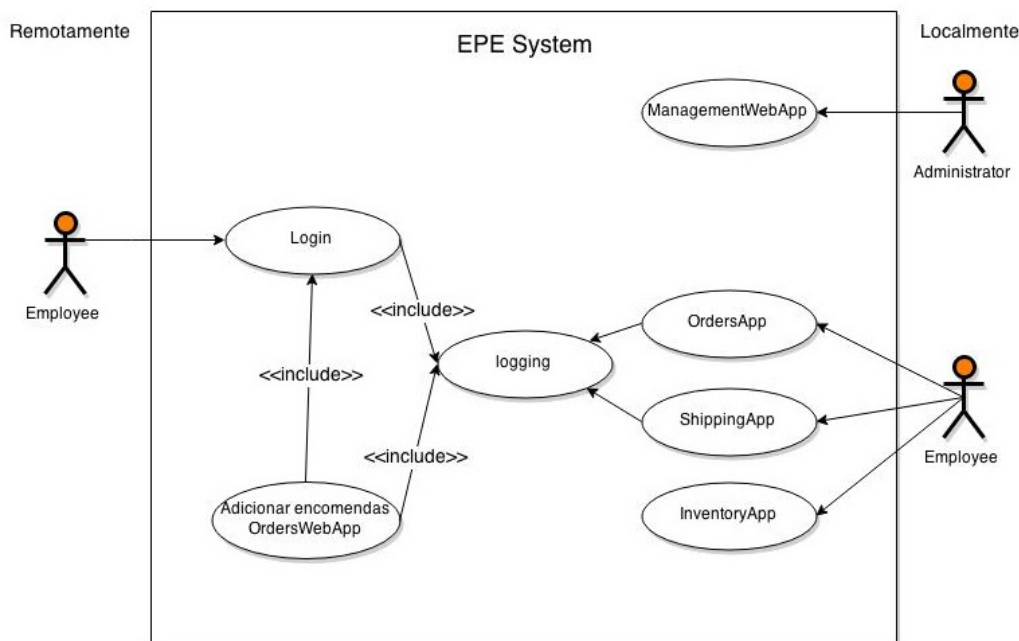


Figura 1: Diagrama de casos de uso.

Para este projecto os requisitos funcionais consistem em:

- **RF1** - criar um modelo de *logging* para manter registo de:

- *log in* e *log out*;
 - introdução de novas encomendas;
 - expedição de encomendas.
- **RF2** - suportar a introdução remota de novas encomendas.

Na Figura 1, através do diagrama de casos de uso, é possível verificar mais pormenorizadamente tanto a especificação dos requisitos funcionais identificados acima, como também os actores envolvidos.

Foram identificados os casos de uso de *Login* (Tabela 1), Adicionar encomendas no OrdersWebApp (Tabela 2), *Logging* (Tabela 3), ManagementWebApp (Tabela 4), OrdersApp (Tabela 5), ShippingApp (Tabela 6) e InventoryApp (Tabela 7).

1.1.2 Atributos de Qualidade

Com base no documento fornecido, identificaram-se os seguintes atributos de qualidade, ordenados por importância:

1. Disponibilidade (Tabela 8);
2. Segurança, (Tabela 9);
3. Consistência, (Tabela 10);
4. Extensibilidade, (Tabela 11);
5. Escalabilidade, (Tabela 12).

Uma vez que a empresa se encontra num período de elevado crescimento e necessita de se expandir, mantendo o seu serviço constantemente activo (**disponibilidade** e **escalabilidade**), também necessita de medidas de **segurança**, de forma a controlar o acesso dos colaboradores (quando os mesmos não se encontram nas instalações centrais da empresa) e restringir acessos indevidos, situação que não estava prevista anteriormente. A criação de novas encomendas é feita através de dois meios diferentes (localmente ou remotamente), pelo que, o sistema deverá manter a **consistência** dos dados, garantindo que para pedidos iguais, se obtêm resultados iguais. Por fim, tendo em conta que o mesmo terá de ser mantido e poderá ser modificado no futuro, foi associado o atributo de **extensibilidade**.

1.1.3 Restrições

Para este projecto foram identificadas três restrições:

- **restrição técnica:** Uso de linguagem *Java*;
- **restrição técnica:** A integração deverá ser feita sem ocorrer tempo de *downtime*, mantendo o sistema antigo completamente funcional;
- **restrição de negócio:** integração do sistema desenvolvido com o sistema legado.

1.1.4 Prioridades

Considera-se que os requisitos funcionais têm menor influência na arquitectura, pois embora estes tenham que ser respeitados, os sistemas desenvolvidos devem respeitar e obedecer especificidades das restrições e dos atributos de qualidade a serem alcançados.

1.2 Quais os factores que motivaram as alterações do sistema? Quais os que influenciaram as suas decisões de *design*?

Foi necessário adicionar uma nova base de dados (**users**) para armazenar os dados dos funcionários de forma persistente. Para além disso, foram ainda criadas algumas restrições (nomeadamente a impossibilidade de existirem atributos nulos e a presença de chaves primárias), de modo a garantir a consistência dos dados.

Para dar resposta à escalabilidade é usado *Java RMI*, para minimizar o número de conexões realizadas à base de dados (accedida apenas localmente). O *Java RMI* permite suportar o crescimento da empresa, dado que não estamos perante uma aplicação complexa e de grandes dimensões. Esta opção foi tomada pela sua simplicidade de implementação, comparativamente com o uso de EJBs, que apesar de também se adequar, oferecendo segurança, gestão de transacções, entre outras características, será mais vocacionado para aplicações de maior porte/complexidade. Foi também necessário aumentar a segurança e para isso foi usado o protocolo *HTTPS* e os dados dos utilizadores também são cifrados com recurso a *MD5*.

Foi escolhido para servidor *web*, *Apache Tomcat 8*, por ser relativamente leve e facilmente configurável, o que leva a ser mais rápido comparado com outros servidores e por uma questão de familiaridade pelos *developers*.

Dividimos o nosso sistema em 3 camadas, de forma a permitir uma maior separação (*model*, *view* e *controller*) para garantir mais segurança, dado que cada camada só comunica com as camadas contíguas, não sendo possível a camada de apresentação comunicar directamente com a base de dados.

1.2.1 Utilização de *Java EE*

A plataforma *Java, Enterprise Edition* foi a utilizada para o desenvolvimento do projecto. Esta plataforma caracteriza-se pela adição de bibliotecas que fornecem funcionalidades para implementar *software Java* distribuído, tolerante a falhas e multi-camada.

1.3 Descrição da arquitectura do sistema modificado

Na Figura 2, podemos identificar os diversos componentes da arquitectura. O sistema legado, composto pelas bases de dados, **ShippingApp**, **InventoryApp** e **OrdersApp**, continuam presentes nas instalações centrais da EPE, sendo apenas acedidos localmente. De forma a dar resposta ao requisito funcional referente ao acesso remoto para o registo de novas encomendas, foi idealizado um servidor *Tomcat*, com interface *Java RMI*.

Com o objectivo de garantir uma maior disponibilidade do sistema, foram utilizados dois servidores, que comunicam entre si, através de *heartbeat* (baseado no protocolo UDP), mantendo desta forma um servidor no estado activo e outro no estado passivo. Caso o servidor no estado activo não responda a 3 pedidos de comunicação do passivo, o mesmo passa para o estado activo, reduzindo desta forma o *downtime* do sistema.

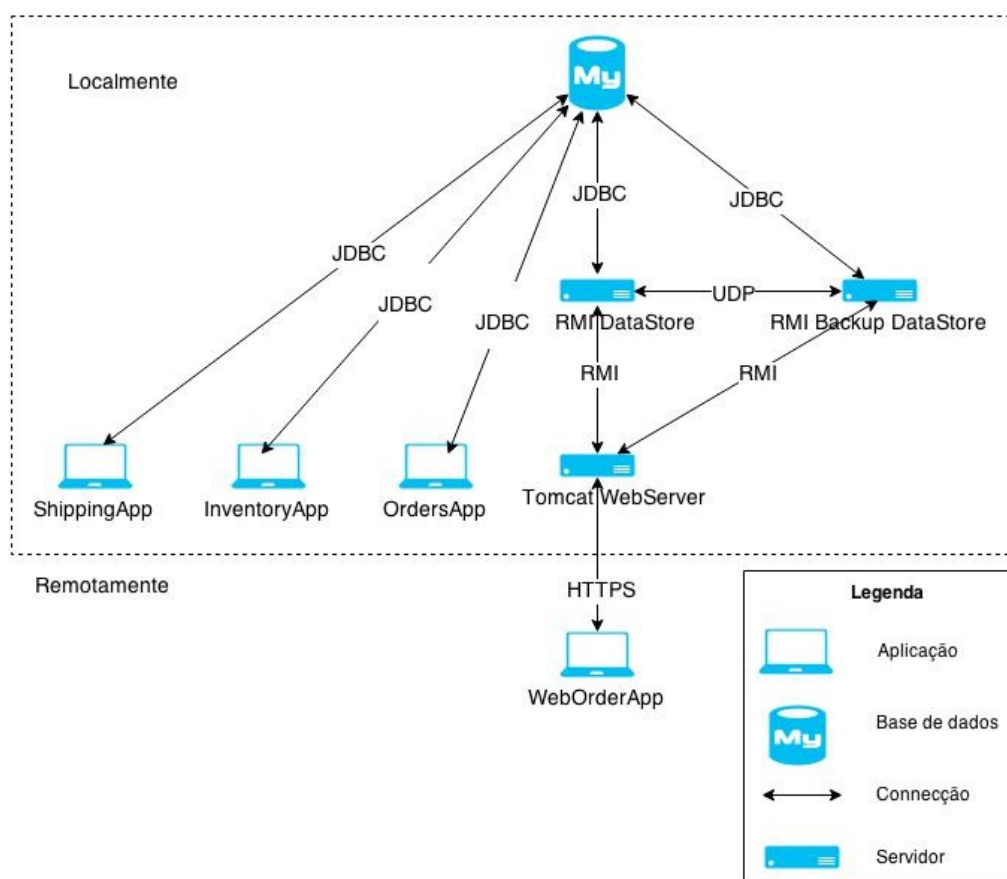


Figura 2: *Arquitectura.*

1.3.1 Vistas

De forma a escolher as vistas² apropriadas para esta arquitectura, é necessário identificar as partes interessadas e as suas necessidades.

Consideraram-se como os principais *Stakeholders* para este projecto: o administrador de TI, o cliente e os utilizadores finais, que vão aceder remotamente ou localmente ao serviço.

O cliente pagará pelo desenvolvimento do projecto, estando tipicamente interessado nos argumentos que reflectem o progresso do projecto, garantindo que a arquitectura e o sistema resultante responderão aos atributos de qualidade e aos requisitos funcionais. Normalmente, estão interessados em diagramas de *deployment* (Figura 5, para saber o suporte físico necessário, para alojar o projecto), análise de resultados (diagramas que provem que os requisitos são cumpridos) e diagramas de componentes-conectores (Figura 4) e/ou módulo de modo a ter uma visão geral do funcionamento do projecto.

Os utilizadores finais são a parte que efectivamente irá usar/trabalhar sobre o projecto e, normalmente, não precisa de saber a arquitectura uma vez que esta é praticamente invisível. No entanto, este utilizador pode encontrar discrepâncias no *design* da arquitectura, como tal, está mais interessado em ver o fluxo de informação do projecto (*inputs/outputs*).

O administrador de TI está normalmente encarregue da manutenção dos sistemas, estando assim, interessado numa vista de módulos (Figura 3, vista de decomposição) de modo a identificar onde será necessário introduzir as modificações pretendidas, provavelmente, numa vista de casos de uso para ter uma ideia do eventual impacto que essa modificação poderá causar e por fim uma vista do modelo de dados (Figura 14), de modo a saber o modo como a informação se encontra organizada.

Na vista de módulos podemos verificar que existem 3 camadas (*Presentation*, *Logic* e *Data*), e que a comunicação é feita entre as camadas *Presentation-Logic* e *Logic-Data*, e nunca entre a *Presentation-Data*.

1.4 Estrutura geral do sistema modificado

Este sistema usa uma arquitectura em 3 camadas com *Struts 2* e *Java RMI*, existindo interacção entre camadas contíguas. A **primeira camada**, destina-se à apresentação, dado que, é com ela que os funcionários interagem de forma a realizar encomendas remotamente, através de *web-pages*. Estas comunicam com a **segunda camada** (*business logic*), onde são tratados os dados recebidos e realizados os reencaminhamentos necessários. Caso estes dados estejam correctos, são enviados para a **terceira camada** (dados).

1.5 Em que medida a estrutura geral do nosso sistema e as nossas escolhas de *design* suportam os objetivos do negócio do sistema, no que diz respeito aos drivers arquitecturais identificados na questão 1.1.

1.5.1 Comparação e escolha da arquitectura

Arquitectura *Pipe-and-Filter*³

Esta é uma arquitectura poderosa e robusta, ideal para dividir uma tarefa de processamento maior numa sequência de etapas menores, independentes (*filters*) que são conectados por canais (*pipes*). A maior desvantagem é a necessidade de os filtros exigirem que os dados estejam no menor denominador comum, tipicamente fluxos de *bytes* ou caracteres, ao longo de todo o *pipeline* e não é apropriado para aplicações interactivas. Seria uma opção viável se este problema exigisse um grande número de transformações para executar, o que não é o caso.

Service Oriented Architecture (SOA)

Este tipo de arquitectura de *software* tem como princípio fundamental a disponibilização de funcionalidades através de um conjunto de serviços. Uma possibilidade de implementação é recorrendo a *Web Services*. Nesta arquitectura, um sistema é constituído por entidades autónomas (sub-sistemas, também chamados de serviços), que actuam independentemente umas das outras. Geralmente, os motivos que levam a escolher esta abordagem visam solucionar a integração de diversos sistemas diferentes. Apresenta como vantagens o facto de permitir evoluir futuramente, facilitando a substituição dos serviços existentes e a adição de novos. Desta forma, permite minimizar as interrupções dos processos de negócio de uma organização.

Contudo, esta arquitectura possui alguns *drawbacks*, que nos fizeram optar por outra alternativa, dado que afecta a performance (depende do servidor onde o serviço está publicado, como também da rede), disponibilidade (uma queda na rede ou no servidor deixa todos os serviços indisponíveis) e segurança (os serviços estão disponíveis na rede, onde qualquer aplicação pode consumir esse serviço).

Arquitectura de 1 camada (monolítica)

Nesta arquitectura todos os componentes existem apenas numa camada, ou seja o equivalente a correr todas as aplicações numa única máquina. **Vantagens:** Fácil de desenhar, modularidade, dado que se pode implementar uma camada sem exigir modificação das outras.

Desvantagens: Não é escalável e aumenta a complexidade e o tamanho do código.

Arquitectura de 2 camadas

Nesta arquitectura, a camada de apresentação é executada pelo cliente e a camada de dados é guardada no servidor. **Vantagens:** as aplicações são mais fáceis de desenvolver devido à simplicidade da arquitectura; a geração de protótipos das aplicações é rápida; a camada de dados e a lógica de negócio encontram-se fisicamente perto, o que resulta em maior performance. **Desvantagens:** tem problemas de segurança uma vez que grande parte do processamento é feito no lado do cliente e acede directamente à camada de dados, problemas de escalabilidade uma vez que suporta um número limitado de clientes.

Arquitectura de 3 camadas - a escolhida

Uma das principais razões para a escolha de uma arquitectura de 3 camadas está na possibilidade de ser possível reutilizar a lógica de negócio já existente e poder facilmente adicionar os requisitos pretendidos. Esta arquitectura possibilita também uma melhor escalabilidade apenas presente em arquitecturas com 3 camadas, na medida em que, tipicamente, é difícil escalar verticalmente os *web servers* (número de conexões é limitado pelo número de *threads* disponíveis). Este problema é resolvido com recurso a (*dispatch queues*) entre o *web server* e o (*application server*). Esta arquitectura também melhora a segurança geral do sistema uma vez que é colocada uma camada entre o (*web server*) e a base de dados, o que impossibilita o acesso directo à base de dados por parte do utilizador. Por fim a disponibilidade também é melhorada pois em casos em que a camada lógica se encontra indisponível e exista cache suficiente, é possível processar pedidos Web através da camada de apresentação. **Vantagens:** maior facilidade em escalar cada camada, pois cada uma é independente das restantes; maior disponibilidade e extensibilidade dado que é relativamente simples adicionar novos componentes nas camadas; maior segurança pois cada pedido dos utilizadores passa primeiro por uma camada intermédia antes de chegar à base de dados.

Desvantagens: maior complexidade na comunicação entre componentes,

maior esforço na sua implementação.

1.5.2 Atributos de Qualidade

Para conseguir atingir estes atributos de qualidade foram usadas várias combinações de táticas de modo atingir o respectivo atributo.

A **segurança** é assegurada com recurso a um mecanismo de autenticação de *users*, usando o protocolo HTTPS entre o cliente e o servidor, dando apenas acesso a utilizadores (funcionários) registados à tabela pretendida (*users* ou *orders*), protegendo a interface contra *sql-injection*, cifrando os dados do utilizador e por fim guardando todos os dados no servidor.

Em relação à **escalabilidade**, é possível escalar a camada de apresentação e de aplicação conforme as necessidades (por exemplo facilmente se adicionam mais servidores web a camada de apresentação), por fim como a lógica de negócio se encontra na camada da aplicação qualquer mudança que seja efectuada será reflectida em todo o sistema.

O atributo de **disponibilidade** é obtido recorrendo a servidores de *backup*, que entram em funcionamento no caso de falha do servidor principal.

O atributo de **consistência**, é obtido usando *message queues*, para o caso da ligação entre componentes falhar as mensagens não são duplicadas. A base de dados de *users*, têm regras específicas para os seus campos e faz uso de chaves primárias (id).

O atributo de **extensibilidade** também é favorecido usando esta arquitectura, sendo fácil a integração de novos módulos.

1.5.3 Requisitos Funcionais

Com a criação de uma tabela *users* na base de dados e com o desenvolvimento de um sistema de *login/logout* (de modo a guardar os dados dos diversos funcionários e as suas credenciais), baseado em *Struts 2* é assim cumprido o Requisito **RF1**. Para além disso, foi criada também uma página *web* onde podem ser feitas as encomendas remotamente, cumprindo desta forma o Requisito **RF2**.

Os ficheiros de *log* são também criados e armazenados localmente no servidor podendo ser consultados pelo administrador.

1.6 *Tradeoffs* realizados na concepção do sistema. Que outras alternativas teriam? O que vos fez optar pelas vossas soluções em detrimento de outras?

1.6.1 RMI vs EJBs vs *Web Services*

A opção por *Java* RMI justifica-se pela sua rapidez, leveza, simplicidade e capacidade de resposta a este problema pouco complexo, levando, assim, vantagem sobre os *Enterprise JavaBeans* (EJB) que funcionam, também, sobre RMI. Os mesmos, seriam também uma alternativa viável no desenvolvimento de uma solução de maior complexidade no âmbito de transacções, segurança e concorrência, dado que estes oferecem "automaticamente" alguns benefícios a esse nível. A Figura 9 ilustra a diferença do *Java* RMI *plain* com o usado nos EJBs. Como o desenvolvimento do novo sistema também será em *Java*, não se justifica a utilização de *Web Services*, que apesar de evidenciarem uma maior vantagem de comunicação entre serviços, apresentam alguns *drawbacks*, como elevada complexidade (apesar do serviço SOAP ser um *standart*) e desempenho, quando comparado com o RMI, por exemplo, uma vez que as operações de *Marshalling* consomem ciclos de CPU para permitir comunicação entre o cliente e as diversas partes.

1.6.2 Disponibilidade vs Segurança

Estes dois atributos são aqueles que se consideraram mais importantes no estudo deste projecto. Tratando-se de uma empresa *business-to-business*, considera-se que será mais prejudicial enquanto negócio se não oferecer um serviço de alta **disponibilidade**. Não significa, contudo, que o atributo de qualidade associado à **segurança** tenha sido deteriorado, até porque como a empresa passará a ter o serviço de registo de encomendas descentralizado, é de elevada magnitude o interesse em manter o controlo do acesso ao sistema. É também valorizado o não repúdio da informação criada. Assim, o *tradeoff* entre estes atributos não assume que algum tenha sido deteriorado, mas justifica as razões dadas para ordenar a importância de cada um.

2 Plano de Implementação

Embora não seja prioritário para o cliente, são recomendadas algumas mudanças ao nível da base de dados, nomeadamente na inserção e actualização dos dados. De seguida, são referidos os aspectos implementados na versão que segue em anexo a este relatório:

- registo remoto de novas encomendas (RMI, Tomcat);
- *log* de entradas e saídas de funcionários no sistema remoto;
- *log* de entradas e expedições de encomendas;
- interface *web* para registo de novas encomendas e de novos funcionários.

3 Conclusão

Conclui-se que existe um grande conjunto de arquitecturas de *software* com vantagens e desvantagens associadas. Não existe o conceito de *silver-bullet*, pelo que cabe ao arquitecto adaptar aquela que mais se adequa aos *drivers* arquitecturais do projecto. É, também, importante ter em consideração as prioridades dos atributos de qualidade de forma a poder analisar os *tradeoffs* existentes. Deve ter-se em consideração quais os *stakeholders* de forma a concentrar esforços nas vistas que lhes trazem o nível de detalhe adequado.

4 Tabelas

Caso de uso	Login
Id	id1as
Actores Principais	Funcionário
Descrição	Funcionário coloca <i>username</i> e <i>password</i> . As credencias são verificadas na base de dados e se forem corretas, o funcionário entra no sistema, caso contrário, recebe uma mensagem de erro na página de <i>login</i> .
Pré-condições	Ter credenciais válidas para aceder ao sistema.
Pós-condições	Aceder à secção de registo remoto de encomendas e registar a entrada em <i>log</i>
Input	<i>Username</i> e <i>password</i>
Output	Inteiro - Id do Utilizador ou -1 em caso de falha
Prioridade	Normal
Excepções	Credenciais inválidas

Tabela 1: Caso de uso “Login”

Caso de uso	Adicionar encomendas no OrdersWebApp
Id	id2as
Actores Principais	Funcionário
Descrição	Funcionário coloca os dados do cliente e adiciona as unidades dos produtos pretendidos. Finaliza a encomenda, registando-a na base de dados.
Pré-condições	Ter entrado no sistema com sucesso.
Pós-condições	<i>Feedback</i> da entrada da encomenda no sistema. Registo em <i>log</i> do número da encomenda, nome do utilizador, hora e data da introdução.
Input	Dados do cliente (primeiro e último nomes, telefone e morada), artigos a encomendar
Output	Mensagem de confirmação do registo da encomenda
Prioridade	Normal
Exceções	Não existir quantidade suficiente do(s) artigo(s) desejado(s)

Tabela 2: Caso de uso “Adicionar encomendas no OrdersWebApp”

Caso de uso	<i>Logging</i>
Id	id3as
Actores Principais	Funcionário
Descrição	Registo automático das acções de <i>login</i> e <i>logout</i> , entrada e despacho de encomendas
Pré-condições	Ter entrado no sistema com sucesso.
Pós-condições	Gravar em ficheiro todo o registo de entradas e saídas no sistema, assim como o registo da entrada e despacho de encomendas
Input	Momento do acesso/saída ao sistema, dados da encomenda
Output	
Prioridade	Normal
Exceções	Erro na escrita em ficheiro

Tabela 3: Caso de uso “Logging”

Caso de uso	ManagementWebApp
Id	id4as
Actores Principais	Administrador
Descrição	Registo de funcionários
Pré-condições	Ter entrado no sistema como Administrador com sucesso.
Pós-condições	
Input	Dados do funcionário
Output	Registo efectuado com sucesso ou não.
Prioridade	Normal
Excepções	

Tabela 4: Caso de uso “ManagementWebApp”

Caso de uso	OrdersApp
Id	id5as
Actores Principais	Funcionário
Descrição	Registo local de encomendas
Pré-condições	Ser funcionário da instalação central e ter acesso ao programa
Pós-condições	Mensagem de <i>feedback</i> do registo da encomenda quer em log, quer para o utilizador
Input	Dados do cliente (primeiro e último nomes, morada e telefone), produtos a encomendar
Output	Mensagem de <i>feedback</i> da acção realizada
Prioridade	Normal
Excepções	

Tabela 5: Caso de uso “OrdersApp” (já existente)

Caso de uso	ShippingApp
Id	id6s
Actores Principais	Funcionário
Descrição	Despachar as encomendas localmente
Pré-condições	Ser funcionário da instalação central e ter acesso ao programa
Pós-condições	Mensagem de <i>feedback</i> do despacho da encomenda, quer em log, quer para o utilizador
Input	Id da encomenda a despachar
Output	Mensagem de <i>feedback</i> da acção realizada
Prioridade	Normal
Excepções	

Tabela 6: Caso de uso “ShippingApp” (já existente)

Caso de uso	InventoryApp
Id	id7as
Actores Principais	Funcionário
Descrição	Adicionar produtos ao stock localmente
Pré-condições	Ser funcionário da instalação central e ter acesso ao programa
Pós-condições	Submissão de novo produto no stock
Input	Descrição, id, tipo, preço e quantidade do produto
Output	Listagem do stock
Prioridade	Normal
Excepções	

Tabela 7: Caso de uso “InventoryApp” (já existente)

<i>Raw Quality Attribute</i>	Disponibilidade
<i>Stimulus</i>	Servidor <i>RMI</i> fica indisponível
<i>Source of the Stimulus</i>	Interna
<i>Relevant Environmental Conditions</i>	Em condições normais
<i>Architectural Elements</i>	Servidor <i>RMI</i>
<i>System Response</i>	Servidor <i>RMI</i> de backup, entra em funcionamento para substituir o servidor principal, uma vez que este se encontra em baixo.
<i>Response Measure(s)</i>	O servidor de <i>backup</i> entra em funcionamento após 3 tentativas falhadas de comunicação com o servidor principal.

Tabela 8: Cenário de disponibilidade

<i>Raw Quality Attribute</i>	Segurança
<i>Stimulus</i>	Um utilizador tenta modificar a Base de Dados
<i>Source of the Stimulus</i>	utilizador
<i>Relevant Environmental Conditions</i>	O sistema final pode ser acedido remotamente
<i>Architectural Elements</i>	Camada de apresentação
<i>System Response</i>	O sistema indica ao utilizador que introduziu dados incorrectos
<i>Response Measure(s)</i>	Os dados permanecem inalterados

Tabela 9: Cenário de segurança

<i>Raw Quality Attribute</i>	Consistência
<i>Stimulus</i>	O funcionário cria uma encomenda sem preencher a morada do cliente
<i>Source of the Stimulus</i>	Funcionário
<i>Relevant Environmental Conditions</i>	Em condições normais
<i>Architectural Elements</i>	Camada Lógica
<i>System Response</i>	Em condições normais
<i>Response Measure(s)</i>	A nova entrada não é inserida na base de dados e o funcionário é notificado dessa ocorrência.

Tabela 10: Cenário de consistência

<i>Raw Quality Attribute</i>	Extensibilidade
<i>Stimulus</i>	Adição de uma funcionalidade para saber o histórico de compras do cliente
<i>Source of the Stimulus</i>	Interna
<i>Relevant Environmental Conditions</i>	Em condições normais
<i>Architectural Elements</i>	Camada de Lógica
<i>System Response</i>	Operações normais
<i>Response Measure(s)</i>	Será necessário adicionar a funcionalidade pretendida nas diversas camadas.

Tabela 11: Cenário de extensibilidade

<i>Raw Quality Attribute</i>	Escalabilidade
<i>Stimulus</i>	O servidor web deixa de conseguir responder ao elevado número de pedidos
<i>Source of the Stimulus</i>	Externa
<i>Relevant Environmental Conditions</i>	Sobrecarga
<i>Architectural Elements</i>	Servidor Web
<i>System Response</i>	Maior intervalo de resposta por parte do servidor para os clientes
<i>Response Measure(s)</i>	A adição de novos servidores irá mitigar o problema de sobrecarga uma vez que a mesma será distribuída pelos servidores totais

Tabela 12: Cenário de escalabilidade

5 Diagramas e Vistas

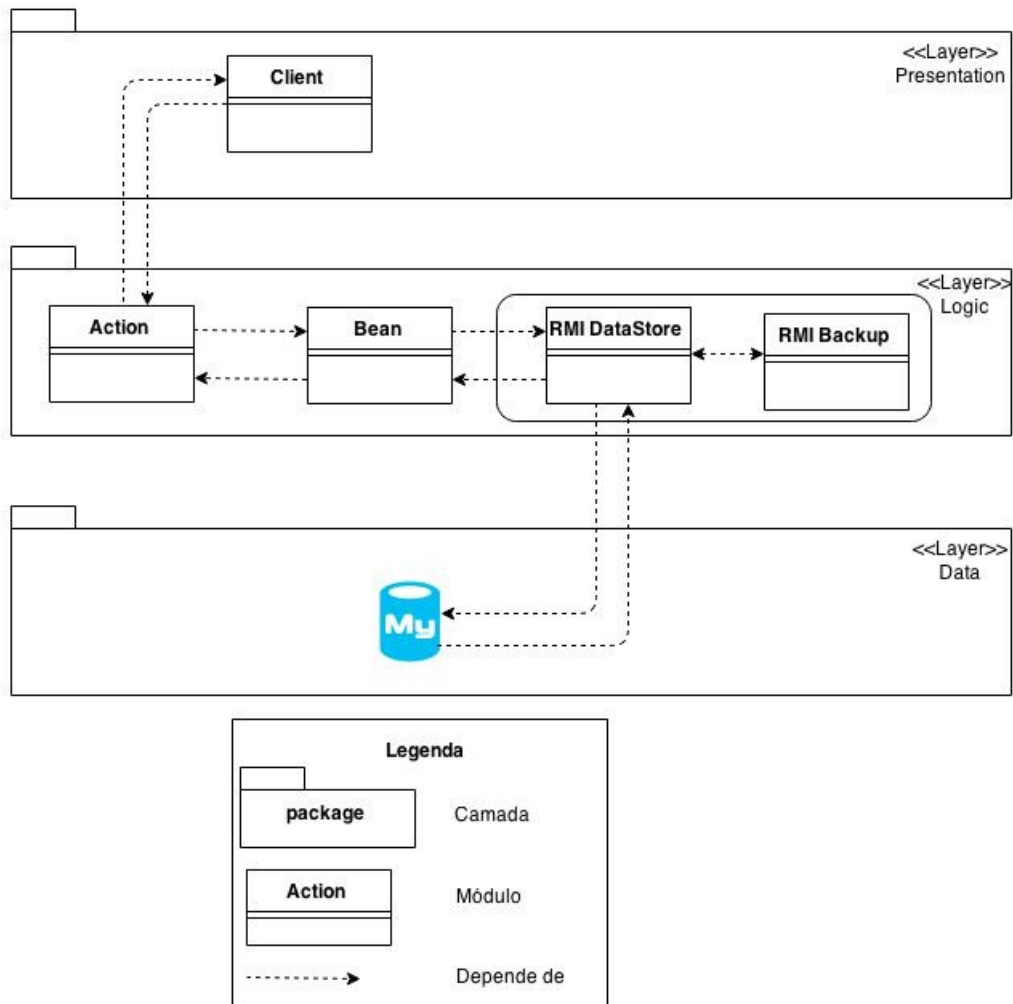


Figura 3: Vista de Módulos - Vista de Composição.

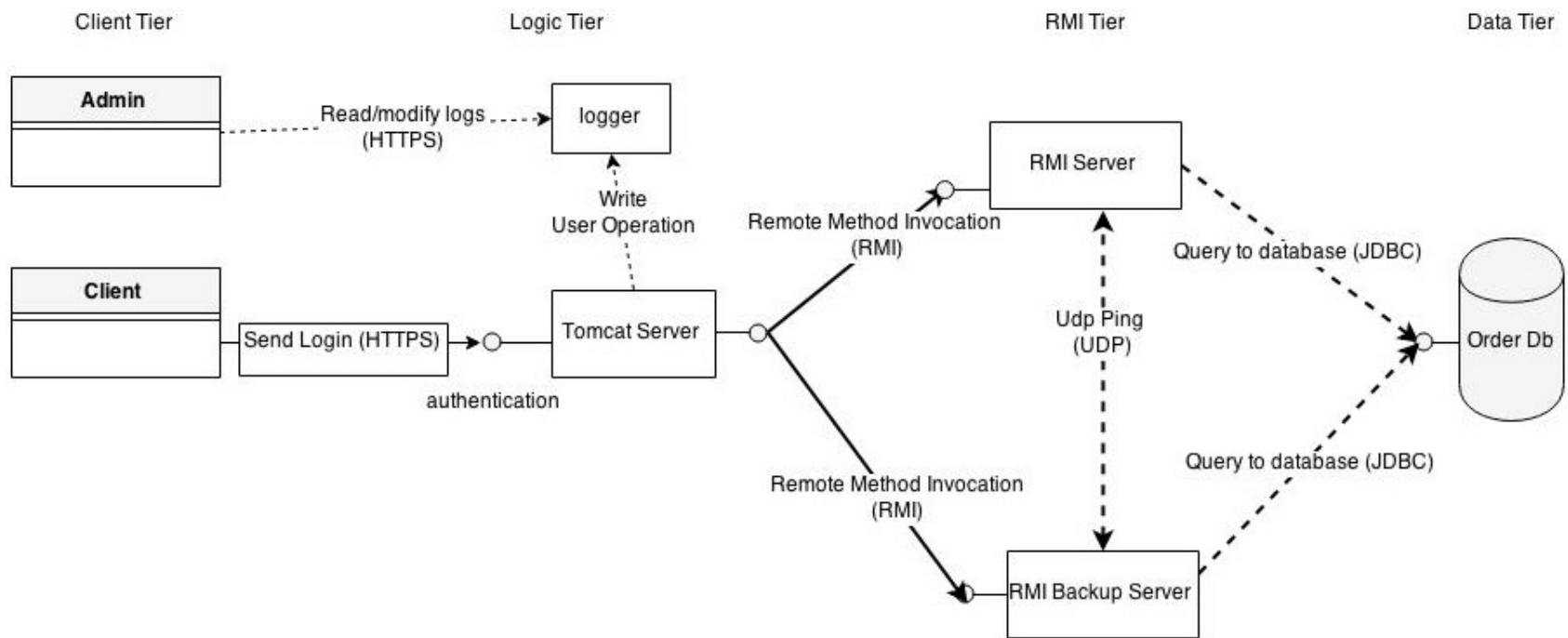


Figura 4: Vista de Componente e Conector - Shared Data.

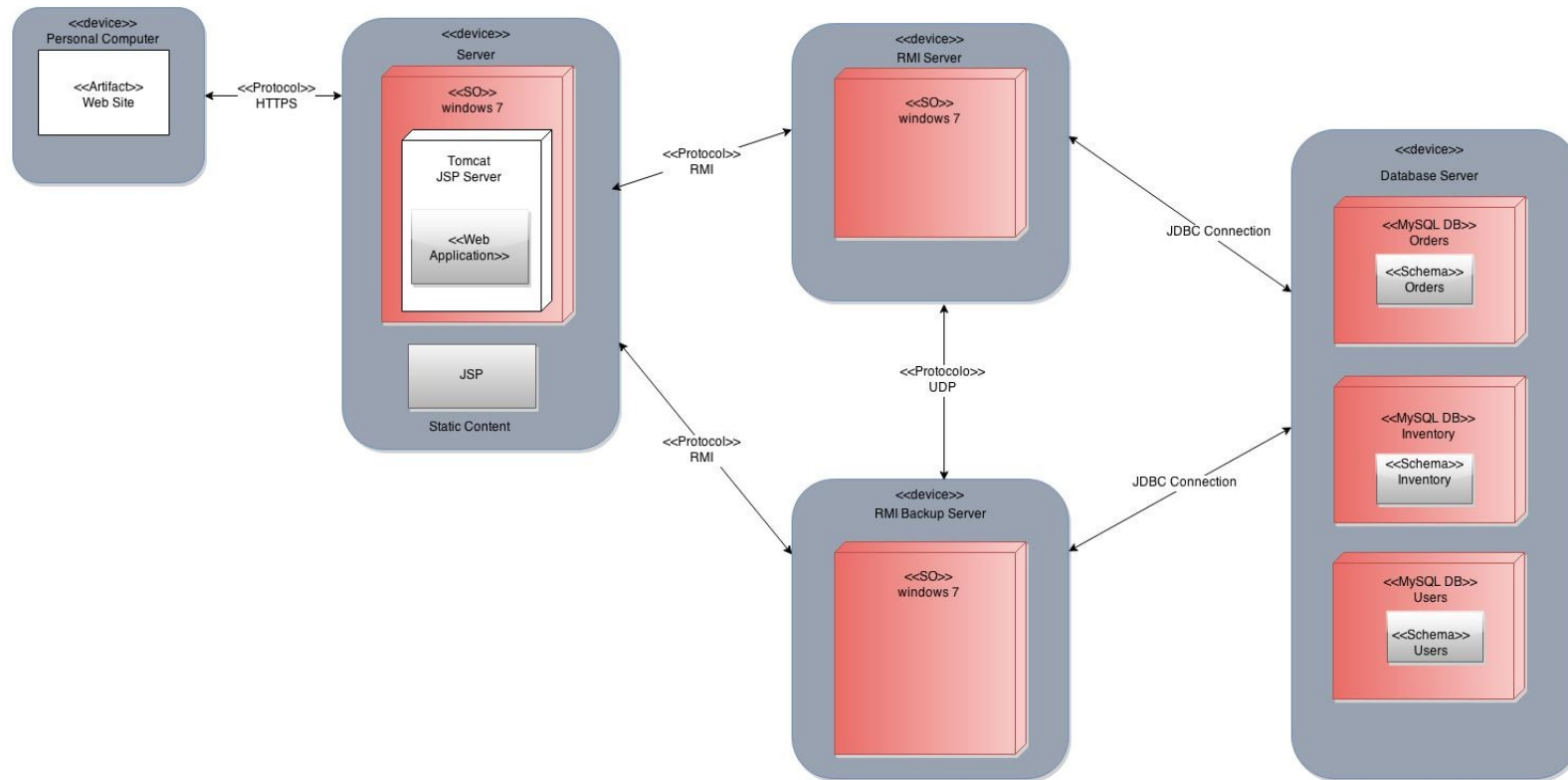


Figura 5: Diagrama de alocação.

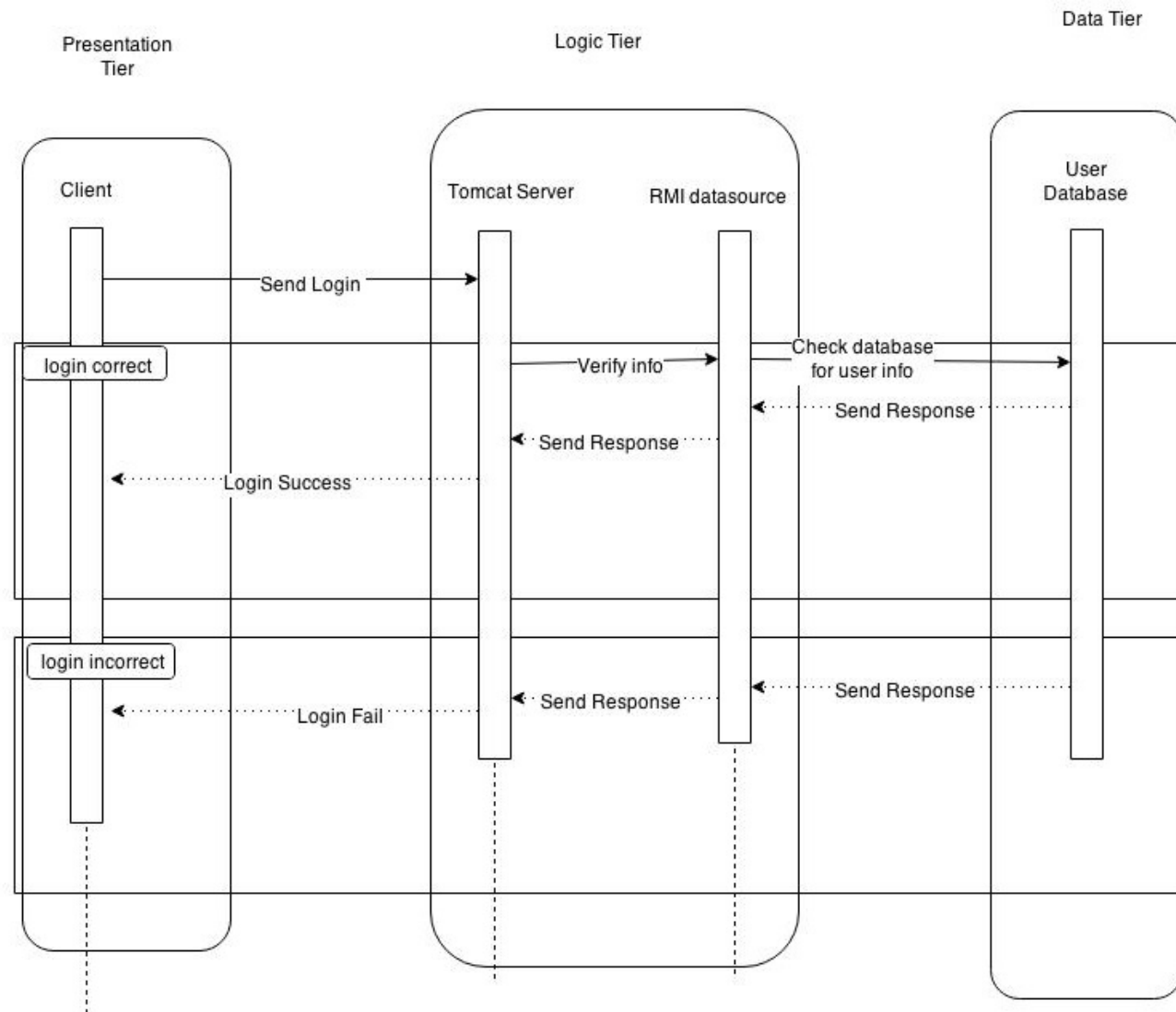


Figura 6: Diagrama de sequência - Login.

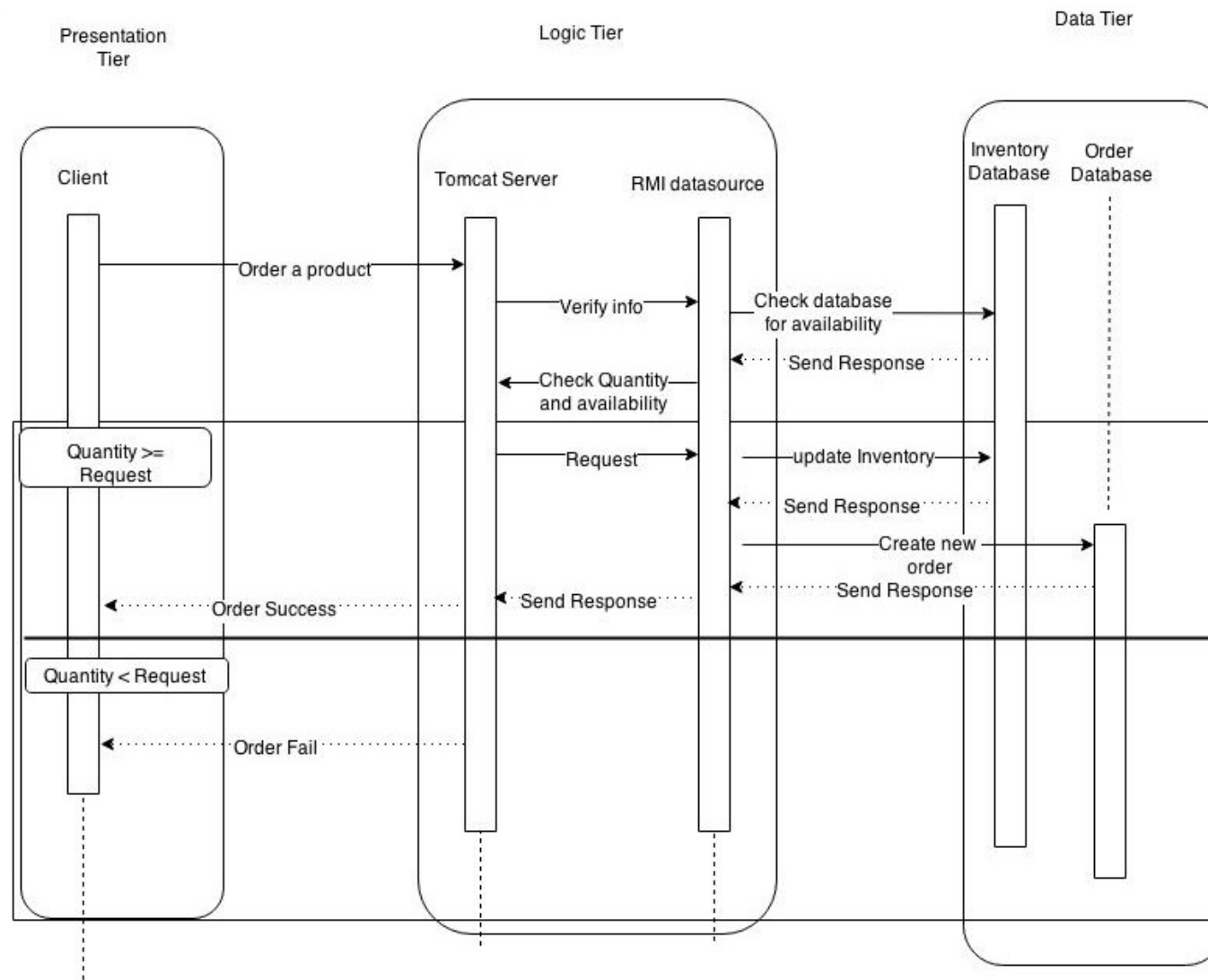


Figura 7: Diagrama de sequência - Orders.

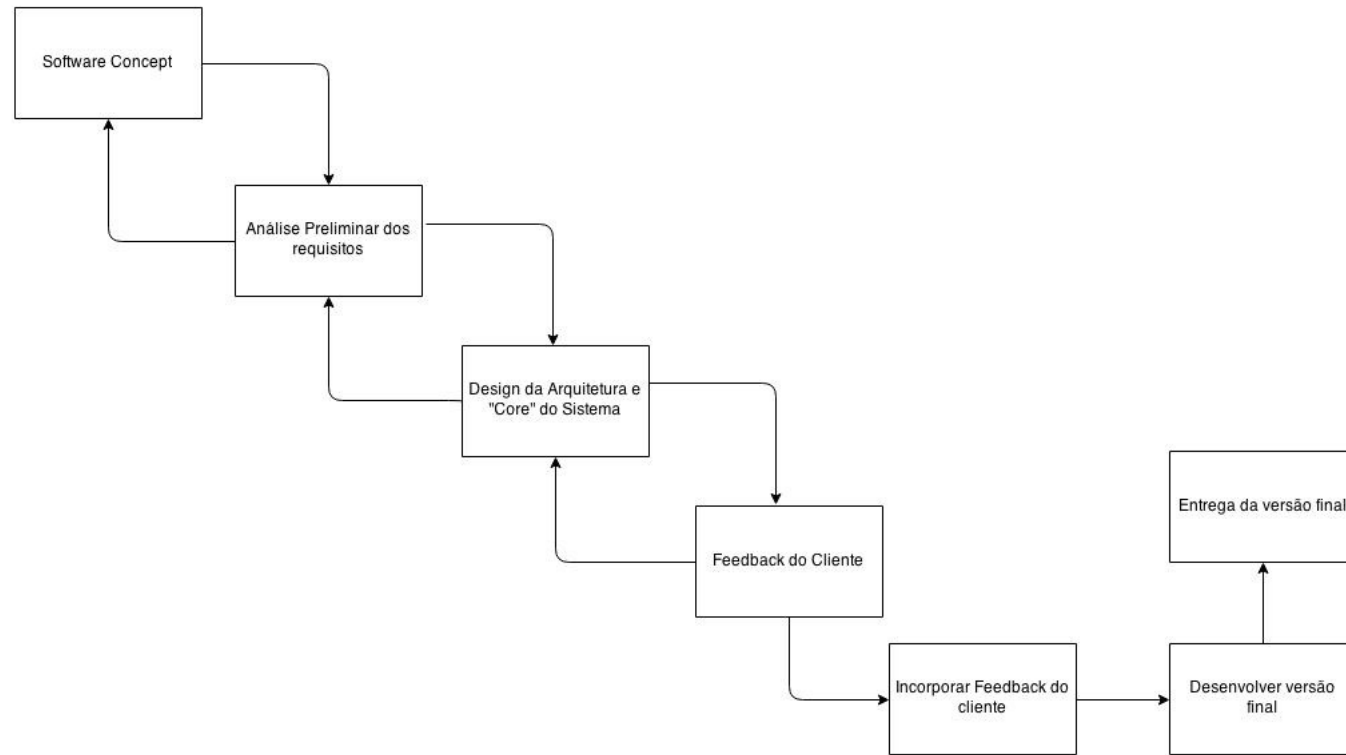
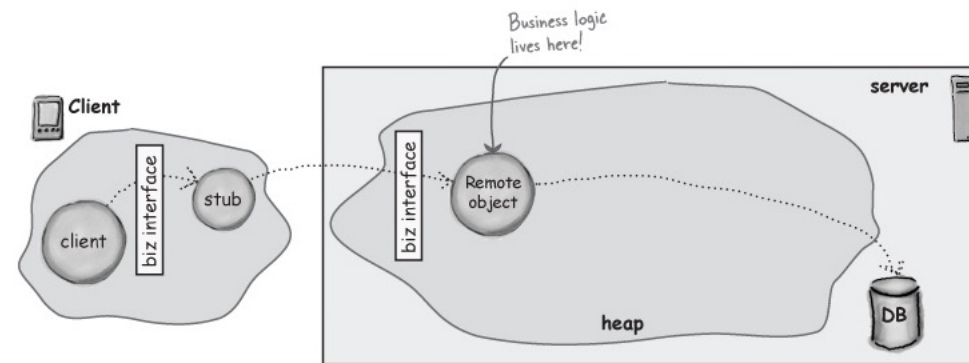


Figura 8: *Architecture Life Cycle.*

Plain RMI



RMI in EJB

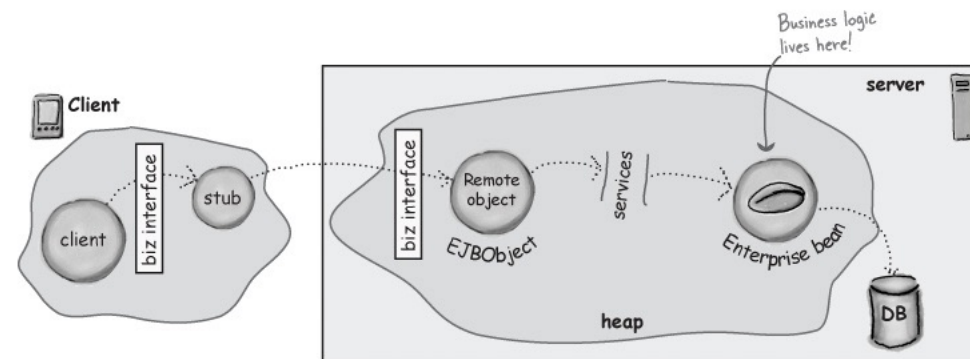


Figura 9: Plain RMI vs RMI in ESB.¹

Welcome to EPE System

Login:

Username: gsp

Password:

Login

Figura 10: *Layout do login.*

EPE - Order System

gsp [Log Out](#)

- Login com sucesso

Trees

Seeds

Shrubs

Cart

Gonçalo

Pereira

RU Polo II-I Piso 3 - Rua Miguel Bombarda

911112937

EF001 | Elephant Foot | 6 | 800.00

JYM001 | Japanese Yama Maple | 500 | 42.00

YK001 | Yemen Khat | 21 | 625.00

AT001 | Asian Teak | 83 | 260.00

WW001 | Worm Wood | 91 | 115.00

WW001 | Worm Wood | 91 | 115.00

LC001 | Lemon Cypress | 280 | 67.00

EOM001 | European Olive | 62 | 225.00

RSM001 | Red Snakebark Maple | 200 | 98.00

AT001 | Asian Teak | 83 | 260.00

CT001 | Cigar Tree | 10 | 83.00

Total: 2690

Encomendar

Código	Descrição	Quantidade	Preço
EF001	Elephant Foot	6	800.00
BB001	Black Bamboo	30	100.00
BF001	Banyan Fig	13	325.00
RSM001	Red Snakebark Maple	200	98.00
BE001	Box Elder	400	36.00
JYM001	Japanese Yama Maple	500	42.00
EOM001	European Olive	62	225.00
YK001	Yemen Khat	21	625.00
AT001	Asian Teak	83	260.00
WW001	Worm Wood	91	115.00
LC001	Lemon Cypress	280	67.00
GB001	Ginkgo Biloba	75	80.00
CT001	Cigar Tree	10	83.00
AM002	Arden Maple	40	70.00
FL002	Finger Leaf Elm	16	75.00
123456	tree	50	10.00

Figura 11: *Layout dos produtos em stock.*

EPE - Admin Menu

root Log Out

• Login com sucesso

×

Registrar novo funcionário:

Email:

Introduza o email

Username:

Introduza o username

Firstname:

Introduza o primeiro nome

Lastname:

Introduza o último nome

Address:

Introduza a morada

Phone:

Introduza o telefone

Password:

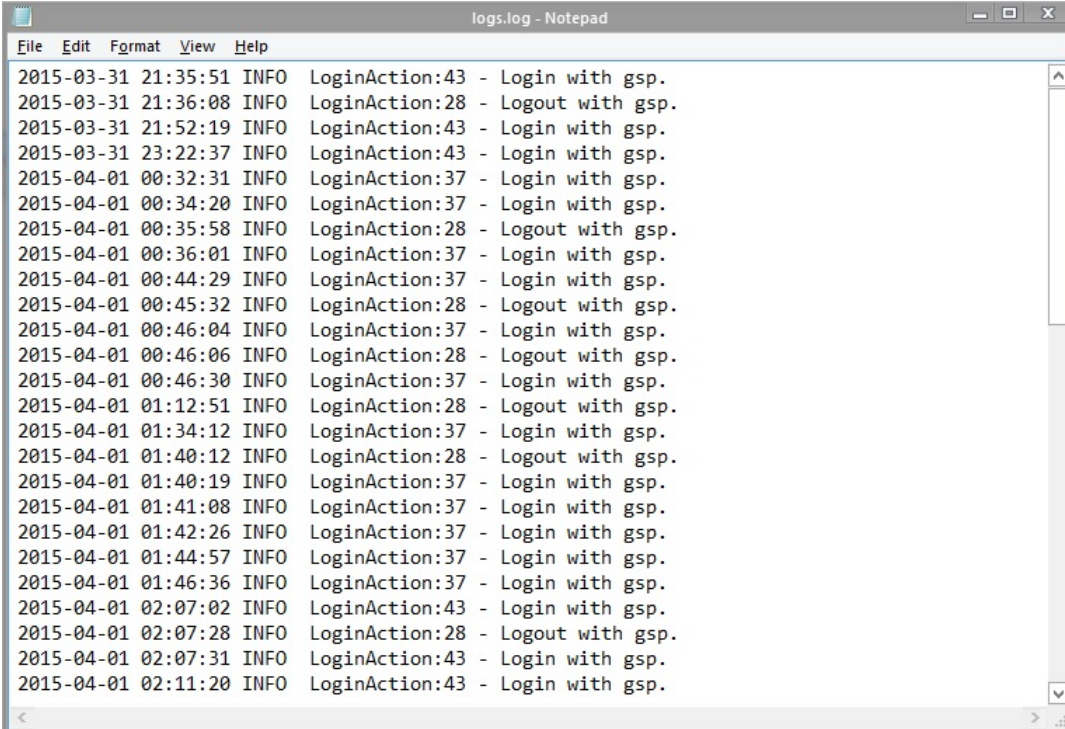
Introduza a password

Password:

Confirme a password

Registrar

Figura 12: *Layout do menu do Admin.*



```
logs.log - Notepad
File Edit Format View Help
2015-03-31 21:35:51 INFO LoginAction:43 - Login with gsp.
2015-03-31 21:36:08 INFO LoginAction:28 - Logout with gsp.
2015-03-31 21:52:19 INFO LoginAction:43 - Login with gsp.
2015-03-31 23:22:37 INFO LoginAction:43 - Login with gsp.
2015-04-01 00:32:31 INFO LoginAction:37 - Login with gsp.
2015-04-01 00:34:20 INFO LoginAction:37 - Login with gsp.
2015-04-01 00:35:58 INFO LoginAction:28 - Logout with gsp.
2015-04-01 00:36:01 INFO LoginAction:37 - Login with gsp.
2015-04-01 00:44:29 INFO LoginAction:37 - Login with gsp.
2015-04-01 00:45:32 INFO LoginAction:28 - Logout with gsp.
2015-04-01 00:46:04 INFO LoginAction:37 - Login with gsp.
2015-04-01 00:46:06 INFO LoginAction:28 - Logout with gsp.
2015-04-01 00:46:30 INFO LoginAction:37 - Login with gsp.
2015-04-01 01:12:51 INFO LoginAction:28 - Logout with gsp.
2015-04-01 01:34:12 INFO LoginAction:37 - Login with gsp.
2015-04-01 01:40:12 INFO LoginAction:28 - Logout with gsp.
2015-04-01 01:40:19 INFO LoginAction:37 - Login with gsp.
2015-04-01 01:41:08 INFO LoginAction:37 - Login with gsp.
2015-04-01 01:42:26 INFO LoginAction:37 - Login with gsp.
2015-04-01 01:44:57 INFO LoginAction:37 - Login with gsp.
2015-04-01 01:46:36 INFO LoginAction:37 - Login with gsp.
2015-04-01 02:07:02 INFO LoginAction:43 - Login with gsp.
2015-04-01 02:07:28 INFO LoginAction:28 - Logout with gsp.
2015-04-01 02:07:31 INFO LoginAction:43 - Login with gsp.
2015-04-01 02:11:20 INFO LoginAction:43 - Login with gsp.
```

Figura 13: *Exemplo de ficheiro de logs relativamente ao Login e Logout.*

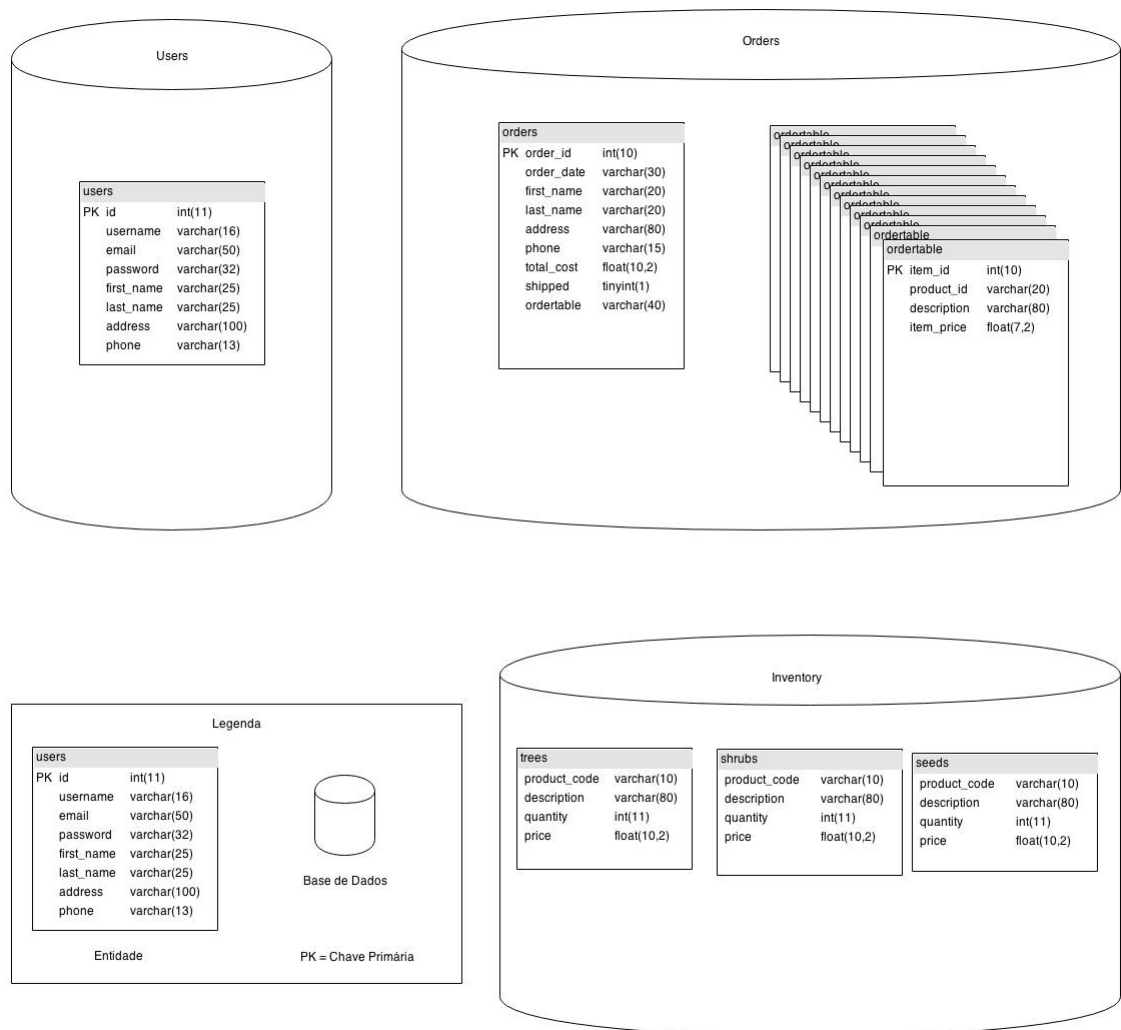


Figura 14: *Data Model View* .

Referências

- ¹ Safaribooksonline, “How ejb uses rmi,” 2015, @29-Março-2015. [Online]. Available: <https://www.safaribooksonline.com/library/view/head-first-ejb/0596005717/ch02s09.html>
- ² P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, *Documenting software architectures: views and beyond*. Pearson Education, 2002.
- ³ Dossier-andreas, “Pipe-and-filter,” 2015, @30-Março-2015. [Online]. Available: http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html
- ⁴ B. L. K. R. Clements, Paul, *Software Architecture in Practice, Second Edition*. Addison Wesley, 2003.
- ⁵ “Microsoft application architecture guide, second edition,” 2009, @29-Março-2015. [Online]. Available: <https://msdn.microsoft.com/en-us/library/ff650706.aspx>