|  |
| --- |
|  |
| Universidade do Minho |
| Licenciatura em Engenharia Informática |

Relatório Final

Projecto da U.C. de Modelação e Desenvolvimento

Braga, 7 de Janeiro de 2013

**Elaborado por:**

Grupo 33

|  |  |
| --- | --- |
| C:\Users\marcelo\Desktop\piças.jpg | C:\Users\marcelo\Desktop\paxa.jpg |
| Pedro Miguel Costa Maia – A61078 | Luís Carlos Caseiro – A61009 |
| C:\Users\marcelo\Desktop\ces.jpg | C:\Users\marcelo\Pictures\EU\Pedro Faria - Cópia.jpg |
| César Morais Perdigão – A61007 | Pedro Miguel Oliveira Faria- A60998 |

Índice

[Índice Tabelas: 3](#_Toc345332715)

[Índice Figuras: 4](#_Toc345332716)

[4 – Introdução 5](#_Toc345332717)

[5 – Desenvolvimento 6](#_Toc345332718)

[5.1 - Esquema Conceptual 6](#_Toc345332719)

[5.2 - Funcionalidades a implementar na aplicação Windows. 8](#_Toc345332720)

[5.3 - Funcionalidades a implementar na aplicação Web 9](#_Toc345332721)

[6 – Camadas 10](#_Toc345332722)

[6.1 - Camada de Apresentação (Interface do Utilizador) 10](#_Toc345332723)

[6.2 - Camada de Dados 18](#_Toc345332724)

[6.3 - Camada Business 19](#_Toc345332725)

[6.3.1 – Class BuyKing 19](#_Toc345332726)

[7 – Conclusão 22](#_Toc345332727)

[8 – Anexos 23](#_Toc345332728)

[6.3.2 – Class Leilão 23](#_Toc345332729)

[6.3.3 – Class Produto 24](#_Toc345332730)

[6.3.4 – Class Suspeita 24](#_Toc345332731)

[6.3.5 – Class Troca 25](#_Toc345332732)

[6.3.6 – Class Venda 26](#_Toc345332733)

[6.3.7 – Class Utilizador 27](#_Toc345332734)

[6.3.8 – Class Licitação 28](#_Toc345332735)

[6.3.9 – Class Leilao Fechado Exception 28](#_Toc345332736)

[6.3.10 – Class Classificação 29](#_Toc345332737)

[6.3.11 - Class Baixa Licitação Exception 29](#_Toc345332738)

# Índice Tabelas:

# Não foi encontrada nenhuma entrada do índice de ilustrações.

# Índice Figuras:

[Fig. 1- Esquema Conceptual da Base Dados 6](#_Toc345333551)

[Fig. 2-Login 11](#_Toc345333552)

[Fig. 3 - Favoritos 11](#_Toc345333553)

[Fig. 4 - Principal 12](#_Toc345333554)

[Fig. 5 – Vender 12](#_Toc345333555)

[Fig. 6 - Leiloar 13](#_Toc345333556)

[Fig. 7 - Pesquisa avançada 13](#_Toc345333557)

[Fig. 8 - Produto em leilão 14](#_Toc345333558)

[Fig. 9 - Registar 14](#_Toc345333559)

[Fig. 10 - Produto à venda 15](#_Toc345333560)

[Fig. 11 – Pesquisa Simples 15](#_Toc345333561)

[Fig. 12 - A vender 16](#_Toc345333562)

[Fig. 13 – Inicio 16](#_Toc345333563)

[Fig. 14 - Esquema conceptual da base de dados elemento a elemento 29](#_Toc345333564)

# 4 – Introdução

Este relatório surge no âmbito da realização do trabalho da disciplina

de Base de Dados, do curso de Engenharia Informática da Universidade do Minho. Neste trabalho pretende-se desenvolver uma aplicação informática que suporta o comércio eletrónico, permitindo aos utilizadores comprar e vender produtos.

Desta forma procedemos á análise do enunciado do projecto debatendo ideias sobre como deveríamos estruturar o nosso esquema conceptual da base de dados, sobre as funcionalidades que deveríamos implementar na aplicação Windows e na aplicação web.

***????????????***Decidimos alterar um pouco com o previsto pois prescindimos de que o programa suportasse uma funcionalidade de Trocas, a única opção que nos restava era criar outro sistema para os utilizadores fazerem trocas entre si, o que não foi feito pois ia em contra com os objetivos programa de Comercio eletrónico.

O objetivo principal do nosso sistema é permitir aos seus utilizadores vender/comprar/leiloar produtos sem que estes precisem de se preocupar mais, pois o sistema trata de entregar os produtos ao comprador e o valor do produto ao vendedor.

Este relatório descreve então esses passos tomados no desenvolvimento do projecto, servindo assim de apoio a este projecto.

# 5 – Desenvolvimento

## 5.1 - Esquema Conceptual

Sabendo que as chaves primárias, candidatas e estrangeiras deverão estar claramente identificadas e também as tabelas deverão estar indexadas pelos atributos usados em seleções, ordenações e agrupamentos.  
 Após todos os elementos do grupo se terem reunido na escolha das chaves primárias de cada uma das tabelas foi complicado pois começamos por fazer a normalização dos dados.

Como podemos ver, a Figura 3.1mostra todo o esquema conceptual que tendo algumas tabelas como chave primária com mais de um atributo.

|  |
| --- |
|  |
| Fig. 1- Esquema Conceptual da Base Dados |

Fazendo uma breve análise sobre a Fig. 1 temos o nome de cada tabela nas células preenchidas de cor cinzenta, e as chaves primárias de cada tabela com o texto a negrito.  
Quanto as ligações que são feitas, são chaves estrangeiras ou primarias que estarão presentes e que tomaram valores únicos.

Quanto as cores que preenche cada célula, temo a cor amarela corresponde a Inteiros, a cor vermelha a VarChar2 e a cor verde a tipo data.

Em [**anexo**](#_Anexos)terá a imagem de cada tabela.

## 5.2 - Funcionalidades a implementar na aplicação Windows.

O ***Java*** é uma linguagem de programação orientada a objectos. Para a aplicação que funcionará em ambiente ***Windows,*** decidimos optar pela linguagem ***Java*** pelas seguintes razões:

* Sendo uma linguagem de programação de fácil uso.
* Funciona para todas as plataformas.
* Existência de packages que ajudam no tempo de construção.

Suporta vendas, trocas e leilões entre utilizadores, bem com uma lista de favoritos e um sistema de classificação de utilizadores.

## 5.3 - Funcionalidades a implementar na aplicação Web

Relativamente há implementação da aplicação web utilizará ***PHP*** pois é uma linguagem de programação ***Server-Side*** para internet. As razões que nos levaram adoptar a esta linguagem foram as seguintes:

* Funciona em qualquer plataforma onde for possível instalar um servidor Web.
* É rápido e raramente deixa o servidor lento.
* Conecta facilmente a sistemas ***Oracle***.
* Contem elementos das linguagens ***java*** e ***C***, com as quais já trabalhamos.
* É uma programação Orientada a Objetos.

# 6 – Camadas

### 6.1 - Camada de Apresentação (Interface do Utilizador)

Apesar de no ano anterior na **Unidade Curricular de LI3** termos feito já uma pequena interface, neste projeto queríamos fazer uma interface mais personalizada, com melhor manuseamento por parte do utilizador e também com uma boa performance. Para isso pesquisamos sobre as utilidades do **java swing** com fim de utilizar as suas funcionalidades para realizar uma interface com as características acima referidas.

**Ícones:** Como é mais fácil para um Utilizador associar um ícone a uma função do projeto, por exemplo a função pesquisar é associada a uma lupa, tentamos colocar ícones sugestivos em toda a interface. Isto vem oferecer um melhor manuseamento e uma interface mais limpa e divertida. Alguns dos ícones foram mesmo desenhados por nós de forma a representarem melhor aquilo que nós queremos transmitir.

**Janelas:** Neste projeto tentamos utilizar o mais possível uma janela de forma a o utilizador não se confundir tanto. Para isso usamos um layout disponibilizado pelo java chamado **CardLayout**, este layout permite alterar os **JPanels** de acordo com o que nós queremos, não criando novas janelas pois estas iriam tornar a interface muito confusa porque teria um grande numero de janelas o que poderia induzir o próprio Utilizador a erro. Depois de alguma pesquisa e alguma prática conseguimos trabalhar com este layout de forma a deixar a interface da forma pretendida, ou seja mais simples e menos confusa.

Tal como pedido procedemos á criação de uma interface gráfica tendo em conta todas as funcionalidades da análise de requisitos.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\login.png |
| Fig. 2-Login |

|  |
| --- |
| C:\Users\Pedro\Dropbox\DSS\Interface\Favoritos.png |
| Fig. 3 - Favoritos |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\depoisLogin.png |
| Fig. 4 - Principal |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\comoPorVenda.jpg |
| Fig. 5 – Vender |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\comoLeiloar.jpg |
| Fig. 6 - Leiloar |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\Pesquisa Avançada.jpg |
| Fig. 7 - Pesquisa avançada |

|  |
| --- |
| C:\Users\Pedro\Dropbox\DSS\Interface\produto leilão.png |
| Fig. 8 - Produto em leilão |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\registar.jpg |
| Fig. 9 - Registar |

|  |
| --- |
| C:\Users\Pedro\Dropbox\DSS\Interface\produto vender.png |
| Fig. 10 - Produto à venda |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\pesquisaSimples.png |
| Fig. 11 – Pesquisa Simples |

|  |
| --- |
| C:\Users\Pedro\Dropbox\DSS\Interface\venderleiloar.png |
| Fig. 12 - A vender |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\DSS\2-Fase\Manual\inicio.png |
| Fig. 13 – Inicio  (Utilizador sem efectuar login) |

### 6.2 - Camada de Dados

Como durante as aulas da disciplina não nos foi ensinado como aceder à nossa base de dados, nós tivemos que realizar alguma pesquisa de forma a sabermos como definir os métodos para este problema.

Primeiro como nos foi sugerido, analisamos o **JDBC** do java (*Java Database Connectivity* ou JDBC é um conjunto de classes e interfaces (*API*) escritas em Java que fazem o envio de instruções *SQL* para qualquer banco de dados relacional).

Depois de analisar vimos quais seriam os métodos desta API que podíamos utilizar e que nos iriam oferecer uma maior performance e facilidade de acesso à Base de Dados.

Por fim, conciliamos o nosso código com o código *SQL*, para aceder à base de Dados com o objetivo de obter os resultados pretendidos em cada método através de *selects*, *inserts* e *updates*.

### 6.3 - Camada Business

A camada Business e uma camada logica que faz a ligação aos dados e à interface

### 6.3.1 – Class BuyKing

|  |
| --- |
| package Business;  import Data.LeiloesDAO;  import Data.ProdutosDAO;  import Data.SuspeitasDAO;  import Data.TrocasDAO;  import Data.UserDAO;  import Data.VendasDAO;  import java.math.BigInteger;  import java.security.MessageDigest;  import java.security.NoSuchAlgorithmException;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.GregorianCalendar;  import java.util.List;  import java.util.logging.Level;  import java.util.logging.Logger;  public class BuyKing {  private UserDAO \_utilizadores;  private VendasDAO \_vendas;  private LeiloesDAO \_leiloes;  private SuspeitasDAO \_suspeitas\_;  private TrocasDAO \_trocas\_;  private ProdutosDAO \_produtos;  public List<Venda> pesquisaVendasSimples(String aPchave, String aCat) throws SQLException  {  return pesquisaVendasAvançada(aPchave, aCat, 0, Float.MAX\_VALUE);  }    public List<Venda> pesquisaVendasAvançada(String aPchave, String aCat, float minP, float maxP) throws SQLException {  //throw new UnsupportedOperationException();  List<Venda> l = \_vendas.getVendasAbertas();  List<Venda> res = new ArrayList<Venda>();  for(Venda v:l)  {  if((v.getPreco()>minP)&&(v.getPreco()<maxP)&&  (aPchave.matches(v.getProduto().getNome())||aPchave.matches(v.getProduto().getDescricao()))  &&(aCat.equals(v.getProduto().getCategoria())))  res.add(v);  }  return res;  }  public List<Leilao> pesquisaLeilao(String aPchave, String aCat) {  throw new UnsupportedOperationException();  }  public boolean login(String aUsername, String aPassword) throws SQLException {  //throw new UnsupportedOperationException();  Utilizador u = \_utilizadores.get(aUsername);  String md5 = BuyKing.md5crypt(aPassword);  return ((u != null) && (u.getPassmd5().equals(md5)));  }  public boolean registar(Utilizador u) throws SQLException {  //throw new UnsupportedOperationException();  return \_utilizadores.add(u);  }  public static String md5crypt(String s) {  MessageDigest md = null;  try {  md = MessageDigest.getInstance("MD5");  } catch (NoSuchAlgorithmException ex) {  Logger.getLogger(BuyKing.class.getName()).log(Level.SEVERE, null, ex);  }  byte[] digest = md.digest(s.getBytes());  BigInteger big = new BigInteger(1, digest);  String md5 = big.toString(16);  while (md5.length() < 32) {  md5 = "0" + md5;  }  return md5;  }  public boolean registaCompra(Utilizador u, Venda v) throws SQLException {  v.setComprador(u);  GregorianCalendar d = new GregorianCalendar();  d.add(GregorianCalendar.DAY\_OF\_MONTH, 7); //7 dias para conclusao da venda  v.setDataLimiteVenda(d);  boolean res = \_vendas.update(v);  return res;  }    public boolean vender(Utilizador u, Produto p, float preco) throws SQLException  {  boolean res=false;  List<Produto> pr = \_produtos.getFromUser(u);  if(pr.contains(p)&&!\_vendas.aVenda(p)&&!\_leiloes.emLeilao(p))  {  Venda v = new Venda(VendasDAO.getNewId(),preco, new GregorianCalendar(), null, null, null,p, u, null);  res=\_vendas.add(v);  }  return false;  }    public boolean licitar(Utilizador u, Leilao l, float v) throws LeilaoFechadoException, BaixaLicitacaoException, SQLException  {  boolean res=false;  if(!l.fechado())  {  Licitacao lit = new Licitacao(u, v, new GregorianCalendar());  res=l.registaLicitacao(lit);  if(res&&(v>=l.getTecto()))  {  GregorianCalendar d = new GregorianCalendar();  d.add(GregorianCalendar.DAY\_OF\_MONTH, 7);  l.setDataLimiteLeilao(d);  \_leiloes.update(l);  }  }  return res;  }    public boolean leiloar(Utilizador u, Produto p, float base, float tecto) throws SQLException  {  boolean res=false;  List<Produto> pr = \_produtos.getFromUser(u);  if(pr.contains(p)&&!\_vendas.aVenda(p)&&!\_leiloes.emLeilao(p))  {  GregorianCalendar hoje = new GregorianCalendar();  GregorianCalendar fecho = (GregorianCalendar) hoje.clone();  fecho.add(GregorianCalendar.DAY\_OF\_MONTH,14);  GregorianCalendar limite = (GregorianCalendar) fecho.clone();  limite.add(GregorianCalendar.DAY\_OF\_MONTH, 7);  Leilao l = new Leilao(LeiloesDAO.getNewId(), u, p, hoje, fecho, limite, null, null, base, tecto);  res=\_leiloes.add(l);  }  return res;  }  } |

# 7 – Conclusão

Ao longo da realização deste trabalho, deparamo-nos com inúmeras dificuldades, que fomos superando à medida que nos tornamos mais familiarizados com a UC.

Como esperávamos a elaboração desta modelação gerou várias discussões, principalmente a nível da escolha das entidades *vs* atributos do modelo de domínio, na escolha correcta das actividades dos actores,???????????????????. Após várias tentativas e optando pela simplicidade, ficamos com os diversos modelos simples e de fácil compreensão.

O grupo conseguiu ter sempre em mente os objectivos a que se propôs e a cumpri-los com sucesso, contribuindo assim para um bom trabalho e uma boa formação individual.

Após a realização deste trabalho, chegamos à conclusão que adquirimos conhecimentos importantes sobre a planificação e modelação de projectos complexos. Os conhecimentos adquiridos durante a realização do projecto e a frequência da Unidade Curricular permitir-nos-ão abordar problemas de uma forma mais metódica e standardizada.

# 8 – Anexos

### 6.3.2 – Class Leilão

|  |
| --- |
| package Business;  import Data.LicitacoesDAO;  import java.sql.SQLException;  import java.util.GregorianCalendar;  public class Leilao {    private int id;  private Utilizador leiloador;  private Produto p;  private GregorianCalendar dataLeilao;  private GregorianCalendar \_dataFecho;  private GregorianCalendar \_dataLimiteLeilao;  private GregorianCalendar dataPagamento;  private GregorianCalendar dataEnvioProduto;  private float \_base;  private float \_tecto;  private LicitacoesDAO licitacoes;  public Leilao(int id, Utilizador leiloador, Produto p, GregorianCalendar dataLeilao, GregorianCalendar \_dataFecho, GregorianCalendar \_dataLimiteLeilao, GregorianCalendar dataPagamento, GregorianCalendar dataEnvioProduto, float \_base, float \_tecto) {  this.id = id;  this.leiloador = leiloador;  this.p = p;  this.dataLeilao = dataLeilao;  this.\_dataFecho = \_dataFecho;  this.\_dataLimiteLeilao = \_dataLimiteLeilao;  this.dataPagamento = dataPagamento;  this.dataEnvioProduto = dataEnvioProduto;  this.\_base = \_base;  this.\_tecto = \_tecto;  this.licitacoes=new LicitacoesDAO(id);  }  } |

### 6.3.3 – Class Produto

|  |
| --- |
| package Business;  import Data.SuspeitasDAO;  import java.awt.image.BufferedImage;  public class Produto {  private int \_id;  private String \_nome;  private BufferedImage \_imagem;  private String descricao;  private String categoria;  private SuspeitasDAO suspeitas;  public Produto(int \_id, String \_nome, BufferedImage \_imagem, String descricao, String categoria) {  this.\_id = \_id;  this.\_nome = \_nome;  this.\_imagem = \_imagem;  this.descricao = descricao;  this.categoria = categoria;  suspeitas = new SuspeitasDAO(\_id);  }  } |

### 6.3.4 – Class Suspeita

|  |
| --- |
| package Business;  public class Suspeita {  private Utilizador \_queixoso;    private String \_just;  public Suspeita(Utilizador \_queixoso, String \_just) {  this.\_queixoso = \_queixoso;    this.\_just = \_just;  } |

### 6.3.5 – Class Troca

|  |
| --- |
| package Business;  import java.util.GregorianCalendar;  public class Troca {  private Utilizador \_prop;  private Utilizador \_convidado;  private Produto desejado;  private Produto oferta;  private GregorianCalendar dataProposta;  private GregorianCalendar dataConfirmacao;  private GregorianCalendar dataConclusao;  private GregorianCalendar dataLimite;  private int id;  public Troca(Utilizador \_prop, Utilizador \_convidado, Produto desejado, Produto oferta, GregorianCalendar dataProposta, GregorianCalendar dataConfirmacao, GregorianCalendar dataConclusao, GregorianCalendar dataLimite, int id) {  this.\_prop = \_prop;  this.\_convidado = \_convidado;  this.desejado = desejado;  this.oferta = oferta;  this.dataProposta = dataProposta;  this.dataConfirmacao = dataConfirmacao;  this.dataConclusao = dataConclusao;  this.dataLimite = dataLimite;  this.id = id;  }  } |

### 6.3.6 – Class Venda

|  |
| --- |
| package Business;  import java.util.GregorianCalendar;  public class Venda {  private int id;  private float \_preco;  private GregorianCalendar dataVenda;  private GregorianCalendar \_dataLimiteVenda;  private GregorianCalendar dataEnvioProduto;  private GregorianCalendar dataPagamento;  private Produto \_produto;  private Utilizador vendedor;  private Utilizador comprador;      public Venda(int id, float \_preco, GregorianCalendar dataVenda, GregorianCalendar \_dataLimiteVenda, GregorianCalendar dataEnvioProduto, GregorianCalendar dataPagamento, Produto \_produto, Utilizador vendedor, Utilizador comprador) {  this.id = id;  this.\_preco = \_preco;  this.dataVenda = dataVenda;  this.\_dataLimiteVenda = \_dataLimiteVenda;  this.dataEnvioProduto = dataEnvioProduto;  this.dataPagamento = dataPagamento;  this.\_produto = \_produto;  this.vendedor = vendedor;  this.comprador = comprador;  }  } |

### 6.3.7 – Class Utilizador

|  |
| --- |
| package Business;  import Data.ClassificacaoDAO;  import Data.FavoritosDAO;  import Data.ProdutosDAO;  import java.awt.image.BufferedImage;  import java.util.GregorianCalendar;  public class Utilizador {  private String \_username;  private String \_passmd5;  private String email;  private String localidade;  private GregorianCalendar dataNascimento;  private GregorianCalendar dataRegisto;  private BufferedImage imagem;  private ClassificacaoDAO \_classificacao;  private FavoritosDAO \_wishlist;  private ProdutosDAO meusProds;      public Utilizador(String name, String pass, String mail, String l, GregorianCalendar dN, GregorianCalendar dR, BufferedImage i)  {  \_username=name;  \_passmd5=pass;  email=mail;  localidade=l;  dataNascimento= dN;  dataRegisto=dR;  imagem=i;  \_classificacao=new ClassificacaoDAO(name);  \_wishlist=new FavoritosDAO(name);  meusProds=new ProdutosDAO();  }  } |

### 6.3.8 – Class Licitação

|  |
| --- |
| package Business;  import java.util.GregorianCalendar;  public class Licitacao {    private Utilizador u;  private float v;  private GregorianCalendar data;    public Licitacao(Utilizador user, float valor,GregorianCalendar data)  {  u=user;  v=valor;  this.data = data;  }  } |

### 6.3.9 – Class Leilao Fechado Exception

|  |
| --- |
| package Business;  public class LeilaoFechadoException extends Exception{  public LeilaoFechadoException() {  }  public LeilaoFechadoException(String message) {  super(message);  }    } |

### 6.3.10 – Class Classificação

|  |
| --- |
| package Business;  import java.util.GregorianCalendar;  public class Classificacao {  public Classificacao(Utilizador classificador, GregorianCalendar data, int valor) {  this.classificador = classificador;  this.data = data;  this.valor = valor;  } |

### 6.3.11 - Class Baixa Licitação Exception

|  |
| --- |
| package Business;  public class BaixaLicitacaoException extends Exception{  public BaixaLicitacaoException() {  }  public BaixaLicitacaoException(String message) {  super(message);  }    } |

As cores de cada figura têm a seguinte correspondência:

* Vermelho- tipo VarChar2.
* Verde – tipo Data.
* Amarelo- tipo Inteiro.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |
| Fig. 14 - Esquema conceptual da base de dados elemento a elemento |