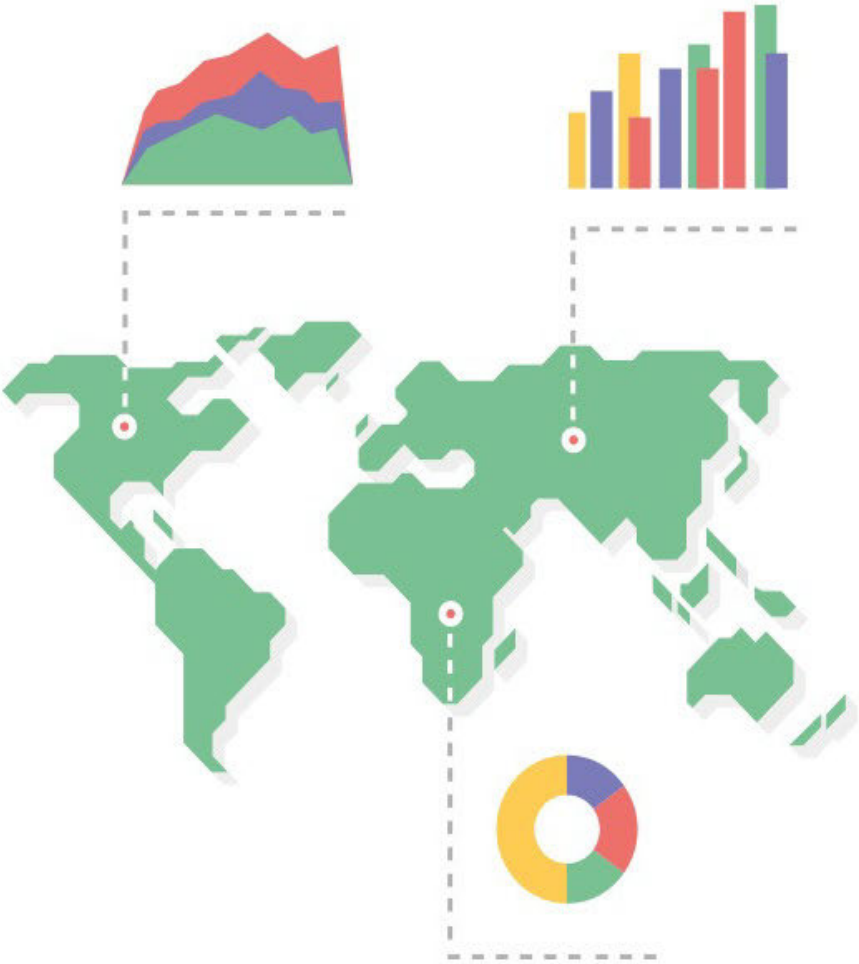
iReport

Crie relatórios práticos e elegantes



Casa do Código

MAURÍCIO MORAIS

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida
Vivian Matsui

Revisão

Bianca Hubert
Vivian Matsui

[2016]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

SOBRE O AUTOR

Maurício Morais é analista de sistemas, mestrando em Informática Aplicada e pós-graduado em desenvolvimento de sistemas com ênfase na arquitetura J2EE pela Universidade de Fortaleza. Possui mais de 20 anos de experiência profissional em desenvolvimento de software.

Tutor a distância do curso de licenciatura em Computação a Distância na Universidade Estadual do Ceará, e professor do curso de desenvolvimento web com JSF, Hibernate e iReport na Universidade de Fortaleza.

AGRADECIMENTOS

Primeiramente, agradeço a Deus, que até aqui nos tem proporcionado uma vida cheia de adversidade, mas também de vitória, acima de tudo.

Aos meus pais, Gerson Augusto de Oliveira (em memória) e Hulda Morais de Oliveira, que me ensinaram princípios e valores éticos que têm norteado a minha existência e que venho tentando repassar aos meus filhos.

A minha esposa Vera Lúcia, que sempre me apoia, incentiva e está sempre ao meu lado.

Aos meus filhos Júlia e Lucas, que a cada ano que passa sempre me enchem de orgulho com suas conquistas.

Aos meus tios Gentil e Teresa Augusto de Oliveira e família, pelo o apoio e acolhimento quando morei em sua casa no Rio de Janeiro.

Às minhas tias Raimunda, Onezilda (em memória), Amália e Noemi, que sempre apoiaram meus pais e contribuiram significativamente na minha educação.

Aos meus irmãos Marcia, Marta, Gerson Junior, Marcílio e Matheus, que sempre me incentivaram e apoiaram em situações adversas.

Aos amigos que fiz ao longo dos 15 anos que morei no Rio de Janeiro e que me proporcionaram momentos épicos. Em especial, Ricardo Braga, Carlos André e Eduardo Fonseca.

Aos meus alunos do curso de desenvolvimento web com JSF, Hibernate e iReport na Universidade de Fortaleza. Suas críticas e

sugestões contribuíram para melhoria do material didático que utilizo no curso e que serviram para nortear o conteúdo deste livro.

Ao departamento de educação continuada da Universidade de Fortaleza pela parceria no curso de desenvolvimento web com JSF, hibernate e iReport.

Ao Roberto Gadelha da Secrel de Fortaleza, que me apresentou o iReport, e que teve paciência, tranquilidade e dedicação ao me ensinar a implementar meus primeiros relatórios com iReport.

Ao editor Márcio Marcelli, que me orientou durante a escrita do livro, me passando dicas e sugestões, e tirando dúvidas.

Não poderia deixar de agradecer aos editores Paulo Silveira e Adriano Almeida, que acreditaram e apoiaram o meu projeto de escrever um livro sobre iReport.

Em fim, aos meus colegas de trabalho que, ao longo destes anos, no dia a dia contribuíram direta ou indiretamente para meu crescimento pessoal e profissional.

PREFÁCIO

Criar o design do relatório diretamente em XML pode ser uma tarefa muito demorada e improdutiva. Seria bom se existisse uma ferramenta fácil de usar e intuitiva que automatizasse esse processo. O iReport veio preencher essa lacuna, facilitando a definição e o design do relatório com um ambiente gráfico e com todos os recursos que a biblioteca Jasper oferece.

O iReport facilita a definição de relatórios com designs modernos e complexos, sem a necessidade de escrever uma linha de código em XML, e é todo gerado automaticamente. O ambiente disponibiliza para o desenvolvedor atalhos para tarefas de compilação e visualização do relatório, proporcionando a realização de testes e, consequentemente, uma maior produtividade no processo de design.

É uma ferramenta gráfica que possibilita desenhar e configurar um relatório ao arrastar e soltar componentes, de forma bem parecida com a criação de interfaces e janelas de algumas linguagens de programação com Delphi e Visual Basic.

Ao salvar, automaticamente será gerado um arquivo `JRXML` que será utilizado em uma aplicação. A vantagem é que não é necessário conhecer a fundo o `XML` a ser editado, economizando tempo de desenvolvimento. Ele também disponibiliza um conjunto de modelos (*templates*) que pode ser usado, sendo possível também escrever os próprios modelos para serem reaproveitados sempre que houver necessidade de criar um novo tipo de relatório.

Este livro apresenta, de forma didática e prática, como utilizar os recursos do iReport para implementar relatórios com Java.

O leitor aprenderá como:

1. Utilizar a interface de desenvolvimento do iReport;
2. Criar o design do relatório;
3. Criar parâmetros, atributos e variáveis em um relatório;
4. Gerar relatório utilizando como fonte de dados ArrayList;
5. Gerar relatório utilizando como fonte de dados instruções SQL;
6. Gerar relatório com gráficos;
7. Gerar relatório com sub-relatórios;
8. Gerar relatório com Map;
9. Gerar relatório com Crosstab.

O leitor deste livro terá a oportunidade de entender detalhes de como implementar relatórios utilizando iReport com extrema facilidade e produtividade.

O livro tem por objetivo mostrar para estudantes, programadores e desenvolvedores quais são os conhecimentos necessários para implementação de relatórios com Java usando um dos principais framework da arquitetura Java.

Participe das discussões sobre o livro **iReport: Crie relatórios práticos e elegantes**, em http://forum.casadocodigo.com.br, para tirar dúvidas, críticas e sugestões.

Sumário

1 Introdução **1**

1.1 JasperReports 1

1.2 Instalação do iReport 3

1.3 Interface de desenvolvimento 4

1.4 Criando o primeiro relatório 13

2 Relatório com ArrayList **19**

2.1 Aplicativo de demonstração 19

2.2 Classes que vamos utilizar 20

2.3 Métodos para visualização dos relatórios 23

2.4 Criando um relatório de listagem 26

2.5 Criando relatório com agrupamento 48

2.6 Conclusão 59

3 Relatório com SQL **60**

3.1 Criando relatório de listagem com SQL 60

3.2 Criando relatório com SQL e com parâmetro 74

3.3 Criando relatório com SQL e com agrupamento 78

3.4 Conclusão 81

4 Relatório com gráfico **82**

4.1 Criando relatório com gráfico de pizza 82

4.2 Conclusão 94

5 Relatório com sub-relatório 95

5.1 Criando relatórios com sub-relatório 95

5.2 Conclusão 106

6 Relatório com Map 108

6.1 Criando relatórios com Map 108

6.2 Conclusão 114

7 Relatório com Crosstab 116

7.1 Criando relatórios com Crosstab 116

7.2 Conclusão 123

8 Apêndice 125

8.1 Introdução Jaspersoft Studio 125

8.2 Interface do usuário 126

8.3 Novos recursos do Jaspersoft Studio não disponíveis no iReport Designer 126

9 Bibliografia 129

CAPÍTULO 1

INTRODUÇÃO

Nos próximos tópicos, enfatizaremos as principais características do JasperReports, e também vamos baixar e instalar o iReport para podermos exercitar os nossos exemplos de criação e geração de relatórios que vamos demonstrar.

1.1 JASPERREPORTS

O JasperReports é um framework *open source*, gratuito e o mais usado para geração de relatórios, capaz de criar os mais complexos relatórios para aplicações Java. Como é escrito em Java, também é multiplataforma.

Por meio de uma interface gráfica e intuitiva, o desenvolvedor é capaz de criar diversos tipos de relatórios de forma simples e rápida. Essa biblioteca proporciona uma grande facilidade na organização e apresentação de conteúdo, possibilitando a geração dinâmica de relatórios. Ela também pode ser usada em qualquer aplicação Java, como: aplicações desktop, web e distribuídas.

O iReport é um designer/construtor visual de relatórios de uso fácil e intuitivo para o JasperReports. Por intermédio desta ferramenta, podemos visualmente construir relatórios complexos contendo gráficos, imagens e sub-relatórios, uma vez que o iReport é integrado à biblioteca do JasperReports.

Dentre as funcionalidades do JasperReports, podemos destacar:

- É capaz de exportar relatórios para diversos formatos, tais como PDF, HTML, XML, XLS etc.
- Possibilita a entrada de dados em diversos formatos, tais como arquivo XML ou CSV, conexão com o banco de dados, uma sessão do Hibernate, uma coleção de objetos em memória etc.
- Possibilita o uso de imagens, gráficos e localização geográfica.

Estas características, junto com a facilidade de utilização e custo (open source), faz do JasperReports uma biblioteca de geração de relatório em Java completa.

Fluxo de geração de relatório

O layout do relatório é definido em um arquivo XML, normalmente com a extensão `.jrxml`. Este XML contém todas as informações do relatório e também os campos que serão preenchidos posteriormente de acordo com a fonte de dados utilizada (*data source)*.

As etapas para geração de um relatório são as seguintes:

1. Após compilar o relatório XML, o resultado é um objeto do tipo JasperReport.
2. Usamos a interface *data sources* para preencher o relatório com os dados.
3. O JasperReport processa o arquivo `.jasper` junto com o *data source*, tendo como resultado um objeto do tipo JasperPrint.
4. Podemos então enviar objeto do tipo JasperPrint para impressão diretamente ou exportar para um outro formato, tal como PDF, por exemplo. Na figura a seguir, ilustramos o processo de geração de relatório.

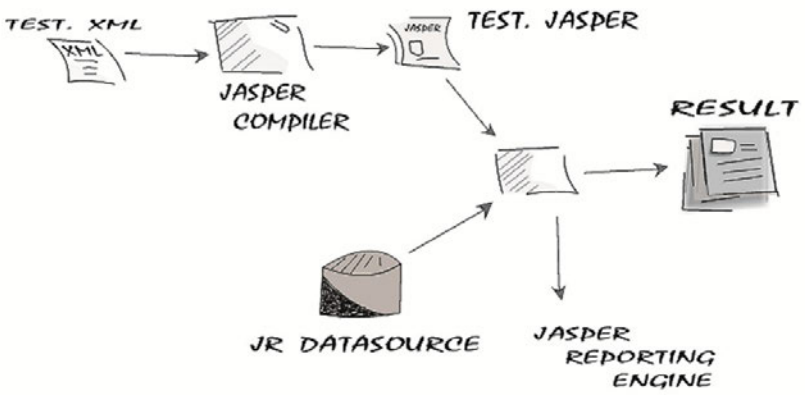


Figura 1.1: Fluxo de geração de relatório

1.2 INSTALAÇÃO DO IREPORT

Quando iniciei a escrita deste livro, a versão que estava disponível para download no site era iReport 5.6.0.

Download iReport Designer

Primeiramente, deve-se acessar o site https://community.jaspersoft.com/project/ireport-designer/.

Então, dê um clique no link `Download iReport Designer`. Você encontrará opção de download para diversos produtos da Jaspersoft, clique no link `iReport Designer`. Lá, haverá a opção de download do instalador para diversos sistemas operacionais, então, selecione a opção equivalente ao seu sistema: Windows ou Linux, conforme o caso. Faça download e salve o instalador na pasta de sua preferência.

No nosso contexto, instalamos a versão Windows: `iReport-5.6.0-windows-installer.exe` e usamos o Java JRE na versão

1.7.

Para quem utiliza o Java 1.8 ou superior, recomendamos baixar o `Jaspersoft Studio` , que tem as mesmas funcionalidades do iReport. Recentemente, foi divulgado no site https://community.jaspersoft.com que o **iReport** será descontinuado e será substituído pelo **Jaspersoft Studio**.

1.3 INTERFACE DE DESENVOLVIMENTO

Ambiente

A figura a seguir mostra a tela principal do iReport. Ela contém todos os recursos necessários para criamos o design e configuração do nosso relatório. Vamos agora entender alguns recursos disponibilizada nela.

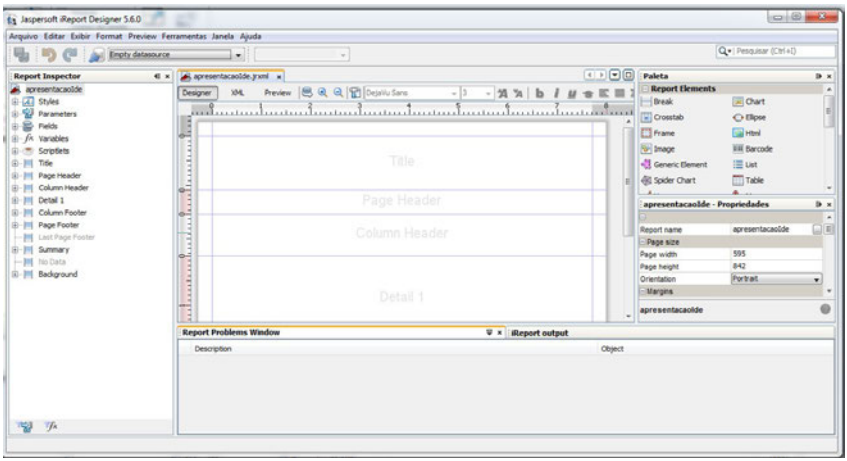


Figura 1.2: Interface de Desenvolvimento

Estrutura do relatório

Podemos ver na figura a tela que utilizamos para desenhamos o relatório. Ele está dividido em 7 seções, sendo cada uma responsável

por uma funcionalidade.

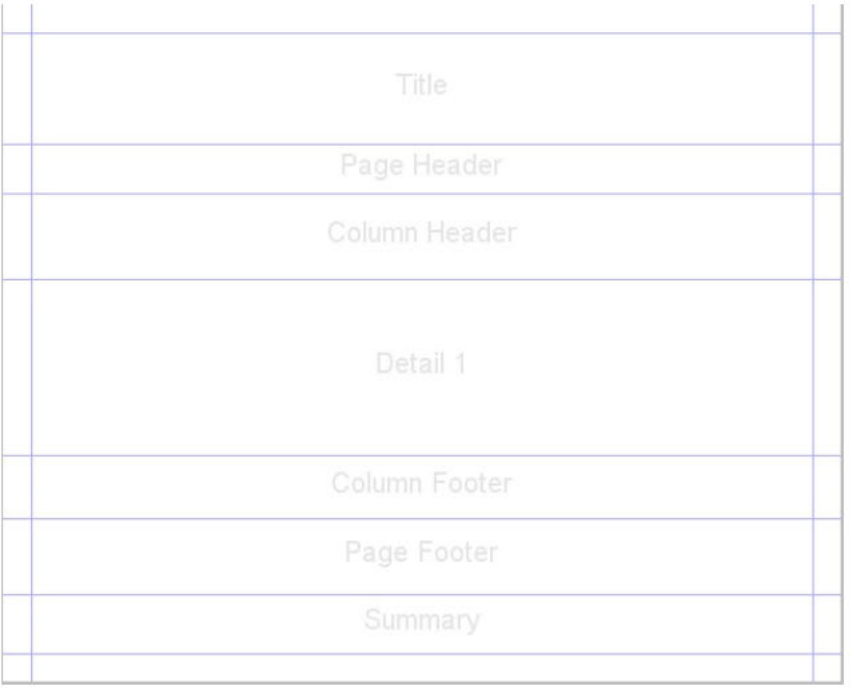


Figura 1.3: Estrutura do relatório

- **Title (título):** é a primeira seção visível na construção de um relatório vazio. Esta seção só aparece uma vez no começo do relatório.
- **PageHeader (cabeçalho da página):** esta seção aparece no começo de cada página impressa.
- **ColumnHeader (cabeçalho da coluna):** esta só aparece no começo de cada interação com a seção `Detail` .
- **Detail (detalhe):** esta seção é o local de exibição dos dados de um objeto data source ou query. Ela se repete enquanto houver linhas para serem colocadas no relatório de acordo com o resultado transmitido.
- **ColumnFooter (rodapé da coluna):** esta aparece abaixo de cada `coluna.análogo` ao `ColumnHeader` .

- **PageFooter (rodapé da página):** representa o rodapé da página. Essa seção aparece no final de cada página.
- **Summary (sumário/resumo):** representa o rodapé do relatório, só aparece uma vez ao término do relatório.

Menu Paleta

Este menu tem todos os componentes que podemos usar em um relatório.

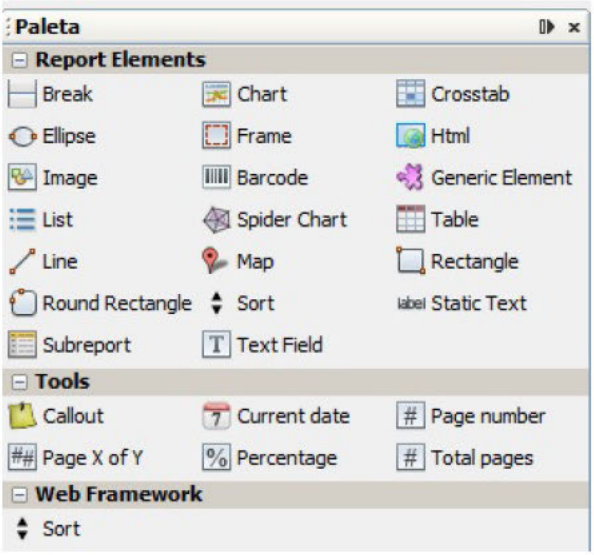


Figura 1.4: Menu Paleta

SubMenu Report Elements

Dentre os elementos disponíveis no menu Paleta, podemos destacar os seguintes componentes:

- **Static Text:** utilizamos este componente para imprimir um label, por exemplo, o título do relatório ou nome de uma coluna.
- **Text Field:** usamos este componente para imprimir o

conteúdo de um atributo, como o conteúdo do atributo código, descrição, sigla etc.

- **Image:** utilizamos este componente para imprimir uma imagem, por exemplo, o logo da empresa ou a foto de um cliente.
- **Chart:** usamos este componente para imprimir gráficos.
- **Subreport:** um relatório pode ser composto de N sub-relatórios. Utilizamos este componente para criar N sub-relatórios.

SubMenu Tools

O iReport disponibiliza algumas variáveis que usamos no nosso dia a dia. Dentre elas, podemos destacar:

- **Current date:** utilizamos esta variável para imprimir a data atual, como a data de impressão do relatório.
- **Pager number:** usamos esta variável para imprimir o número da página.
- **Page X of Y:** utilizamos esta variável para imprimir o número da página com contagem em regressiva, por exemplo, 1 de 10, 2 de 10, 3 de 10 etc.
- **Total pages:** usamos esta variável para imprimir o número total de página do relatório.

Menu Report Inspector

A seguir, veja o menu que usamos para adicionar e configurar recursos ao relatório.

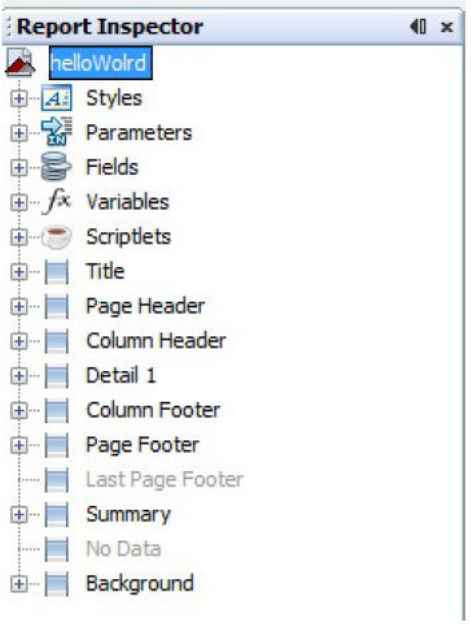


Figura 1.5: Menu Report Inspector

Dentre os recursos que podemos adicionar ao relatório, podemos destacar:

- **Parameters:** nesta opção, adicionamos os parâmetros, por exemplo, o período do relatório, o path de uma imagem, o nome da empresa.
- **Fields:** aqui, adicionamos os atributos que usaremos no relatório.
- **Variables:** nesta opção, adicionamos as variáveis que vamos utilizar no relatório, como Total Faturamento Diário, Total da População de um Estado etc.

Menu Propriedades

Utilizamos este menu para configurar os componentes que adicionamos ao relatório.

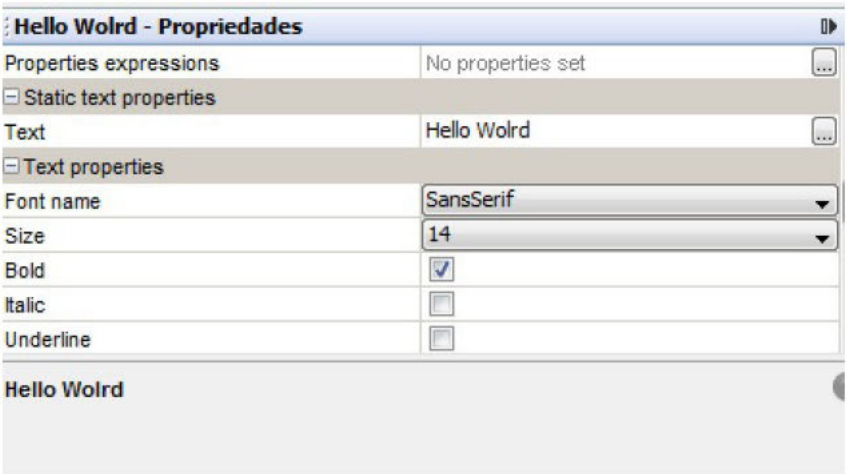


Figura 1.6: Menu Propriedades

Dentre as propriedades de um componente Static text, por exemplo, podemos destacar:

- **Font name:** selecionamos o tipo de fonte.
- **Size:** selecionamos o tamanho da fonte.
- **Bold:** opcionalmente, podemos imprimir um label em negrito.

Configuração da página

Para mudarmos a configuração da pagina do relatório, precisamos ir ao menu do iReport. Após criar um relatório, clique na opção `Arquivo`. Será visualizado um submenu, clique na opção `Configurar página`.

Nesta tela, podemos configurar a página de acordo com as nossas necessidades:

- **Orientação:** disponibiliza 2 opções de orientação.
- **Tamanho:** disponibiliza um lista de tamanhos.
- **Margens:** podemos alterar as margens.

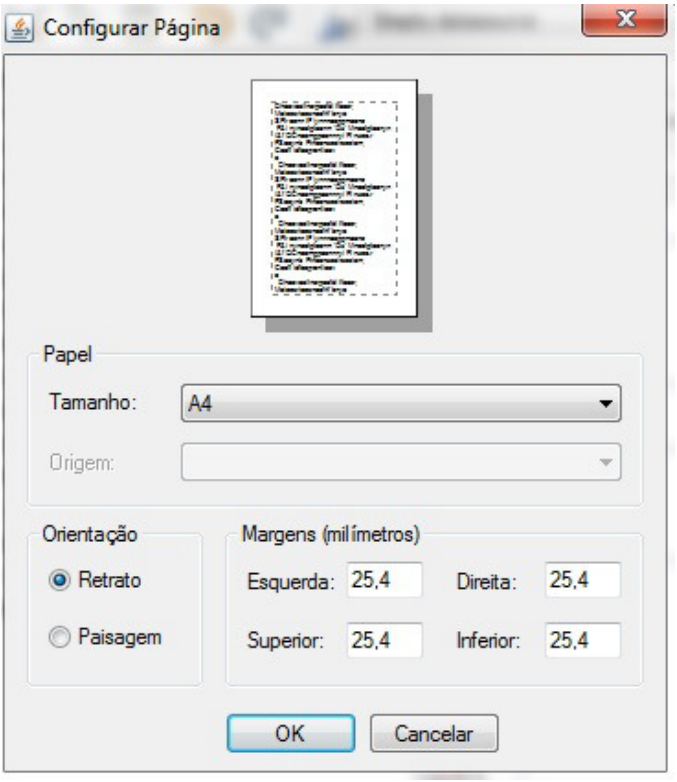


Figura 1.7: Configurando a página

Adicionando menu na Interface de Desenvolvimento

Observe a figura seguinte no lado direito, não temos a opção do menu Paleta, mas precisamos dele para adicionar componentes ao relatório. Neste caso, o que fazer?

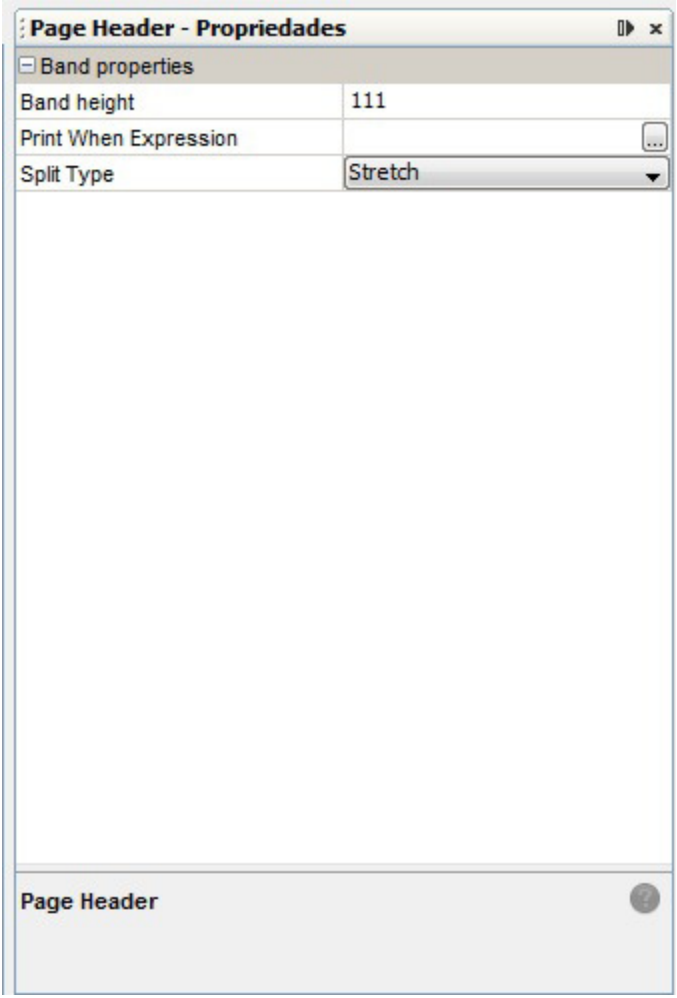


Figura 1.8: Adicionando Menu

No menu do iReport, selecione a opção `Janela`. Será disponibilizado o submenu com diversas opções. Sempre que precisarmos adicionar recursos a nossa interface de desenvolvimento, utilizamos a opção `Janela` do menu do iReport. No nosso contexto, para adicionar o menu `Paleta`, temos duas opções:

1. Clicar na opção `Paleta` .
2. Clicar na opção `Redefinir Janelas` .

Redimensionando área

Para redimensionar as áreas dos relatórios, temos duas opções:

1. Visualmente, clicando nas linhas que separa as áreas e arrastando para cima ou para baixo.
2. No menu `Report Inspector` , clique na área desejada e, em seguida, no menu `Propriedades` , altere o valor da propriedade `Band height` , informando o valor desejado.

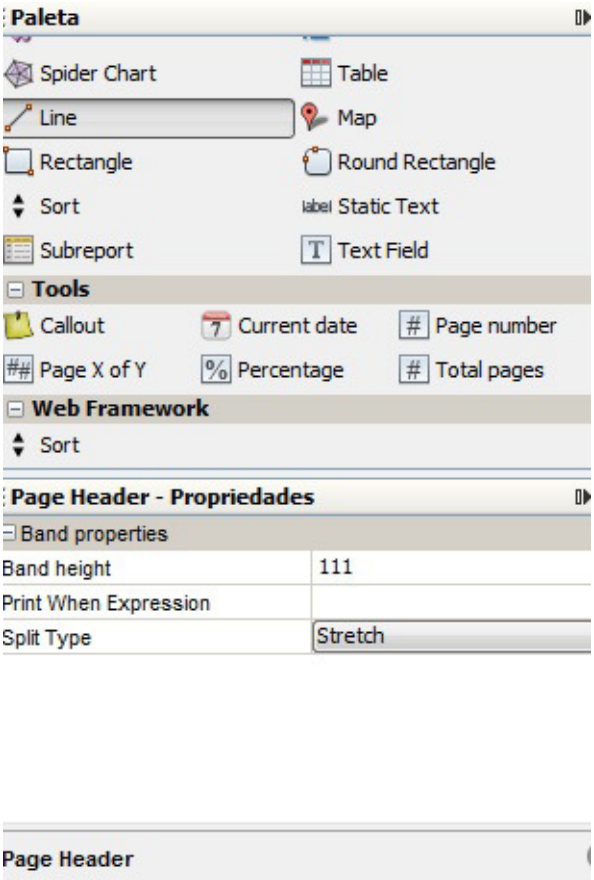


Figura 1.9: Redimensionando

1.4 CRIANDO O PRIMEIRO RELATÓRIO

Vamos, agora, criar nosso primeiro relatório, que chamaremos de `Hello World`.

Hello World

Esta é a tela inicial do iReport e, a partir dela, damos início à criação de relatórios.

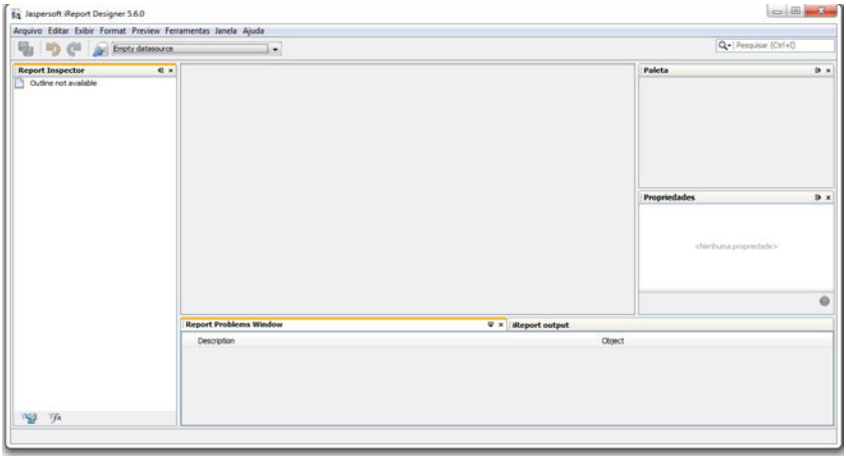


Figura 1.10: Tela inicial do iReport

Após um clique na opção `arquivo`, será visualizado um submenu com as seguintes opções habilitadas: `New`, `Open`, `Open Recent File`, `Configurar página` e `Sair`. Clique na opção `New`.

A figura a seguir mostra a tela em que você define o nome, o relatório e a pasta de sua preferência onde você salvará o arquivo do relatório. Neste caso, o nome do relatório é `helloWorld`, e estou gravando na pasta `C:\Mauricio\Ireport\Relatorio`.

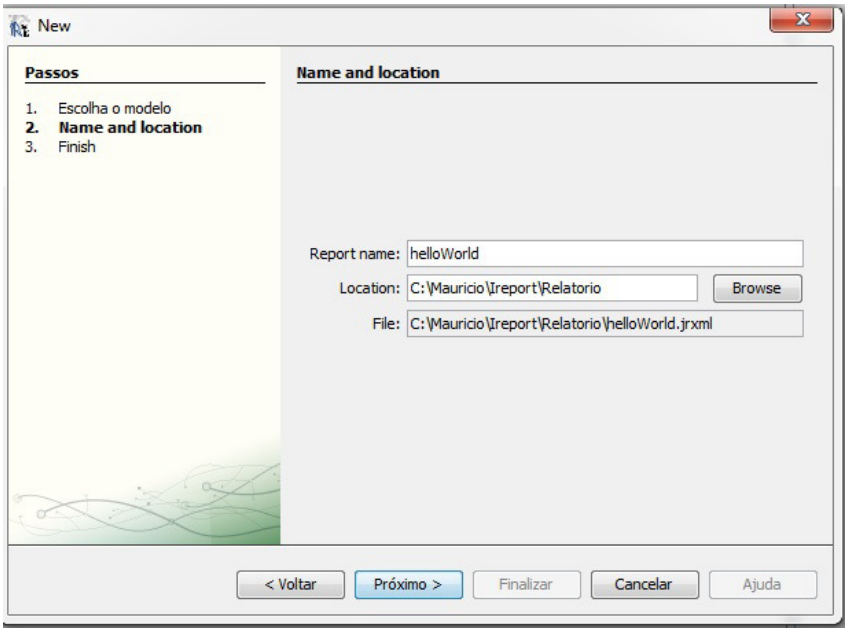


Figura 1.11: Definindo o nome do relatório

Na tela mostrada pela figura adiante, faremos o design do nosso relatório, pois ela contém todos os recursos necessários para criamos o design e a configuração do nosso relatório *Hello World*.

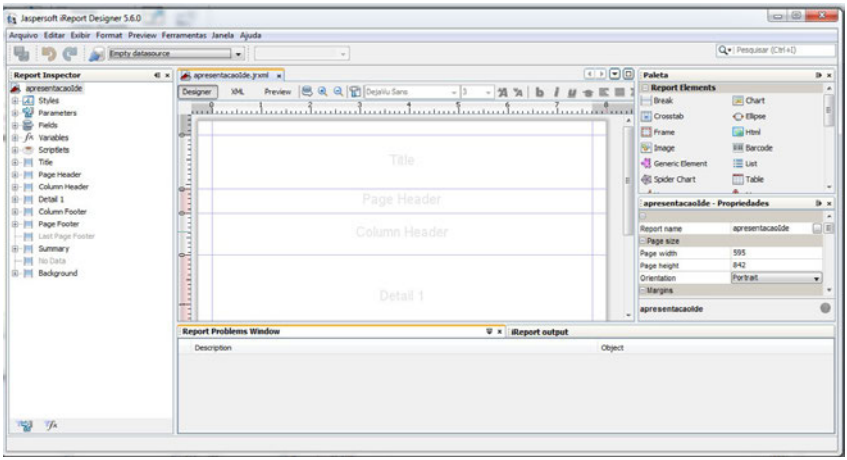


Figura 1.12: Interface de desenvolvimento

Veja a seleção dos componentes na figura a seguir.

1. Clique no componente `Static Text` no menu `Paleta`, e arraste para a seção `Title` do relatório.
2. Dê um clique no componente `Static Text` que colocamos na seção `Title`, e altere a descrição para *Hello World*.

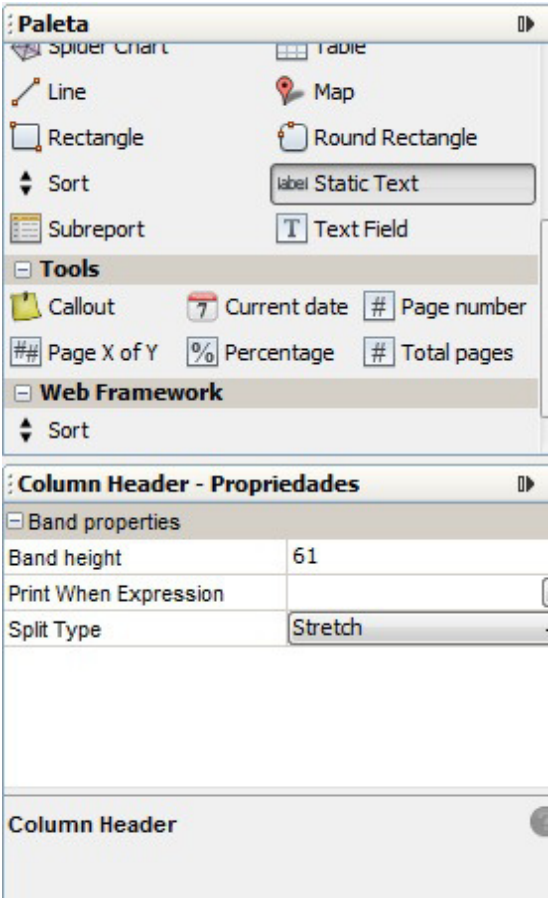


Figura 1.13: Adicionando componente

Com o componente Static Text selecionado, vamos agora colocar a descrição em negrito, selecionar a fonte e aumentar seu tamanho.

Menu Propriedade

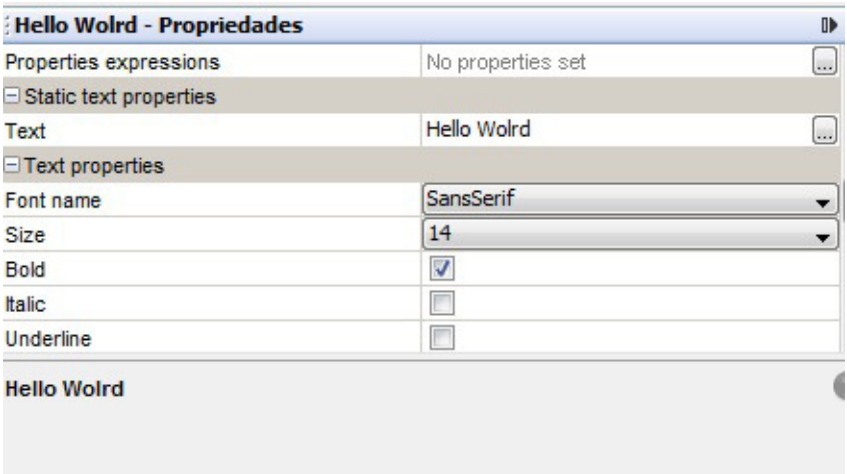


Figura 1.14: Configurando propriedades

1. Na propriedade `Fonte name`, selecione a fonte de sua preferência.
2. Na propriedade `Size`, selecione o tamanho de sua preferência.
3. Marque a propriedade `Bold` para descrição ser impressa em negrito.

Dê um clique na aba `XML` e será mostrado o XML do relatório:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    <property name="ireport.zoom" value="1.0"/>
    <property name="ireport.x" value="0"/>
    <property name="ireport.y" value="0"/>
    <style name="style1"/>
    <background>
        <band splitType="Stretch"/>
    </background>
    <title>
        <band height="79" splitType="Stretch">
            <staticText>
                <reportElement x="125" y="26" width="161" height="20" uuid="ff71232b-ff7d-48ff-b5f5-d83c1a1838e7"/>
                <textElement textAlignment="Justified">
                    <font size="14" isBold="true"/>
                </textElement>
                <text><![CDATA[Hello World]]></text>
            </staticText>
        </band>
    </title>
    <pageHeader>
```

Figura 1.15: Hello World XML

Dê um clique na aba `Preview` e será mostrado o relatório em tempo de execução:

Figura 1.16: Hello World

Agora que instalamos o iReport, aprendemos alguns recursos que a IDE proporciona, criamos e executamos o relatório *Hello World*, podemos partir para os próximos capítulos em que vamos aprender a criar relatórios um pouco mais complexos.

CAPÍTULO 2

RELATÓRIO COM ARRAYLIST

2.1 APLICATIVO DE DEMONSTRAÇÃO

Para que possamos testar e rodar os relatórios que vamos criar no decorrer dos capítulos deste livro, implementaremos um aplicativo de demonstração com todas as funcionalidades necessárias para poder executá-los.

O aplicativo será implementado utilizando tecnologia Java web. Os relatórios que implementaremos fazem parte do Sistema de Gestão de Cliente, um aplicativo de E-mail Marketing da startup MMO DEVELOPER, especializada em desenvolvimento de software localizada em Fortaleza – CE , para gerenciar o relacionamento de uma empresa com seus clientes.

O aplicativo será implementado utilizando os seguintes recursos:

1. Tomcat como servidor de aplicação – Para maiores informações sobre Tomcat acesse o link http://tomcat.apache.org/;
2. JSF 2.0 com ênfase em *primefaces* – Para maiores informações sobre *primefaces* acesse o link http://www.primefaces.org/showcase/;
3. Como banco de dados, será usado o PostgresSQL, para

armazenar as informações do sistema – Para maiores informações sobre PostgresSQL, acesse o link http://www.postgresql.org/.

O aplicativo está disponível para download no GitHub em: https://github.com/mmmauricio/ireportCrieRelatoriosPraticosEelegantes.

Adicionando Libs

Para o perfeito funcionamento de relatórios `*.jasper` em uma aplicação Java, faz-se necessário adicionar as *libs* mostradas na sequência.

- `commons-beanutils`
- `commons-collections`
- `commons-digester`
- `commons-logging`
- `Groovy-all`
- `iText`
- `poi`
- `jasperreports`
- `jcommon`
- `jfreechart`

As libs `jcommon` e `jfreechart` são obrigatórias apenas em relatórios com gráficos; nos outros, elas são opcionais. Entretanto, é sempre bom adicioná-las também.

Como o contexto de nossa aplicação é uma aplicação web, é preciso adicionar a lib `servlet-api` à nossa aplicação para que possamos exibir o relatório.

2.2 CLASSES QUE VAMOS UTILIZAR

Nossa aplicação de demonstração é baseada na arquitetura MVC e é composta das seguintes classes:

Camada model

- `UF` – Responsável pelas informações da UF (Unidade Federativa).
- `Municipio` – Responsável pelas informações do Município.
- `Cliente` – Responsável pelas informações do Cliente.

UF

Para implementar nossa classe `UF`, usaremos o seguinte código:

```
public class UF {

    private Integer id;
    private String nome;
    private String sigla;

    Get e Set
}
```

Município

Para implementar nossa classe `Municipio`, usaremos o seguinte código:

```
public class Municipio {

    private Integer id;
    private String nome;
    private UF uf;
    private Long populacao;

    Get e Set
}
```

Cliente

Para implementar nossa classe `Cliente` , usaremos o seguinte código:

```
public class Cliente {

    private Integer id;
    private String nome;
    private String endereco;
    private String observacao;
    private Double dlatitude;
    private Double dlongitude;
    private Municipio municipio;
    private UF uf;

     Get e Set

}
```

Camada DAO

Na camada DAO ficam as nossas classes que interagem com o banco de dados, realizando as inclusões, alterações, consultas e exclusões.

- `UFDao` – Responsável pela Inclusão, Alteração, Consulta e Exclusão das informações da UF.
- `MunicipioDao` – Responsável pela Inclusão, Alteração, Consulta e Exclusão das informações do Município.
- `ClienteDao` – Responsável pela Inclusão, Alteração, Consulta e Exclusão das informações do Cliente.

Camada controle

Na camada de controle ficam as classes responsáveis por capturar as requisições vindas da camada de visão.

- `AbstractMB` – Classe abstrata que implementaremos

com todos os métodos que serão herdados pelas demais classes controle.

- `UFMB` – Responsável por capturar as requisições vindas da camada de visão e repassar para classe `UFdao` e, após o retorno da classe, `UFDao` retorna o resultado para a camada de visão.

- `MunicipioMB` – Responsável por capturar as requisições vindas da camada de visão e repassar para classe `MunicipioDao` e, após o retorno da classe, `MunicipioDao` retorna o resultado para a camada de visão.

- `ClienteMB` – Responsável por capturar as requisições vindas da camada de visão e repassar para classe `ClienteDao` e, após o retorno da classe, `ClienteDao` retorna o resultado para a camada de visão.

2.3 MÉTODOS PARA VISUALIZAÇÃO DOS RELATÓRIOS

Como vimos anteriormente, um relatório com JaperReport pode ter diversas fontes de dados. Neste capítulo, a fonte de dados que usaremos será ArrayList, ou seja, uma estrutura de armazenamento que pode conter N objetos.

Método gerarRelatorio

Na classe `AbstractMB`, adicionamos o método `gerarRelatorio`. Este implementa o tópico *Fluxo de geração de relatório* que vimos no capítulo anterior.

O método `gerarRelatorio` é responsável pela execução e visualização do relatório, como também por acrescentar parâmetros

comuns para todos os relatórios, como, por exemplo, `imagemLogo` .

Inicialmente, instanciamos a classe `HttpServletResponse` para que o usuário possa visualizar o relatório.

```
public void gerarRelatorio(String nomeRelatorio
HashMap paramRel,
List listaRel)
throws Exception {
    FacesContext context =
    FacesContext.getCurrentInstance();
    HttpServletResponse response =
        (HttpServletResponse) context.getExternalContext()
                                    .getResponse();
    ServletContext sc =
        (ServletContext) context.getExternalContext()
                                .getContext();
    String relPath = sc.getRealPath("/");
    String imagemLogo =
        relPath + "resources/imagens/logo_mmo.jpg";
    paramRel.put("imagemLogo", imagemLogo);
    paramRel.put("nmSistema", Constants.NOME_SISTEMA);
    paramRel.put("REPORT_LOCALE", new Locale("pt", "BR"));
    JasperPrint print = null;
```

A classe `JRBeanCollectionDataSource` transforma o `Arraylist` em um `datasource` e, em seguida, a classe `JasperFillManager` gera o relatório.

```
    JRBeanCollectionDataSource rel =
        new JRBeanCollectionDataSource(listaRel);
    print = JasperFillManager.
        fillReport(relPath + "relatorios/" + nomeRelatorio +
            ".jasper", paramRel,rel);
```

Com o relatório criado, nesse contexto configuramos o objeto `response` para mostrar o relatório no formato `.pdf` , e a classe `JasperExportManager` exporta o objeto `print` para `.pdf` .

```
    response.setContentType("application/pdf");
    response.addHeader("Content-disposition", "attachment;
    filename=\"" + nomeRelatorio + ".pdf\"");
    JasperExportManager.exportReportToPdfStream(print,
    response.getOutputStream());
```

```
    ServletOutputStream responseStream =
            response.getOutputStream();
    responseStream.flush();
    responseStream.close();
    FacesContext.getCurrentInstance().renderResponse();
    FacesContext.getCurrentInstance().responseComplete();
}
```

Classe UFMB

Na classe `UFMB`, adicionamos o método `relatório`, que será responsável pela chamada da consulta, pela preparação dos parâmetros do relatório e pela chamada do método `gerarRelatório`. Nela passamos os seguintes parâmetros: o nome do relatório que deve ser visualizado, os parâmetros do relatório e o arraylist com as informações que serão disponibilizadas.

```
public class UFMB extends AbstractMB {
    public  void relatorio() throws Exception {
        try {
            List<UF> listagemResultado = ufDao.consulta(uf);

            HashMap paramRel = new HashMap();
            String nomeRelatorio = "relUF";
            gerarRelatorio(nomeRelatorio, paramRel,
                listagemResultado);
        } catch (NegocioException e) {
            addMsgErro(e.getMessage());
        }
    }
}
```

UF.xhtml

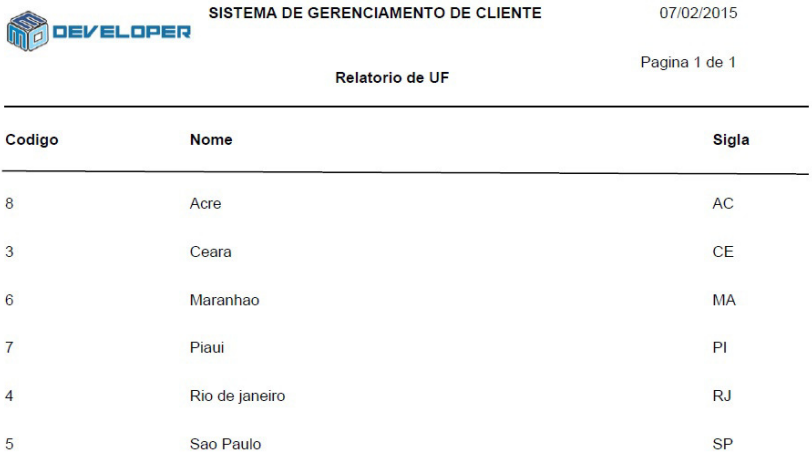
Na página `UF.xhtml` de nossa aplicação, incluímos um botão da tag `PrimeFaces` para poder iniciar a execução do relatório.

```
<p:commandButton
    value="Relatório" ajax="false"
    actionListener="#{ufMB.relatorio}">
</p:commandButton>
```

2.4 CRIANDO UM RELATÓRIO DE LISTAGEM

Layout do relatório

Nosso desafio inicial é implementar o relatório de unidade federativa, conforme layout da figura a seguir.



SISTEMA DE GERENCIAMENTO DE CLIENTE 07/02/2015

Pagina 1 de 1

Relatorio de UF

Codigo	Nome	Sigla
8	Acre	AC
3	Ceara	CE
6	Maranhao	MA
7	Piaui	PI
4	Rio de janeiro	RJ
5	Sao Paulo	SP

Figura 2.1: Layout do relatório

O relatório é composto de um cabeçalho, que contém:

1. Uma imagem com o logo da empresa;
2. O nome do sistema;
3. A data de impressão;
4. Contador de páginas;
5. O nome de relatório.

Em seguida, uma área é reservada para colocar o nome das colunas do relatório. Normalmente, é o nome que identifica o atributo que será mostrado na mesma coluna na área `Detail`. Finalmente, temos a área `Detail` que é onde colocamos as informações da UF que serão mostradas no relatório, que é o seu objetivo principal.

Adicionando parâmetros no relatório

Para a implementação do nosso relatório, vamos usar a interface de desenvolvimento do iReport. Como já foi dito anteriormente, fazer o relatório diretamente em um arquivo `xml` é uma tarefa muito custosa e improdutiva. Para isso, utilizaremos a interface do iReport por ser bastante intuitiva e de fácil manuseio.

Após criar o relatório `relUF`, estando na interface de desenvolvimento no menu `Report Inspector`:

Primeiramente, vamos adicionar o parâmetro `imagemLogo` que será usada para mostrar a imagem do logo da empresa.

1. Clique na opção `Parameter`.
2. Dê o clique no botão direito do mouse.
3. Será visualizado um submenu, conforme figura adiante.
4. Clique na opção `Adicionar Parameter`.
5. Em seguida, será visualizada a figura seguinte:

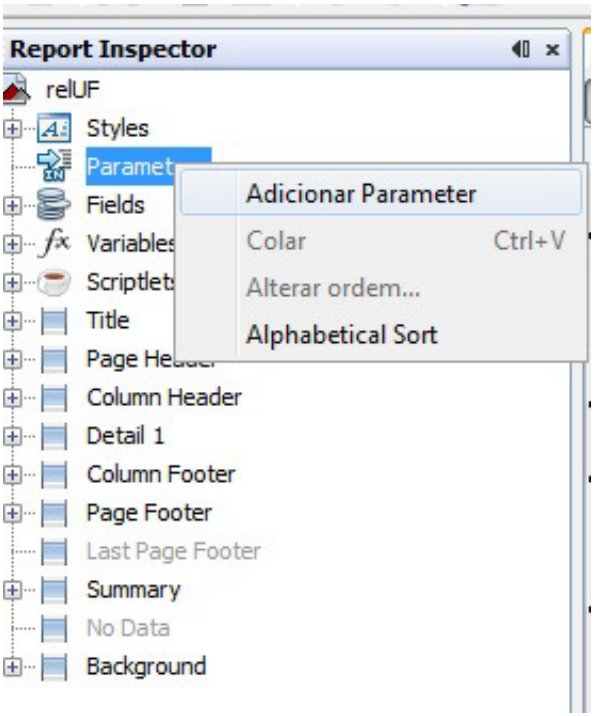


Figura 2.2: Adicionando Parameter

No menu de propriedades, vamos configurar o parâmetro, conforme pode ser visto a seguir.

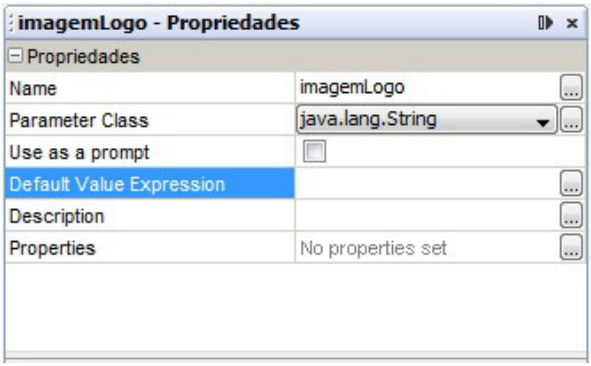


Figura 2.3: Configurando o Parameter imagemLogo

1. Na propriedade `name`, colocaremos o nome do parâmetro, neste caso, `imagemLogo`.
2. Na propriedade `parameter Class`, visualizaremos uma lista de classes. Selecione a classe `String`.

Adicione parâmetro `nmSistema` que será utilizada para mostrar o nome do sistema.

1. Na propriedade `name`, vamos colocar o nome do parâmetro, neste caso, `nmSistema`.
2. Na propriedade `parameter Class`, visualizaremos uma lista de classes. Selecione a classe `String` novamente.

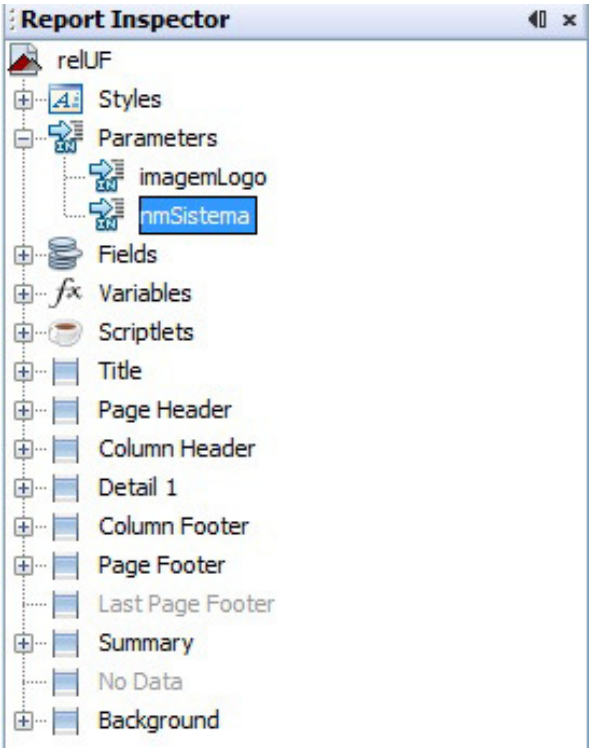


Figura 2.4: Configurando o Parameter nmSistema

Adicionamos e configuramos os parâmetros do relatório. Agora,

ele está apto para receber parâmetros. O importante é que haja uma correspondência entre o nome dos parâmetros no iReport e o nome do parâmetros no HashMap passado com parâmetro para o iReport.

Adicionando fields no relatório

Estando na interface de desenvolvimento, no menu `Report Inspector`:

1. Clique na opção `Field`.
2. Dê o clique no botão direito do mouse.
3. Será visualizado um submenu, conforme a figura adiante.
4. Clique na opção `Adicionar Field`.
5. Em seguida, será vista a figura a seguir.

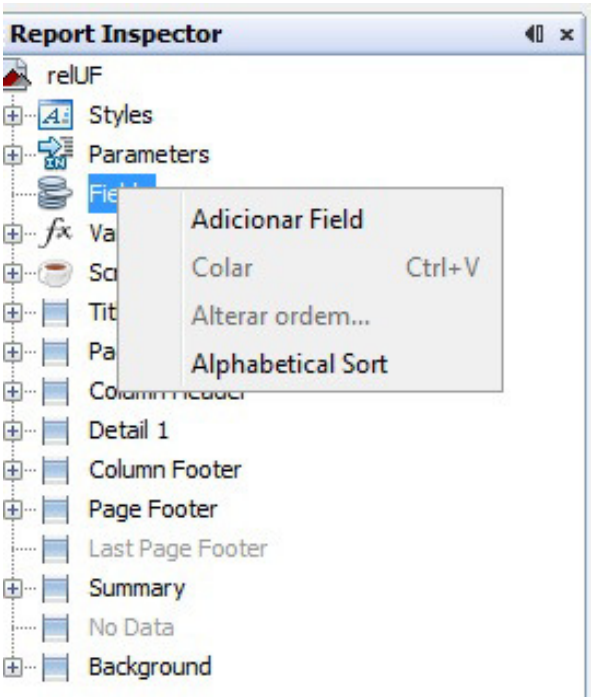


Figura 2.5: Adicionando fields

No menu de propriedades, vamos configurar o `field` .

É oportuno lembrar de que, no nosso contexto, a fonte de dados será um ArrayList. Assim, o nome do atributo (`field`) deve ser o mesmo nome do atributo do objeto armazenado no ArrayList. Ou seja, deve haver uma correspondência entre o nome dos atributos da classe `UF.java` e o dos `fields relUF.jrxml` .

1. Na propriedade `name` , colocaremos o nome do `field` de `id` .
2. Na propriedade `Parameter Class` , visualizamos uma lista de classes. Selecione a classe `Integer` .

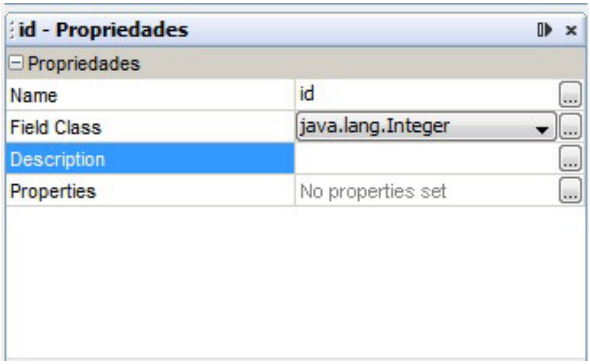


Figura 2.6: Configurando o field id

Agora, vamos adicionar outro `Field` . Assim, na propriedade `name` , colocaremos o nome do `field` de nome e o do `field` de sigla. Já na propriedade `Parameter Class` , visualizamos uma lista de classes. Selecione a classe `String` para ambos.

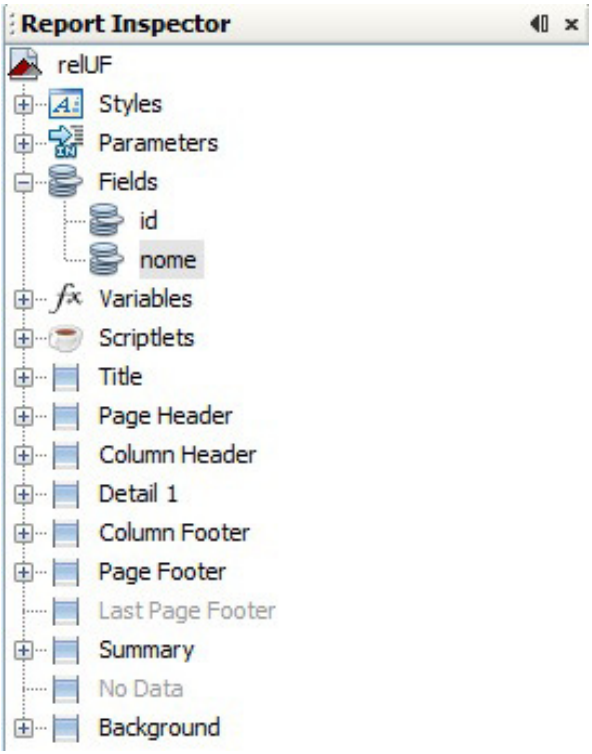


Figura 2.7: Configurando field nome

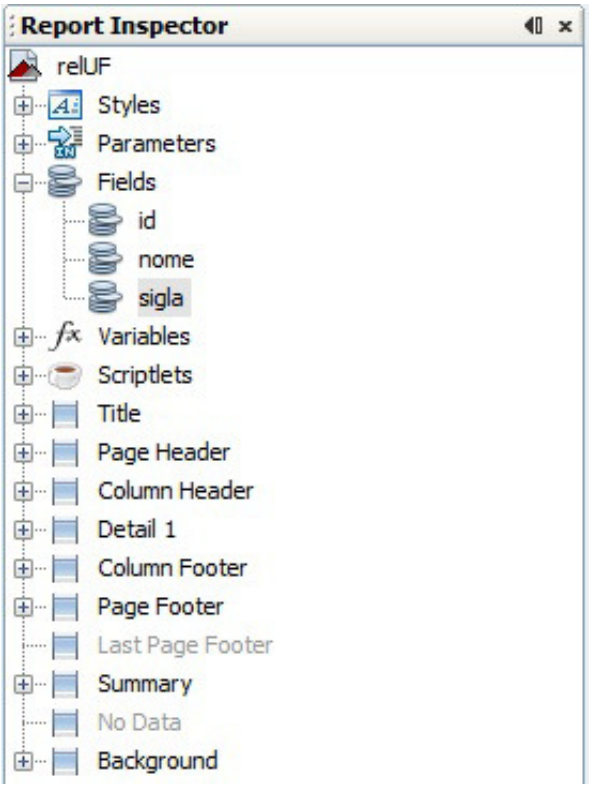


Figura 2.8: Adicionando o field sigla

Agora que adicionamos os parâmetros e atributos, o nosso relatório está apto para receber os parâmetros e atributos.

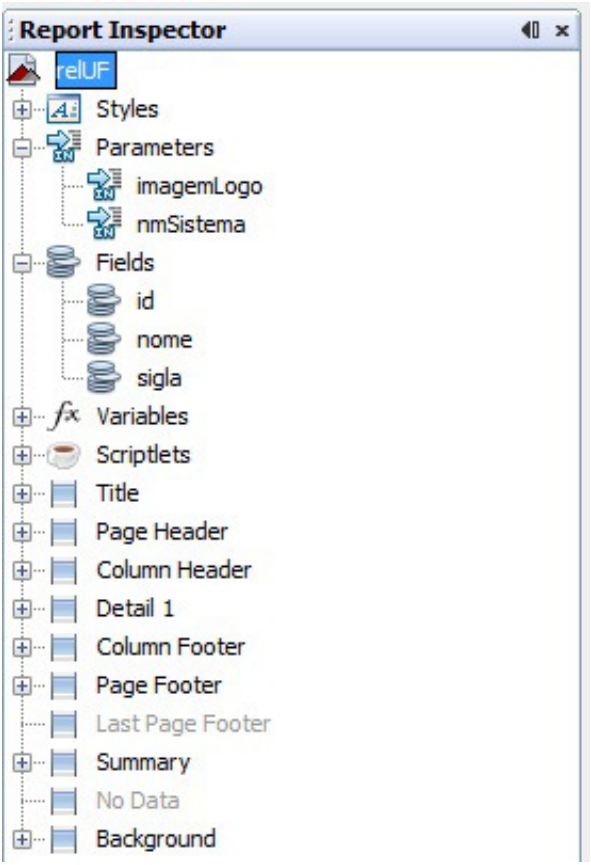


Figura 2.9: Parâmetros e atributos adicionados

Cabeçalho do relatório

Vamos agora iniciar a implementação do cabeçalho. Na interface de desenvolvimento, no menu `Report Inspector` :

1. Clique no parâmetro `nmSistema` .
2. Arraste o parâmetro `nmSistema` para a área `Page Header` .
3. Redimensione e posicione o parâmetro `nmSistema` .

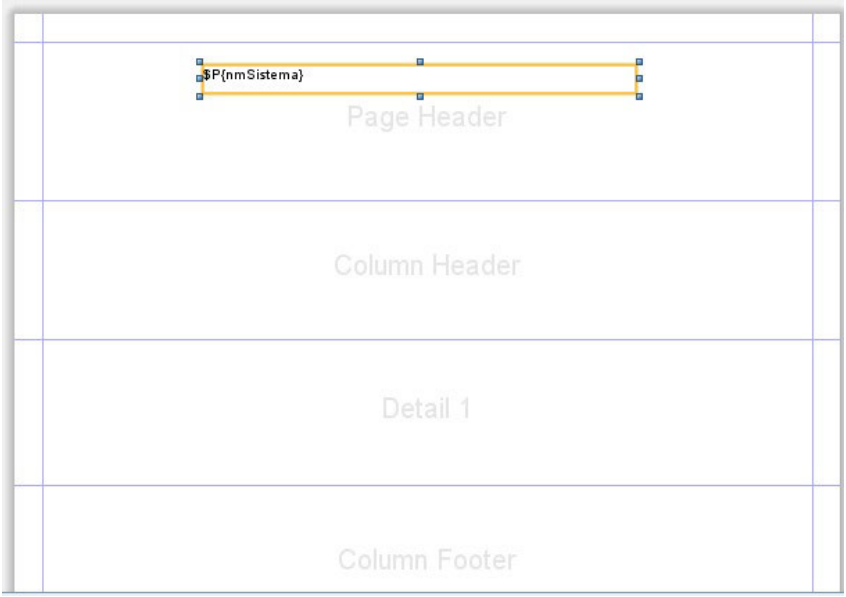


Figura 2.10: Adicionando o parâmetro nmSistema no cabeçalho

Agora, no menu `Paleta` .

1. Clique no componente `Image` .
2. Arraste o componente `Image` para a área `Page Header` .
3. Quando soltá-lo na área, será disponibilizada uma tela conforme a figura a seguir.

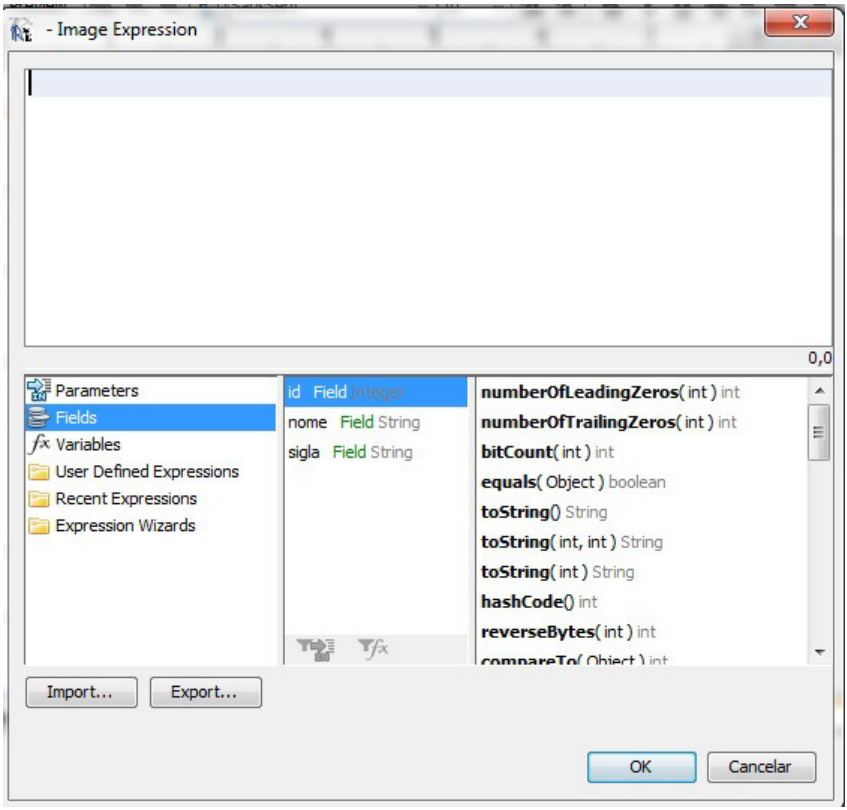


Figura 2.11: Adicionando o componente Image no cabeçalho

Nesta tela, são dispostas opções de seleção para associarmos ao componente `Image` .

Então, clique na opção `Parameters` . Na parte inferior no centro, será visualizada uma lista de parâmetros. Dê um duplo clique na opção `imagemLogo` , certifique-se de que o nome do parâmetro `imagemLogo` está sendo visualizado na parte superior da tela, e clique no botão `Ok` .

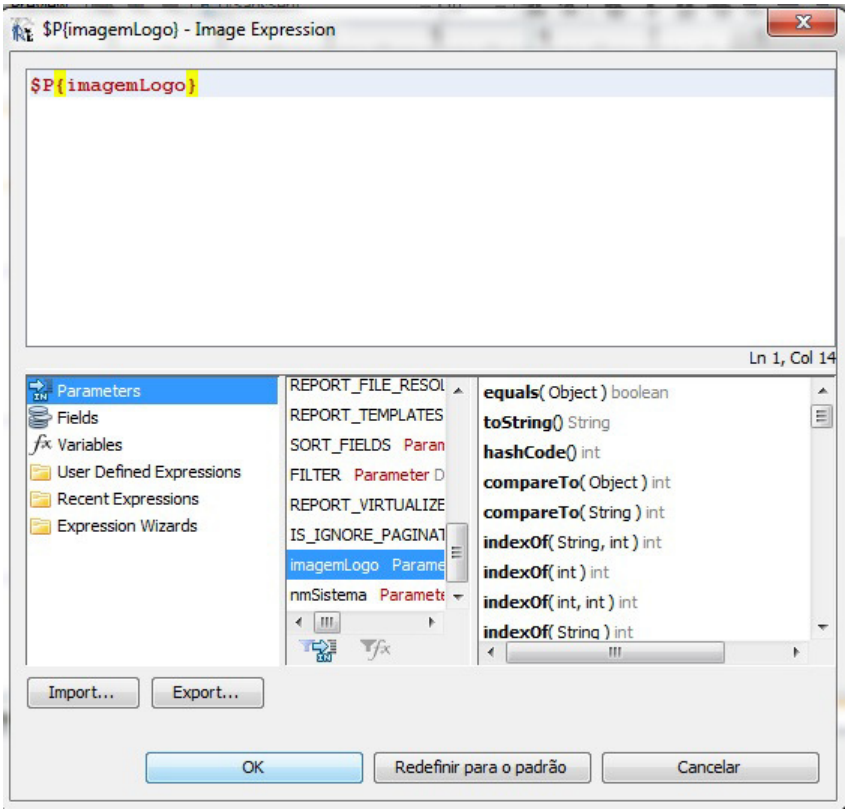


Figura 2.12: Configuração do componente Image

Agora vamos redimensionar e posicionar o componente `Image` .

Opcionalmente, na propriedade `Image Expression` do menu `Propriedades` , podemos alterar a configuração do componente `Image` .

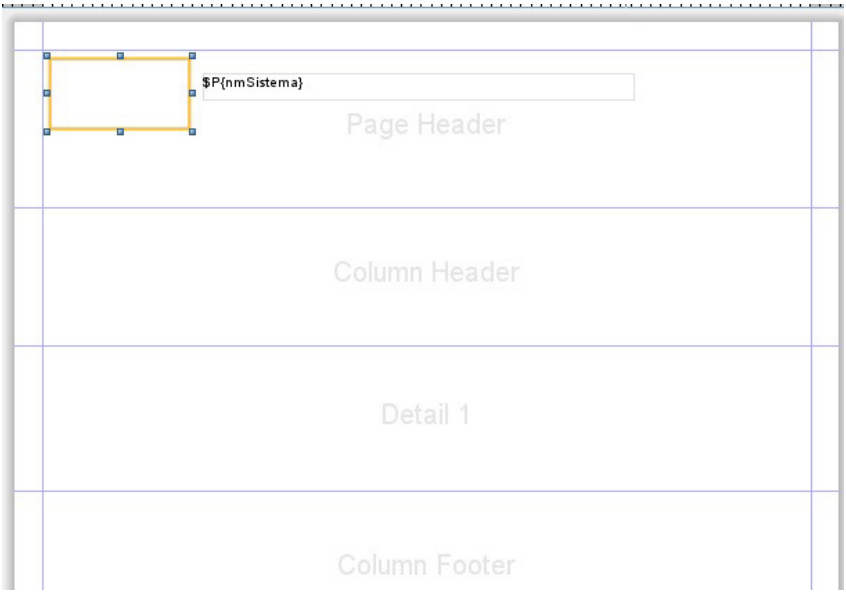


Figura 2.13: Alterando o componente Image

No menu `Paleta`, no submenu `Tools`:

1. Clique no componente `Current date`.
2. Arraste-o para a área `Page Header`.
3. Quando soltarmos o componente na área, será disponibilizada uma tela conforme vemos na figura seguinte.

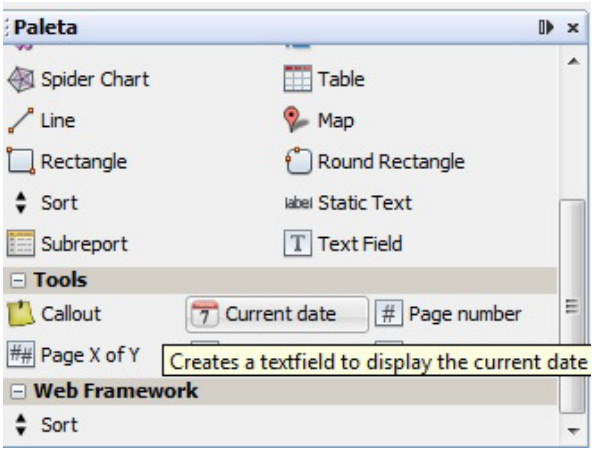


Figura 2.14: Selecionando o componente Current date

Selecione a formatação `dd/mm/yyyy` , e clique no botão `Apply` .

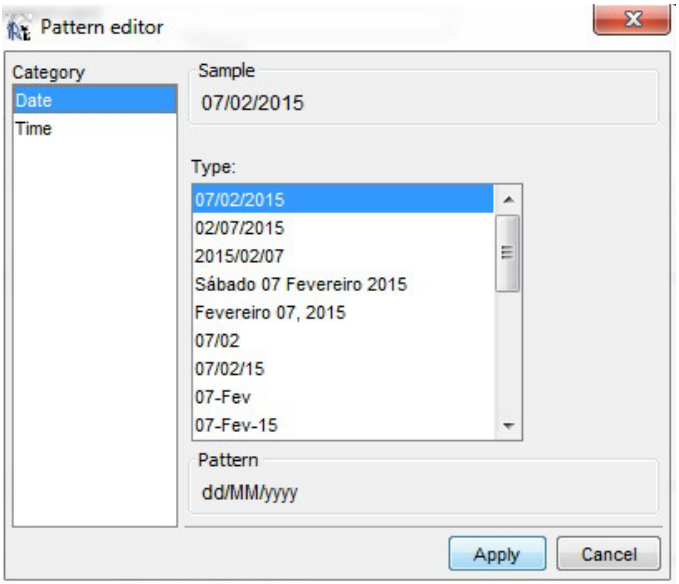


Figura 2.15: Definição de formatação de data

Ainda no menu `Paleta` no submenu `Tools` , faça o mesmo procedimento com o `Static Text` , clicando e arrastando-o para a

`Page Header`.

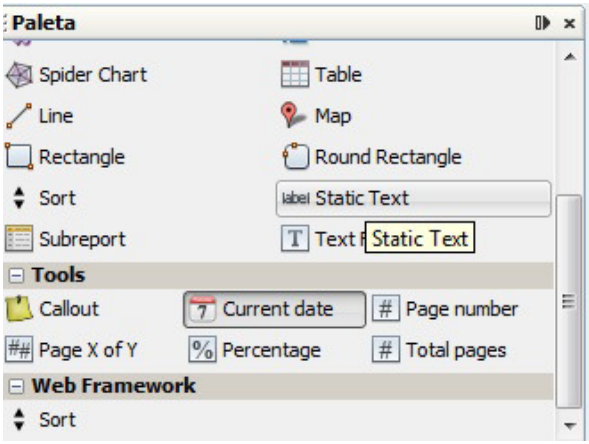


Figura 2.16: Selecionando o componente Static Text

1. Altere a propriedade `Text` do menu `Propriedades` para `Relatório de UF`.
2. Redimensione e posicione o componente `Static Text`.

Figura 2.17: Configurando o componente Static Text

Novamente no menu `Paleta` no submenu `Tools`:

1. Clique no componente `Page X of Y`.
2. Arraste-o para a área `Page Header`.
3. Redimensione e posicione-o.

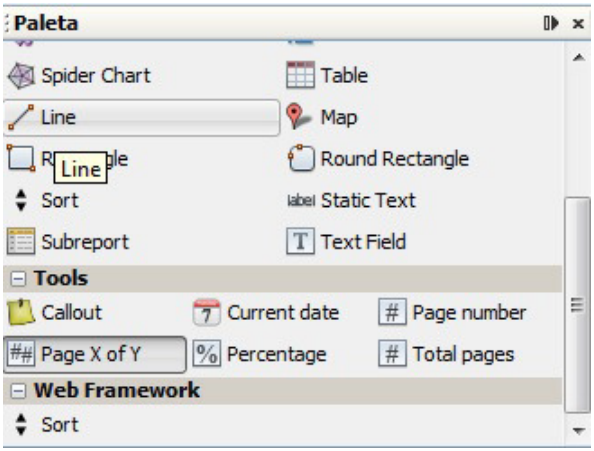


Figura 2.18: Adicionando o componente Page X of Y

Agora, apenas no menu `Paleta`, faça o mesmo para o componente `Line`: clique, arraste para `Page Header` e posicione-o.

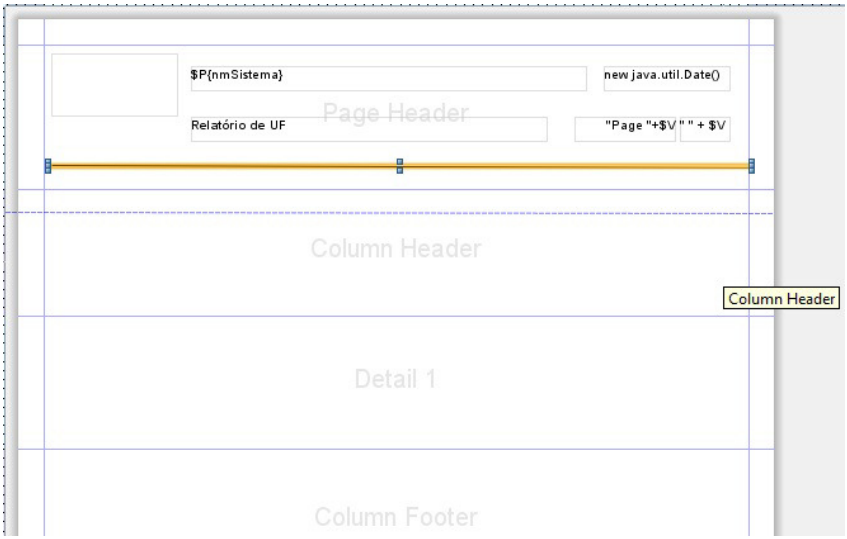


Figura 2.19: Adicionando o componente Line

Assim, concluímos a implementação do cabeçalho.

Detalhe do relatório

Na área `Detail`, colocamos os `fields` que serão mostrados no relatório. Vá para o menu `Report Inspector`:

1. Clique na opção `fields`.
2. Clique em `field id`.
3. Arraste-o para área `Detail`, redimensionando e posicionando-o.
4. Será disponibilizado um componente `Static Text` na área `Column Head` com o nome da coluna já definido. Opcionalmente, podemos alterá-lo.
5. Clique no `Label id` na área `Column Head` e, em seguida, altere a propriedade `Text` do menu `Propriedades` para `Código`.

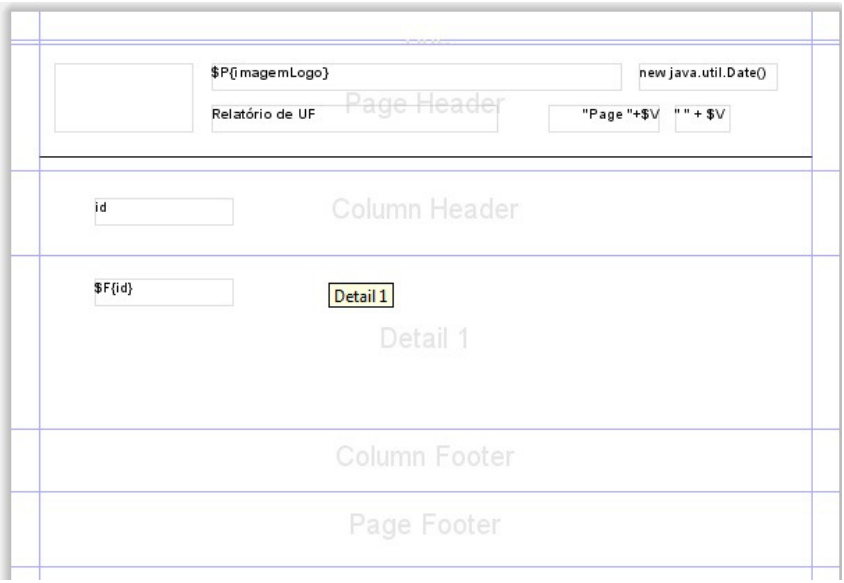


Figura 2.20: Adicionando o field id no detalhe do relatório

Agora que você já aprendeu a adicionar um `field` no relatório, adicione os demais `field`, o do nome e da sigla.

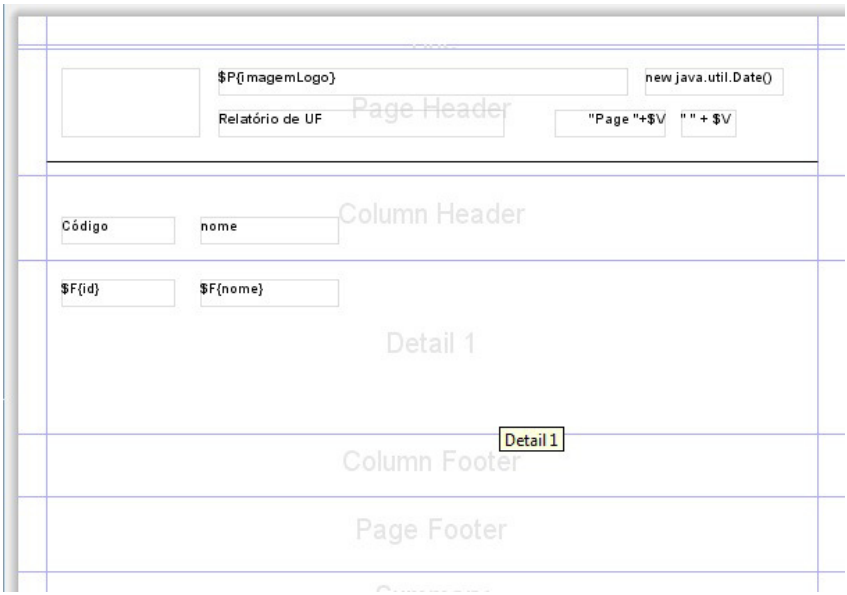


Figura 2.21: Adicionando o field nome no detalhe do relatório

Novamente no menu `Paleta` :

1. Clique no componente `Line` .
2. Arraste-o para área `Column Head` .
3. Redimensione e posicione o componente.

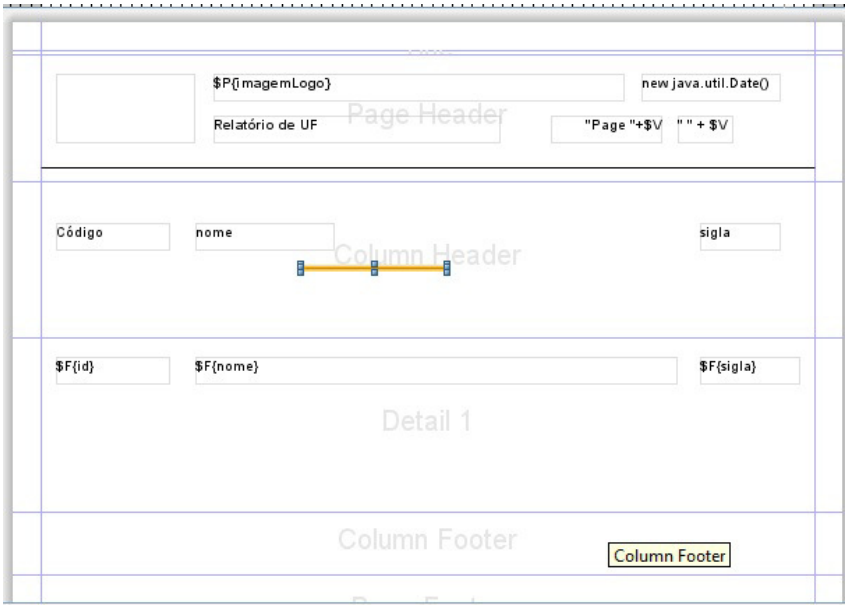


Figura 2.22: Adicionando o componente Line

Na área `Column Head`, clique no label `Código`. Serão disponibilizadas para configuração as suas propriedades. Para imprimir em negrito, marque a opção `Bold`.

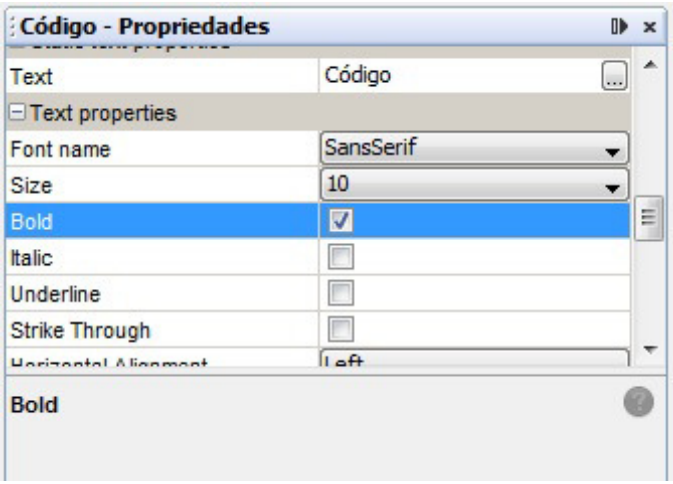


Figura 2.23: Configurando o label Código

Para que todos os label tenham a mesma altura, temos as seguintes opções:

1. Configurar visualmente arrastando os labels e colocando todos na mesma altura.
2. Clique em cada label e, na propriedade `Top` do menu `Propriedades`, atribua o mesmo valor a todos.

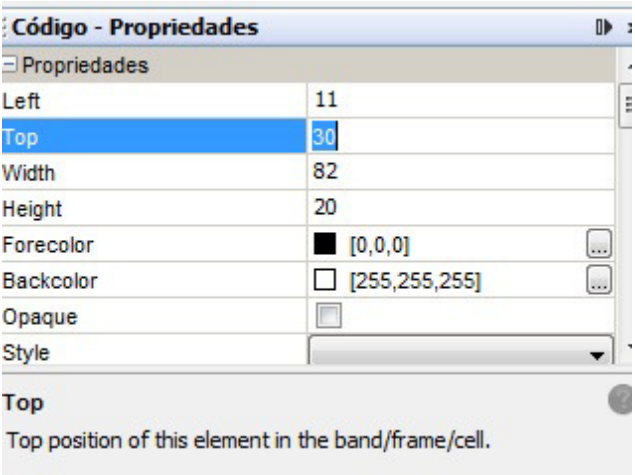


Figura 2.24: Configurando a propriedade Top do label

Fazemos o mesmo processo para colocar todos os fields na mesma altura, assim como fizemos com os labels: ou configuramos visualmente, ou também clicamos em cada field e, na propriedade `Top` do menu `Propriedades`, atribuímos o mesmo valor para todos.

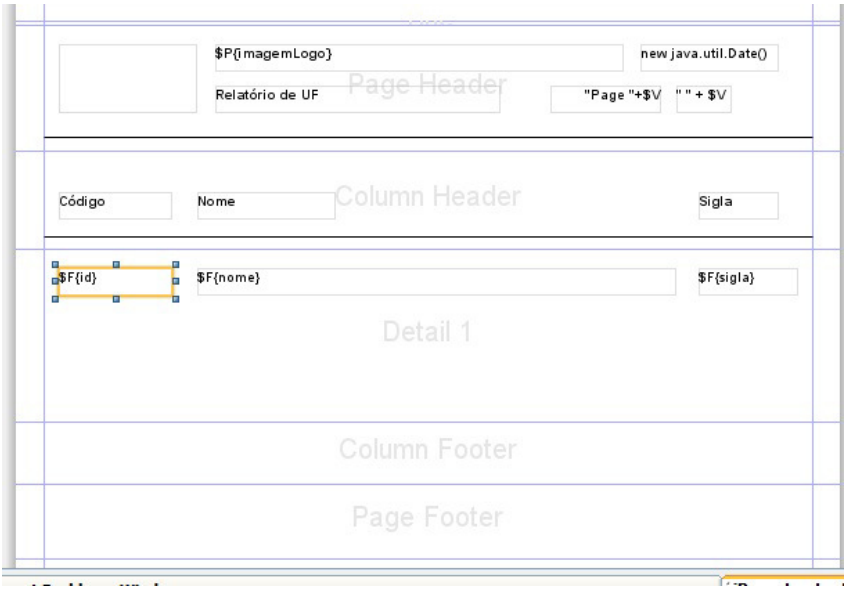


Figura 2.25: Configurando a propriedade top dos fields

Dessa forma, concluímos a implementação. Agora, é preciso compilar. Para isso, clique na figura do martelo do lado do nome da fonte. Na parte inferior na aba `Report output`, podemos visualizar o caminho completo onde foi gerado o arquivo `relUF.jasper`. Agora é só testar a execução do relatório de UF.

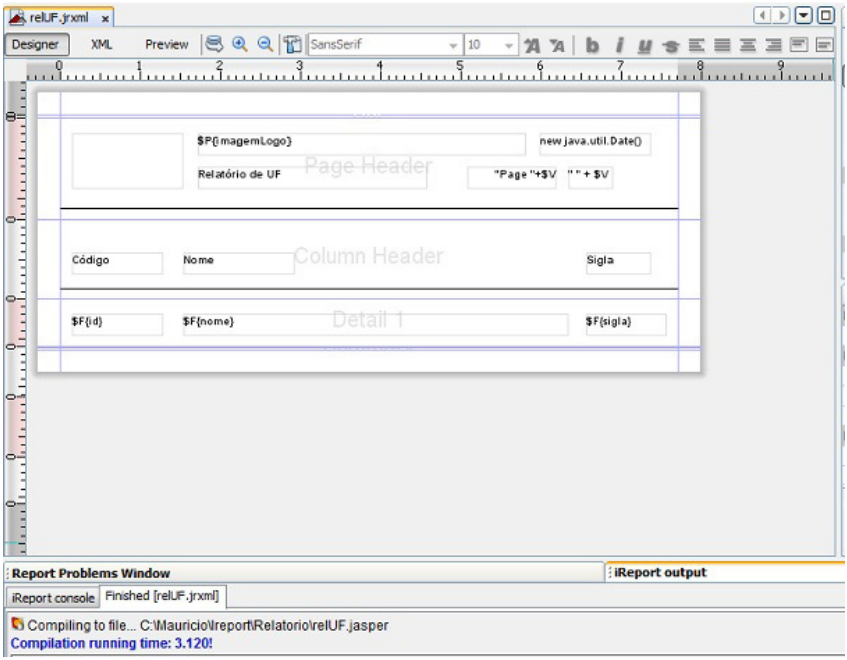


Figura 2.26: Compilando o relatório

2.5 CRIANDO RELATÓRIO COM AGRUPAMENTO

Layout do Relatório

Nosso desafio agora é implementar o relatório com agrupamento por unidades federativas, conforme o layout mostrado na figura a seguir.

DEVELOPER SISTEMA DE GERENCIAMENTO DE CLIENTE 07/02/2015

Pagina 1 de 1

Relatorio de Municipio

UF Ceara

Código	Nome		População
4	aquiraz		50
1	fortaleza		100
5	juazeiro		80
Média Populacional	76,67	Total	230

UF Rio de janeiro

Código	Nome		População
2	rio de janeiro		200
Média Populacional	200,00	Total	200

UF Sao Paulo

Código	Nome		População
3	sao paulo		300
Média Populacional	300,00	Total	300

Figura 2.27: Layout do relatório

O relatório é composto de um cabeçalho, que contém:

1. Uma imagem com o logo da empresa;
2. O nome do sistema;
3. A data de impressão;
4. Contador de páginas;
5. O nome de relatório.

Em seguida, temos um agrupado por unidades federativas e uma área reservada para colocar o nome das colunas do relatório. Temos também uma área detalhada com as informações do Município pertencente à unidade federativa, e, finalmente, uma área reservada para as variáveis que vão calcular a média populacional e o total da população da unidade federativa. É oportuno lembrar de que os valores da população não reflete a realidade, são meramente didáticos.

Adicionando Parameters, Fields e Cabeçalho no relatório

Já sabemos como adicionar parâmetros, fields e implementar o cabeçalho do relatório. Se ainda persistir alguma dúvida sobre esses temas, retorne à seção *Criando um relatório de listagem*. O que veremos de diferente aqui será como adicionar grupo e variáveis.

Gostaríamos apenas de enfatizar que a classe `Municipio.java` possui um atributo do tipo `UF.java` . Observe que adicionamos os fields: `uf.id` e `uf.nome` .

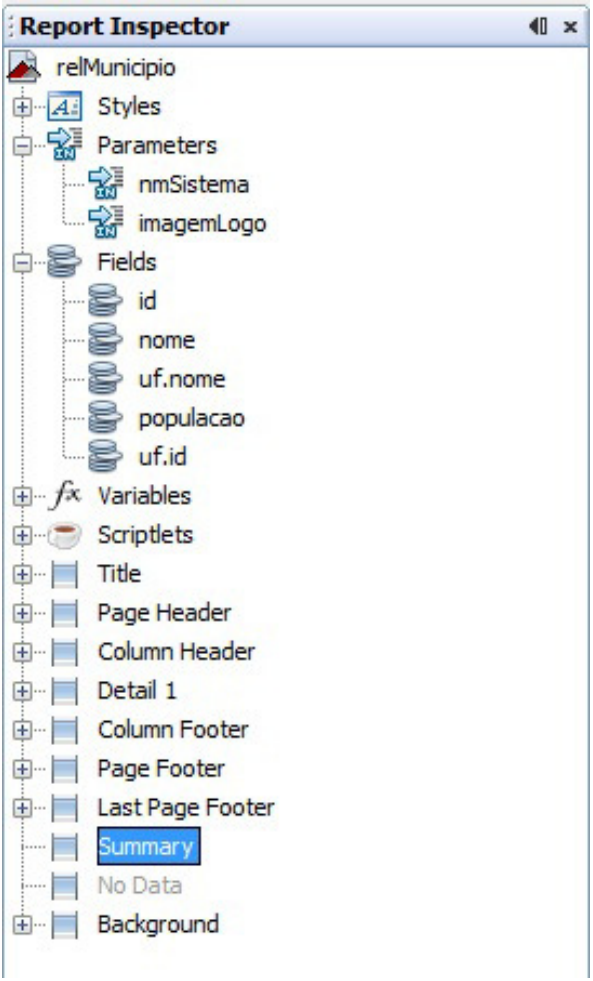


Figura 2.28: Ênfase no field UF

Adicionando Grupo

Neste relatório, vamos criar um grupo para que possamos totalizar a população por unidade federativa e também obter uma média da população por unidade federativa.

No menu `Report Inspector` :

1. Clique na opção do nome do relatório; no nosso contexto, em `relMunicipio`.
2. Clique no botão direito do mouse para visualizar o submenu.
3. Clique na opção `Add Report Group`.
4. Será disponibilizada uma tela conforme a figura a seguir.

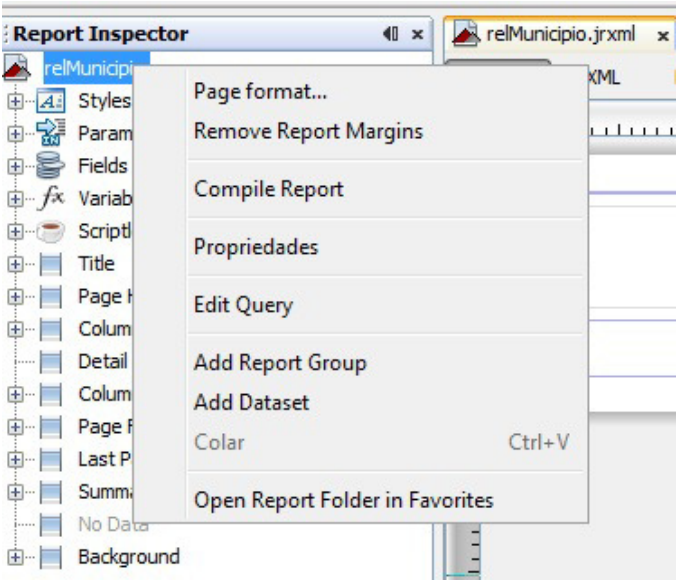


Figura 2.29: Adicionando Grupo

Nesta tela, configuramos o grupo colocando o nome de `UF`, agrupado por `uf.id`.

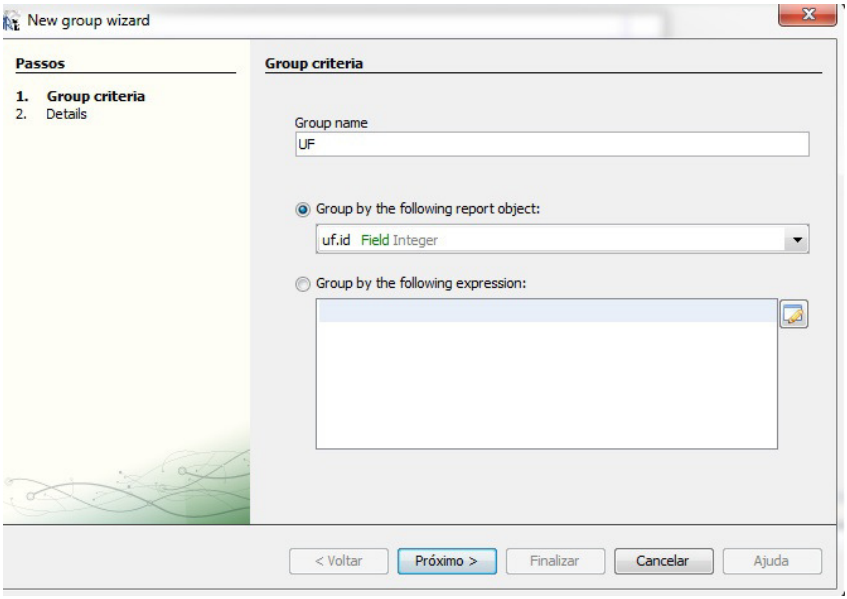


Figura 2.30: Configurando Grupo

Como as opções `group header` e `group footer` estão marcadas, elas serão adicionadas ao relatório. Depois, clique no botão de finalizar.

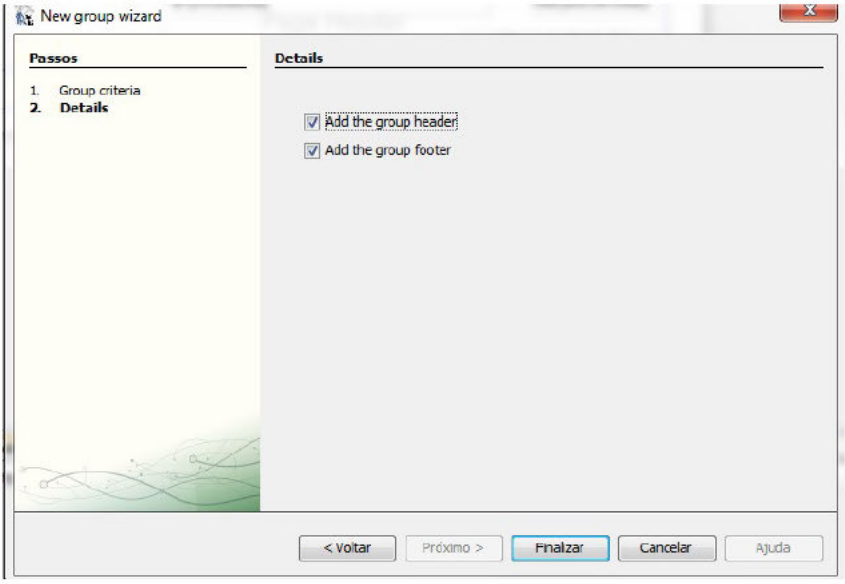


Figura 2.31: Finalizando configurando do Grupo

Criando variáveis no relatório

No menu `Report Inspector` , clicando na opção `variables` , as variáveis são disponibilizadas. O iReport já disponibiliza 5 variáveis nativas.

- `PAGE_NUMBER`
- `COMMLUM_NUMBER`
- `REPORT_COUNT`
- `PAGE_COUNT`
- `COMMLUM_COUNT`

Agora, no menu `Report Inspector` :

1. Clique na opção `variables` .
2. Clique no botão direito do mouse para visualizar o submenu.
3. Clique na opção `Add variable` .
4. Será adicionada uma `variable` conforme a figura a seguir.

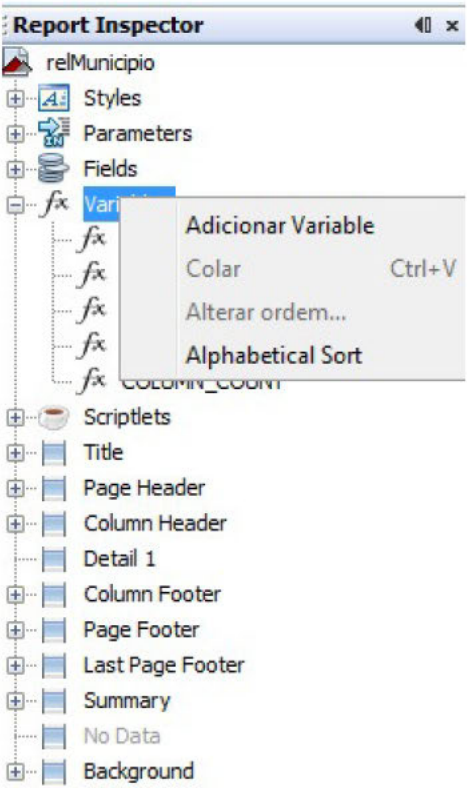


Figura 2.32: Selecionando adicionar variáveis

Nesta tela, vá para o menu `Propriedades` e configure a `variable`.

- Na propriedade `Name`, altere para `totalPopulacao`.
- Na propriedade `Variable Class`, é disposta uma lista de classe. Selecione a classe `Integer`.
- Na propriedade `Calculation`, é disponibilizada uma lista de cálculo. Selecione a opção `sum`. É oportuno lembrar de que essa propriedade somará a população dos municípios cadastrados.
- Na propriedade `Reset Type`, é apresentada uma lista de `Reset`. Selecione a opção `Group`.

- Na propriedade `Reset Group` , selecione a opção `UF` .

As propriedades `Resset Type` e `Reset Group` determinam que, sempre que uma variável mudar de unidade federativa, ela será inicializada.

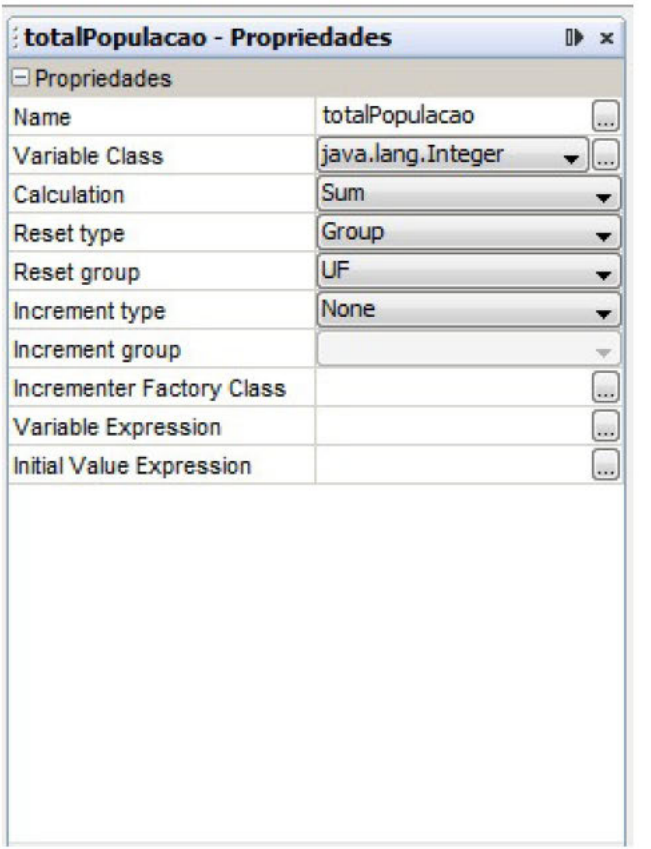


Figura 2.33: Configurando variável totalPopulacao

Vamos, agora, adicionar mais uma variável. Para isso, vá para o menu `Propriedades` e configure a `variable` da seguinte forma:

1. Na propriedade `Name` , altere para `mediaPopulacao` .
2. Na propriedade `Variable Class` , é disponibilizada uma lista de classe. Selecione a classe `Double` .

3. Na propriedade `Calculation`, é fornecida uma lista de cálculo. Selecione a opção `average`. É válido lembrar de que essa propriedade calculará a média da população da unidade federativa.
4. Na propriedade `Reset Type`, é apresentada uma lista de `Reset`. Selecione a opção `Group`.
5. Na propriedade `Reset Group`, selecione a opção `UF`.

Como já visto, as propriedades `Resset Type` e `Reset Group` determinam que, sempre que uma variável mudar de unidade federativa, ela será inicializada.

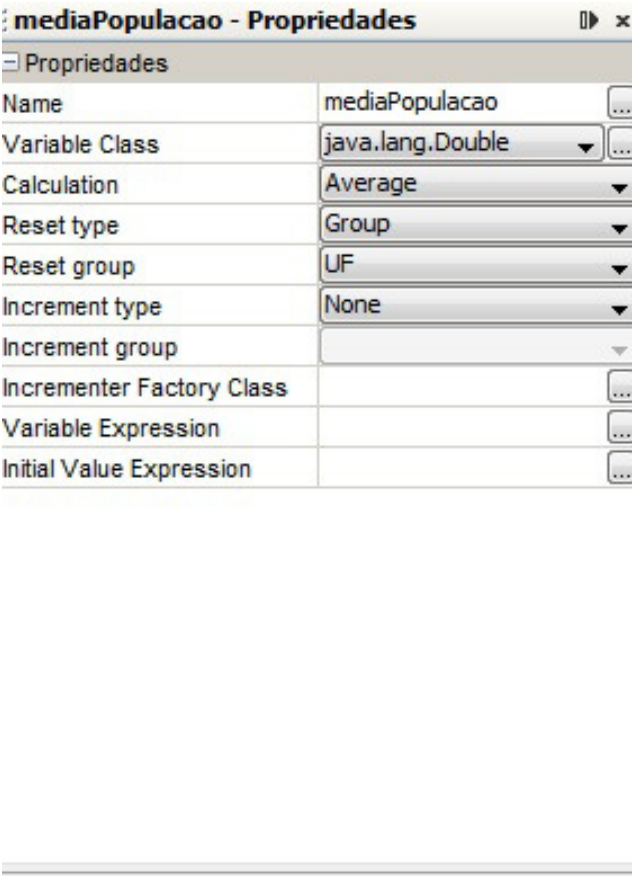


Figura 2.34: Configurando variável mediaPopulacao

Como já aprendemos anteriormente, não vamos repetir o processo de adicionar e configurar fields e labels. Se ainda persistir alguma dúvida, retorne à seção *Criando um relatório de listagem*.

Gostaria apenas de enfatizar que as informações da unidade federativa e os labels das colunas do relatório ficam na área `UF Group Header`. Assim, arraste as variáveis que adicionamos para a área `UF Group Footer`.

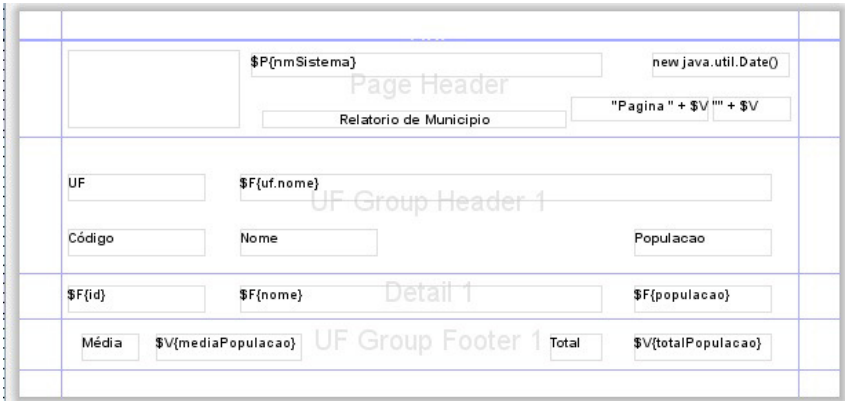


Figura 2.35: Finalizando design do relatório

Concluímos a implementação, mas precisamos compilar. Para isso, clique no martelo do lado do nome da fonte. Então, na parte inferior na aba `Report output`, podemos visualizar o caminho completo em que o arquivo `relMunicipio.jasper` foi gerado. Agora é só testar a execução do relatório de Município.

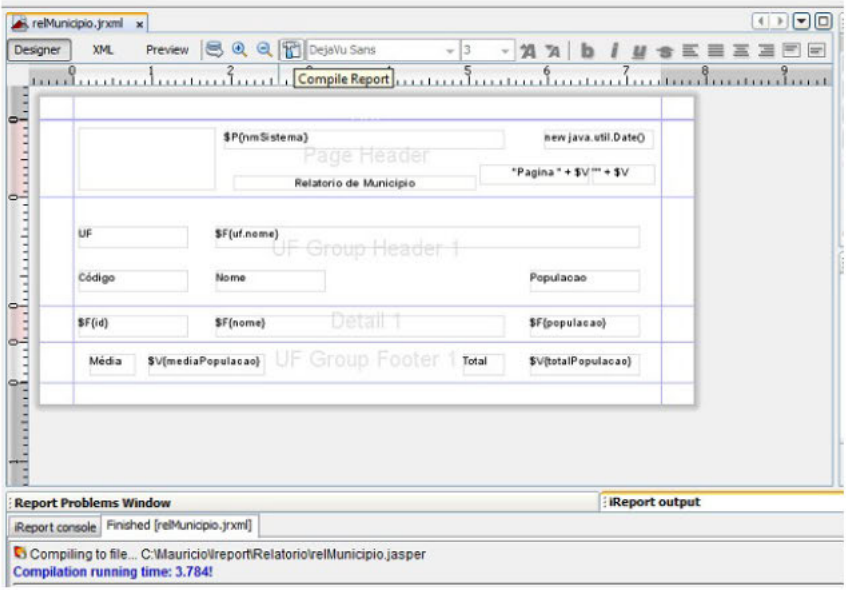


Figura 2.36: Compilando relatório

2.6 CONCLUSÃO

Neste capitulo aprendemos a:

- Adicionar as libs necessárias para executar o relatório em uma aplicação Java.
- Implementar o método necessário para disponibilizar o relatório no formato PDF.
- Adicionar e configurar Parameters, Fields e Variables.
- Implementar um relatório de listagem usando como fonte de dados ArrayList.
- Implementar um relatório com agrupamento usando como fonte de dados ArrayList.

No próximo capítulo, vamos aprender a implementar estes mesmos relatórios usando com fonte da dados instrução SQL.

CAPÍTULO 3

RELATÓRIO COM SQL

3.1 CRIANDO RELATÓRIO DE LISTAGEM COM SQL

Layout do relatório

Nosso desafio será implementar o relatório de unidade federativa conforme mostra a figura a seguir, que criamos anteriormente utilizando como fonte de dados ArrayList. Porém, desta vez, em vez de usar ArrayList, usaremos instrução SQL.

DEVELOPER

SISTEMA DE GERENCIAMENTO DE CLIENTE 07/02/2015

Pagina 1 de 1

Relatorio de UF

Codigo	Nome	Sigla
8	Acre	AC
3	Ceara	CE
6	Maranhao	MA
7	Piaui	PI
4	Rio de janeiro	RJ
5	Sao Paulo	SP

Figura 3.1: Layout do relatório

Método gerarRelatorio

Na classe `AbstractMB`, adicionamos o método `gerarRelatorio`. Este método é similar ao que vimos no capítulo anterior. A diferença é que ele não utiliza ArrayList, mas sim uma conexão JDBC (*Java Database Connectivity*).

O método `gerarRelatorio` é responsável pela execução e visualização do relatório, e também por acrescentar parâmetros comum a todos os relatórios, como por exemplo `imagemLogo`.

Instanciamos a classe `HttpServletResponse` para que o usuário possa visualizar o relatório.

```
public void gerarRelatorio(String nomeRelatorio,
  HashMap paramRel){
   FacesContext context = FacesContext.getCurrentInstance();
   HttpServletResponse response =
       (HttpServletResponse) context.getExternalContext()
                                     .getResponse();
   ServletContext sc =
       (ServletContext) context.getExternalContext().getContext();

   String relPath = sc.getRealPath("/");
   String imagemLogo =
       relPath + "resources/imagens/logo_mmo.jpg";
   paramRel.put("imagemLogo", imagemLogo);
   paramRel.put("nmSistema", Constants.NOME_SISTEMA);
   paramRel.put("REPORT_LOCALE", new Locale("pt", "BR"));
```

Instanciamos a classe `Connection`, que é utilizada para fazer a conexão com o banco de dados, e a classe `JasperFillManager` gera o relatório.

```
   try  {
        JasperPrint print = null;
        String url = "jdbc:postgresql://localhost:5432/sgc";
        String user = "postgres";
        String pass = "123456";
        Connection connection =
            DriverManager.getConnection(url, user,pass);
        print = JasperFillManager.fillReport(relPath +
            "relatorios/"+nomeRelatorio+".jasper",
            paramRel,connection);
```

Com o relatório criado, configuramos o objeto `response` para mostrar o relatório no formato `.pdf`. A classe `JasperExportManager` exporta o objeto `print` para `pdf`.

```
        response.setContentType("application/pdf");
        response.addHeader("Content-disposition", "attachment;
            filename=\"" + nomeRelatorio + ".pdf\"");
        JasperExportManager.exportReportToPdfStream(print,
            response.getOutputStream());
        ServletOutputStream responseStream =
            response.getOutputStream();
        responseStream.flush();
        responseStream.close();
        FacesContext.getCurrentInstance().renderResponse();
        FacesContext.getCurrentInstance().responseComplete();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Classe UFMB

Na classe `UFMB`, adicionamos o método `relatorio`, que será responsável pela preparação dos parâmetros do relatório e pela chamada do método `gerarRelatório`. Um dos parâmetros passados é o nome do relatório que deve ser visualizado. Como o objetivo é um relatório com SQL, neste método não é feita nenhuma consulta e também não é passado nenhum ArrayList.

```
public class UFMB extends AbstractMB {
   public  void relatorio() throws Exception {
          try {
             HashMa p paramRel = new HashMap();
             String nomeRelatorio = "relUF";
             gerarRelatorio(nomeRelatorio, paramRel);
          } catch (NegocioException e) {
             addMsgErro(e.getMessage());
          }
      }
}
```

Para iniciar a configuração da conexão, na tela principal da `IDE`, clique na figura do lado da palavra `Empty datasource` para

ser disponibilizada uma tela de configuração, conforme mostra a figura seguinte.

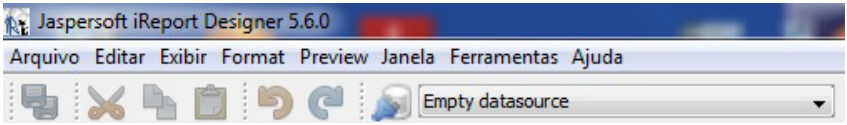


Figura 3.2: Interface de desenvolvimento

Na tela `Connections / DataSources` , clique no botão `New` e, depois, avance para a tela seguinte. Veja a figura a seguir:

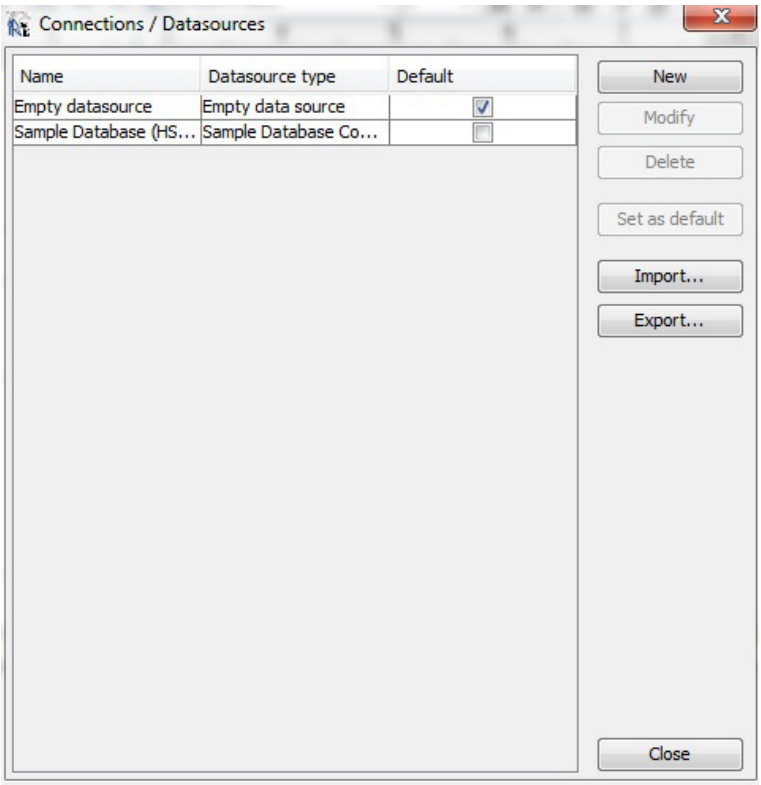


Figura 3.3: Connections/DataSources

Na tela `DataSource` , será apresentada uma lista de `DataSource` . Selecione a opção `Database JDBC connection` e

clique no botão `Next` .

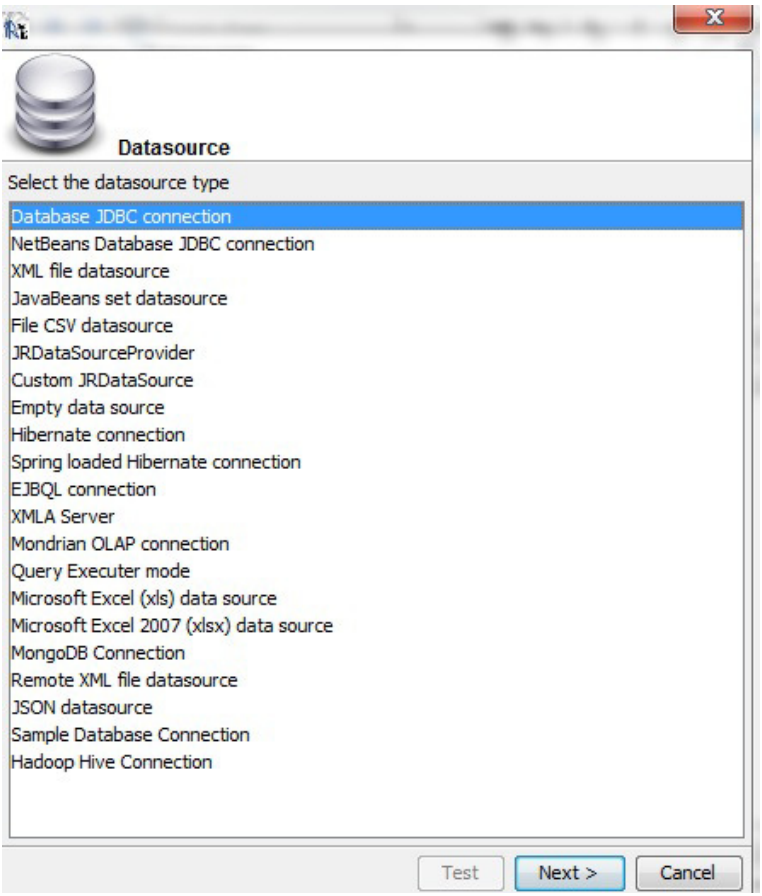


Figura 3.4: DataSources

A tela `Database JDBC connection` é onde informamos os dados da conexão, conforme mostra a figura a seguir. É oportuno lembrar que o *username* e o *password* são os que você usou para configurar o seu banco de dados PostgreSQL.

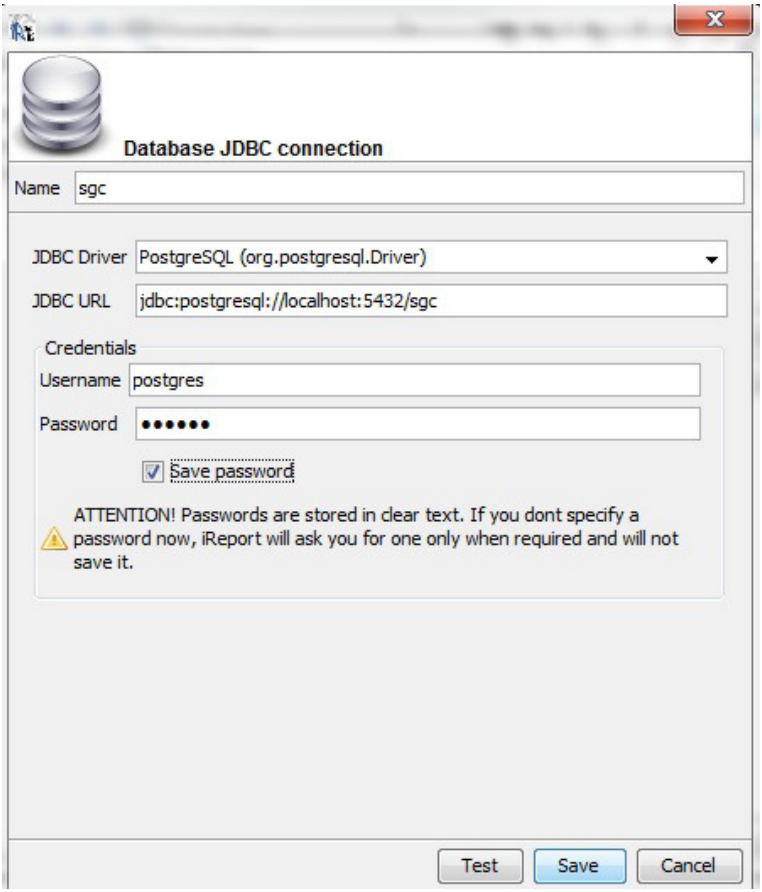


Figura 3.5: Database JDBC connection

Após terminar a configuração, clique no botão `Test` para verificar se a conexão foi configurada com sucesso, como verificamos na figura adiante. Se a conexão foi configurada com sucesso, clique no botão `Save` .

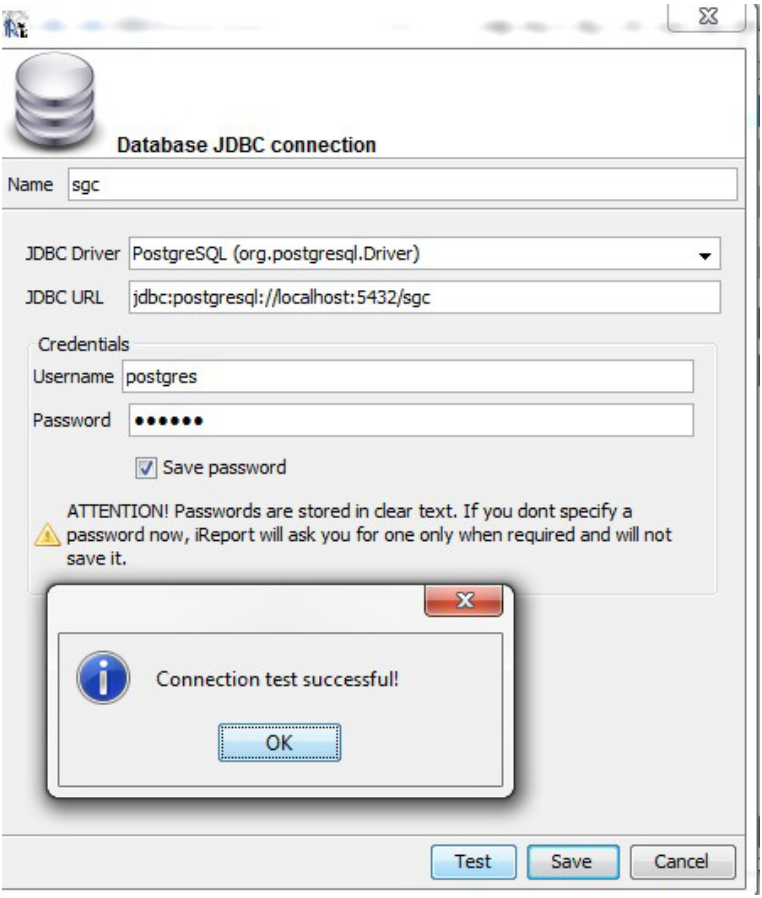


Figura 3.6: Sucesso no teste de conexão

Na tela `Database JDBC connection` , caso haja a necessidade de alterar a configuração do `datasource sgc` , basta um clique na opção `sgc` , e o botão `Modify` será habilitado. Em seguida, clique nele e a tela `Database JDBC connection` será disponibilizada para alterar a configuração.

Já se houver a necessidade de excluir a configuração, basta um clique na opção `sgc` e o botão `Delete` será habilitado. Ao clicar nele, ela será excluída.

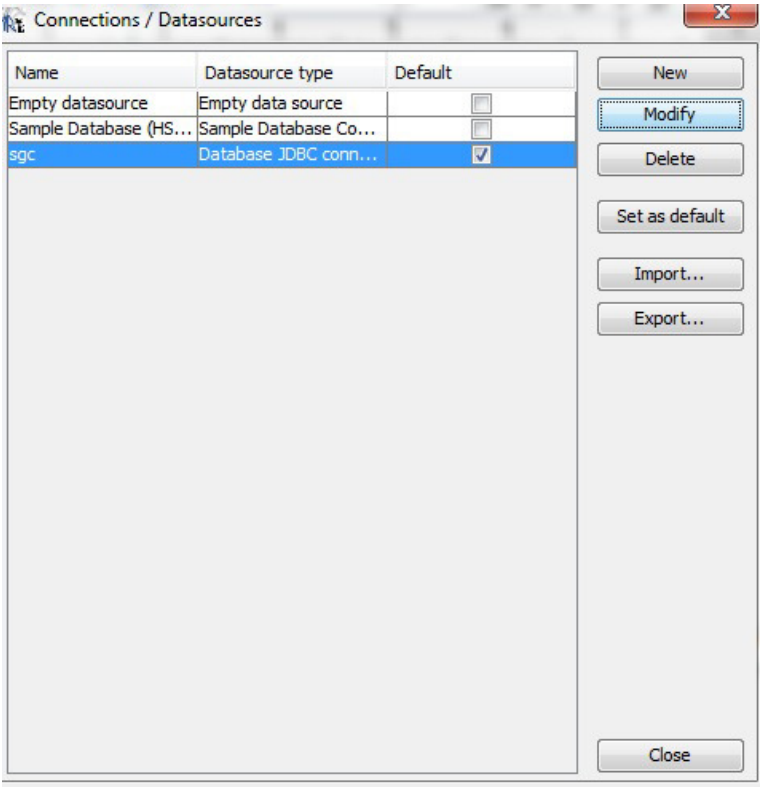


Figura 3.7: Selecionando conexão sgc

Agora que já criamos a conexão, vá para a tela principal da `IDE` e clique na figura do lado da palavra `preview` . Será disponibilizada uma tela, como verificamos na figura a seguir:

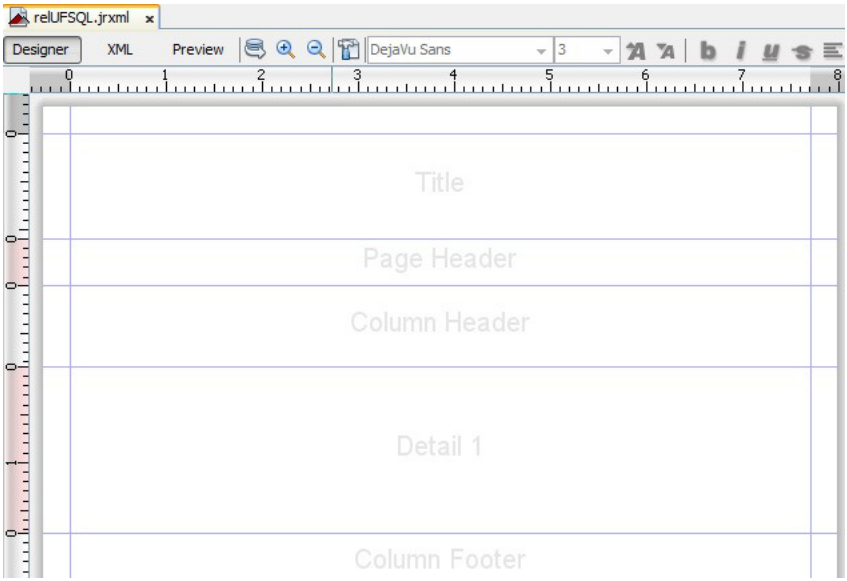


Figura 3.8: Selecionando a opção Report query

Na tela `Report query`, visualizamos diversas opções de `Datasource`. Selecione a opção de `Report query` e, em `Quary language`, veremos as diversas opções de linguagens. Clique na opção `SQL`, conforme mostra a figura a seguir.

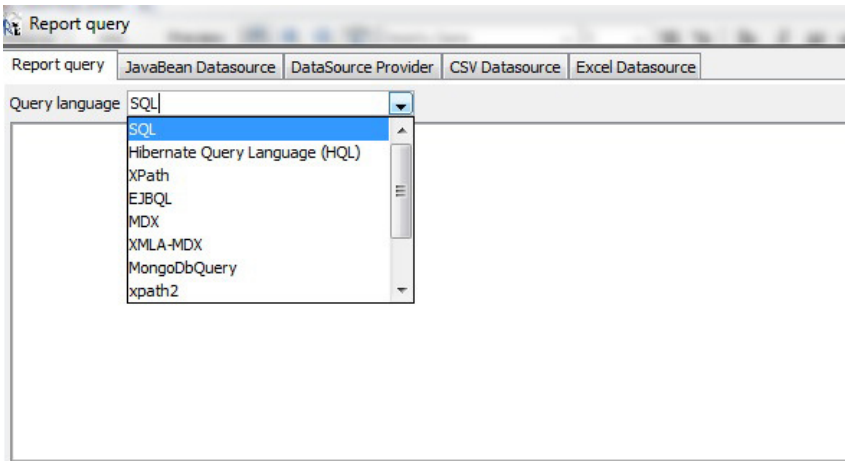


Figura 3.9: Selecionando a opção Query language

Assim, criamos a nossa instrução de consulta à tabela `UF` e, simultaneamente, os fields do relatório são criados.

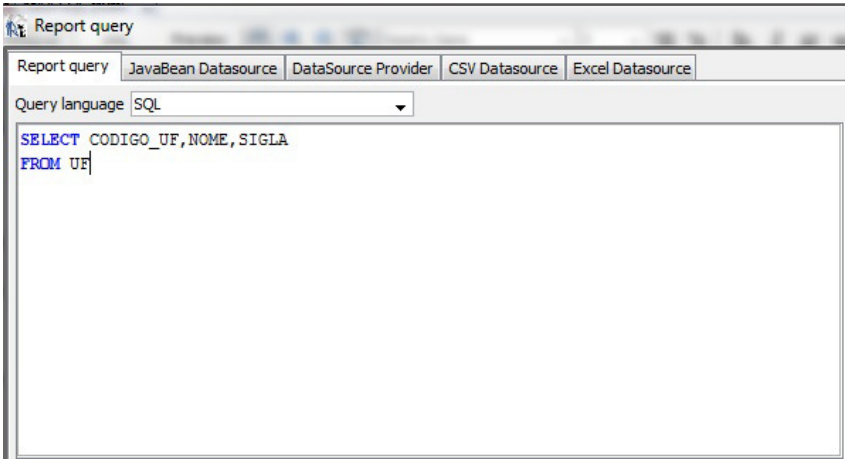


Figura 3.10: Script SQL

Load Query

Outra forma de criarmos nossa instrução SQL é quando criamos a instrução previamente e salvamos em uma pasta. Na tela `Report query` , clique no botão `Load Query` e será disponibilizada uma tela conforme mostra a figura a seguir. Selecione o arquivo na pasta onda está o arquivo e clique em abrir.

Conforme o arquivo selecionado, a instrução será criada simultaneamente aos fields do relatório.

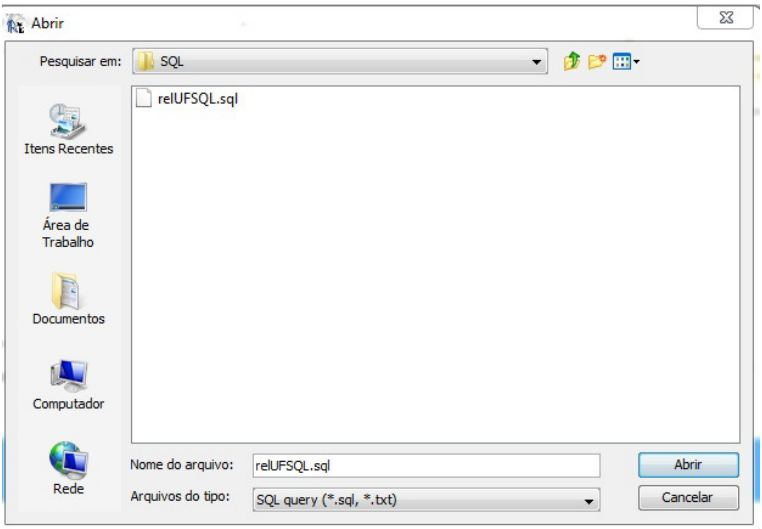


Figura 3.11: Selecionando script na pasta

Na tela `Report query`, clique no botão `New Parameter` para a tela de adicionar parâmetros ser exibida. Será nela que vamos adicionar os parâmetros do nosso relatório, conforme mostra a figura a seguir.

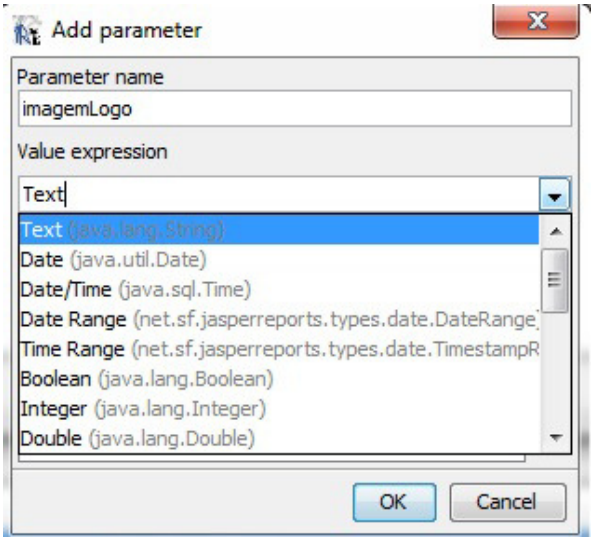


Figura 3.12: Adicionando parâmetro

Após criar a instrução SQL e os parâmetros do relatório, clique em `OK` para sair da tela `Report query`.

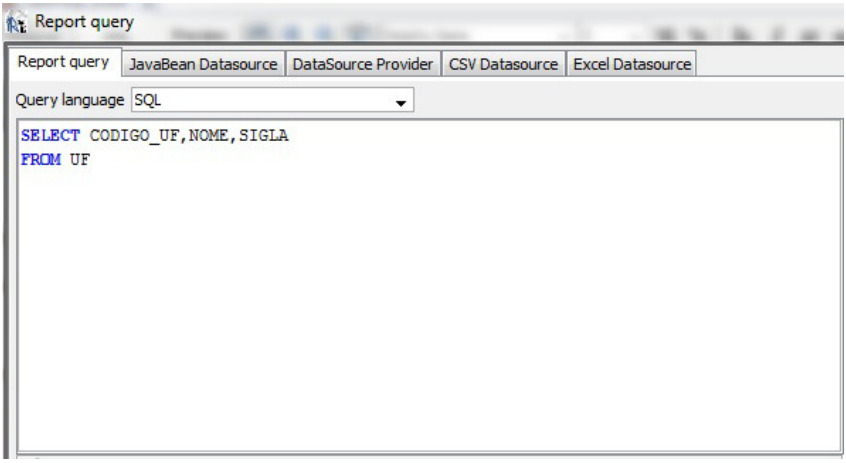


Figura 3.13: Report query finalizado

Retornando à tela principal da `IDE`, verifique que os parâmetros e os fields já estão criados. Agora é só desenhar o relatório conforme já foi visto anteriormente.

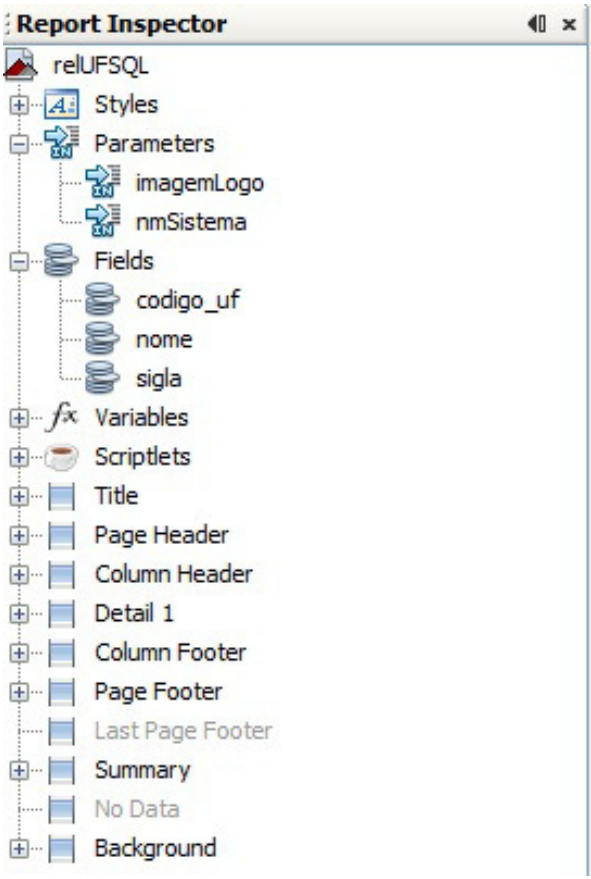


Figura 3.14: Parâmetro e fields adicionados

Após o término do desenho do relatório, é só compilar e testar na aplicação. Como se trata de uma fonte de dados `Report query`, outra forma de testar é dando um clique na aba `preview`. Assim, será disponibilizada uma tela para informar a localização do parâmetro `imagemLogo`, como verificamos na figura adiante.

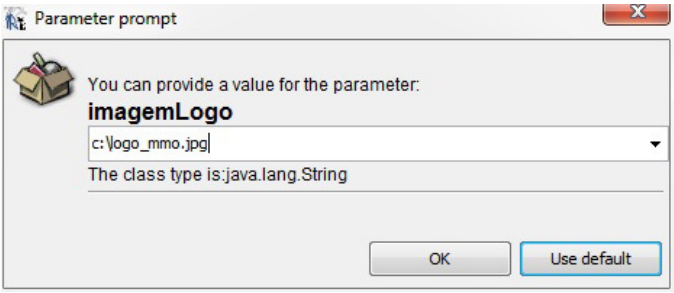


Figura 3.15: Informa path do parâmetro imagemLogo

Depois, será exibida uma tela para informar o parâmetro `Nome do Sistema`.

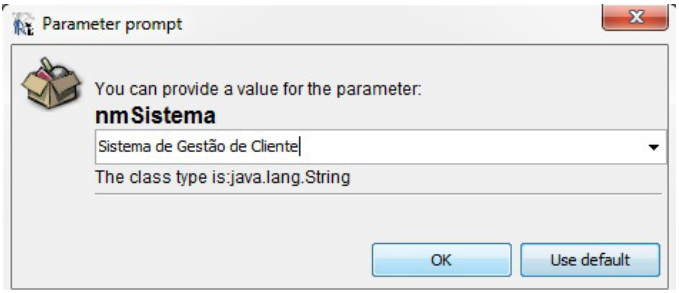


Figura 3.16: Informa nome do sistema

Finalmente, o relatório será visualizado.

Sistema de Gestão de Cliente 10/02/2015

Relatório de UF Page 1 of 1

codigo_uf	nome	sigla
6	Maranhao	MA
8	Acre	AC
3	Ceara	CE
7	Piaui	PI
4	Rio de janeiro	RJ
5	Sao Paulo	SP

Figura 3.17: Preview do relatório

3.2 CRIANDO RELATÓRIO COM SQL E COM PARÂMETRO

Layout do Relatório

Nosso desafio será implementar o relatório de unidade federativa, como visto na figura adiante, que criamos no tópico `Criando relatório de listagem com SQL`. Porém, desta vez, vamos acrescentar uma cláusula `where` na instrução SQL e vamos passar o `id` da UF como parâmetro.

SISTEMA DE GERENCIAMENTO DE CLIENTE 07/02/2015

Pagina 1 de 1

Relatorio de UF

Codigo	Nome	Sigla
8	Acre	AC
3	Ceara	CE
6	Maranhao	MA
7	Piaui	PI
4	Rio de janeiro	RJ
5	Sao Paulo	SP

Figura 3.18: Layout do relatório

Classe UFMB

No método `relatorio`, passamos o `id` da UF que será usado na `where` da nossa instrução SQL do relatório que vamos criar.

```
public void relatorioSQLCP() {
    HashMap paramRel = new HashMap();
    if (uf.getId() > 0) {
        paramRel.put("codigo",uf.getId());
    }
    String nomeRelatorio = "relUFSQLCP";
    gerarRelatorio(nomeRelatorio, paramRel);
}
```

Na linha 4, só é criado o parâmetro se o `id` da UF for informado.

Na tela `Report query`, clique no botão `New Parameter` para adicionar o parâmetro `codigo`. Já aprendemos no livro a adicionar os parâmetros `imagemLogo` e `nmSistema` no capítulo anterior.

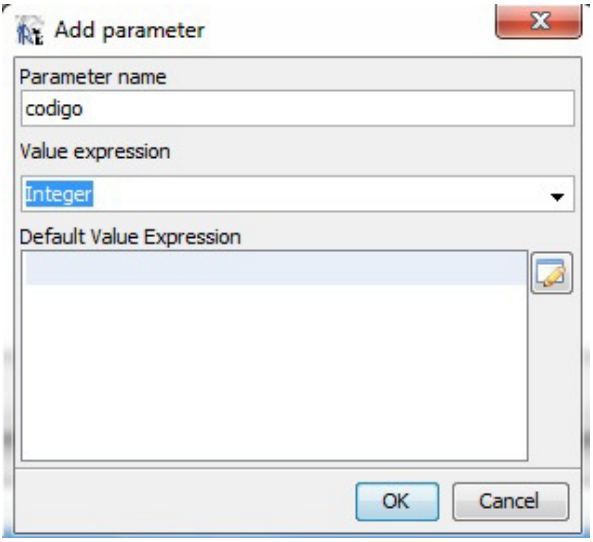


Figura 3.19: Adicionar parâmetro

Criamos, então, a instrução SQL e colocamos a `where` , como mostra a figura a seguir:

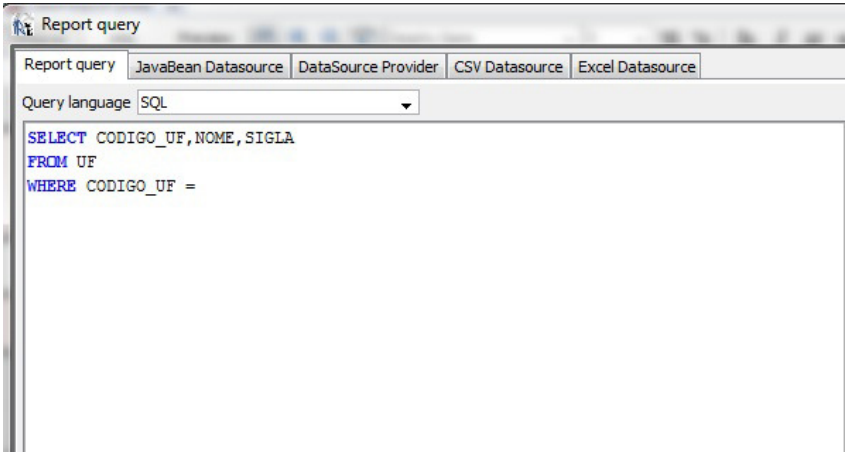


Figura 3.20: Criando instrução SQL

Em seguida, clique no parâmetro `codigo` e arraste para `where` da nossa instrução SQL. Veja a seguir:

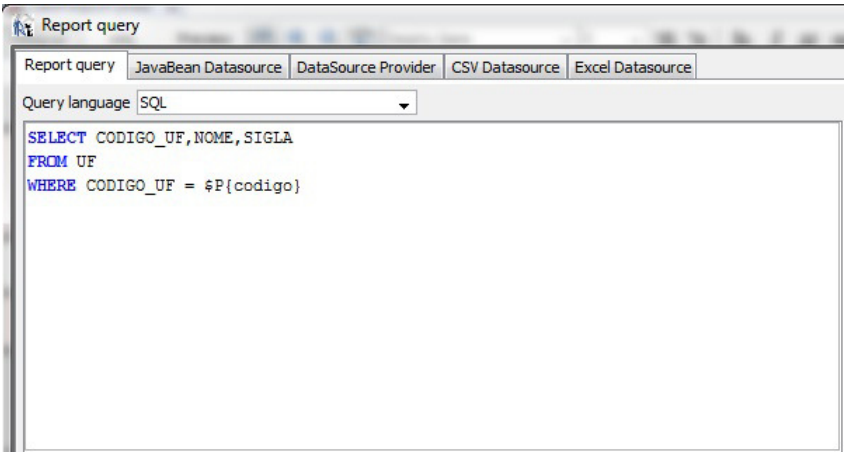


Figura 3.21: Definindo where

Retornando para a tela principal da `IDE`, verifique que os parâmetros e os fields já estão criados. Agora é só desenhar o relatório conforme já foi visto anteriormente.

Figura 3.22: Desing do relatório

Após o término da implementação do relatório, é só compilar e testar na aplicação. Outra forma de testar, como já aprendemos, é dando um clique na aba `preview`. Será disponibilizada a tela para informar a localização do parâmetro `imagemLogo`, o nome do sistema e o código da UF a ser exibida.

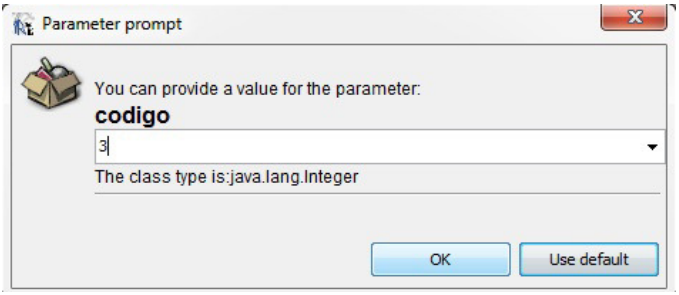


Figura 3.23: Informe código

Finalmente, o relatório será visualizado.

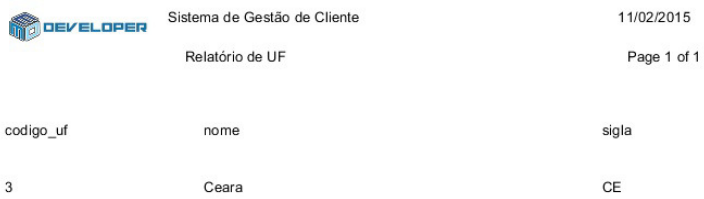


Figura 3.24: Preview do relatório

3.3 CRIANDO RELATÓRIO COM SQL E COM AGRUPAMENTO

Layout do relatório

Agora, nosso desafio será implementar o relatório de município conforme mostra a figura a seguir, que criamos anteriormente usando ArrayList como fonte de dados. Porém, desta vez, vamos criar o relatório com agrupamento utilizando como fonte de dados instrução SQL.

DEVELOPER SISTEMA DE GERENCIAMENTO DE CLIENTE 07/02/2015

Pagina 1 de 1

Relatorio de Municipio

UF Ceara

Código	Nome		População
4	aquiraz		50
1	fortaleza		100
5	juazeiro		80
Média Populacional	76,67	Total	230

UF Rio de janeiro

Código	Nome		População
2	rio de janeiro		200
Média Populacional	200,00	Total	200

UF Sao Paulo

Código	Nome		População
3	sao paulo		300
Média Populacional	300,00	Total	300

Figura 3.25: Layout do relatório

Após criar o relatório, crie um grupo assim como vimos na seção *Criando relatório com agrupamento* do capítulo anterior. Em seguida, vá para a tela `Report query` e crie a instrução SQL.

Veja que não é preciso criar uma instrução SQL com funções de agrupamento e nem com `group by`. Uma instrução simples e trivial já atende às nossas necessidades. Após criar a instrução SQL, crie os parâmetros do relatório, e clique em `OK` para sair da tela

`Report query`.

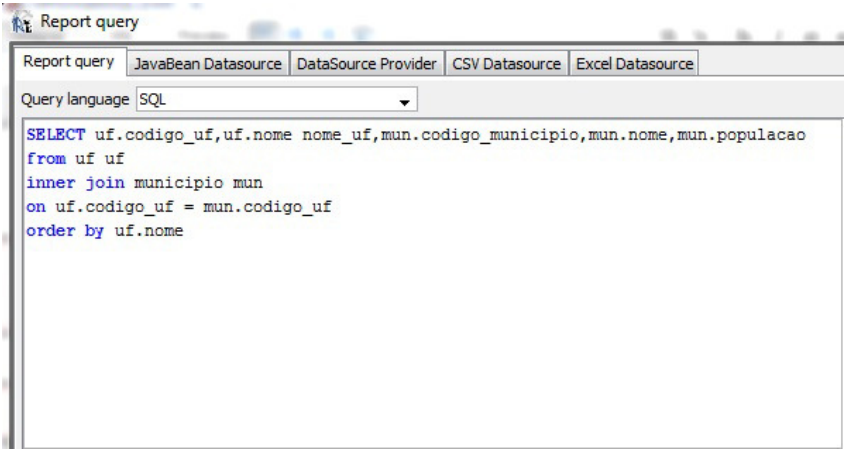


Figura 3.26: Criando instrução SQL

Após retornar para a tela principal da `IDE`, crie as variáveis e faça o design do relatório, conforme vimos anteriormente.

Após o término da implementação do relatório, é só compilar e testar na aplicação. Não se esqueça da outra forma de testar dando um clique na aba `preview`.

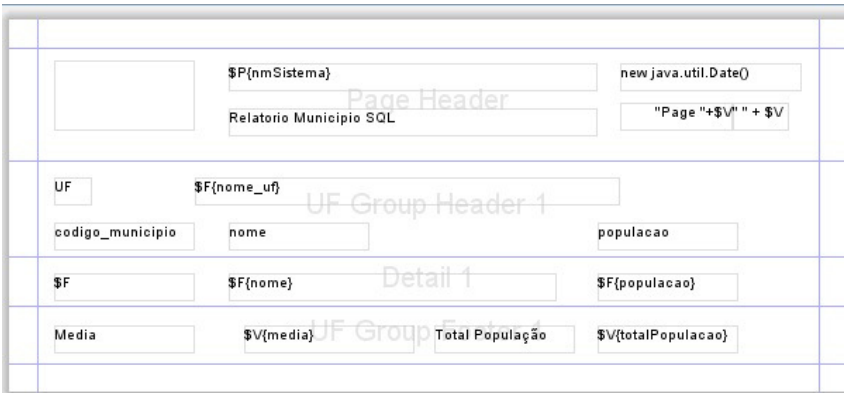


Figura 3.27: Design do relatório

3.4 CONCLUSÃO

Neste capítulo, aprendemos a:

- Implementar um relatório de listagem usando como fonte de dados instrução SQL.
- Implementar um relatório com agrupamento usando como fonte de dados instrução SQL.
- Testar o relatório utilizando o Preview do iReport.
- Implementar o método necessário para gerar um relatório usando uma conexão com banco de dados.

No próximo capítulo, aprenderemos a implementar relatório usando gráfico.

Capítulo 4

RELATÓRIO COM GRÁFICO

4.1 CRIANDO RELATÓRIO COM GRÁFICO DE PIZZA

Layout do relatório

Nosso desafio agora será implementar um relatório com um gráfico, pois faz parte do nosso cotidiano fazer este tipo de relatório. Quando lidamos com porcentagem e frações, o gráfico que será implementado é em forma de pizza, em que cada fatia representa a população de um município, ou relatórios financeiros onde cada fatia representa o faturamento do mês.

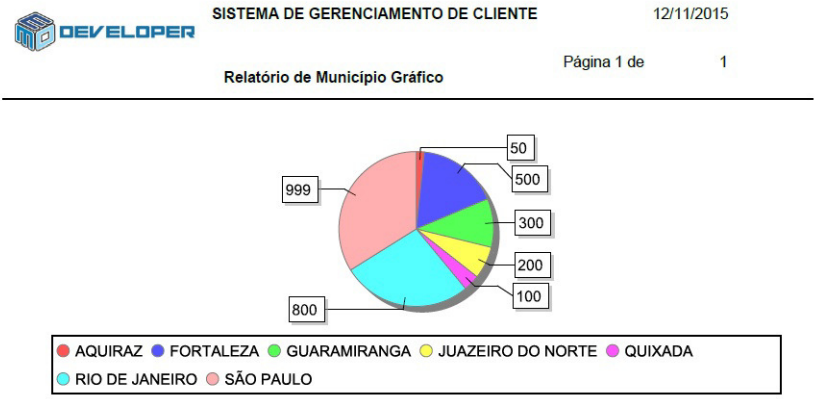


Figura 4.1: Layout do relatório

Classe MunicipioMB

Na classe `MunicipioMB`, adicionamos o método `relatorio`, que será responsável pela chamada da consulta, pela preparação dos parâmetros do relatório e pela chamada do método `gerarRelatório`. Um dos parâmetros passados é o nome do relatório que deve ser visualizado. No nosso caso, demos o nome de `relMunicipio`.

```
public class MunicipioMB extends AbstractMB {
    public  void relatorio() throws Exception {
        try {
            List<UF> listagemResultado = municipioDao.consulta(uf)
;
            HashMap paramRel = new HashMap();
            String nomeRelatorio = "relMunicipio";
            gerarRelatorio(nomeRelatorio, paramRel,
                listagemResultado);
        } catch (NegocioException e) {
            addMsgErro(e.getMessage());
        }
    }
}
```

Municipio.xhtml

Na página `Municipio.xhtml`, incluímos um botão da tag `PrimeFaces` para iniciar a execução do relatório.

```
<p:commandButton
    value="Relatório" ajax="false"
    actionListener="#{municipioMB.relatorio}">
</p:commandButton>
```

Após criarmos os parâmetros, os fields e o cabeçalho, e inibirmos todas as áreas do relatório deixando bem dimensionada apenas a área `Summary`, conforme mostra a figura a seguir, vamos para o submenu `Report Elements`.

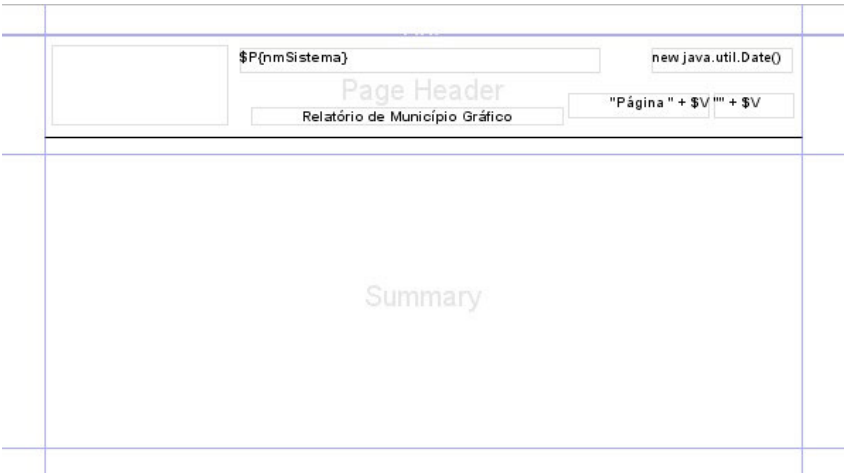


Figura 4.2: Iniciando relatório

No submenu `Report Elements`, clique no componente `Chart` e arraste-o para a área `Summary`.

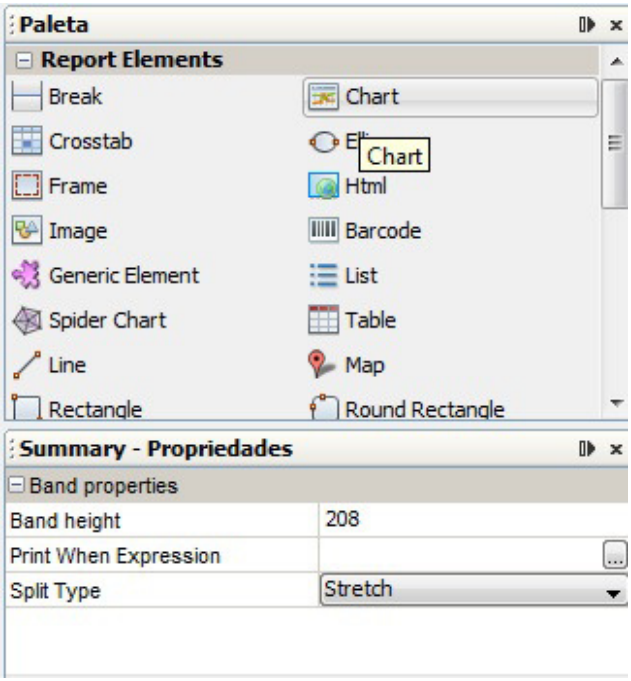


Figura 4.3: Adicionando Chart

Ao soltar o componente `Chart` na área `Summary`, será disponibilizada uma tela com diversos tipos de gráficos, como verificamos na figura adiante. Clique no gráfico de pizza e, depois, no botão `OK`.

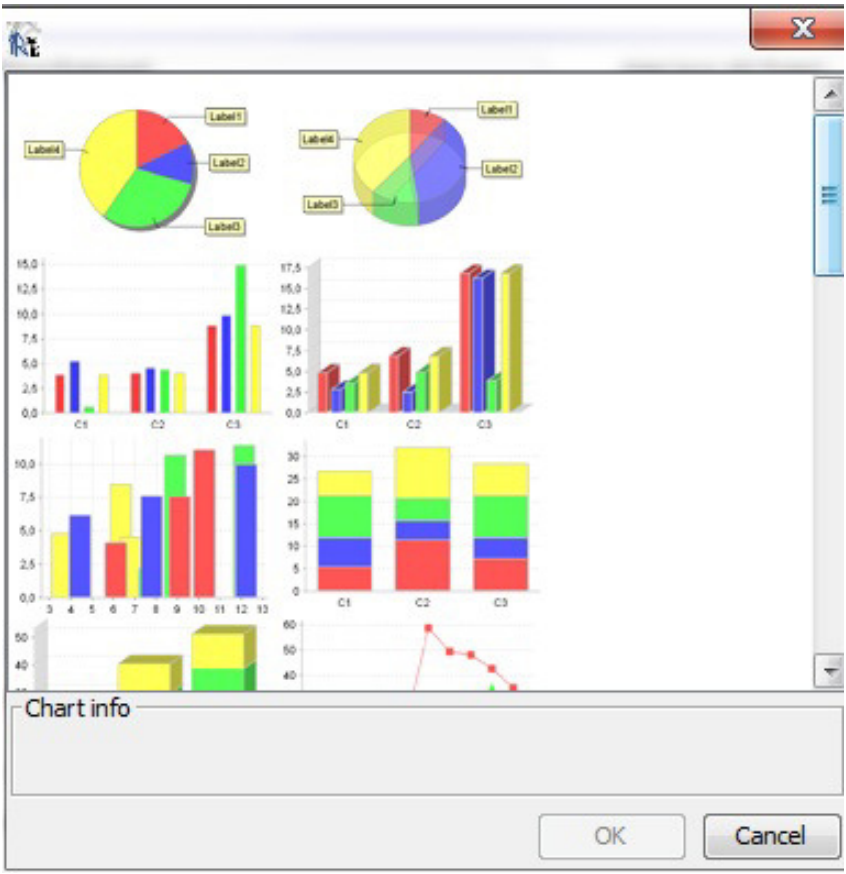


Figura 4.4: Selecionando o gráfico

Será disponibilizada uma tela de `Wizard` para configuração do componente `Chart`. Na opção `Dataset`, mantenha selecionado `Main report dataset`, clique no botão `próximo` e avance para a tela seguinte.

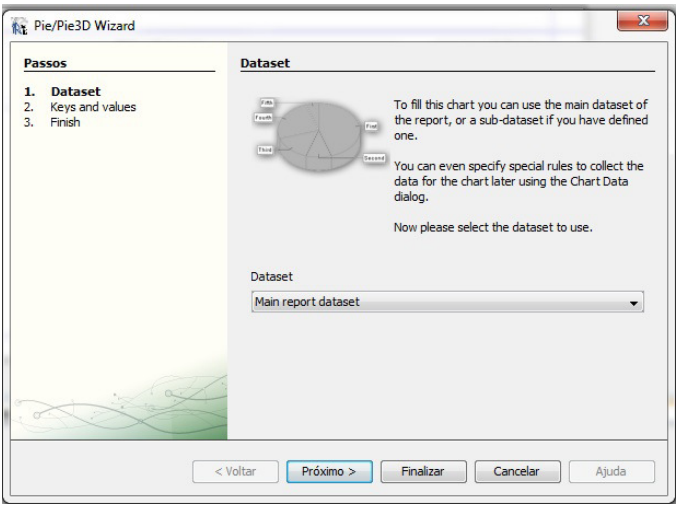


Figura 4.5: Configurando Chart

Nesta tela, inicialmente configuraremos a propriedade `unique identifier`. Clique no botão localizado na extremidade da direita, da área de *input* da propriedade `unique identifier`.

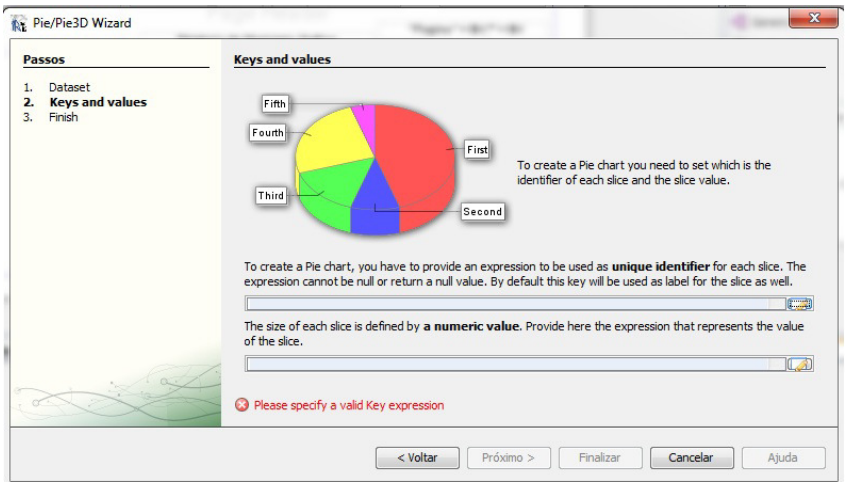


Figura 4.6: Configurando unique identifier

Será disponibilizada uma tela conforme mostra a figura a seguir.

Selecione `fields` e clique duas vezes na opção do field `nome`. Depois, clique no botão `apply`.

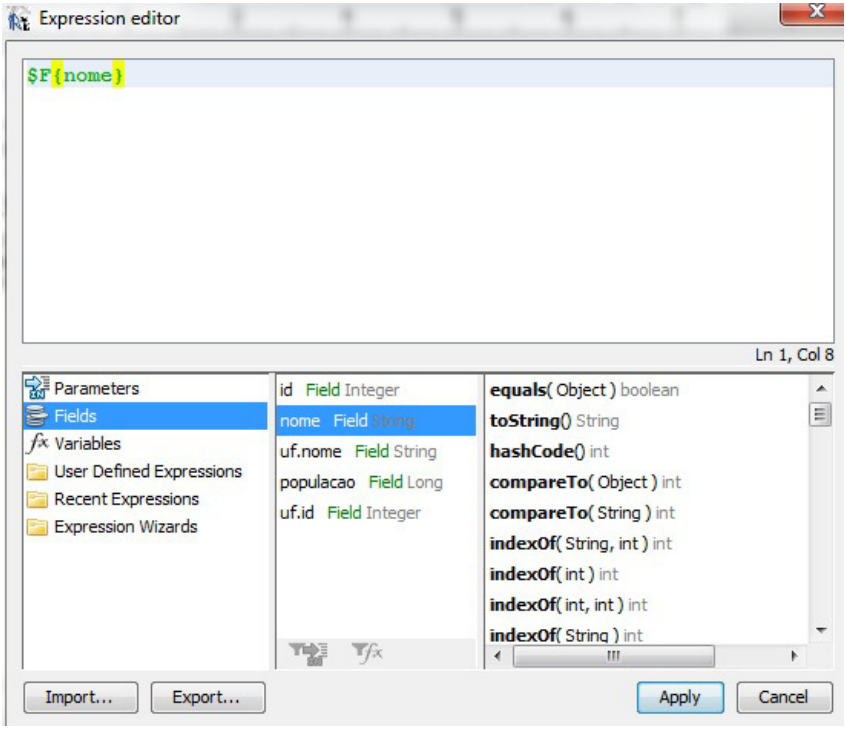


Figura 4.7: Selecionando field nome

Faremos o mesmo processo para configurar a propriedade `numeric value`. Clique no botão localizado na extremidade direita da área de *input* da propriedade.

Será mostrada uma tela igual à figura seguinte. Selecione `fields` e clique duas vezes na opção do `field populacao`, em seguida, clique no botão `apply`.

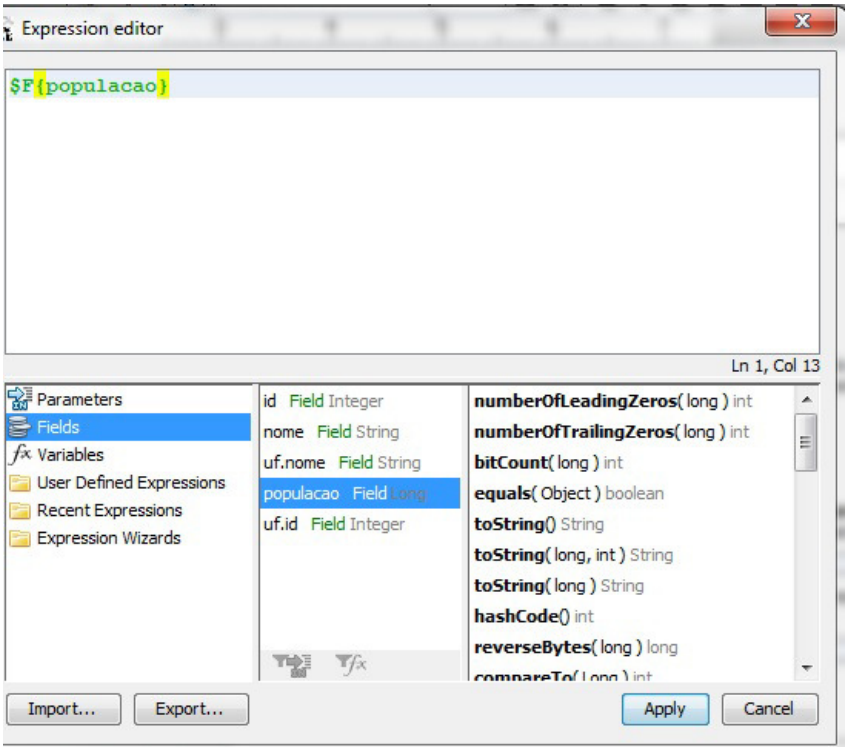


Figura 4.8: Selecionando field populacao

Após configurar as duas propriedades, clique no botão `Próximo`.

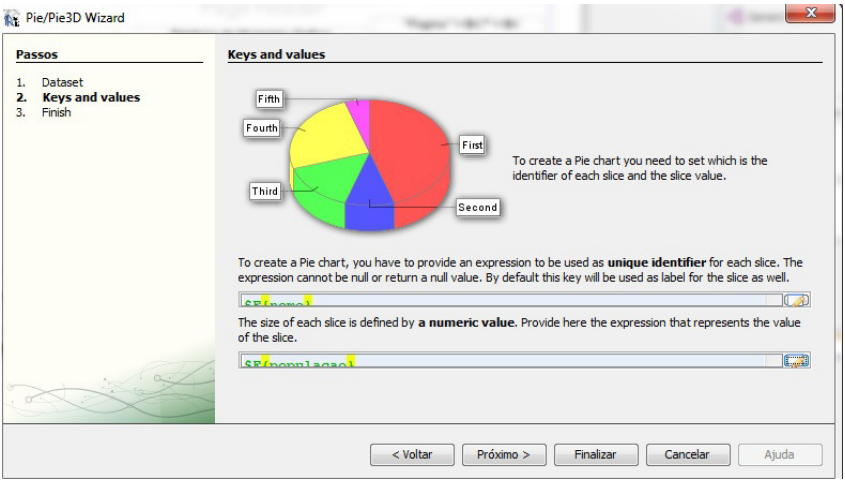


Figura 4.9: Clique no botão Próximo

Para finalizar, clique no botão `Finalizar`, e redimensione e posicione o componente `Chart`. Clique no componente e, em seguida, no botão direito do mouse. Será disponibilizado um menu conforme mostra a figura:

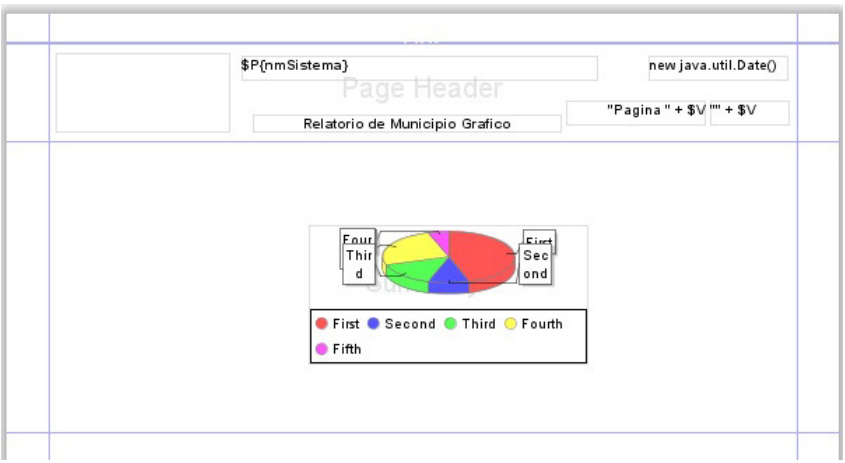


Figura 4.10: Desing do relatório

Clique na opção `Chart Data`. Assim, a tela de configuração

como verificamos na figura a seguir será exibida.

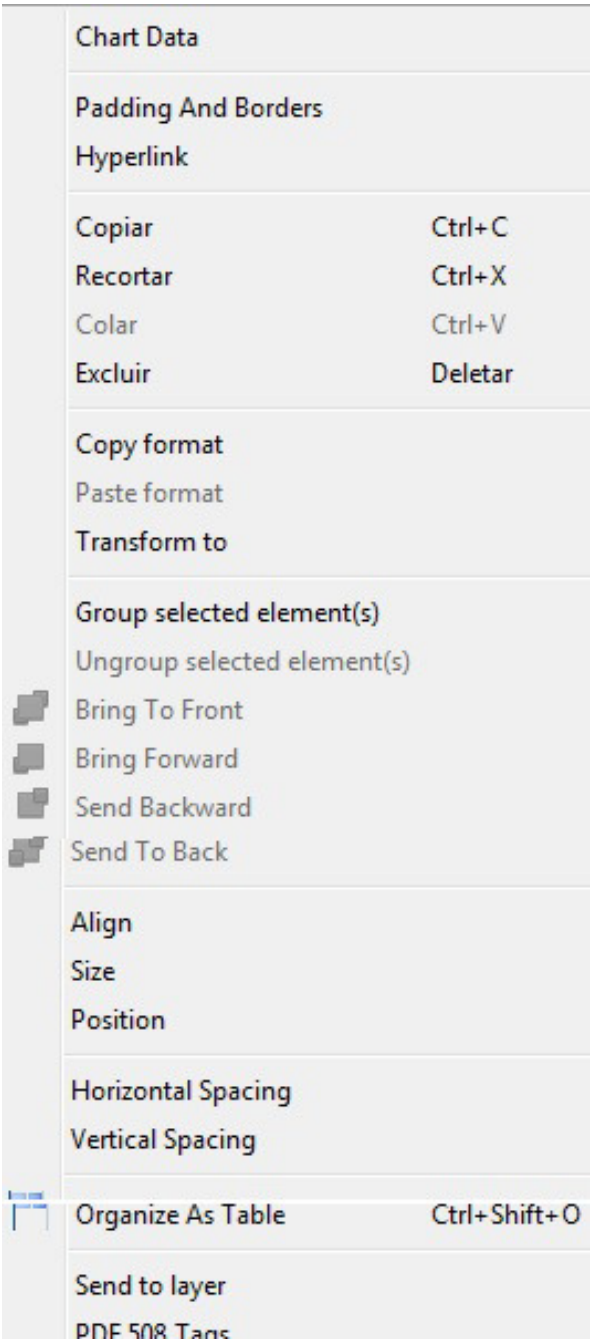


Figura 4.11: Configurando Chart

Na tela `Chart Details`, clique na aba `Details` e será disponibilizada uma tela com os detalhes da configuração com as duas propriedades que já configuramos, conforme mostra a figura:

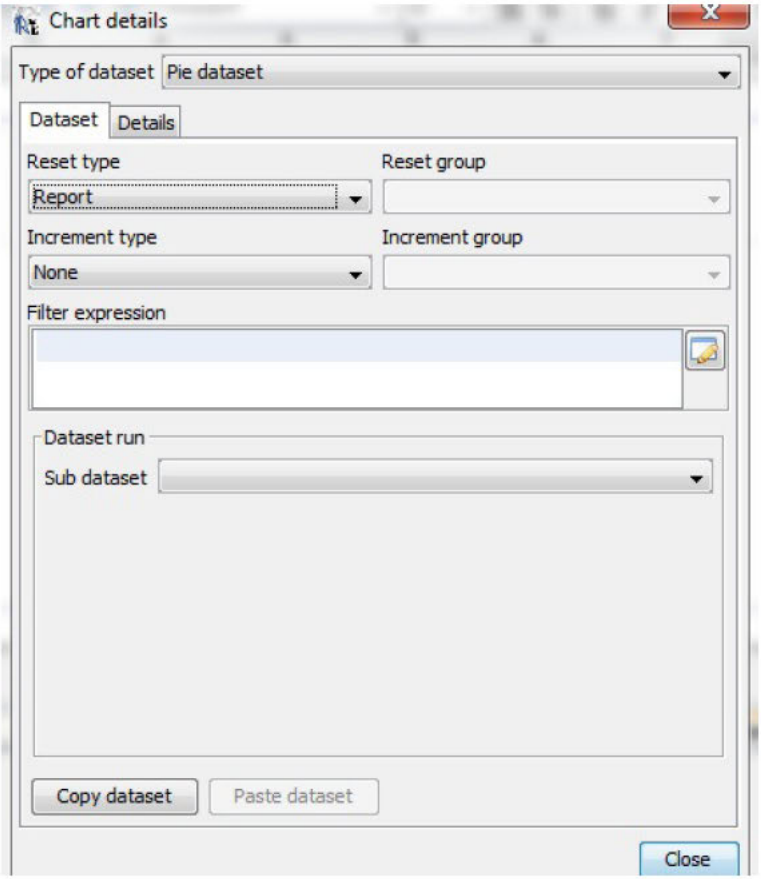


Figura 4.12: Aba Details

- A propriedade `Key expression` representa o nome do Município que será visualizado na legenda.
- A propriedade `value expression` determina o tamanho da fatia da pizza.
- A propriedade `Label expression` determina um

label para a fatia da pizza. Essa propriedade só aceita tipo `String` e, como no nosso contexto quero que mostre a população e esta é do tipo `long`, tenho de concatenar com espaço em branco para transformar população em `String`; caso contrário, ocorreria um erro.

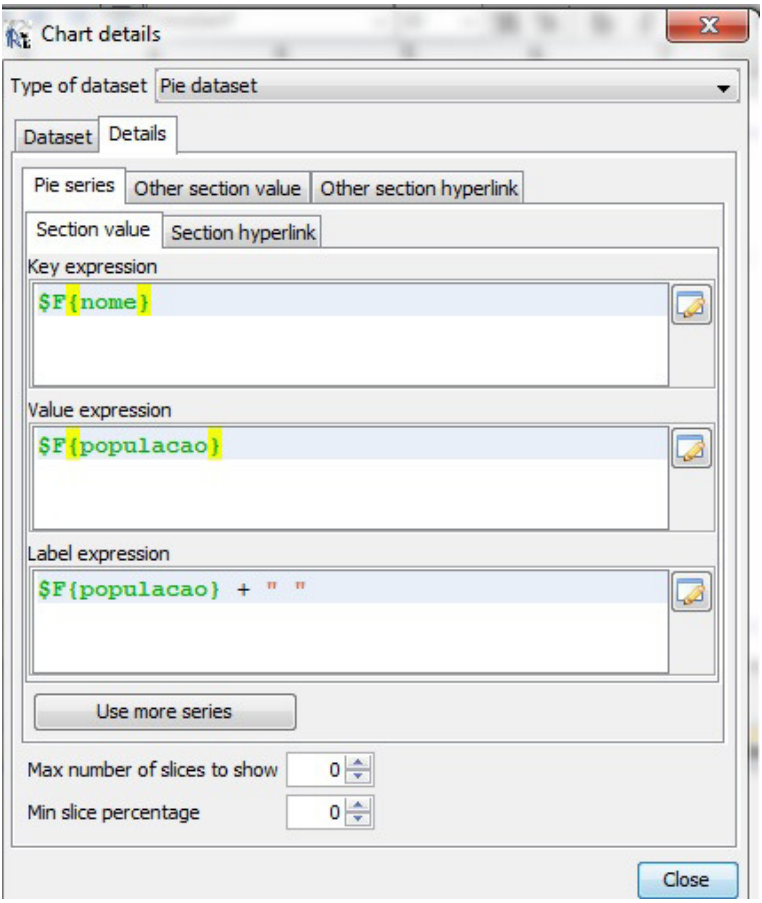


Figura 4.13: Configurando label expression

Clique em `close`. Terminamos a implementação do relatório gráfico; agora, é só compilar e testar.

4.2 CONCLUSÃO

Neste capitulo aprendemos a:

- Adicionar e configurar o componente Chart.
- Implementar um relatório com gráfico.

No próximo capítulo, aprenderemos a implementar relatório com sub-relatório.

Capítulo 5

RELATÓRIO COM SUB-RELATÓRIO

5.1 CRIANDO RELATÓRIOS COM SUB-RELATÓRIO

Layout do relatório

Neste capítulo, implementaremos o relatório de unidade federativa com sub-relatório de uma listagem de clientes que moram na unidade federativa selecionada. No nosso exemplo, vamos buscar uma lista de clientes que moram no Ceará, conforme mostra a figura seguinte.

SISTEMA DE GERENCIAMENTO DE CLIENTE 05/07/2015

Pagina 1 de 1

Relatorio de UF com SubRelatório

Codigo	Nome	Sigla
1	CEARÁ	CE

Lista de Clientes

Código	Nome
1	GABRIELA LAGUNA
2	JULIA MAIA
3	LUCAS MORAIS
4	MATHEUS FILHO
5	TIMOTEO LOPES

Figura 5.1: Layout do relatório

Método gerarRelatorio

O método `gerarRelatorioSub` é responsável pela execução e visualização do relatório, como também por acrescentar parâmetros comuns para todos os relatórios. Como seu objetivo é gerar um relatório com sub-relatório, nos parâmetros do relatório, acrescentaremos o caminho completo da localização do arquivo do sub-relatório.

Instanciamos a classe `HttpServletResponse` para que o usuário possa visualizar o relatório.

```
public void gerarRelatorioSub(String nomeRelatorio,HashMap
 paramRel,List listaRel,List listaRelSub,
 String subNomeRelatorio){
  FacesContext context = FacesContext.getCurrentInstance();
    HttpServletResponse response =
      (HttpServletResponse) context.getExternalContext()
```

```
                                .getResponse();
    ServletContext sc =
     (ServletContext) context.getExternalContext().getContext();
    String relPath = sc.getRealPath("/");
    String imagemLogo =
      relPath + "resources/imagens/logo_mmo.jpg";
    paramRel.put("imagemLogo", imagemLogo);
  paramRel.put("nmSistema", Constants.NOME_SISTEMA);
    paramRel.put("REPORT_LOCALE", new Locale("pt", "BR"));
  subNomeRelatorio =
    relPath + "relatorios/" + subNomeRelatorio+".jasper";
  paramRel.put("subNomeRelatorio",subNomeRelatorio);
```

Após receber dois ArrayList como parâmetro, o ArrayList `listaRel` com as informações do relatório principal e o `listaRelSub` com as informações do sub-relatório, a classe `JRBeanCollectionDataSource` transforma-os em dois `datasource`, são eles: `rel` e `relSub`. Acrescentamos como parâmetro de relatório o *datasource* `relSub` e, em seguida, a classe `JasperFillManager` gera o relatório.

```
try {
    JRBeanCollectionDataSource rel =
      new JRBeanCollectionDataSource(listaRel);
    JRBeanCollectionDataSource relSub =
      new JRBeanCollectionDataSource(listaRelSub);
    paramRel.put("relSub",relSub);
    JasperPrint print = JasperFillManager.fillReport(relPath +
       "relatorios/"+nomeRelatorio+".jasper",paramRel,rel);
```

Com o relatório criado, configuraremos o objeto `response` para mostrar o relatório no formato `.pdf`, e a classe `JasperExportManager` exporta o objeto `print` para PDF.

```
      response.setContentType("application/pdf");
      response.addHeader("Content-disposition",
        "attachment; filename=\"" + nomeRelatorio + ".pdf\"");
      JasperExportManager.exportReportToPdfStream(print,
        response.getOutputStream());
      ServletOutputStream responseStream =
        response.getOutputStream();
      responseStream.flush();
      responseStream.close();
      FacesContext.getCurrentInstance().renderResponse();
```

```
      FacesContext.getCurrentInstance().responseComplete();
    } catch (Exception e) {
      e.printStackTrace();
    }
}
```

Classe UFMB

Na classe `UFMB`, adicionamos o método `relatorio`, que será responsável pela execução do método `gerarRelatorioSub`. Neste caso, passamos como parâmetro do relatório o nome do relatório e do sub-relatório. Acrescentamos dois ArrayList na passagem de parâmetro do método `gerarRelatório`: `listagemResultadoBusca` com as informações do relatório principal, e o `listagemSubRel` com as informações do sub-relatório.

```
public void relatorioSubRel() {
  listagemResultadoBusca =  getUfDao().consulta(uf);
    List<Cliente> listagemSubRel =
      getClienteDao().clientePorUF(uf);
    HashMap paramRel = new HashMap();
    String nomeRelatorio = "relUFCSREL";
    String subNomeRelatorio = "subRelCliente";
    gerarRelatorioSub(nomeRelatorio, paramRel,
      listagemResultadoBusca,listagemSubRel,subNomeRelatorio);
}
```

Iniciaremos pelo sub-relatório. Neste caso, não temos parâmetros e nem cabeçalho, redimensionamos somente as áreas `Column Header` e `Detail`, e criamos somente os fields. Veja a figura a seguir:

Figura 5.2: Criando sub-relatório

Em seguida, desenhamos o relatório, conforme mostra a figura adiante. Ao finalizar, é importante compilar.

Figura 5.3: Desenho do sub-relatório

No relatório que será o principal, criamos os parâmetros, os fields, o cabeçalho e `detail`, e redimensionamos a área `Summary`.

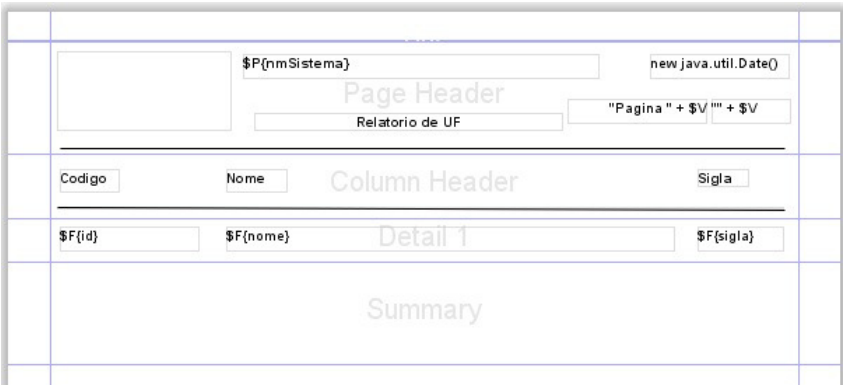


Figura 5.4: Criando parâmetro, fields, detail e cabeçalho

Criamos o parâmetro `subNomeRelatório` que receberá o path do arquivo do sub-relatório e o parâmetro `relSub` que receberá o datasource que será usado pelo sub-relatório.

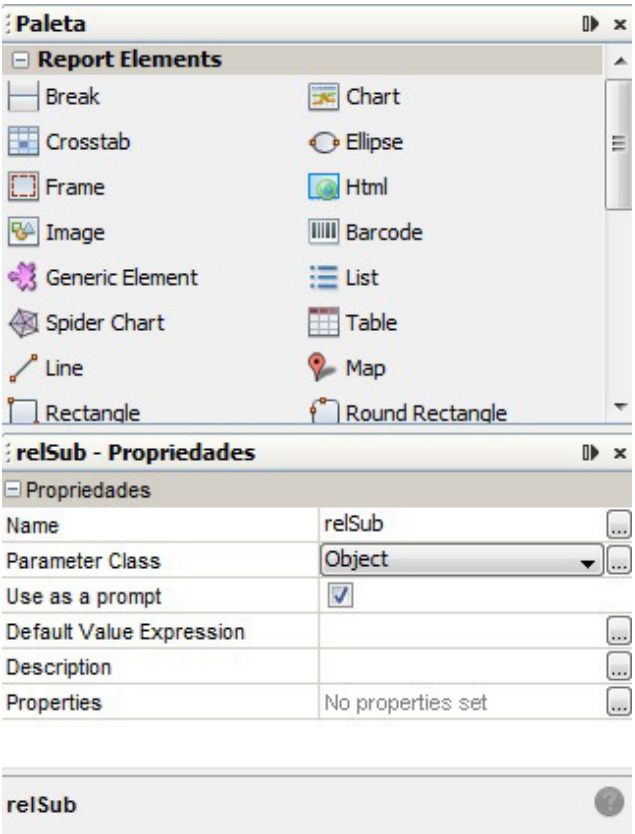


Figura 5.5: Parâmetro sub-relatório

No menu `Paleta`, selecione o componente `subReport`, arraste-o para a área `Summary` e solte. Será visualizada uma tela de wizard, conforme mostra a figura:

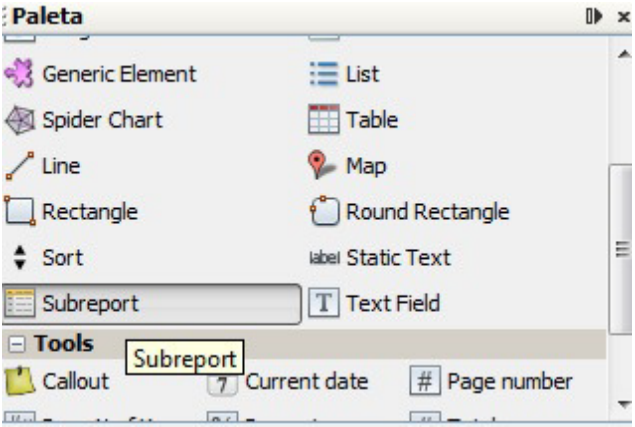


Figura 5.6: Selecionando componente SubReport

Na tela `Subreport wizard`, marque a opção `Just create the subreport element` e, depois, clique no botão para finalizar.

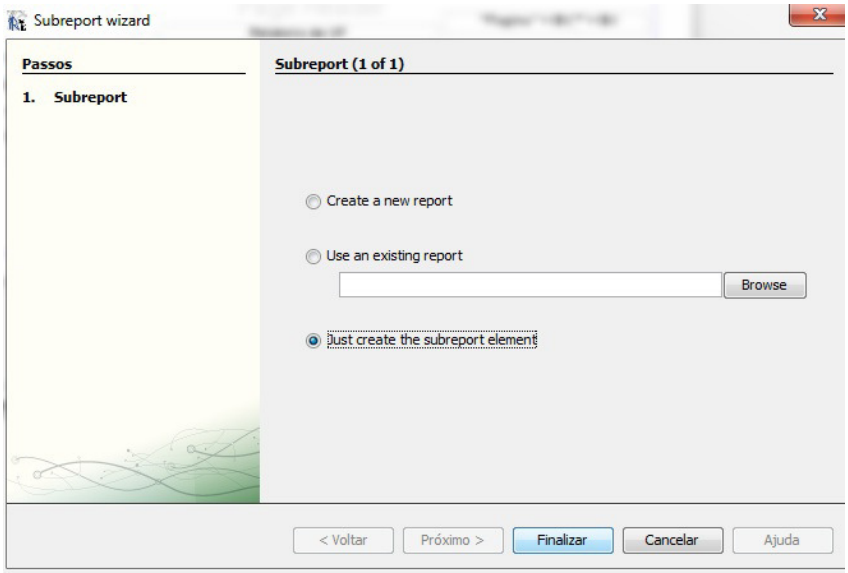


Figura 5.7: Finalizando Wizard

Redimensione e posicione o componente `Subreport`.

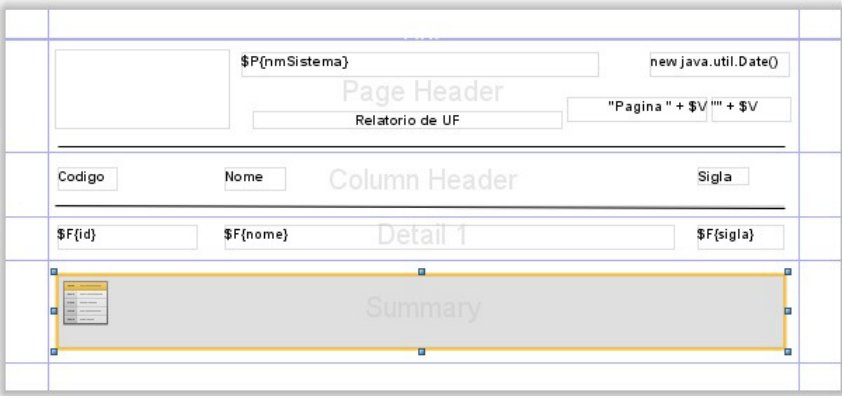


Figura 5.8: Redimensionando o componente Subreport

Clique no componente `Subreport` e, no menu `Propriedades` na propriedade `Subreport Expression`, clique no botão. Assim, uma tela para configuração será mostrada, como verificamos na figura:

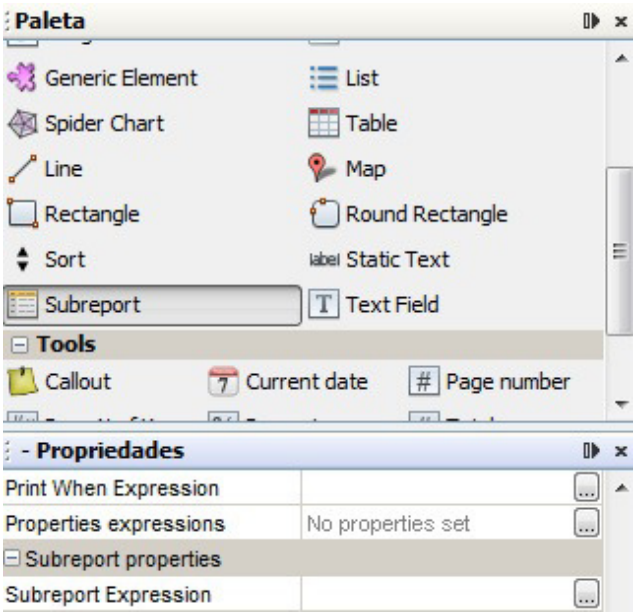


Figura 5.9: Configurando o Subreport

Clique na opção `Parameters` e, na lista de parâmetro, clique duas vezes no parâmetro `subNomeRelatório` . Depois, clique no botão `OK` .

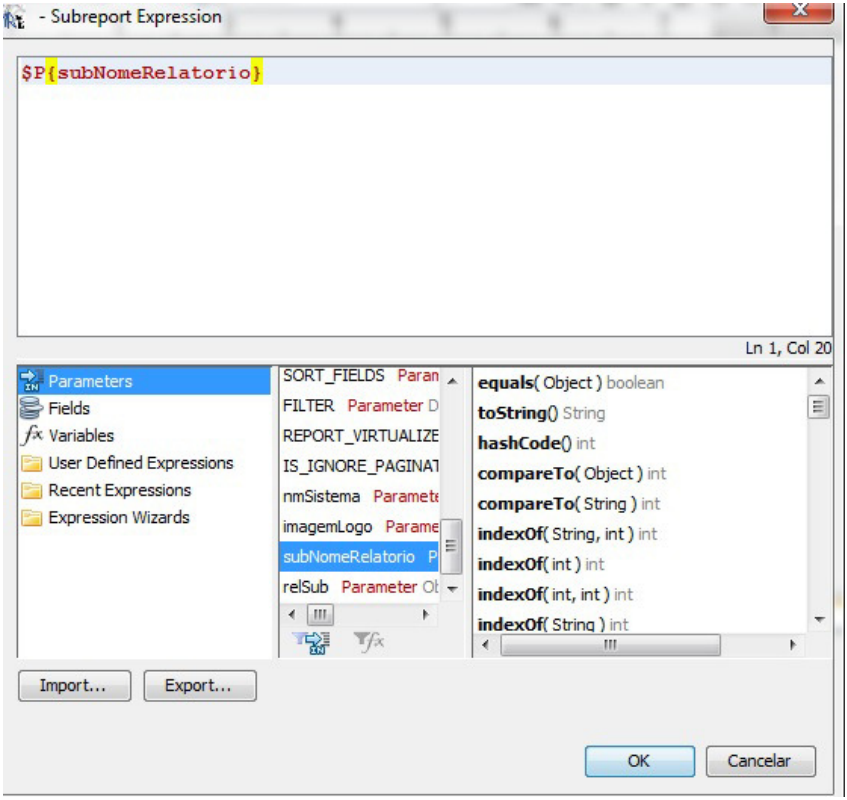


Figura 5.10: Configurando o parâmetro subNomeRelatório

Clique no componente `Subreport` e, no menu `Propriedades` na propriedade `connection type` , será disponibilizada uma lista de tipos. Selecione a opção `Use a datasource expression` e, em seguida, será habilitada a propriedade `Data Source Expression` , conforme mostra a figura:

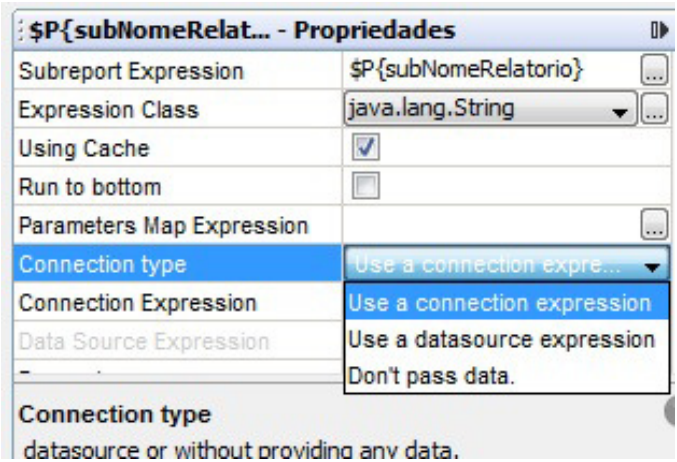


Figura 5.11: Configurando a connection type

Clique no componente `Subreport` e, no menu `Propriedades` na propriedade `Data Source Expression`, clique no botão e será mostrada uma tela para configuração.

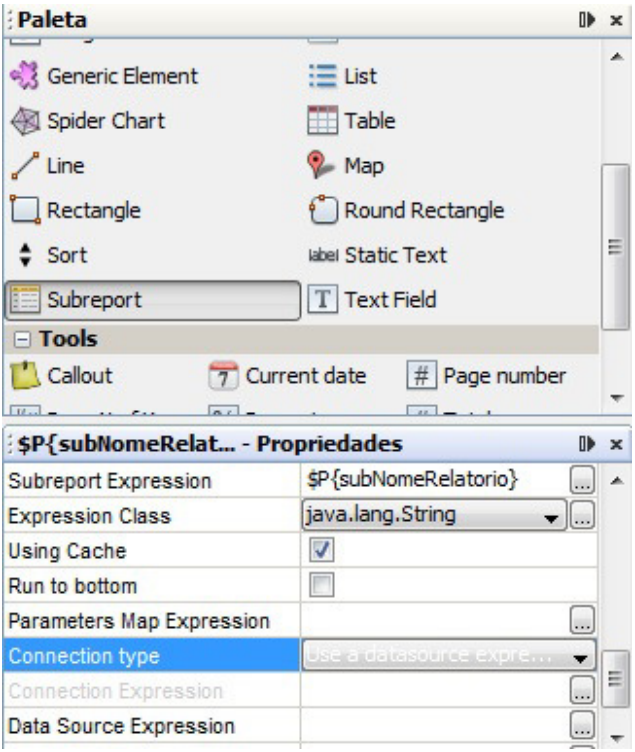


Figura 5.12: Configurando Data Source

Clique na opção `Parameters` e, na lista de parâmetros, clique duas vezes no `relSub` e, depois, no botão `OK`.

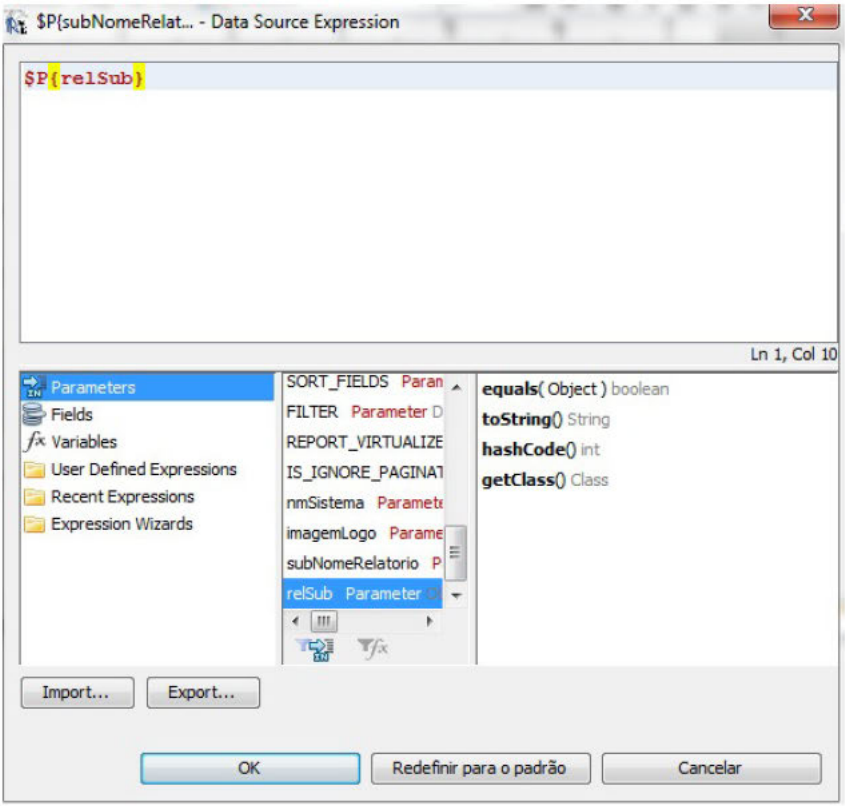


Figura 5.13: Selecionando Data Source

Terminamos a implementação do relatório com sub-relatório. Agora é só compilar e testar.

5.2 CONCLUSÃO

Neste capitulo aprendemos a:

- Adicionar e configurar o componente SubReport.
- Implementar um relatório com sub-relatório.
- Implementar o método necessário para gerar um relatório com sub-relatório.

No próximo capítulo, aprenderemos a implementar relatório usando map.

Capítulo 6

RELATÓRIO COM MAP

6.1 CRIANDO RELATÓRIOS COM MAP

Layout do relatório

Neste capítulo, nosso desafio será implementar o relatório de cliente com um mapa da localização do seu endereço. No nosso exemplo, vamos mostrar o mapa da localização da casa em que nasceu o escritor cearense José de Alencar em Fortaleza - CE, conforme mostra a figura:

SISTEMA DE GERENCIAMENTO DE CLIENTE 17/02/2015

Relatório de ClIMap Page 1 of 1

Código 8 Latitude -3.811666 Longitude -38.47834

Nome CASA DE JOSÉ DE ALENCAR

Endereço Avenida Washington Soares, 6055

Observação

A Casa de José de Alencar está situada no Sítio Alagadiço Novo, no bairro de Messejana, Fortaleza-CE e foi adquirido em 1825 pelo padre José Martiniano de Alencar, pai do escritor cearense José de Alencar, personagem principal da nossa história.Por nove anos, este espaço foi o lar do escritor, autor dos mais renomados títulos da Literatura Nacional, com destaque para as obras "Iracema" e "O Guarani", que foram fortemente influenciadas pelas belezas naturais do estado do Ceará. Fonte : http://www.cja.ufc.br/

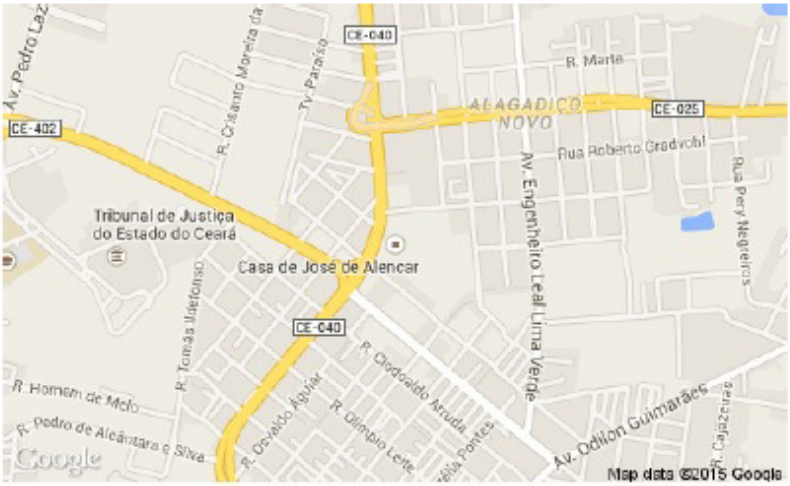


Figura 6.1: Layout do relatório

Classe ClienteMB

Na classe `ClienteMB` , adicionamos o método `relatorio` que será responsável pela execução do método `gerarRelatorio` .

```
public void relatorio() {
```

```
        listagemResultadoBusca = getClienteDao().consulta(cliente)
;
        HashMap paramRel = new HashMap();
        String nomeRelatorio = "relCliMap";
        gerarRelatorio(nomeRelatorio, paramRel,listagemResultadoBu
sca);
    }
```

Iniciamos esse relatório adicionando parâmetros, fields e cabeçalho, redimensionando as áreas `Column Header` e `Summary`, e inibindo as demais.

Vale ressaltar que os fields de latitude e longitude é do tipo `float` e receberam as coordenadas para definição da localização do mapa. Em seguida, desenhamos o relatório, como mostra a figura a seguir.

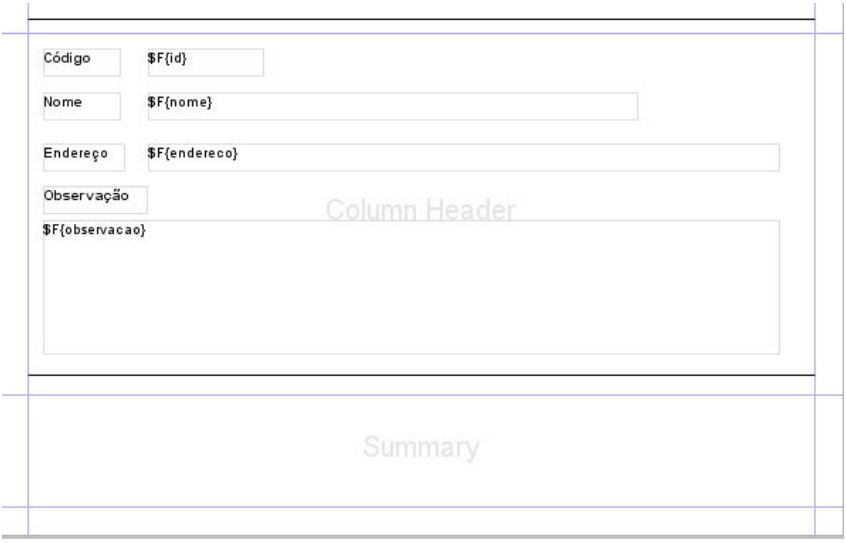


Figura 6.2: Desing do relatório

No menu `Paleta`, selecione o componente `Map`, arraste-o para a área `Summary` e solte. Redimensione e posicione-o, assim como mostra a figura:

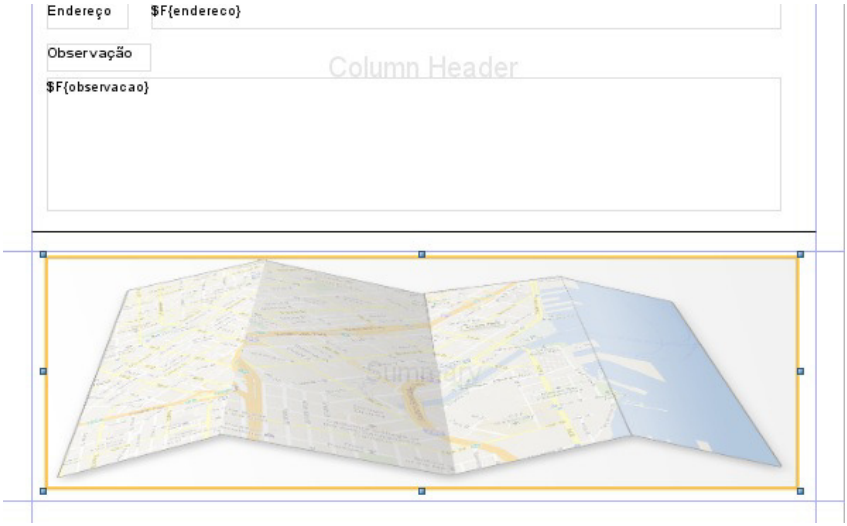


Figura 6.3: Selecionando o componente Map

Clique no componente `Map` e, no menu `Propriedades` na propriedade `latitude` , para exibir a tela de configuração, basta clicar no botão e uma tela de configuração será disponibilizada. Clique na opção `Fields` e, na lista de fields, clique duas vezes no field `latitude` e depois no botão `OK` .

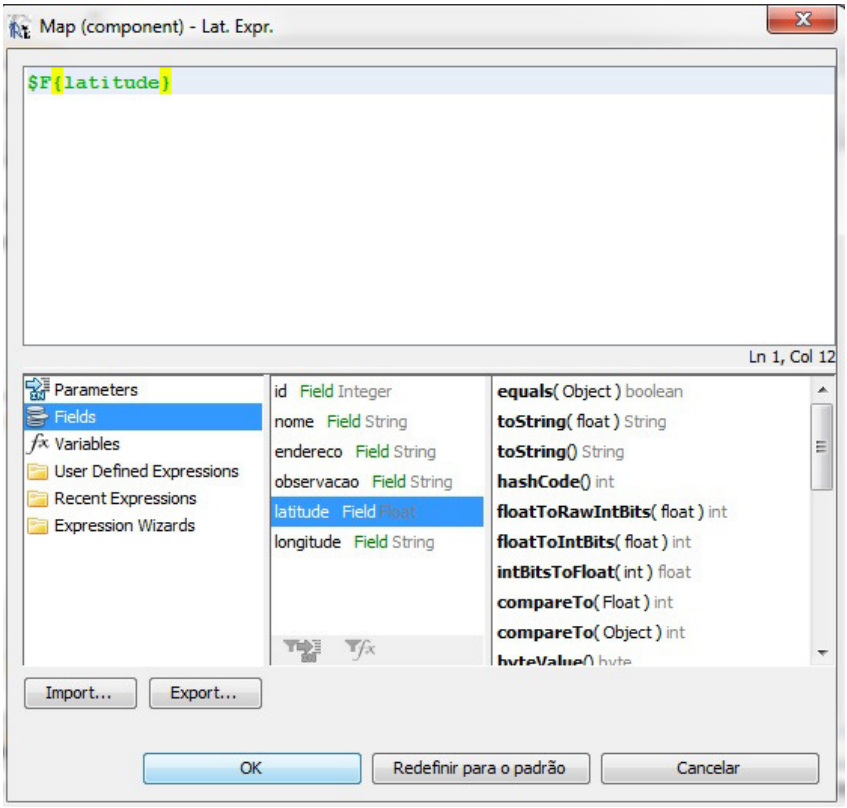


Figura 6.4: Configurando Latitude

Clique novamente no componente `Map` e, no menu `Propriedades` na propriedade `longitude`, para exibir a tela de configuração, basta clicar no botão e uma tela de configuração será disponibilizada. Clique na opção `Fields` e, na lista de fields, clique duas vezes no field `longitude` e depois em `OK`.

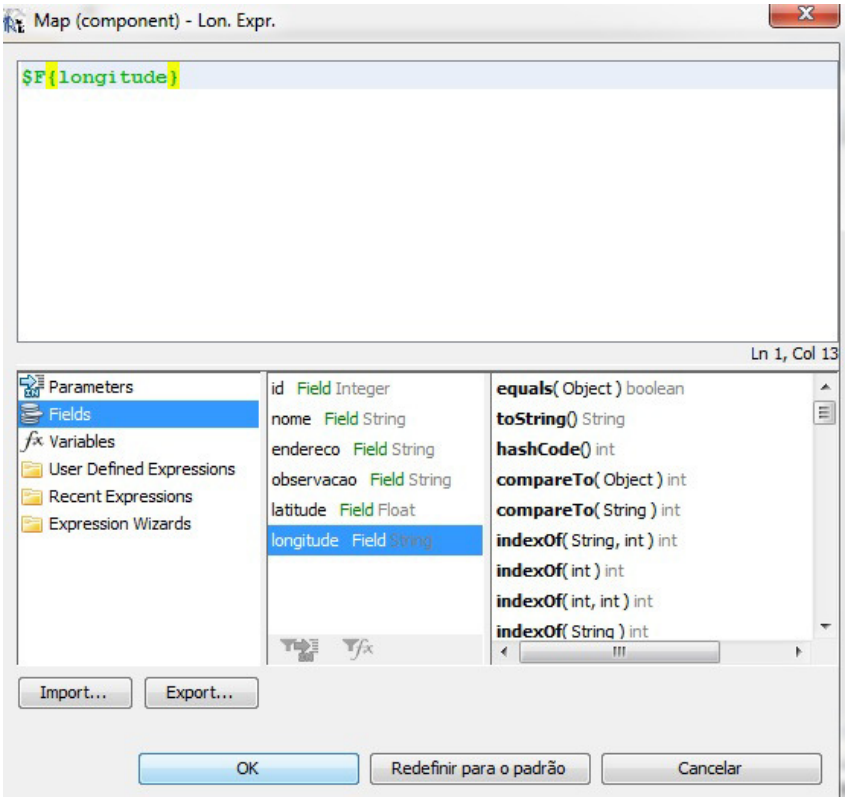


Figura 6.5: Configurando Longitude

Por último, clique mais uma vez no componente `Map` e, no menu `Propriedades` na propriedade `Zoom Expr`, informe o valor **15**, como é visto na figura:

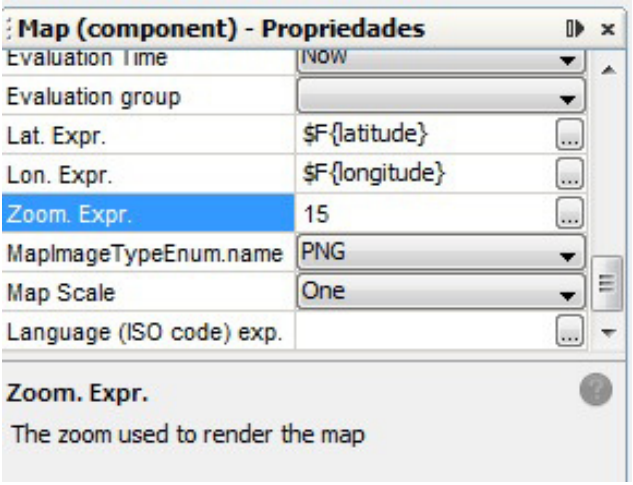


Figura 6.6: Configurando o Zoom Expr

Terminamos a implementação do relatório com Map. Agora é só compilar e testar!

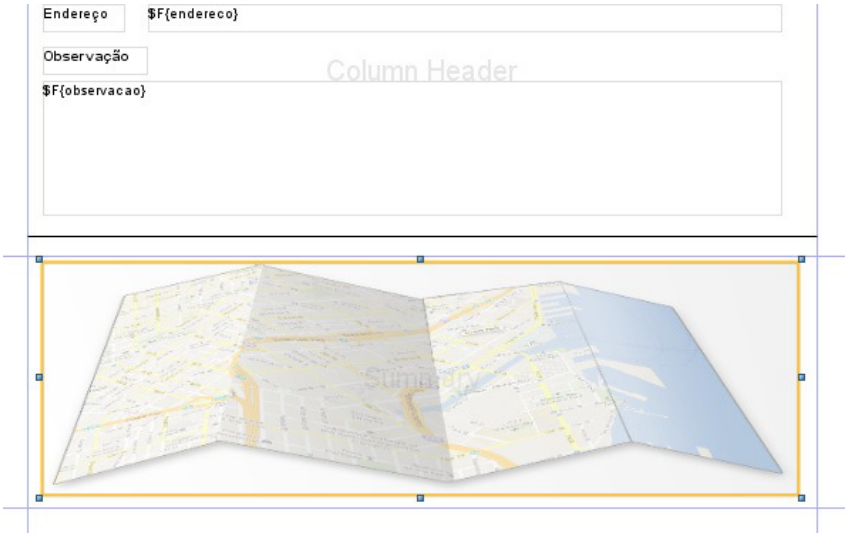


Figura 6.7: Finalizando o relatório

6.2 CONCLUSÃO

Neste capítulo aprendemos a:

- Adicionar e configurar o componente Map.
- Implementar um relatório com Map.

No próximo capítulo, aprenderemos a implementar relatório usando Crosstab.

CAPÍTULO 7

RELATÓRIO COM CROSSTAB

7.1 CRIANDO RELATÓRIOS COM CROSSTAB

Layout do relatório

Nosso último desafio será implementar o relatório de Municípios com Crosstab, conforme mostra a figura:

DEVELOPER

SISTEMA DE GERENCIAMENTO DE CLIENTE 22/02/2015

Pagina 1 de 1

Relatorio de Municipio

	Ceara	Rio de janeiro	Sao Paulo	Total Linha
Fortaleza	500	0	0	500
Aquiraz	50	0	0	50
Guaramiranga	300	0	0	300
Juazeiro do	200	0	0	200
Quixada	100	0	0	100
Rio de Janeiro	0	800	0	800
Sao Paulo	0	0	999	999
Total Coluna	1150	800	999	2949

Figura 7.1: Layout do relatório

Classe MunicipioMB

Na classe `MunicipioMB` , adicionamos o método `relatorio`

que será responsável pela execução do método `gerarRelatorioSub`.

```
public void relatorioCrossTab {
     HashMap paramRel = new HashMap();
     List<Municipio> listaRel =
         getMunicipioDao().consulta(municipio);
     String nomeRelatorio = "relMunicipioCT";
     gerarRelatorio(nomeRelatorio, paramRel,listaRel);
}
```

Iniciamos esse relatório adicionando parâmetros, fields, cabeçalho e agrupamento por UF, redimensionando a área `Summary`, e inibindo as demais. Veja a figura a seguir:

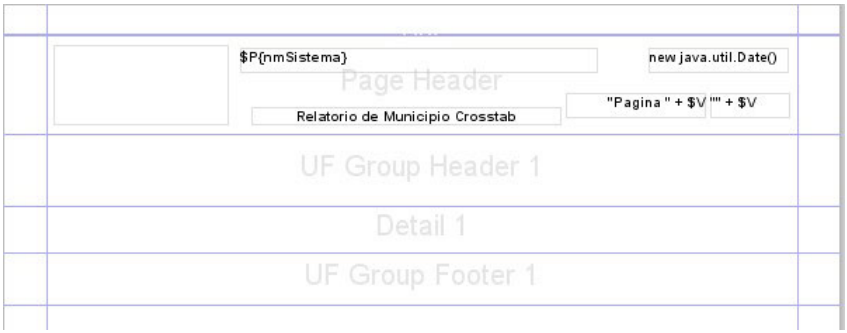


Figura 7.2: Desing do relatório

No menu `Paleta`, selecione o componente `Crosstab`, arraste-o para a área `Summary` e solte. Será visualizada uma tela de wizard, conforme mostra a figura:

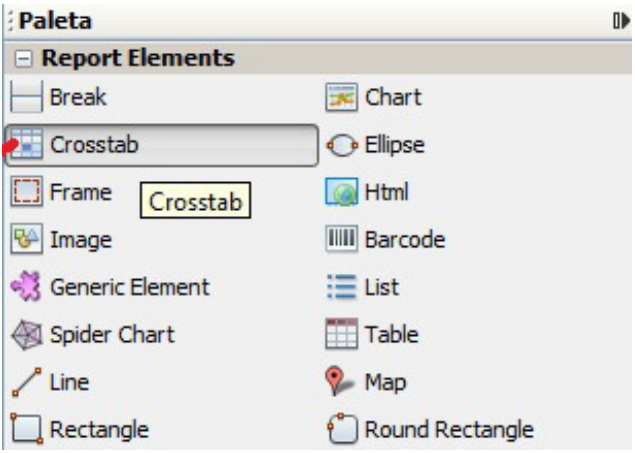


Figura 7.3: Selecionando componente Crosstab

Na tela inicial do `New crosstab`, clique no botão `Próximo`.

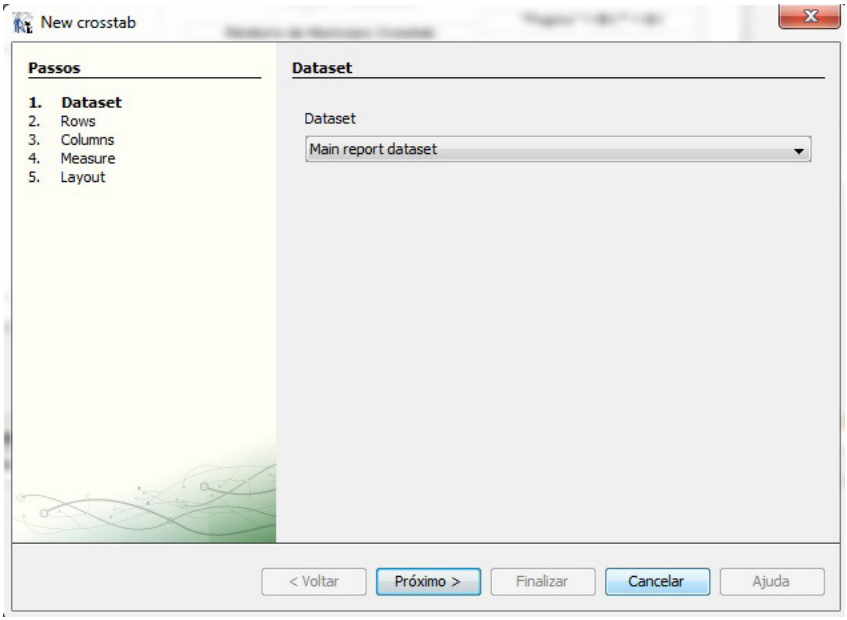


Figura 7.4: Iniciando configuração Crosstab

No passo `Rows`, dando início à configuração da `grid` na

`combo box group`, selecione o atributo `nome` que será impresso na primeira coluna da grid, conforme mostra a figura a seguir. Em seguida, clique no botão `Próximo`.

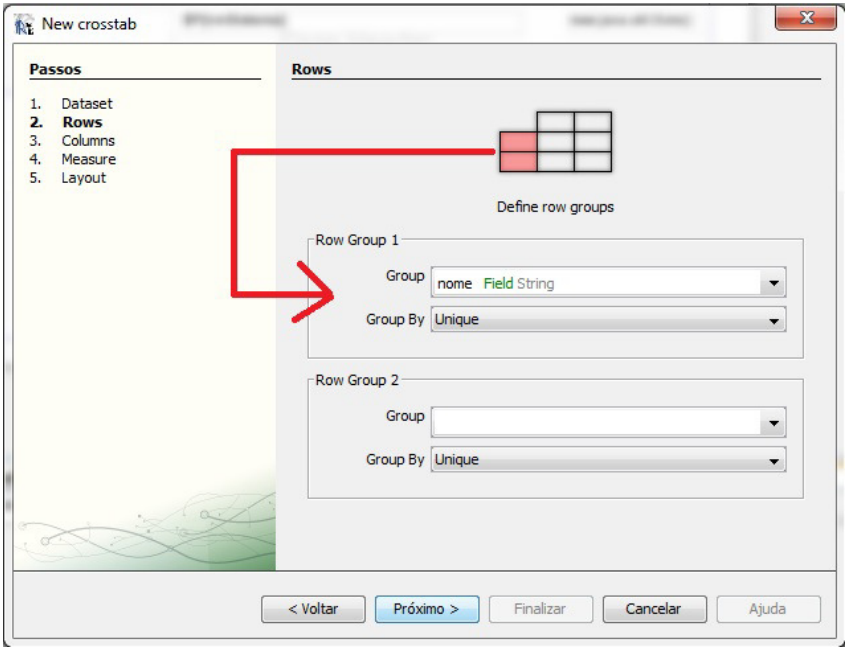


Figura 7.5: Selecionando field nome

No passo `Columns`, na `combo box group`, selecione o field `uf.nome` que será impresso na primeira linha da grid, como a figura a seguir. Depois, clique no botão `Próximo`.

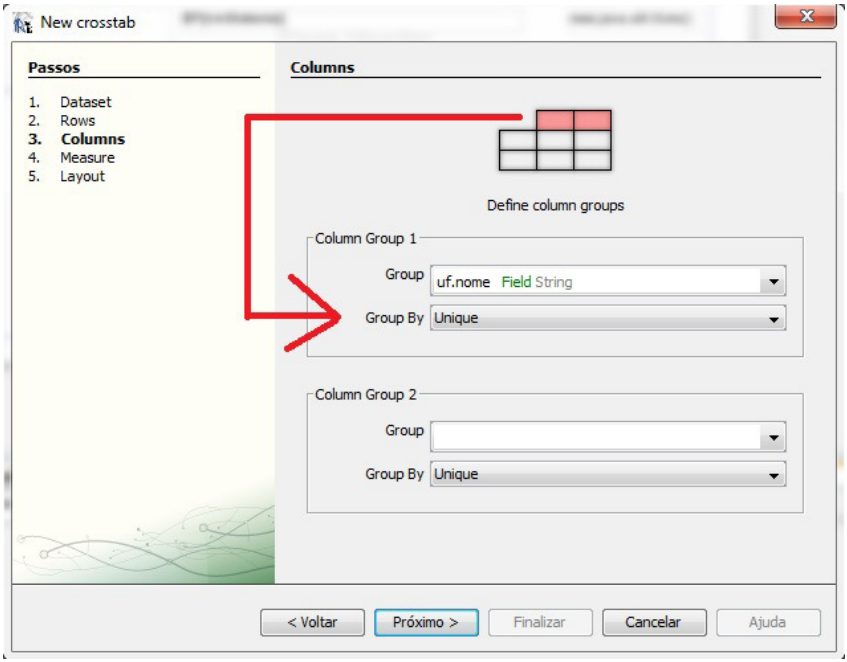


Figura 7.6: Selecionando field uf.nome

No passo `Measure`, na `combo box measure`, selecione o field `população` e, na `combo box function`, selecione a função `Sum`, como visto na figura adiante. Então, clique no botão `Próximo`.

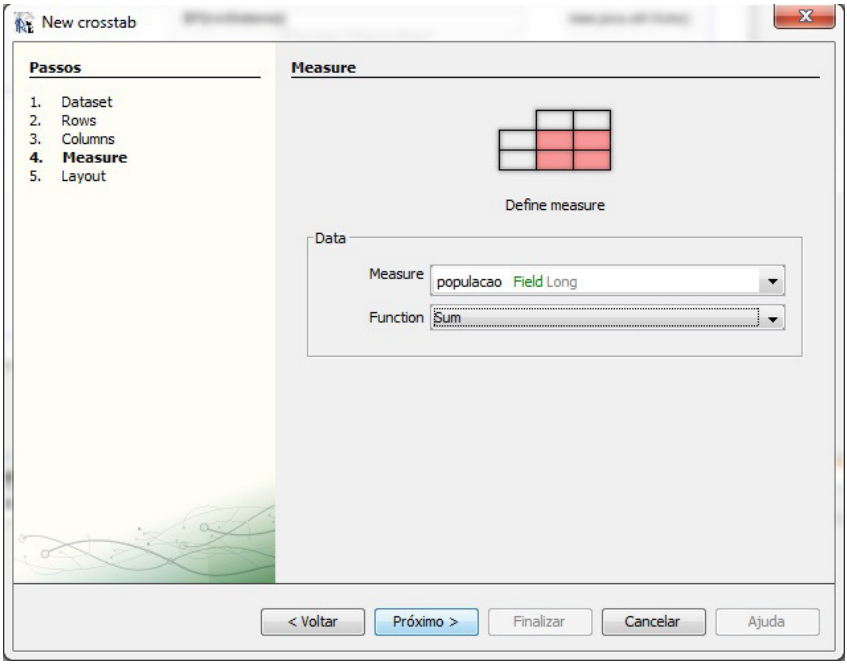


Figura 7.7: Selecionando field populacao

No passo `Layout`, para finalizar a configuração, opcionalmente podemos adicionar totais por linha e por coluna. Em nosso contexto, vamos manter os totais, conforme mostra a figura adiante. Em seguida, clique no botão `Finalizar`.

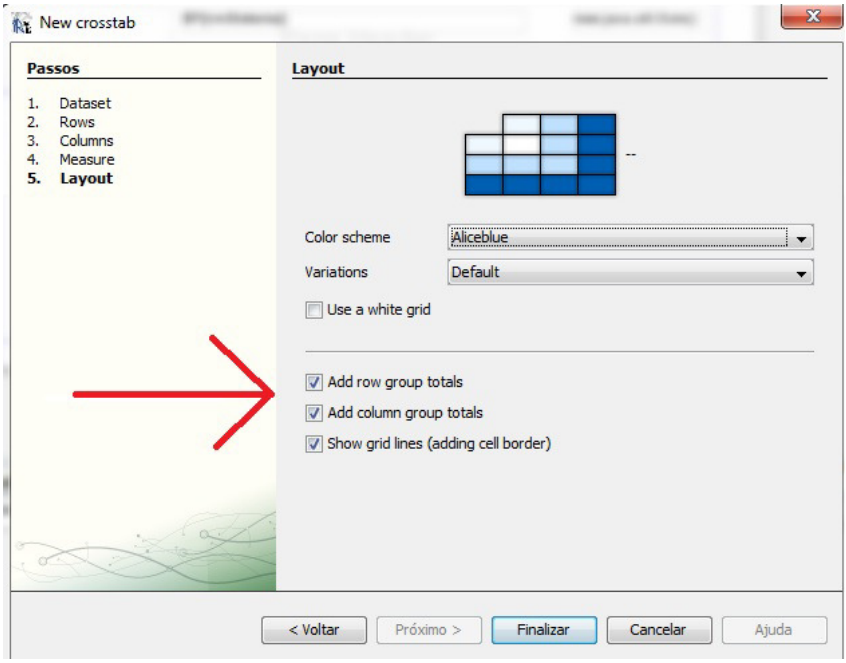


Figura 7.8: Adicionando Totais

Na tela de designer, clique na aba `Crosstab` .

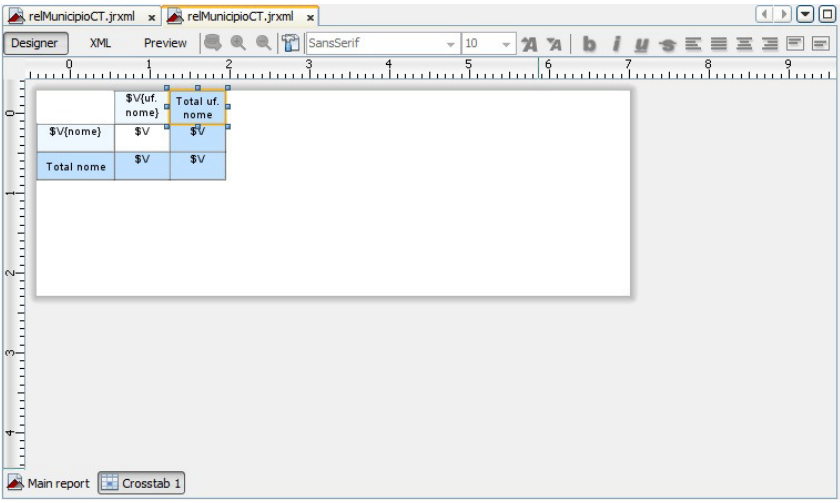


Figura 7.9: Selecionando aba crosstab

Nesta aba, opcionalmente podemos alterar o nome do label dos totais e redimensionar as colunas, como pode ser visto na figura:

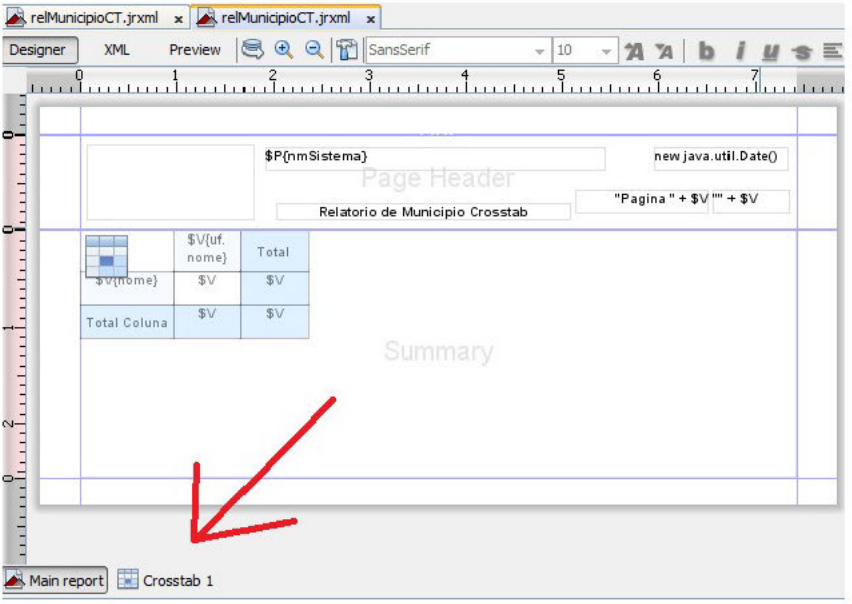


Figura 7.10: Alterar label de totais

Terminamos a implementação do relatório com Crosstab! Só resta compilar e testar.

7.2 CONCLUSÃO

Neste capítulo aprendemos a:

- Adicionar e configurar o componente Crosstab.
- Implementar um relatório com Crosstab.

Concluímos aqui o nosso livro que teve como objetivo ensinar a implementar os principais tipos de relatórios utilizando o iReport. Espero que o leitor possa aproveitar os conhecimentos adquiridos com a leitura deste livro no seu dia a dia como desenvolvedor.

Participe do nosso fórum de discussão, em https://forum.casadocodigo.com.br, para tirar dúvidas, críticas e sugestões.

Para baixar o código-fonte do aplicativo de demonstração, utilize o link do GitHub: https://github.com/mmmauricio/ireportCrieRelatoriosPraticosEelegantes.

CAPÍTULO 8

APÊNDICE

8.1 INTRODUÇÃO JASPERSOFT STUDIO

Segundo o site Jaspersoft Community, o Jaspersoft Studio é a nova ferramenta de geração de relatório baseado em Eclipse para JasperReports e JasperReports Server. É uma reescrita completa do iReport Designer, disponível como plugin Eclipse, e como um aplicativo independente. Jaspersoft Studio permite que você crie layouts de relatórios sofisticados que contêm gráficos, imagens, sub-relatórios, tabelas de referência cruzada e muito mais. Você pode acessar seus dados através de JDBC, TableModels, JavaBeans, XML, Hibernate, CSV, e fontes de costume, em seguida, publicar seus relatórios como PDF, RTF, XML, XLS, CSV, HTML, XHTML, texto, DOCX ou OpenOffice.

O principal objetivo da Jaspersoft Studio é fornecer os mesmos recursos e funcionalidades do já conhecido editor de relatório Jaspersoft, disponível no iReport Designer. Tendo seus fundamentos na plataforma Eclipse, Jaspersoft Studio é uma solução mais completa.

Designer iReport e Jaspersoft Studio também permitem configurar fontes de dados e utilizá-los para testar seus relatórios. Em muitos casos, os assistentes orientados por dados podem ajudar você a criar seus relatórios muito mais rápido. iReport Designer inclui o motor JasperReports, que permite que você visualize sua saída do relatório, teste e refine seus relatórios.

8.2 INTERFACE DO USUÁRIO

Jaspersoft Studio é oferecido em duas versões diferentes: um produto RCP autônomo, e uma versão plug-in do Eclipse. Pessoas que trabalharam com Eclipse vão estar familiarizados com a interface do usuário, enquanto para os novos usuários, ou os que estão familiarizados apenas com iReport Designer, a disposição dos elementos mostrados aparece bem diferente. Ambos o standalone e a versão plug-in têm uma interface similar. Na imagem a seguir, você pode ver uma prévia da interface Jaspersoft Studio, com as principais áreas em destaque:

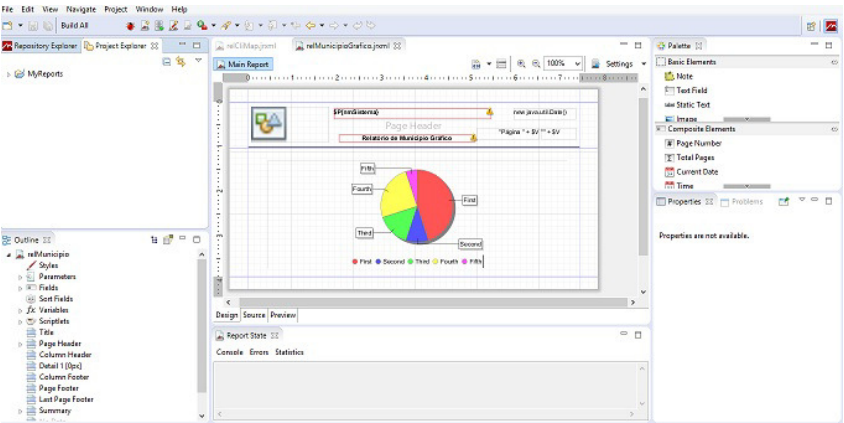


Figura 8.1: Interface

8.3 NOVOS RECURSOS DO JASPERSOFT STUDIO NÃO DISPONÍVEIS NO IREPORT DESIGNER

1. Um novo poderoso editor de expressão com a capacidade de usar funções embutidas e criar novas.
2. Herança Estilo.
3. A ajuda contextual disponível através da aplicação.
4. Exemplos disponíveis através do "New Project" wizard.

5. "Feedback" diálogo: permite enviar emissão / solicitação de recurso / bugs etc. diretamente para a comunidade, anexando registros úteis ou informações de software e hardware.
6. O suporte nativo para adaptadores de dados (permitindo ter provedor de editores / campos já durante a criação do relatório).
7. Melhoria no mecanismo para lidar com o relatório publicado e modificado no servidor.
8. Capacidade de fornecer valores de tamanho e posição recorrendo a diferentes valores de pixes para imagens (ou seja, digitando '0,5 centímetros').
9. Melhor gerenciamento classpath.
10. Melhoria (pixel) de precisão na concepção de relatório (especialmente quando se utiliza zoom).
11. Melhoria na maneira de criar elemento. No iReport, você tem que clicar na paleta; manter pressionado e soltar o elemento no editor.
12. Apoio a projetos.
13. Melhoria do ambiente de execução, com a possibilidade de definir dinamicamente o relatório.
14. O ambiente de execução fornece relatórios de estatísticas de execução precisos, como número de registro, o tamanho do arquivo, execução e tempo de execução.
15. Os relatórios são gerados de forma assíncrona, permitindo visualizar quase imediatamente as primeiras páginas de relatórios.
16. Capacidade de executar um relatório interativo e vê-lo dentro do visualizador de relatórios web integrada Jaspersoft Studio.
17. Habilidade para escrever e testar componentes JR e adaptadores de dados no Eclipse e testá-los diretamente no Jaspersoft Studio instalado como plugin do Eclipse.
18. Capacidade de gerenciar o suporte para fontes e componentes personalizados por projeto.

19. Licença EPL que permite a reutilização do Jaspersoft Studio em aplicações baseadas em Eclipse RCP.
20. Recursos disponíveis no Eclipse incluem a integração de controle de versão e histórico de alterações.

Jaspersoft Studio já está disponível para os sistemas operacionais Windows 64 bits ou de 32 bits, Linux 64 bits ou de 32 bits, Mac OS X 64 bits e Java Development Kit (JDK) 1.6 ou mais recente a partir deste.

Para maiores informações sobre Jaspersoft Studio você também pode consultar a documentação oficial.

Introduction to Jaspersoft Studio

http://community.jaspersoft.com/wiki/jaspersoft-studio-tutorials-archive

Jaspersoft Studio Features

http://community.jaspersoft.com/wiki/jaspersoft-studio-features

Capítulo 9

BIBLIOGRAFIA

JASPERSOFT COMMUNITY. *iReport Ultimate Guide*. 15 jan. 2013. Disponível em: http://community.jaspersoft.com/documentation.