UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

SISTEMA PARA GERENCIAMENTO DA DISTRIBUIÇÃO DO AUXÍLIO TRANSPORTE PARA ESTUDANTES

WILLY MANKE

BLUMENAU
2015

2015/2-24

WILLY MANKE

SISTEMA PARA GERENCIAMENTO DA DISTRIBUIÇÃO DO AUXÍLIO TRANSPORTE PARA ESTUDANTES

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Alexander Roberto Valdameri, Mestre - Orientador

BLUMENAU
2015

2015/2-24

SISTEMA PARA GERENCIAMENTO DA DISTRIBUIÇÃO DO AUXÍLIO TRANSPORTE PARA ESTUDANTES

Por

WILLY MANKE

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri, Mestre - Orientador, FURB
Membro:	Prof. Airan Arinê Possamai, Mestre – FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, dia de mês de ano [data da apresentação]

Dedico este trabalho aos familiares, amigos, professores e a todos que torceram, acreditaram e contribuíram para que este trabalho fosse realizado.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que sempre esteve presente.

Aos meus amigos, pelas colaborações e também pelas brincadeiras.

Ao meu orientador, professor Alexander Valdameri, por ter acreditado na conclusão deste trabalho e por ter colaborado com valiosas orientações.

No meio da dificuldade encontra-se a oportunidade.

Albert Einstein

RESUMO

O auxílio transporte oferecido por municípios é um fator de estímulo para os seus munícipes quanto a procura por formação em grandes centros. Dessa forma os estudantes podem desfrutar de uma vasta variedade de cursos oferecidos por instituições de ensino, normalmente localizadas em municípios maiores. Como pode ser observado em um município da região, a gerência dessa distribuição normalmente é realizada através de planilhas, a qual dificulta o levantamento de determinadas informações e acaba não sendo uma maneira segura de armazená-las. A carência de um sistema específico para o processo descrito foi motivador para o desenvolvimento deste trabalho. O objetivo principal é desenvolver um sistema para gerenciamento do auxílio transporte para prefeituras que ofereçam tal benefício, sendo que os objetivos específicos seriam administrar e gerenciar os estudantes beneficiados pelo vale transporte, controlar as empresas, os veículos e os motoristas que realizam o transporte, e possibilitar análise quantitativa a partir de relatórios do sistema para facilitar as tomadas de decisão. Tais objetivos foram alcançados por meio da construção de um sistema de gerenciamento de auxílio transporte, utilizando a linguagem JavaFX e a ferramenta Scene Builder. Para armazenamento das informações foi utilizado o MySQL versão 5.6 e para persistência dos dados o EclipseLink. Na questão dos relatórios foi utilizado o JasperSoft Studio versão 6.1.

Palavras-chave: Sistemas gerenciais. Auxílio transporte. JavaFX.

ABSTRACT

The transportation assistance provided by municipalities is a stimulating factor for their citizens as the demand for training in big cities. Thus students can enjoy a wide variety of courses offered by educational institutions, usually located in larger municipalities. As can be seen in a municipality of the region, the management of this distribution is usually performed through spreadsheets, which complicates lifting certain information and ends up not being a safe way to store them. The lack of a specific system for the described process was motivating for the development of this work. The main objective is to develop a system for managing transportation assistance to municipalities that provide this benefit, with the specific objectives would be to administer and manage the students benefited from the bus passes, control the companies, vehicles and drivers which transports and enabling quantitative analysis from system reports to facilitate decision-making. These objectives were achieved through the construction of a transportation allowance management system, using the JavaFX language and Scene Builder tool. For storage of the information we used the MySQL version 5.6 and persistence of data EclipseLink. In the matter of the reports was used Jaspersoft Studio version 6.1.

Key-words: Management systems. Transportation assistance. JavaFX.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ANSI – American National Standards Institute

API – *Application Programming Interface*

CRUD – *Create, Read, Update e Delete*

CSS - *Cascading Style Sheets*

DAO – *Data Access Object*

IDE – *Integrated Development Environment*

JDBC – Java Database Connectivity

JDK - Java Development Kit

JPA – Java Persistence API

JRE – Java Runtime Environment

JSR – Java Specification Request

JVM – Java Virtual Machine

ORM – *Object Relational Mapper*

SQL – *Structured Query Language*

URL – *Uniform Resource Locator*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Na última década no Brasil aumentou o número de alunos matriculados em cursos superiores, sendo que em 2012 a marca chegou a 7 milhões, o que representou um aumento de 4,4% em relação ao ano de 2011 (BRASIL, 2013a). Além disso, no Brasil em 2013 houve novamente um aumento de 3,8% nas matriculas, o que demonstra uma forte tendência neste setor que cresceu acima de 50% na última década, passando de 2,8 milhões de matrículas em 2000 para cerca de 6 milhões em 2010 (BRASIL, 2013b).

Ainda no Brasil em 2012 os 7.037.688 matriculados em cursos de graduação estavam distribuídos em 31.866 cursos oferecidos por 304 instituições públicas e 2.112 particulares (BRASIL, 2013a). Além disso, segundo o Ministério da Educação que divulgou uma lista das melhores universidades do Brasil, é notável que a maioria das universidades localizam-se em grandes centros, o que acaba atraindo a grande parte dos estudantes devido à variedade e qualidade dos cursos oferecidos (2013 apud PATI, 2014).

Em relação aos dados apresentados, é perceptível que diversos municípios comecem ou já forneçam auxílio ao transporte de estudantes até as instituições de ensino. Como exemplo na região sudeste, o município de Itatiba-SP oferece auxilio transporte aos universitários que frequentam cursos não existentes nas instituições da cidade, sendo que o auxílio varia entre R$ 60 e R$ 240 conforme a distância (ITATIBA, 2015).

Diante deste cenário o presente trabalho propõe desenvolver um software que facilite a gestão da distribuição do auxílio transporte. Esta ferramenta tem por objetivo possibilitar gerenciamento por parte do município na manutenção de um cadastro dos estudantes, das empresas, dos veículos que realizam o transporte, além de disponibilizar a geração de relatórios. Um exemplo é o relatório que demostre quantos estudantes frequentam um determinado curso ou um relatório que indique os veículos que estão sendo utilizados para transporte dos estudantes. Estes poderão ser utilizados como auxílio nas tomadas de decisão, como por exemplo, solicitar que a empresa que realiza o transporte forneça um veículo com um número maior de lugares, ou até mesmo um veículo extra para atender o número de estudantes.

1.1 OBJETIVOS

O objetivo geral do trabalho é desenvolver um sistema para gerenciamento do auxílio de transporte para prefeituras que ofereçam tal benefício. Os objetivos específicos são:

a) administrar e gerenciar os estudantes beneficiados pelo vale transporte;
b) controlar as empresas, os veículos e os motoristas que realizam o transporte;

c) possibilitar análise quantitativa a partir de relatórios do sistema para facilitar as tomadas de decisão.

1.2 ESTRUTURA

Este trabalho está dividido em quatro partes, no qual o segundo capítulo apresenta a fundamentação teórica com conceitos de sistemas gerenciais, leis municipais, gestão do transporte de pessoas, JavaFX, Java Persistence API (JPA), além de apresentar alguns trabalhos correlatos sobre o assunto ou tecnologia utilizada para que se possa realizar uma comparação com o presente trabalho.

O terceiro capítulo apresenta as tecnologias utilizadas para o desenvolvimento da aplicação, sendo demostrados alguns trechos de código, ferramentas e telas do sistema. Fazem parte desta especificação os diagramas de caso de uso os quais são descritos no Apêndice A e o modelo entidade-relacionamento. Além disso são apresentados alguns resultados e discussões em relação aos trabalhos correlatos, tecnologias utilizadas e entrevistas realizadas. O quarto capítulo apresenta conclusões sobre o trabalho desenvolvido e sugestões de trabalhos futuros, demostrando melhorias ou funcionalidades que poderiam ser acrescentadas à aplicação proposta.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos como sistemas gerenciais, leis municipais, gestão de transporte de pessoas, JavaFX, Java Persistence API, além dos trabalhos correlatos.

2.1 SISTEMAS GERENCIAS

Os sistemas de informação em geral possibilitam uma empresa avaliar melhor os seus processos e consequentemente controlar os seus resultados e objetivos esperados. Segundo Melo (2002, p. 30) “[...] todo e qualquer sistema que tem informações como entrada visando gerar informações de saída [...] para satisfazer determinadas necessidades, corresponde ao objetivo geral dos Sistemas de Informação”.

Os sistemas gerenciais devem ser analisados pelos executivos como uma possibilidade de adequação da empresa em relação a um cenário provável, economias nacionais ou internacionais, produtividade, custos, tecnologia, capacidade logística, capacidade de investir ou de produzir, dentre outros (OLIVEIRA, 2012, p. 29). Com isso os sistemas gerenciais tornam-se de elevada abrangência e importância nas empresas, gerando diversos benefícios como: redução de custos, melhoria no acesso às informações, melhoria na produtividade, melhoria na tomada de decisão, melhoria na adaptação (OLIVEIRA, 2012, p. 29).

Segundo Oliveira (2012, p. 24), “Um Sistema Gerencial é o desenvolvimento e a consolidação do processo administrativo, representado pelas funções de planejamento, organização, direção, gestão de pessoas e controle, voltado para a otimização dos resultados[...]”. A Figura 1 demonstra essas funções.

Figura 1 - Funções do processo administrativo

Fonte: Gestor Eficaz (2012).

Existem muitas situações em que empresas utilizam de maneira inadequada o processo administrativo, na maior parte das vezes, elas não se preocupam com o objetivo esperado e os meios de chegar até lá (OLIVEIRA, 2012, p. 23). Os executivos acabam não utilizando a sustentação administrativa, consequentemente não consegue manter o controle dos processos e nem avaliar nenhuma situação inerente a empresa (OLIVEIRA, 2012, p. 24).

2.2 LEIS MUNICIPAIS

Nos últimos cinco anos diversos municípios oferecem um incentivo às pessoas que desejam estudar. Muitos deles acabam concedendo auxílio ao transporte, para que os munícipes possam se deslocar a outros municípios em busca de tais oportunidades de estudo. Esse incentivo faz com que o município ofereça as empresas locais mais mão de obra qualificada e profissionais aptos.

A prefeitura de Massaranduba por exemplo, criou um conjunto de leis que garantem auxílio aos alunos. Conforme artigo 1º da lei nº 1285/2011: “O Município de Massaranduba prestará auxílio financeiro intitulado “BOLSA TRANSPORTE” aos estudantes de ensino superior e ensino médio profissionalizante, na forma da presente lei” (MASSARANDUBA, 2011). A lei nº 1285/2011 segue complementada pelo artigo 2º: “O valor do benefício concedido a qualquer aluno que comprove matrícula [...], será de 50% do valor gasto com o transporte escolar tomando como ponto de partida o centro de Massaranduba”

(MASSARANDUBA, 2011). A lei nº 1610/2014 altera o artigo 5º da lei nº 1285/2011 ficando com a seguinte redação: “Os alunos beneficiados com o Bolsa Transporte deverão cumprir, a critério da Administração Municipal e de acordo com a necessidade, prestação de serviços em trabalhos voluntários, programas, atividades e projetos sociais [...]” (MASSARANDUBA, 2014).

O Bolsa Transporte é disponibilizado em 50% e 100%, sendo que os alunos que receberem auxílio de 100% devem se comprometer a cumprir 10 horas e os que receberem 50% devem cumprir 5 horas semestrais acumulativas de serviços comunitários. Alunos que durante o trajeto atrapalharem o bom andamento, realizando algazarras ou praticando trotes, terão seu auxílio suspenso. Aos alunos que durante o semestre desistirem do curso, receberão uma multa de 10% em cima do valor fornecido de auxílio (MASSARANDUBA, 2014).

> A prefeitura de Leme no estado de São Paulo disponibiliza auxilio transporte para os estudantes da cidade através da lei nº 3.284 de 02 de abril de 2013. Artigo 1º - Fica criado o Programa Municipal de Auxílio-Transporte para estudantes universitários, PAE, que institui a transferência de recursos pela Administração Pública Municipal para estudantes matriculados em curso universitário, que tenham por objetivo o deslocamento do Município de Leme, para as instituições de ensino localizadas em outros Municípios. Artigo 2º - O Programa Municipal de Auxílio-Transporte instituído no artigo anterior se destina a beneficiar estudantes comprovadamente e regularmente matriculados em instituições particulares e públicas de ensino de nível superior, concedendo o auxílio, desde que preenchidos os requisitos dessa lei (LEME, 2013).

Além dos municípios citados até o momento tem-se também o município de Ituporanga que concede auxilio transporte aos estudantes da cidade através do programa “Universidade um Salto Para o Futuro”, implantado pela secretaria municipal de educação em 2013. O programa disponibiliza auxilio para estudantes matriculados em cursos de nível superior, profissionalizante ou capacitação que precisam se deslocar para outras cidades através de veículos de empresas particulares (ITUPORANGA, 2013).

> O município de Rio do Oeste no estado de Santa Catarina nos termos da Lei municipal nº 2002/2013, dispõe sobre o Programa Municipal de Auxílio Transporte para estudantes universitários e de centro federal de ensino tecnológico, que tenham por objetivo o deslocamento do Município de Rio do Oeste para as instituições de ensino localizadas em outros Municípios. Art. 1º Fica criado o Programa Municipal de Auxílio Transporte, que institui a transferência de recursos por órgãos e entidades da Administração Pública Direta e Indireta do Município para estudantes universitários e de centro federal de ensino tecnológico, que tenham por objetivo o deslocamento do Município de Rio do Oeste para as instituições de ensino localizadas em outros Municípios, garantindo o acesso dos estudantes. Art. 4º A ajuda financeira, na modalidade Auxílio Transporte, correspondente ao valor mensal de até R$ 60,00 (sessenta reais), será concedida mensalmente ao estudante que preencher os requisitos desta Lei, observando-se o art. 12 desta Lei (RIO DO OESTE, 2013).

Outro exemplo seria a prefeitura de Jacupiranga que criou uma lei que disponibiliza auxílio transporte aos estudantes residentes na cidade. Conforme artigo 1º da Lei nº 1.184:

“Fica instituído o Programa Bolsa – auxílio Transporte no Município de Jacupiranga, como objetivo de viabilizar aos estudantes residentes [...] acesso ao ensino Superior e Técnico Profissionalizante” (JACUPIRANGA, 2015).

2.3 GESTÃO DO TRANSPORTE DE PESSOAS

No período de 2014 a 2015 no Brasil foi movimentado mais de 130 milhões de usuários através de transportes rodoviários interestaduais e internacionais. A fiscalização de permissões e autorizações para essa operação de transporte de pessoas fica por conta da Agência Nacional de Transportes Terrestres que é constituída por Sociedades Empresarias para tal fim (BRASIL, 2015).

O contrato de transporte é bilateral independente da forma, do deslocamento de pessoas ou de coisas de um lugar para outro. Para o transportador torna-se uma específica obrigação de resultado, entregando o bem em local adequado e de maneira integra. Para o beneficiário fica a obrigação de retribuir o ato pagando a passagem ou frete dependendo da situação proposta (PENTEADO, 2015).

No âmbito de transportes rodoviário interestaduais e internacionais existem hoje no país 179.332 ônibus habilitados para prestação de serviços regulares em regime especial, sendo esses responsáveis pelo transporte de 119 milhões de passageiros ao ano. E 25.637 veículos habilitados para realização do transporte fretado que manipulam cerca de 11 milhões de passageiros ao ano, sendo esses dados divulgados para o ano de 2013 (BRASIL, 2015).

A logística vem sendo aprimorada para organizar atividades relacionadas ao transporte de coisas. Segundo Penteado (2015), “O transporte, nos dias que correm, vem associado a necessidades sociais prementes de organização de escoamento de mercadorias e de movimentação de pessoas, [...] da ciência denominado de logística”, o que permite realizar racionalização, otimização de transportes, adequação de horários, dentre outros.

2.4 JAVAFX

Em meados de 2006 surgia o projeto desenvolvido por Chris Oliver, que tinha a intenção de criar uma linguagem que fosse de fácil implementação e que disponibilizasse recursos avançados em interface gráfica. A ideia de Chris era entusiasta, porém a tarefa seria bastante complicada, mas mesmo assim ele deu início ao projeto denominado F3 (OLIVEIRA, 2013).

O projeto F3 acabou chamando a atenção da Sun Microsystems que gostou da proposta de Chris Oliver e acabou adquirindo sua ideia. A linguagem então passou a ser chamada de JavaFX Script, sendo uma linguagem semelhante ao JavaScript, ou seja, não sendo oficialmente Java, o que significa que não faz parte da plataforma Java (OLIVEIRA, 2013). A seguir serão apresentadas as versões do JavaFX, demostrando as mudanças referentes a cada versão e os componentes inseridos.

2.4.1 JavaFX 1.0

A Sun Microsystems anunciou o JavaFX Script em uma conferência da JavaOne em maio de 2007. Na oportunidade a empresa pretendia tornar o JavaFX Script uma linguagem com ampla utilização para ambientes Desktop e Web e futuramente levar a linguagem para dispositivos móveis (OLIVEIRA, 2013).

Em dezembro de 2008 a Sun Microsystems lançou a primeira versão do JavaFX Script, sendo uma linguagem estaticamente tipada, orientada a objetos e projetada para a Java Virtual Machine (JVM). A primeira versão oferecia suporte as aplicações construídas para serem executadas em todos os ambientes, sendo Desktop ou Web, onde a Java Runtime Environment (JRE) estivesse instalada (GIONGO; ARAÚJO; RODRIGUES, 2013).

Em fevereiro de 2009 a Sun Microsystems anunciou a versão 1.1 do JavaFX. Essa versão aprimorou o conceito da linguagem que é “escreva uma vez, execute em qualquer lugar”. Sendo assim a nova versão trouxe suporte para criação de aplicativos para dispositivos móveis, além de melhorias de idiomas, desempenho e estabilidade. Na versão JavaFx 1.1 foram incluídos também novos tipos numéricos como float, double, long, int, short e byte (ORACLE, 2009).

Em seguida, setembro de 2009, foi anunciado o lançamento da versão 1.2 do JavaFX. Nessa nova versão foram adicionadas algumas características como suporte beta para plataformas Linux e Solaris e ainda realizadas algumas mudanças no pacote de *layouts* com a adição de efeitos visuais. Porém essa versão também trouxe mudanças de sintaxes o que deixou alguns códigos da versão anterior incompatíveis (GIONGO; ARAÚJO; RODRIGUES, 2013).

Em abril de 2010 a Sun Microsystems anunciou a versão 1.3 do JavaFX, que foi lançada com atraso pois era esperada para novembro de 2009. Mesmo assim trouxe diversas melhorias, dentre elas controles aperfeiçoados, melhorias de desempenho, suporte do *Integrated Development Environment* (IDE), melhorias para aplicações de TV, dentre outras

(DOEDERLEIN, 2012). A Figura 2 a seguir demostra a evolução dos controles da interface gráfica do usuário em relação as versões citadas até aqui.

Figura 2 - Evolução dos controles do JavaFX

Controle	Introdução
Control	JavaFX 1.1
Skin	
TextBox	
Behavior	JavaFX 1.2
Button	
ButtonBase	
CheckBox	
Hyperlink	
Keystroke	
Label	
Labeled	
ListView	
OverrunStyle	
ProgressBar	
ProgressIndicator	
RadioButton	
ScrollBar	
Skin	
Slider	
TextInputControl	
ToggleButton	
ToggleGroup	

IndexedCell	JavaFX 1.3
ListCell	
PasswordBox	
ScrollBarPolicy	
ScrollView	
Separator	
Toggle	
Tooltip	
CheckMenuItem	JavaFX 1.3 (Preview) JavaFX 1.4 (Release)
CustomMenuItem	
Menu	
MenuBar	
MenuButton	
MenuItem	
MenuItemBase	
PopupMenu	
RadioMenuItem	
SplitMenuButton	
ToolBar	
TreeCell	
TreeItem	
TreeItemBase	
TreeView	

Fonte: Doederlein (2012).

2.4.2 JavaFX 2.0

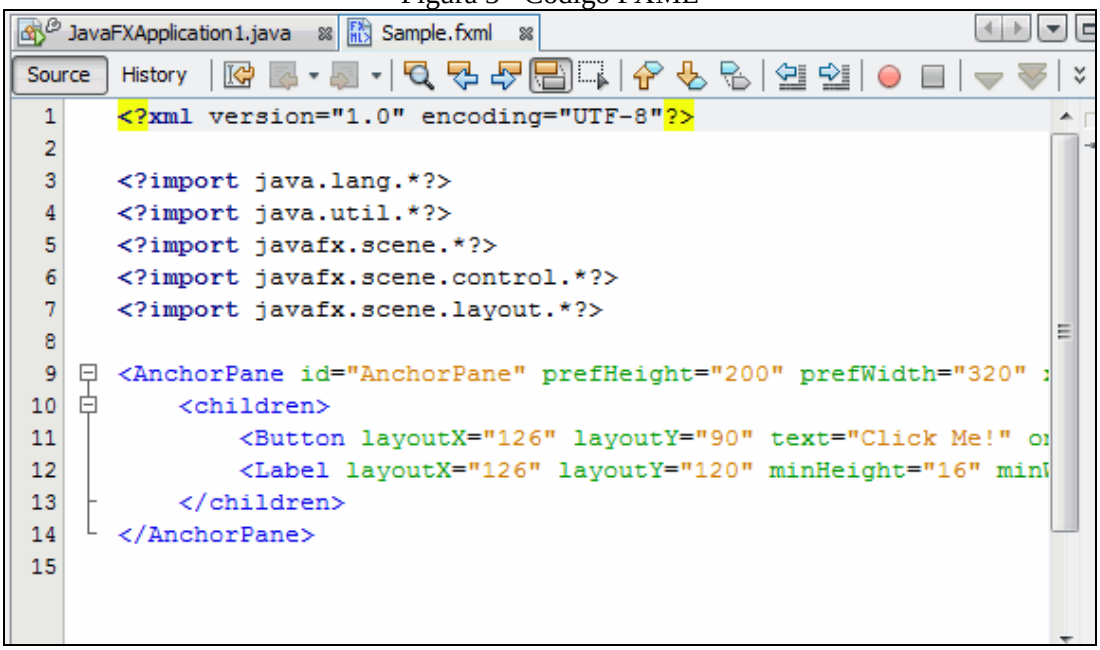
Em outubro de 2011 foi anunciada a versão 2.0 do JavaFX, a qual trouxe a possibilidade e simplicidade de utilização de recursos multimídia através de um conjunto de *Application Programming Interface* (API). Essas bibliotecas trouxeram *engines* de mídia e gráfico que utilizam aceleração de hardware. Além disso, trouxe também uma série de

coleções de controle de interface do usuário, o que facilitou o desenvolvimento da mesma (GIONGO; ARAÚJO; RODRIGUES, 2013).

A versão 2.0, além das mudanças citadas até o momento, trouxe uma grande e importante alteração que foi a substituição da linguagem *scripting* específica chamada de JavaFX Script para o código Java nativo. A mudança foi uma forma de estimular o desenvolvimento de aplicações com JavaFX, sendo que o programador Java tradicional não precisaria aprender uma nova linguagem, podendo aproveitar o seu conhecimento para desenvolvimento de aplicações ricas (QUERINO FILHO, 2013).

Ainda na versão 2.0 outra mudança importante foi a introdução de uma linguagem de marcação chamada de FXML que é baseada em outra linguagem bastante conhecida pelos desenvolvedores o *eXtensible Markup Language* (XML). Essa linguagem permite a criação de interfaces do usuário sendo separada da lógica da aplicação, permitindo realizar uma alteração no layout da aplicação sem precisar recompilar o código toda vez que uma mudança for realizada (GIONGO; ARAÚJO; RODRIGUES, 2013). Na Figura 3 é possível observar um exemplo de utilização da linguagem FXML.

Figura 3 - Código FXML



Fonte: Oracle (2013).

Em abril de 2012, a Oracle anunciou a versão 2.1 do JavaFX, a qual trouxe algumas mudanças em relação a versão 2.0. Foram incluídos controles para caixa de combinação, gráficos empilhados e barra de menu em todo o aplicativo. Além disso, foi adicionado suporte à reprodução de mídia digital no formato MPEG-4 e foi a primeira versão oficial para Mac OS X (ALMEIDA, 2012).

2.4.3 JavaFX 8

Segundo Taman (2015), o JavaFX 8 permite construir facilmente aplicações utilizando o padrão arquitetônico *model, view e controller* (MVC). Além disso, o código é de fácil compreensão e fornece um excelente suporte para recursos 3D e de mídia. Ainda de acordo com Taman (2015), o JavaFX veio à tona com um objetivo principal de ser utilizado em vários tipos de dispositivos, tais como dispositivos embarcados, *smartphones*, TVs, *tablets* e computadores. Sendo assim, o JavaFX 8 foi escrito totalmente do zero em linguagem Java e, portanto, os aplicativos podem ser implantados em *desktops*, *laptops*, *web*, etc.

Conforme Taman (2015), as APIs do JavaFX 8 estão disponíveis como recurso totalmente integrado do Java SE Runtime 8, que foi lançado em março de 2014. Além disso, o JRE e Java Development Kit (JDK) desta versão estão disponíveis para todas as principais plataformas de *desktop*, como Windows, Mac OS X, Solaris e Linux, portanto o JavaFX também será executado em todas as principais plataformas *desktop*.

De acordo com Taman (2015), os seguintes recursos estão inclusos no JavaFX 8:

a) APIs Java: Java FX é uma biblioteca Java que é composta de classes e interfaces que estão escritos em código Java;
b) FXML e Scene Builder: esta é uma marcação declarativa baseada em XML;
c) WebView: este é um componente da web que usa WebKit, para tornar possível incorporar páginas da *web* dentro da aplicação em JavaFX;
d) Swing: aplicações em SWT podem se beneficiar de recursos do JavaFX, como gráficos ricos, reprodução de mídias e incorporar conteúdo web;
e) Built-in controles de interface do usuário e *Cascading Style Sheets* (CSS): fornece os principais controles de interface do usuário, além de fornecer gráficos, paginação e personalização dos componentes através de recursos do CSS;
f) recursos gráficos 3D: suporte para bibliotecas de gráficos 3D;
g) API Canvas: é possível desenhar diretamente dentro de uma cena usando o JavaFX;
h) apoio *Multitouch*: essas operações são suportadas com base na capacidade da plataforma subjacente.

Na Figura 4 pode-se observar uma interface desenvolvida utilizando recursos do JavaFX 8.

Figura 4 – Exemplo de tela em JavaFX 8

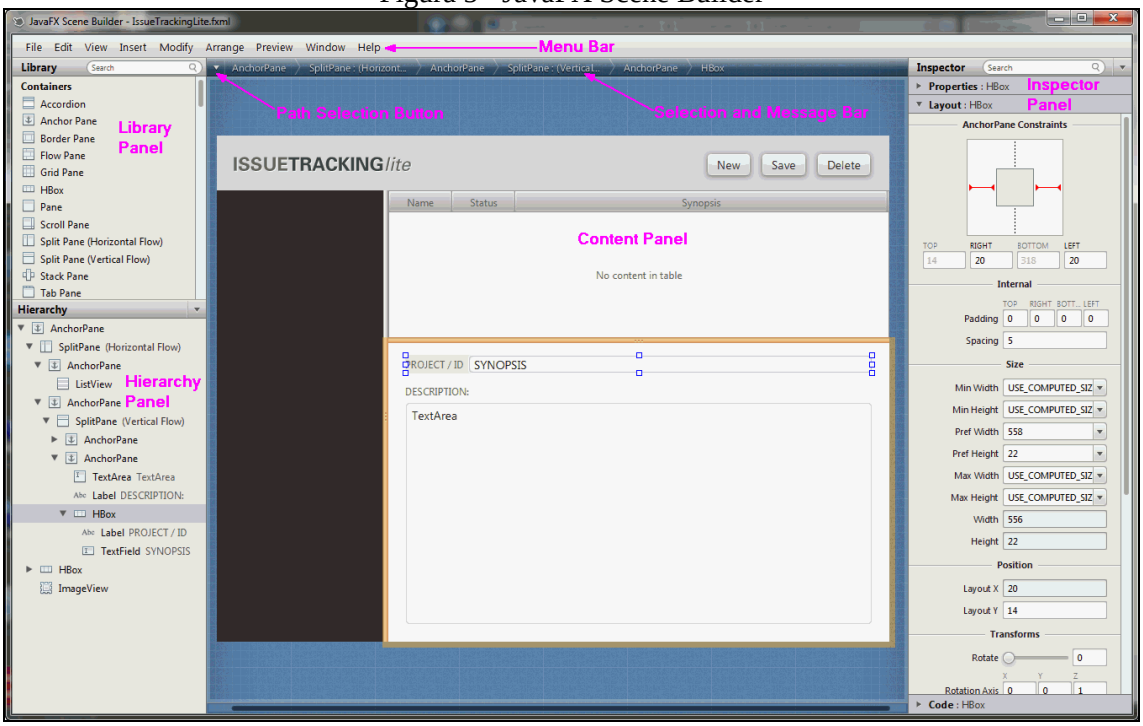
Product	Quantity	Region	Cost
Audi Q5	25	South East	1414.0
Nissan 370Z	2	East	105.32
Mitsubishi Montero	77	West	3193.96
Mitsubishi Montero	12	North East	497.76
Mitsubishi Montero	45	North East	1866.6
Volvo XC60	99	North East	4983.66
Mitsubishi Montero	30	East	1244.4

Fonte: Developer.com (2014).

2.4.4 Scene Builder

O JavaFX Scene Builder foi disponibilizado na versão 2.1 do JavaFX, a qual trouxe também algumas mudanças no desenvolvimento de aplicações com esta linguagem. O JavaFX Scene Builder é uma ferramenta de layout visual para elaboração da interface do usuário que permite a criação de telas de interface simplesmente arrastando e soltando os componentes da paleta para a tela a ser desenvolvida (ALMEIDA, 2012). Na Figura 5 é demonstrada a tela do JavaFX Scene Builder para desenvolvimento da interface.

Figura 5 - JavaFX Scene Builder



Fonte: Oracle (2012b).

O JavaFX Scene Builder é uma ferramenta de *layout* visual que permite aos usuários desenvolverem rapidamente interfaces para aplicações JavaFX. Os componentes podem ser movidos e ajustados na tela da ferramenta, assim como podem ser modificadas as propriedades desses componentes e aplicados folhas de estilos. Sendo assim, a própria ferramenta se encarrega de gerar automaticamente o código FXML para a interface desenvolvida (ORACLE, 2012a).

Na versão 8 do Java e na atualização 40 a Oracle anunciou que o Scene Builder só será liberado como código fonte dentro do projeto OpenJFX, sendo assim a Oracle não disponibilizará mais os instaladores. Porém a Glúon, empresa de software e serviços Java, disponibiliza o instalador do Scene Builder para Mac OS X, Linux e Windows que atualmente está na versão 8.0.0 (SCHULZ, 2013).

2.5 JAVA PERSISTENCE API

Com a popularização do Java em ambientes corporativos, logo se percebeu que diversos desenvolvedores demandavam muito tempo e esforço para codificar *queries Structured Query Language* (SQL) e códigos Java Database Connectivity (JDBC) para acesso ao banco de dados. Além dos problemas de produtividade, outro fator que gerava preocupação era o fato dos bancos de dados, mesmo seguindo um padrão American National Standards

Institute (ANSI), tinham diferenças significativas dependendo do fabricante (CAELUM, 2012).

Com Java o desenvolvedor trabalha com um paradigma chamado orientação a objetos que se difere muito do paradigma seguido pelos Sistemas Gerenciadores de Banco de Dados (SGBD) que adotam um esquema entidade relacional. Logo, no banco de dados as informações são representadas através de tabelas e colunas que se relacionam por meio de chaves primárias e estrangeiras, porém em Java as informações são representadas através de classes e atributos. Portanto, para que se trabalhe com as informações precisamos transformar objetos em registros e registros em objetos (CAELUM, 2012).

Para que os dois paradigmas citados anteriormente possam se comunicar é necessário um mecanismo intermediador chamado *Object Relational Mapper* (ORM). Este mecanismo provê a conversão das informações existentes em um modelo de dados relacional para o orientado a objetos e vice-versa através do mapeamento que descreve a correspondência entre esses modelos (GARCIA, 2014).

Uma solução ORM que existe para a plataforma Java é a JPA. A criação e a atualização da JPA seguem os mesmos procedimentos do Java, que é realizado através de especificações elaborados por meio de um Java Specification Request (JSR). O modo se resume em uma proposta formal com os recursos que deverão ser oferecidos nas implementações, e aprovado por um comitê executivo (GARCIA, 2014).

O Hibernate é uma solução ORM bastante conhecida no mercado, nasceu antes mesmo da JPA, porém hoje utiliza a sua especificação. Além do Hibernate podem ser citadas outras implementações da JPA, como: EclipseLink da Eclipse Foundation e o OpenJPA da Apache. Apesar do Hibernate ter originado a JPA, o EclipseLink é a implementação referencial (CAELUM, 2012).

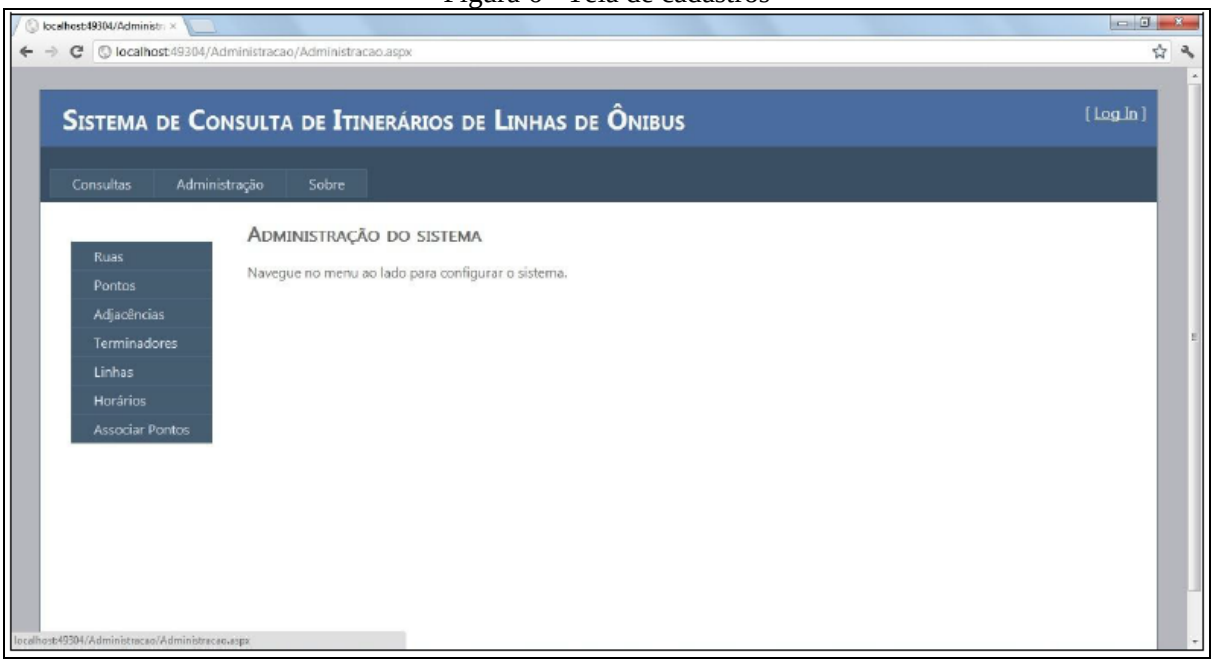
Em maio de 2013 foi liberada a versão 2.1 da JPA através da JSR 338, incorporando melhorias e novos recursos solicitados pela comunidade. Dentre as mudanças podem ser citadas, o *Schema Generation* recurso para geração de esquema de banco de dados; o *Constructor Result* alternativa para construção de resultados de consultas nativas; o *Stored Procedures*; os *Converters* que permitem a conversão dos tipos usados nas entidades para as colunas no banco (GARCIA, 2014).

2.6 TRABALHOS CORRELATOS

Pode-se citar como trabalhos correlatos as monografias realizadas na Universidade Regional de Blumenau pelas alunas Kelly Cristina Boeck e Carine Schaefer para conclusão do curso de Ciência da Computação.

O trabalho de Boeck (2012) foi desenvolver uma aplicação que possibilitasse o cadastro de linhas, horários e itinerários para auxiliar no transporte coletivo urbano. Para desenvolvimento da aplicação foram utilizados o ASP NET, linguagem C# em conjunto com o .Net Framework 4, *HyperText Markup Language* (HTML), Java Script para realização das consultas, a API do Google Maps para representação dos itinerários, e o Visual Studio 2010 para desenvolvimento da aplicação como pode ser observado na Figura 6.

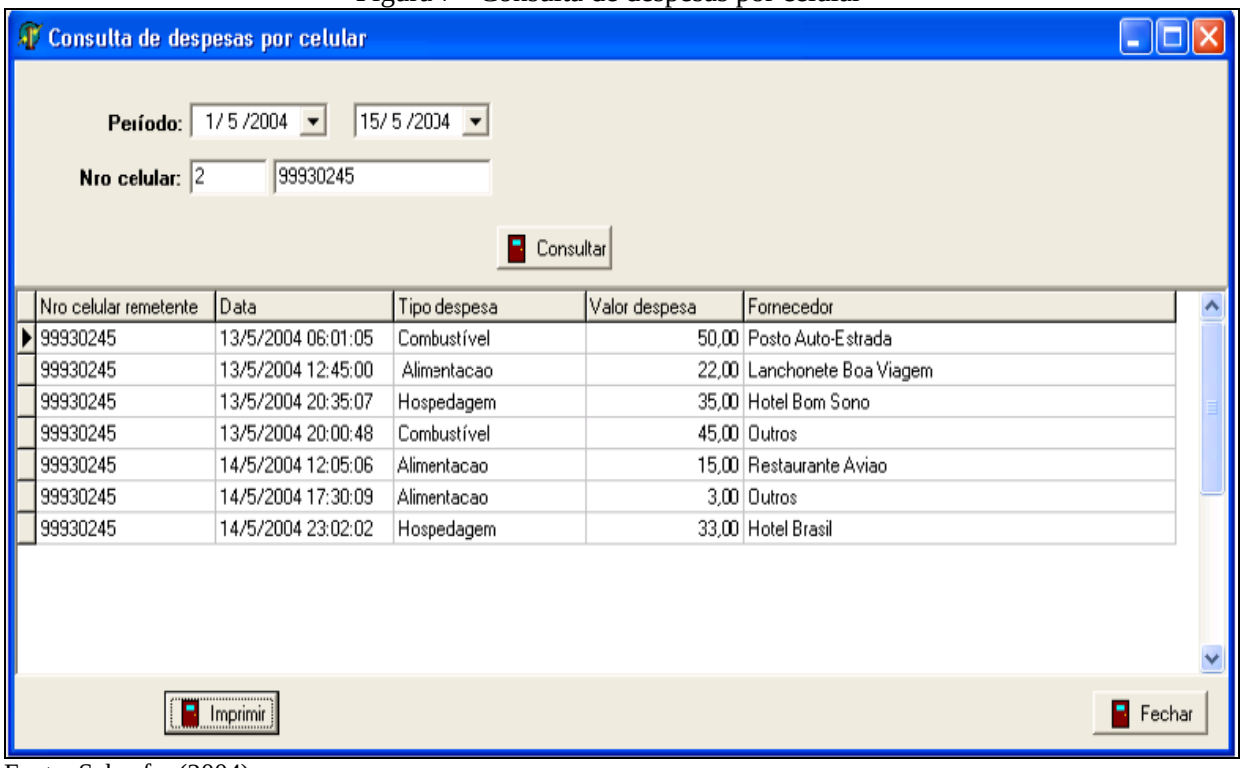
Figura 6 - Tela de cadastros



Fonte: Boeck (2012).

O trabalho de Schaefer (2004) foi desenvolver uma aplicação para uma empresa transportadora para coletar e transmitir dados de um dispositivo móvel para uma base local onde os dados são armazenados e tratados. O desenvolvimento da aplicação foi dividida em dois protótipos, o software do celular que deve cadastrar as despesas do usuário e transmitir os dados informados para um servidor de e-mail, e o software do computador que deve apresentar as despesas recebidas por e-mail e emitir relatórios e gráficos de despesas como pode ser observado na Figura 7.

Figura 7 - Consulta de despesas por celular



Fonte: Schaefer (2004).

3 DESENVOLVIMENTO

Este capítulo demostra o desenvolvimento da aplicação. São apresentados os principais requisitos, especificação, implementação da aplicação, operacionalidade e os resultados.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Foi desenvolvido um sistema para gerenciamento da distribuição do auxílio transporte para estudantes. O sistema permite ao usuário cadastrar cursos, empresas, estudantes, instituições, motoristas, trajetos e veículos, bem como realizar consultas e alterações nos cadastros dos mesmos. Além disso, existe o perfil de administrador que pode cadastrar, consultar ou alterar os usuários. O sistema possui também um painel de bordo, onde é possível o usuário acompanhar a real situação dos registros inseridos através de gráficos e gerar relatórios dos específicos cadastros.

A construção do sistema foi realizada a partir dos seguintes requisitos demostrados no Quadro 1 e Quadro 2.

Quadro 1 - Requisitos funcionais

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deverá permitir o usuário efetuar o *login* no sistema.	UC01
RF02: O sistema deverá permitir o usuário alterar senha do *login*.	UC02
RF03: O sistema deverá permitir o usuário manter estudantes.	UC03
RF04: O sistema deverá permitir o usuário manter empresas de transporte.	UC04
RF05: O sistema deverá permitir o usuário manter veículos para transporte.	UC05
RF06: O sistema deverá permitir o usuário manter motoristas.	UC06
RF07: O sistema deverá permitir o usuário manter instituições de ensino.	UC07
RF08: O sistema deverá permitir o usuário manter cursos.	UC08
RF09: O sistema deverá permitir o usuário manter trajetos para o transporte.	UC09
RF10: O sistema deverá permitir o administrador manter usuários no sistema	UC10
RF11: O sistema deverá emitir relatório de empresas de transporte.	UC11
RF12: O sistema deverá emitir relatório de veículos por empresa.	UC12
RF13: O sistema deverá emitir relatório de cursos por instituição de ensino.	UC13
RF14: O sistema deverá emitir relatório de estudantes por curso e instituição de ensino.	UC14
RF15: O sistema deverá permitir o administrador emitir relatório de usuários.	UC15
RF16: O sistema deverá permitir análise de informações gerenciais.	UC16
RF17: O sistema deverá gerar relatórios gerencias.	UC17

O Quadro 2 lista os requisitos não funcionais do sistema.

Quadro 2 - Requisitos não funcionais

Requisitos Não Funcionais
RNF01: O sistema deverá ser desenvolvido em ambiente Java *Desktop*.
RNF02: O sistema deverá utilizar banco de dados MySQL 5.6.
RNF03: O sistema deve ser compatível com os sistemas operacionais Windows 7 ou superior.

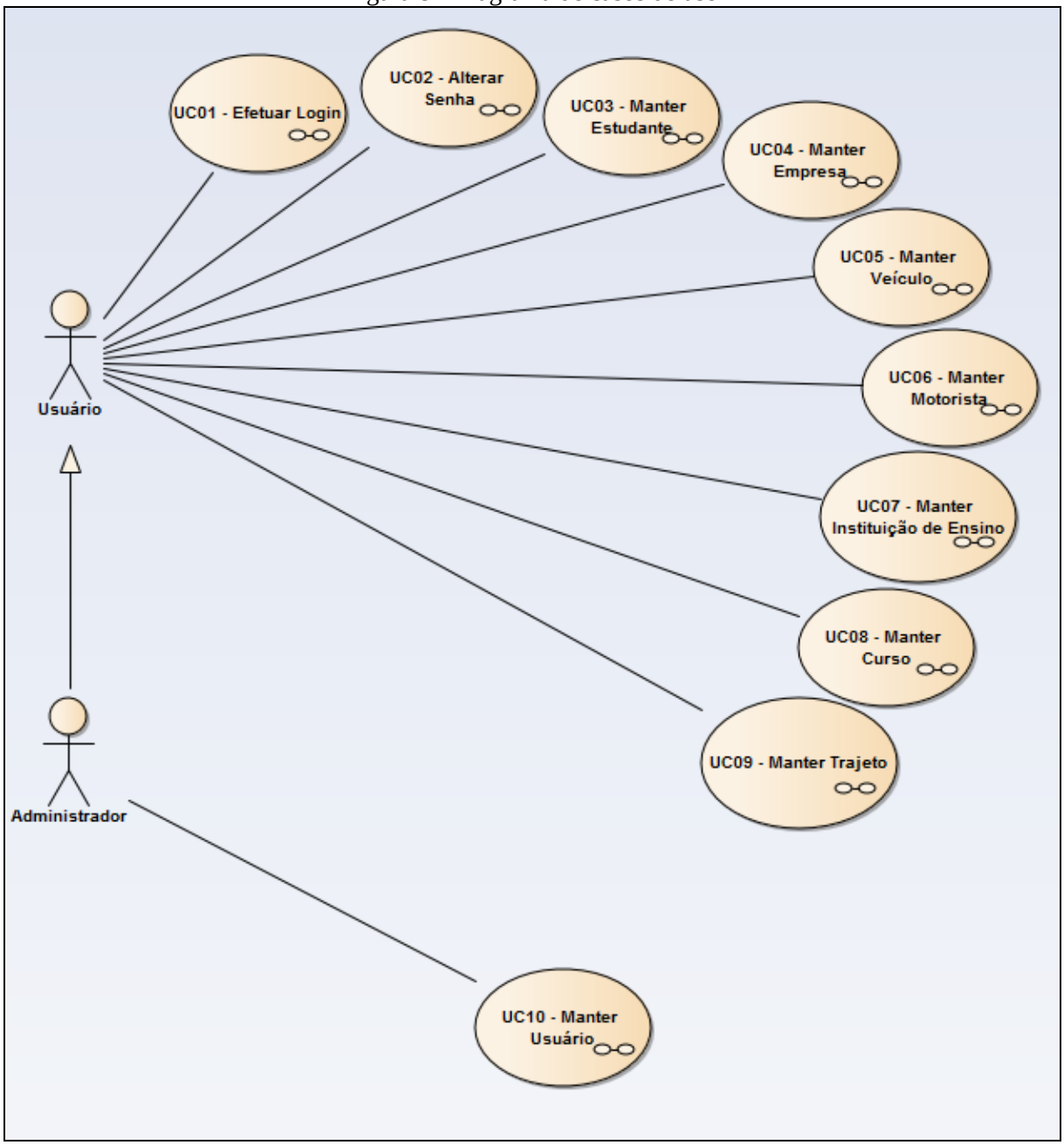
3.2 ESPECIFICAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas na implementação, assim como os diagramas de casos de uso e a estrutura de persistência dos dados.

3.2.1 Casos de Uso

Esta seção apresenta os diagramas de casos de uso do sistema proposto, sendo que o detalhamento dos casos de uso encontra-se no Apêndice A. A Figura 8 demonstra os dez casos de uso e os dois atores disponíveis, o administrador que terá acesso a todas as funcionalidades do sistema e o usuário que terá restrições referentes há alguns casos de uso.

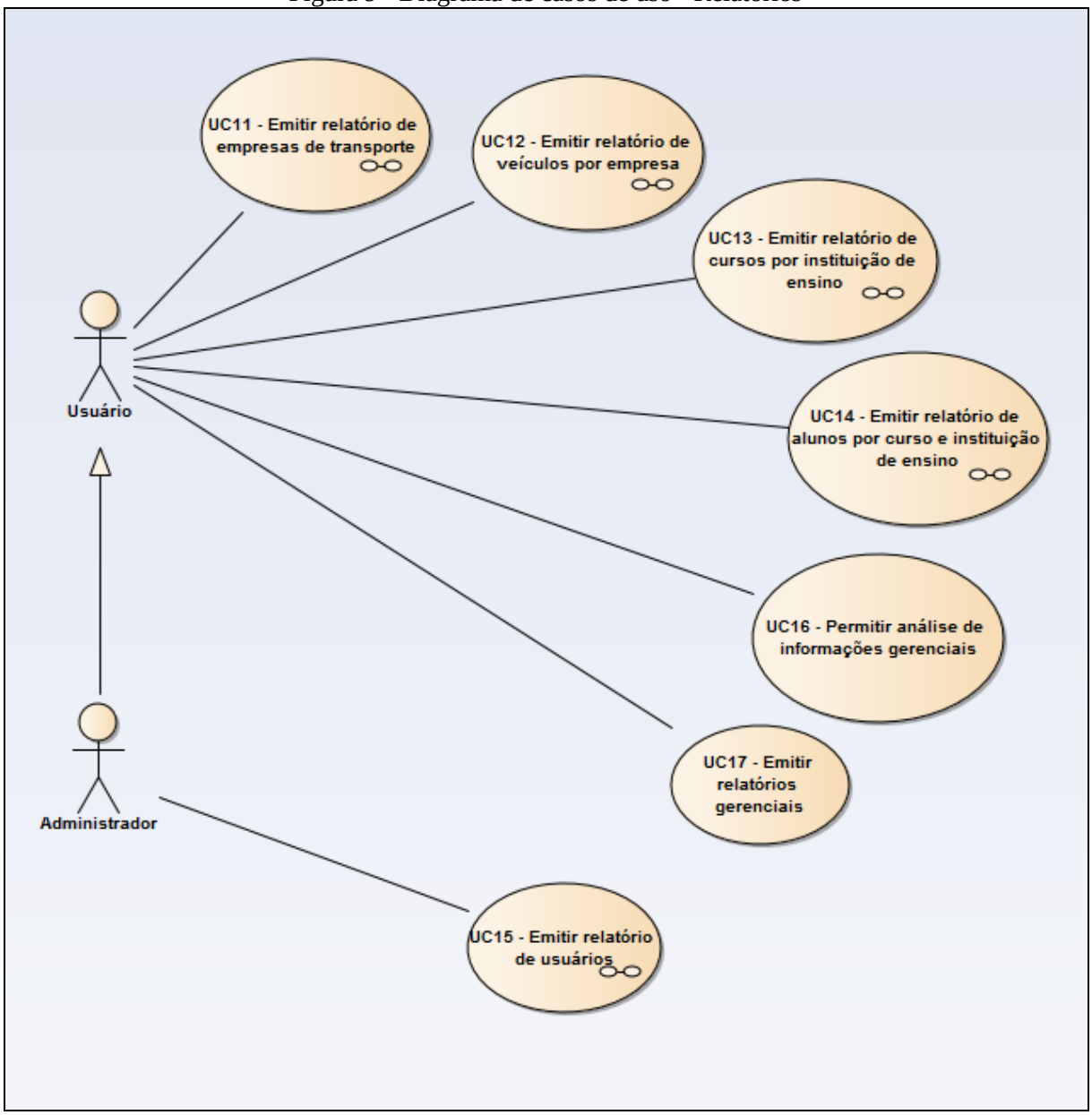
Figura 8 - Diagrama de casos de uso



Na Figura 9, tem-se casos de uso com os atores envolvidos na emissão de relatórios e análise gerenciais, onde o administrador pode emitir relatório de usuários do sistema e o usuário tem a opção de gerar um relatório de empresas de transporte, ou de veículos por

empresa, ou de cursos por instituição de ensino, ou de alunos por curso e instituição, além de emitir relatórios gerencias.

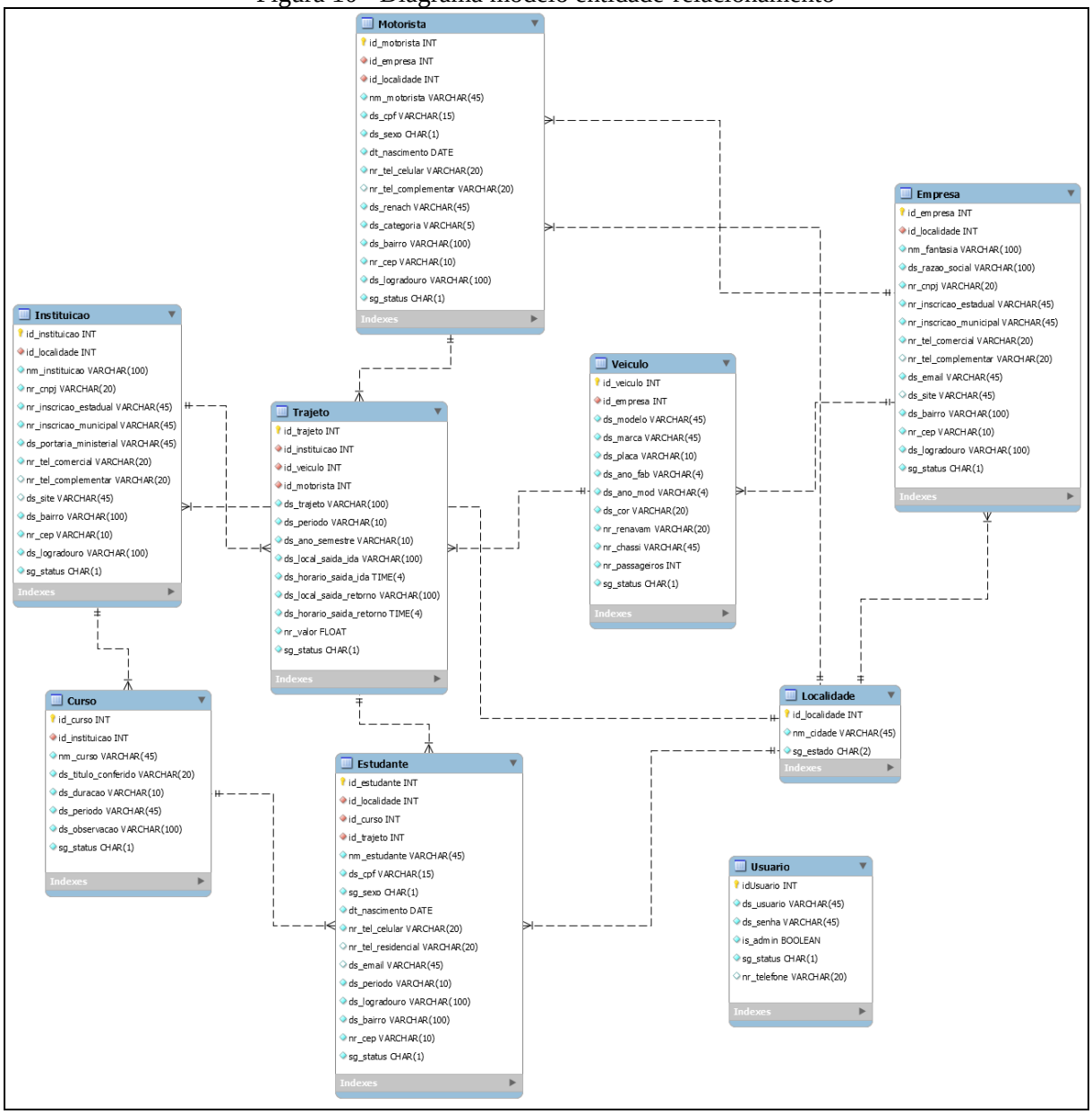
Figura 9 - Diagrama de casos de uso - Relatórios



3.2.2 Modelo Entidade Relacionamento

Na Figura 10 é representado o Modelo Entidade Relacionamento (MER), com a estrutura das tabelas do banco de dados que mantém os dados da aplicação, tais como: `estudante`, `curso`, `instituição`, `empresa`, `veículo`, `motorista`, `trajeto`, `localidade` e `usuário`. O dicionário de dados está descrito no Apêndice B.

Figura 10 - Diagrama modelo entidade-relacionamento



A seguir é apresentada uma breve descrição das entidades criadas para o desenvolvimento do sistema:

a) `Estudante`: entidade que armazena os dados dos estudantes beneficiados;

b) `Instituição`: entidade que armazena os dados das instituições que fornecem cursos para os estudantes;

c) `Curso`: entidade que armazena os dados dos cursos que os estudantes frequentam;

d) `Empresa`: entidade que armazena os dados das empresa que fornecem o transporte para os estudantes;

e) `Veiculo`: entidade que armazena os dados dos veículos que realizam o transporte;
f) `Motorista`: entidade que armazena os dados dos motoristas que conduzem os veículos;
g) `Trajeto`: entidade que armazena os dados dos trajetos para as instituições;
h) `Usuario`: entidade que armazena os usuários do sistema;
i) `Localidade`: entidade que armazena as cidades e os estados para registro dos endereços.

3.3 IMPLEMENTAÇÃO

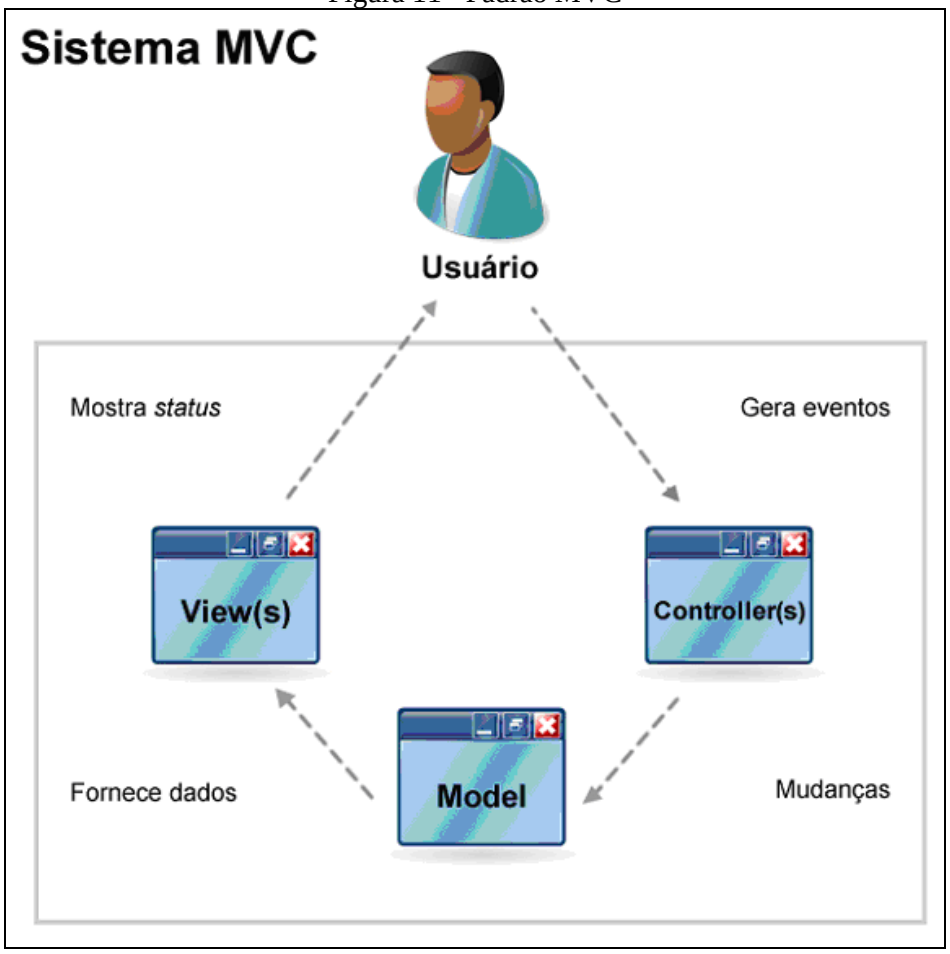
A seguir são descritas as técnicas e ferramentas utilizadas na implementação, assim como algumas rotinas implementadas para a aplicação.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento da aplicação foi utilizado a linguagem de programação Java em conjunto com a API JavaFX, com a IDE Netbeans versão 8.0.2 e o JavaFX Scene Builder versão 2.0. Para persistência dos dados foi utilizado o *framework* EclipseLink e para o armazenamento o MySQL versão 5.6.

Para organização do desenvolvimento foi utilizado a técnica MVC, que divide a aplicação em três partes menores sendo denominadas *model*, *view* e *controller*. Essa divisão permite que as informações fiquem separadas das telas em que o usuário interage. Na Figura 11 podem ser observadas as camadas citadas.

Figura 11 - Padrão MVC



Fonte: Locaweb (2008).

Para persistência dos dados foi utilizado a especificação JPA versão 2.1 sendo o EclipseLink o *framework* escolhido. Sendo assim as classes criadas dentro do pacote `model` receberam a anotação `@Entity`, dessa forma o *framework* entende que aquela classe será uma tabela no banco de dados, como pode ser observado no Quadro 3 que demonstra a anotação na classe estudante.

Quadro 3 - Anotação `@Entity`

```
@Entity
public class Estudante {
```

No Quadro 4 podem ser observados os atributos da classe Estudante, que o *framework* utiliza para criar as colunas da tabela `Estudante` no banco de dados. Através do *framework* foi possível utilizar a anotação `@Column` que permitiu alterar algumas características da coluna, como por exemplo, o termo `name` que possibilitou modificar o nome, o termo `length` que permitiu alterar o tamanho da coluna no banco de dados, além do termo `unique` no atributo `CPF` que restringe a entrada do mesmo CPF mais de uma vez.

Quadro 4 - Anotação `@Column`

```
    @Column(name = "nm_estudante", length = 45)
    private String nome;

    @Column(name = "ds_cpf", unique = true, length = 15)
    private String cpf;

    @Column(name = "sg_sexo", length = 1)
    private String sexo;

    @Temporal(value=TemporalType.DATE)
    @Column(name = "dt_nascimento")
    private Date dtNascimento;

    @Column(name = "ds_tel_celular", length = 20)
    private String telCelular;

    @Column(name = "ds_tel_residencial", length = 20)
    private String telResidencial;

    @Column(name = "ds_email", length = 45)
    private String email;

    @Column(name = "ds_periodo", length = 10)
    private String periodo;

    @Column(name = "ds_logradouro", length = 100)
    private String logradouro;

    @Column(name = "ds_bairro", length = 100)
    private String bairro;

    @Column(name = "nr_cep", length = 10)
    private String cep;

    @Column(name = "sg_status", length = 1)
    private char status;
```

Ainda no Quadro 4 pode ser observado o atributo data de nascimento que recebeu a anotação `@Temporal` que permitiu especificar para o banco de dados o formato em que a data deve ser armazenada. Nesse caso o termo `value=TemporalType.DATE` permitiu armazenar a data, mas é possível utilizar o `TemporalType.TIME` para armazenar somente um horário e o `TemporalType.TIMESTAMP` para armazenar a data e o horário.

Ainda na questão dos atributos foi necessário especificar um atributo chamado `id`, que serve como chave primária para a entidade no banco de dados. Nesse caso, o atributo `id` recebeu anotação `@Id` que determina que ele será a chave primária e também a anotação `@GeneratedValue` que especifica a geração automática dessas chaves como pode ser observado no Quadro 5.

Quadro 5 - Anotação `@Id` e `@GeneratedValue`

```
public class Estudante {

    @Id
    @GeneratedValue
    @Column(name = "id_estudante")
    private int id;
```

Para o relacionamento entre as entidades mapeadas com o JPA foi necessário utilizar a anotação `@ManyToOne`, a `@JoinColumn` e a `@OneToMany`. No Quadro 6 pode ser observado a utilização da anotação `@ManyToOne` em conjunto com a `@JoinColumn` para determinar que há uma lista de estudantes, por exemplo, para cada localidade, curso e trajeto. Nesse caso o `@JoinColumn` especifica qual foi o atributo utilizado da outra classe como chave estrangeira para a classe estudante.

Quadro 6 - Anotação `@ManyToOne` e `@JoinColumn`

```
public class Estudante {

    @ManyToOne
    @JoinColumn(name = "id_localidade")
    private Localidade localidade;

    @ManyToOne
    @JoinColumn(name = "id_curso")
    private Curso curso;

    @ManyToOne
    @JoinColumn(name = "id_trajeto")
    private Trajeto trajeto;
```

Na outra parte do relacionamento, por exemplo de estudante com o curso, foi utilizado a anotação `@OneToMany` que pode ser observada no Quadro 7. Na classe curso tem se uma coleção no atributo estudantes que recebe essa anotação. Nessa situação foi necessário utilizar também o `mappedBy` que determina qual atributo faz referência para essa coleção.

Quadro 7 - Anotação `@OneToMany`

```
public class Curso {
    @OneToMany(mappedBy = "curso")
    private List<Estudante> estudantes;
```

Para que o JPA, no caso o EclipseLink conseguisse se comunicar com o banco de dados MySQL foi necessário criar um arquivo chamado `persistence.xml`. Esse arquivo especifica quais são as classes mapeadas como entidades, qual é a *Uniform Resource Locator* (URL) do banco de dados, qual é o *driver*, qual é o usuário, a senha e o modo de criação como pode ser observado no Quadro 8.

Quadro 8 - `Persistence.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="TCC-AuxilioTransportePU" transaction-
type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>model.entity.Curso</class>
    <class>model.entity.Empresa</class>
    <class>model.entity.Estudante</class>
    <class>model.entity.Instituicao</class>
    <class>model.entity.Localidade</class>
    <class>model.entity.Motorista</class>
    <class>model.entity.Trajeto</class>
    <class>model.entity.Veiculo</class>
    <class>model.entity.Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/tcc_auxilio_transporte?
zeroDateTimeBehavior=convertToNull"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value="root"/>
      <property name="javax.persistence.schema-generation.database.action"
value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Com a utilização do EclipseLink foi necessário criar métodos para o *Create*, *Read*, *Update* e *Delete* (CRUD). Para isso, foi criada uma classe abstrata chamada `BaseDAO`, que implementa esses métodos, cria a conexão com o banco de dados e serve como modelo para os outros *Data Access Object* (DAO) como pode ser observado no Quadro 9. Sendo assim, os outros DAO estendem dessa classe abstrata, reaproveitando os métodos mencionados nela e especificando que entidade ele controla. Além disso, os outros DAO implementam métodos específicos para acesso ou controle dos dados de uma determinada classe utilizando `NamedQuery` como pode ser observado no Quadro 10.

Quadro 9 - BaseDAO

```
abstract class BaseDAO<T> {

    private Class<T> entityClass;

    private static EntityManagerFactory emf;

    private static EntityManagerFactory getFactory() {
        if (emf == null || !emf.isOpen()) {
            emf = Persistence.createEntityManagerFactory("TCC-
AuxilioTransportePU");
        }
        return emf;
    }

    public EntityManager getConnection() {
        return getFactory().createEntityManager();
    }

    public BaseDAO(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    public void insert(T entidade) {
        EntityManager em = getConnection();
        try {
            em.getTransaction().begin();
            em.persist(entidade);
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
        } finally {
            em.close();
        }
    }

    public void update(T entidade) {
        EntityManager em = getConnection();
        try {
            em.getTransaction().begin();
            em.merge(entidade);
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
        } finally {
            em.close();
        }
    }

    public List<T> findAll(T entidade) {
        EntityManager em = getConnection();
        return em.createQuery(("FROM " + entidade.getClass().getName() + "
p"))
                .getResultList();
    }
```

Quadro 10 - `EstudanteDAO`

```
public class EstudanteBaseDAO extends BaseDAO<Estudante> {

    public EstudanteBaseDAO() {
        super(Estudante.class);
    }

    public List<Estudante> getEstudantefindByAtivos() {
        EntityManager em = this.getConnection();
        return
em.createNamedQuery("Estudante.findByAtivos").getResultList();
    }

    public List<Estudante> getEstudantefindByInativos() {
        EntityManager em = this.getConnection();
        return
em.createNamedQuery("Estudante.findByInativos").getResultList();
  }
```

Para que essa `NamedQuery` demostrada no Quadro 10 pudesse ser utilizada, foi necessário criá-la na classe `Estudante`. Dessa forma, se faz um acesso específico aos dados de uma determinada entidade como pode ser observado no Quadro 11.

Quadro 11 - `NamedQuery`

```
@Entity
@NamedQueries({
    @NamedQuery(name = "Estudante.findByAtivos", query = "SELECT i FROM
Estudante i WHERE i.status = 'A' ORDER BY i.nome"),
    @NamedQuery(name = "Estudante.findByInativos", query = "SELECT i FROM
Estudante i WHERE i.status = 'I' ORDER BY i.nome")
})
public class Estudante {
```

O desenvolvimento da aplicação se utilizou da linguagem JavaFX. Essa linguagem separa a classe com os métodos e ações, do arquivo de *layout* da tela conforme descrito na seção 2.4. No Quadro 12 pode-se observar um trecho do código FXML da tela de `cadastro de estudante`, que demonstra os componentes utilizados na tela, a posição que cada componente ocupa e se utiliza alguma ação quando for acionando pelo usuário. O Quadro 13 demonstra a classe de controle do arquivo FXML citado anteriormente. Nessa classe foi necessário criar atributos para todos os componentes da tela e anotá-los com um `@FXML` para que fossem reconhecidos no código FXML. Além disso, a classe de controle implementa a classe `Initializable` que força a criação do método `initialize`(), o qual pode ser utilizado para acionar métodos antes que a tela seja carregada.

Quadro 12 – CadastroEstudante.FXML

```
<AnchorPane id="AnchorPane" prefHeight="650.0" prefWidth="950.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="view.CadastroEstudanteController">
   <children>
      <Label alignment="CENTER" contentDisplay="CENTER" layoutX="-1.0"
layoutY="34.0" prefHeight="53.0" prefWidth="950.0" text="Cadastro de
Estudante" wrapText="true">
         <font>
            <Font size="36.0" />
         </font>
      </Label>
      <VBox alignment="CENTER_RIGHT" layoutX="69.0" layoutY="118.0"
spacing="20.0">
         <children>
            <Label layoutX="27.0" layoutY="109.0" text="Nome Completo:" />
            <Label layoutX="64.0" layoutY="143.0" text="Endereço:" />
            <Label text="Cidade:" />
            <Label layoutX="63.0" layoutY="171.0" text="CPF:" />
            <Label layoutX="35.0" layoutY="200.0" text="Data de Nasc.: "
/>
            <Label layoutX="33.0" layoutY="234.0" text="Tel. Residencial:"
/>
            <Label layoutX="57.0" layoutY="261.0" text="Email:" />
         </children>
      </VBox>
<Button fx:id="btSalvar" alignment="CENTER" layoutX="530.0"
layoutY="563.0" mnemonicParsing="false" onAction="#salvar"
prefHeight="36.0" prefWidth="89.0" text="Salvar">
         <graphic>
            <ImageView fitHeight="28.0" fitWidth="28.0"
pickOnBounds="true" preserveRatio="true">
               <effect>
                  <Blend />
               </effect>
               <image>
                  <Image url="@../images/save.png" />
               </image>
            </ImageView>
         </graphic>
      </Button>
```

Quadro 13 - CadastroEstudanteController

```
public class CadastroEstudanteController implements Initializable,
ControllerCadastro {

    @FXML
    private TextField tfNomeCompleto;
    @FXML
    private TextField tfEndereco;
    @FXML
    private TextField tfBairro;

@Override
    public void initialize(URL url, ResourceBundle rb) {
        mascara();
        localidade();
        instituicao();
        periodo();
        trajeto();
    }
```

Ainda na questão do JavaFX, foram criados métodos na classe de controle para realizar determinadas ações. No Quadro 14 se pode observar que foi criado um método `salvar()`, no qual é realizada a chamada do método `validaCampos()` demonstrado no Quadro 15 para verificar se todos os campos obrigatórios foram preenchidos. Além disso, no método `salvar()` são utilizados os atributos para buscar os dados inseridos na tela e passa-los para a instância criada.

Quadro 14 - Método `Salvar`

```
@Override
    public void salvar() {
        campoPadrao();
        if (!validaCampos()) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Diálogo de Erro");
            alert.setHeaderText("Campos obrigatórios não preenchidos");
            alert.setContentText("Favor preencher todos os campos
obrigatórios!");
            alert.showAndWait();
        } else {
            try {
                EstudanteBaseDAO ebd = new EstudanteBaseDAO();
                Estudante estudante;
                if (editEstudante == null) {
                    estudante = new Estudante();
                    estudante.setStatus('A');
                } else {
                    estudante = editEstudante;
                }
                estudante.setNome(tfNomeCompleto.getText());
                estudante.setLogradouro(tfEndereco.getText());
                estudante.setBairro(tfBairro.getText());
                LocalidadeBaseDAO lbd = new LocalidadeBaseDAO();
                Localidade local =
lbd.getLocalidade(cbCidade.getSelectionModel().getSelectedItem().toString(),
cbUF.getSelectionModel().getSelectedItem().toString());
                estudante.setLocalidade(local);
                estudante.setCep(tfCEP.getText());
                estudante.setCpf(tfCPF.getText());
                if (rbMasculino.isSelected()) {
                    estudante.setSexo("M");
                } else if (rbFeminino.isSelected()) {
                    estudante.setSexo("F");
                }
                SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

estudante.setDtNascimento(sdf.parse(tfDtNascimento.getText()));
                estudante.setTelResidencial(tfTelResidencial.getText());
                estudante.setTelCelular(tfTelCelular.getText());
                estudante.setEmail(tfEmail.getText());
                Curso curso = (Curso)
cbCurso.getSelectionModel().getSelectedItem();
                estudante.setCurso(curso);
                Trajeto trajeto = (Trajeto)
cbTrajeto.getSelectionModel().getSelectedItem();
                estudante.setTrajeto(trajeto);
                String periodo = (String)
cbPeriodo.getSelectionModel().getSelectedItem();
                estudante.setPeriodo(periodo);
                if (editEstudante != null) {
                    ebd.update(estudante);
                    editEstudante = null;
                    Alert alert = new Alert(Alert.AlertType.INFORMATION);
                    alert.setTitle("Diálogo de Informação");
                    alert.setHeaderText(null);
                    alert.setContentText("Estudante alterado com sucesso!");
                    alert.showAndWait();
                } else {
                    ebd.insert(estudante);
```

```
                    Alert alert = new Alert(Alert.AlertType.INFORMATION);
                    alert.setTitle("Diálogo de Informação");
                    alert.setHeaderText(null);
                    alert.setContentText("Estudante inserido com sucesso!");
                    alert.showAndWait();
                }
                limparCampos();
                campoPadrao();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

Quadro 15 - Método `validaCampos`

```
@Override
    public boolean validaCampos() {
        boolean valida = true;
        if (tfNomeCompleto.getText().equals("")) {
            tfNomeCompleto.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfEndereco.getText().equals("")) {
            tfEndereco.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfBairro.getText().equals("")) {
            tfBairro.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfCEP.getText().equals("")) {
            tfCEP.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfCPF.getText().equals("")) {
            tfCPF.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfDtNascimento.getText().equals("")) {
            tfDtNascimento.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfTelResidencial.getText().equals("")) {
            tfTelResidencial.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfTelCelular.getText().equals("")) {
            tfTelCelular.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (tfEmail.getText().equals("")) {
            tfEmail.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (cbCurso.getSelectionModel().getSelectedItem() == null) {
            cbCurso.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (cbPeriodo.getSelectionModel().getSelectedItem() == null) {
            cbPeriodo.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (cbInstituicao.getSelectionModel().getSelectedItem() == null) {
            cbInstituicao.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (cbTrajeto.getSelectionModel().getSelectedItem() == null) {
            cbTrajeto.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (cbCidade.getSelectionModel().getSelectedItem() == null) {
            cbCidade.setStyle("-fx-border-color:red");
            valida = false;
        }
        if (cbUF.getSelectionModel().getSelectedItem() == null) {
```

```
            cbUF.setStyle("-fx-border-color:red");
            valida = false;
        }
        return valida;
    }
```

Na parte do sistema onde são realizadas as consultas, foi implementado o método `preencheTable()` que preenche a `TableView` com uma lista determinada como pode ser observado no Quadro 16. Além disso, é utilizado o `TableColumn` e assim passado o valor para a determinada célula através do `setCellValueFactory()`.

Quadro 16 - Método `preencheTable`

```
@Override
    public void preencheTable() {
        tvEstudantes.setItems(estudantes);
        tcEstudanteNome.setCellValueFactory(new
Callback<TableColumn.CellDataFeatures<Estudante, String>,
ObservableValue<String>>() {

            @Override
            public ObservableValue<String>
call(TableColumn.CellDataFeatures<Estudante, String> param) {
                return new
ReadOnlyObjectWrapper(param.getValue().getNome());
            }
        });
        tcEstudanteCPF.setCellValueFactory(new
Callback<TableColumn.CellDataFeatures<Estudante, String>,
ObservableValue<String>>() {

            @Override
            public ObservableValue<String>
call(TableColumn.CellDataFeatures<Estudante, String> param) {
                return new
ReadOnlyObjectWrapper(param.getValue().getCpf());
            }
        });
    }
```

Ainda na parte da consulta foi implementado o método `filtro()` através da utilização do `FilteredList` e do `SortedList`, como pode ser observado no Quadro 17, que permitem realizar uma comparação e atualizar a `TableView` a cada carácter digitado na `TextField`.

Quadro 17 - Método `filtro`

```
@Override
    public void filtro() {
        //Filtro...
        FilteredList<Estudante> filteredData = new
FilteredList<>(estudantes, c -> true);
        tfFilter.textProperty().addListener((observable, oldValue,
newValue) -> {
            filteredData.setPredicate(estudante -> {
                if (newValue == null || newValue.isEmpty()) {
                    return true;
                }
                String lowerCaseFilter = newValue.toLowerCase();
                if
(estudante.getNome().toLowerCase().contains(lowerCaseFilter)) {
                    return true;
            });
        });
        SortedList<Estudante> sortedData = new SortedList<>(filteredData);

sortedData.comparatorProperty().bind(tvEstudantes.comparatorProperty());
        tvEstudantes.setItems(sortedData);
    }
```

Para que na parte da consulta fosse possível listar os registros ativos e inativos foram implementados dois métodos, o `listInativos()` e o `listAtivos()`, como pode ser observado no Quadro 18. Cada qual com suas determinadas buscas e desabilitando alguns componentes da tela conforme a lista.

Quadro 18 - Método `listAtivos` e `listInativos`

```
@Override
    public void listInativos() {
        if (rbInativos.isSelected()) {
            rbAtivos.setSelected(false);
        }
        estudantes =
FXCollections.observableArrayList(ebd.getEstudantefindByInativos());
        preencheTable();
        showDetalhes(null);
        filtro();
        btAtivar.setDisable(false);
        btDesativar.setDisable(true);
    }

    @Override
    public void listAtivos() {
        if (rbAtivos.isSelected()) {
            rbInativos.setSelected(false);
        }
        estudantes =
FXCollections.observableArrayList(ebd.getEstudantefindByAtivos());
        preencheTable();
        showDetalhes(null);
        filtro();
        btDesativar.setDisable(false);
        btAtivar.setDisable(true);
    }
```

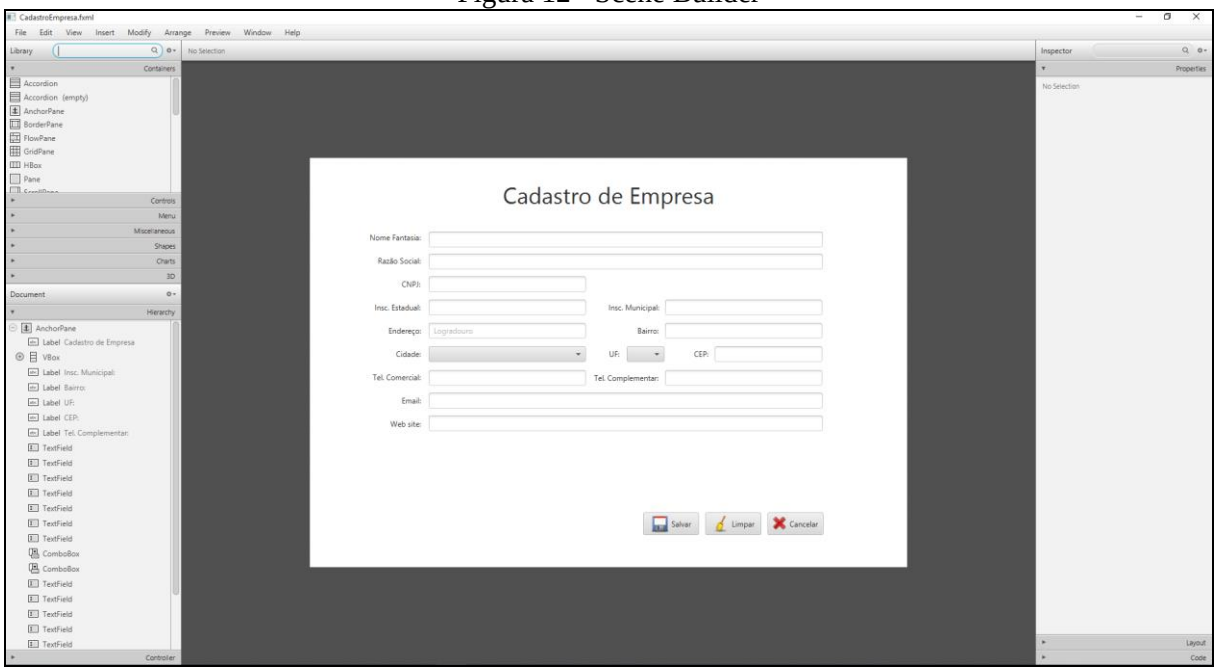
Outra parte do sistema desenvolvido foi o painel de bordo, no qual o usuário consegue ter uma visão do estado atual dos registros inseridos através de gráficos apresentados na tela. Sendo assim foram utilizados `PieChart` que são gráficos de pizza e que possibilitaram essa visualizam. Como pode ser observado no Quadro 19 que demostra o método `PieChartQtdGenero()`, o qual gera um `PieChart` com a quantidade de estudantes por gênero.

Quadro 19 - Método `PieChartQtdGenero`

```
public void PieChartQtdGenero() {
        ObservableList<PieChart.Data> pieChartData =
FXCollections.observableArrayList(
                new PieChart.Data("Masculino",
Double.parseDouble(cbd.getQtdMasculino().toString())),
                new PieChart.Data("Feminino",
Double.parseDouble(cbd.getQtdFeminino().toString()))
        );
        graficoPizza3.setData(pieChartData);
        graficoPizza3.setTitle("Estudantes por Gênero");
    }
```

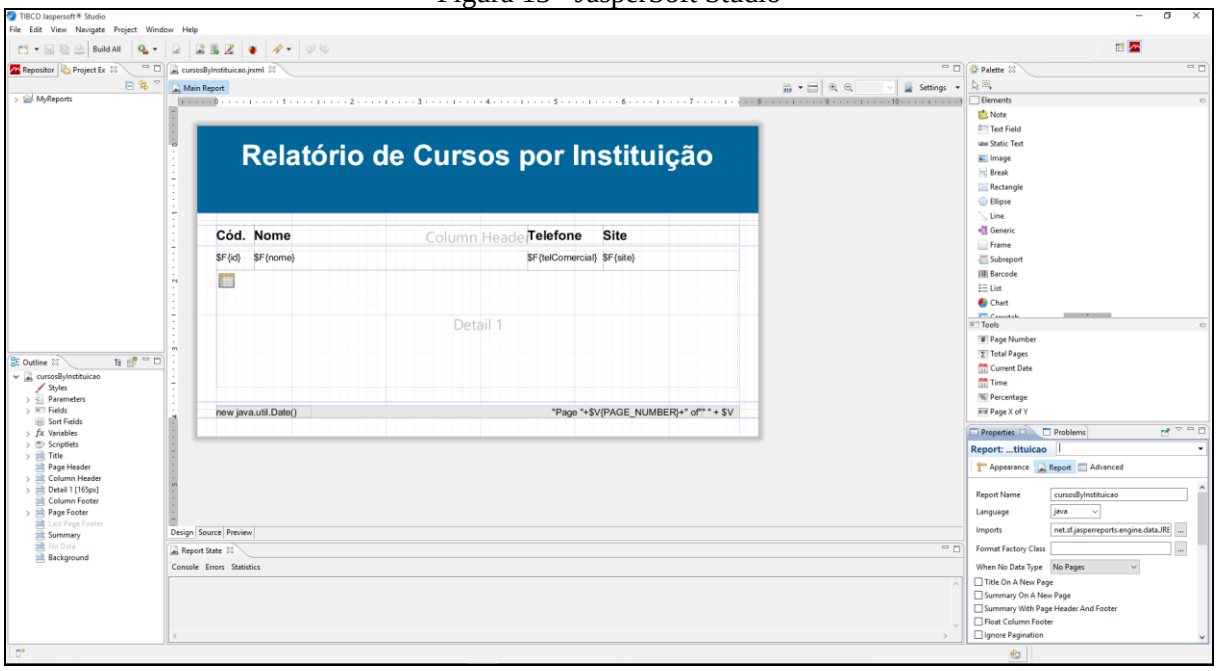
Para o desenvolvimento das telas em JavaFX foi utilizado o Scene Builder que é uma ferramenta que permite a criação de interfaces com rapidez e facilidade, conforme descrito na seção 2.4.4. Conforme a Figura 12, o Scene Builder facilitou o desenvolvimento das telas do protótipo e permitiu prontamente alinhar os componentes da mesma.

Figura 12 - Scene Builder



A Figura 13 demonstra o JasperSoft Studio que foi utilizado para o desenvolvimento dos relatórios.

Figura 13 - JasperSoft Studio



3.3.2 Operacionalidade da implementação

As operacionalidades do sistema são demonstradas através de imagens, sendo apresentadas as principais telas de cadastro, consulta, do painel de bordo e os relatórios mais relevantes.

Conforme Figura 14, o usuário precisa estar cadastrado no sistema para que possa operá-lo. Sendo assim é apresentada a tela de *login* no início da utilização do software, na qual o usuário precisa informar o seu usuário e a sua senha para que possa desfrutar das funcionalidades do sistema.

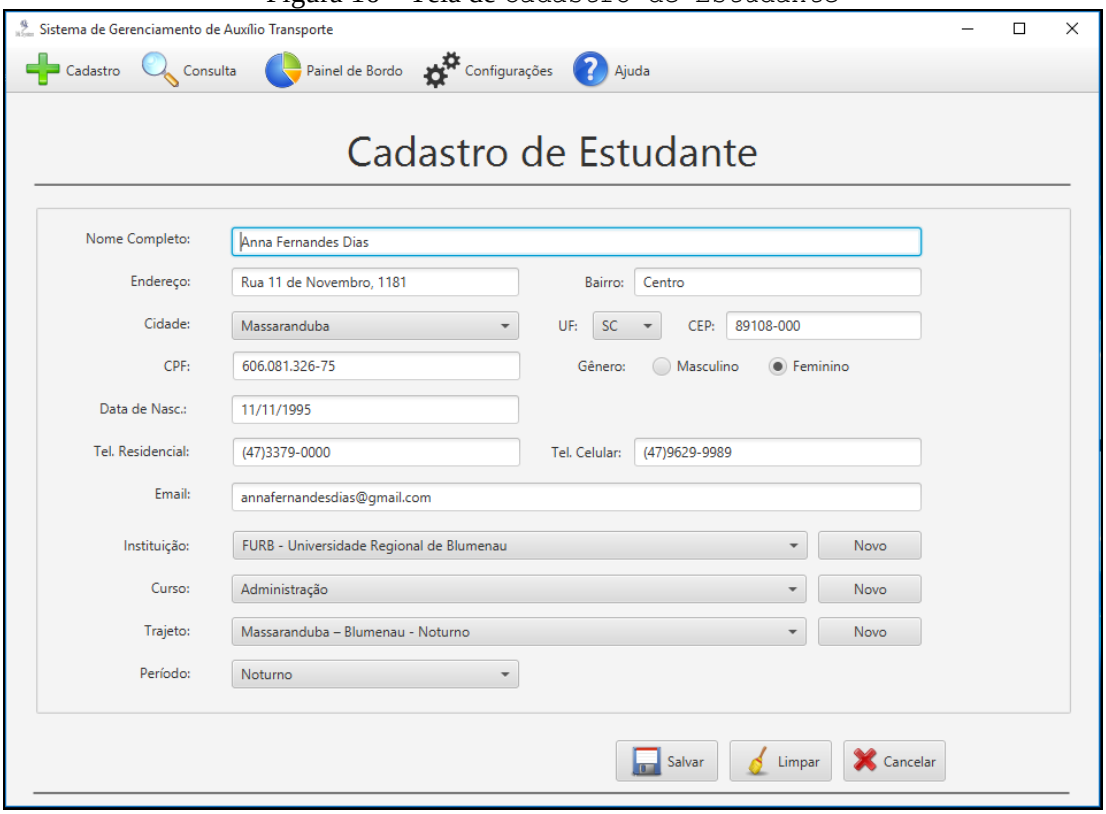
Figura 14 - Tela de *Login*

Conforme Figura 15, são apresentadas as funcionalidades de `cadastro`, `consulta`, `painel de bordo`, `configurações` e `ajuda` da tela principal do sistema.

Figura 15 - Tela principal do sistema

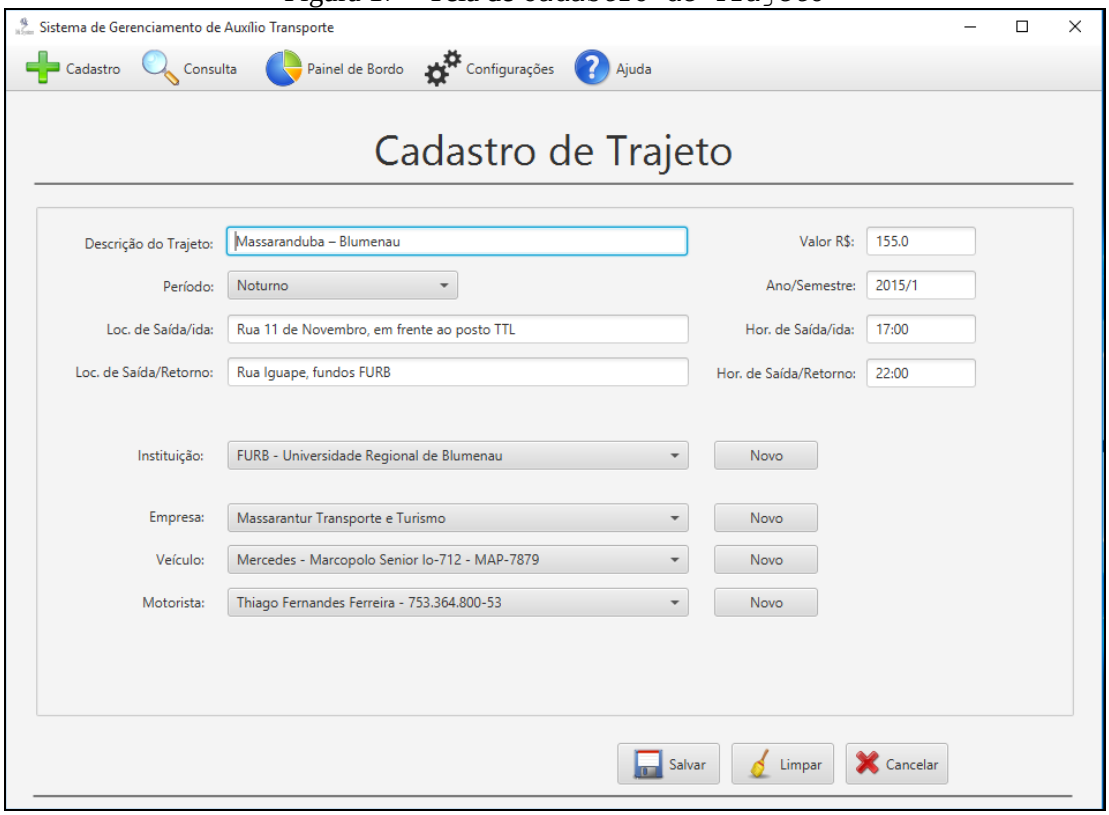
A Figura 16 mostra a funcionalidade de cadastro de estudante.

Figura 16 – Tela de `Cadastro de Estudante`



A Figura 17 mostra a funcionalidade do cadastro de trajeto.

Figura 17 – Tela de `Cadastro de Trajeto`



A tela de consulta de motorista, conforme Figura 18, permite que o usuário selecione algum motorista através da lista apresentada, sendo que as informações são exibidas ao lado. No topo da tela é apresentado um campo buscar, no qual o usuário pode refinar a sua pesquisa. Além disso o usuário pode optar por listar os registros ativos ou inativos, sendo que ele possui privilégio para marcar um registro como ativo ou inativo através dos botões apresentados. Nesse caso, quando um registro for marcado como inativo, ele sofrerá uma exclusão lógica. Ainda na tela de consulta o usuário pode escolher alterar as informações de algum registro através do botão apresentado, sendo que nesse caso o registro é enviado para a tela de cadastro, na qual o usuário pode alterar as informações como pode ser observado na Figura 19.

Figura 18 - Tela de `Consulta de Motorista`

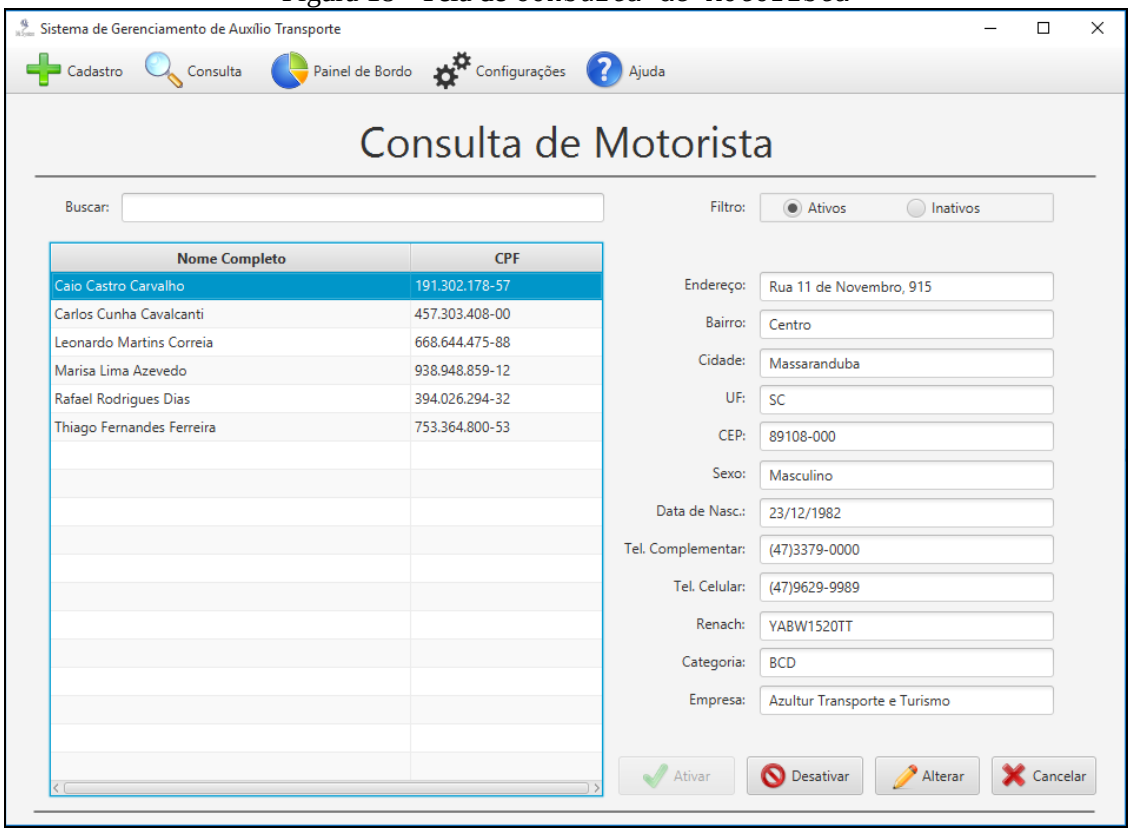
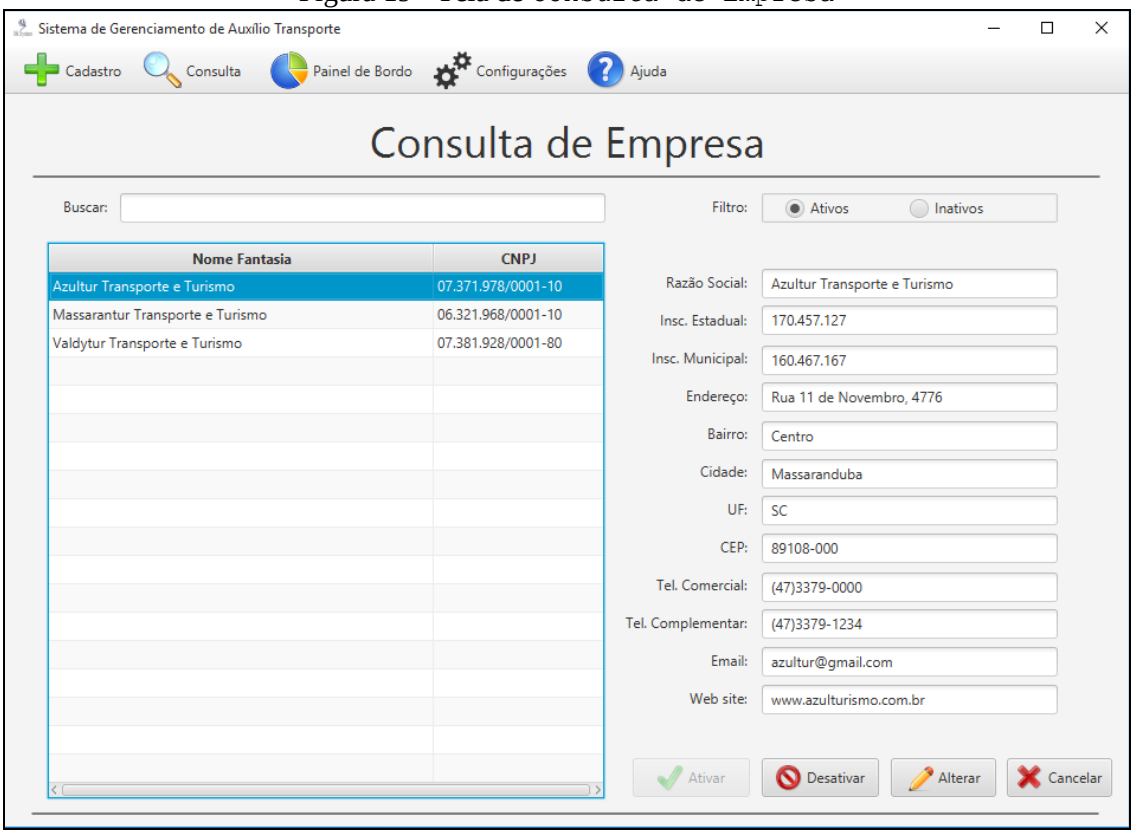


Figura 19 - Tela de `Consulta de Empresa`



Conforme Figura 20, o usuário pode realizar uma análise determinada dos registros dos cursos através dos gráficos apresentados na tela. Na qual se pode visualizar a quantidade de

estudantes por curso, a quantidade de cursos por título conferido ou até mesmo a quantidade de ativos e inativos. Além disso é possível gerar relatórios específicos através dos botões apresentados na tela, sendo que esses botões carregam uma janela do JasperReports que permite visualizar o relatório e salvá-lo em diversos formatos como pode ser observado na Figura 21.

Figura 20 - Tela de `Análise de Cursos`

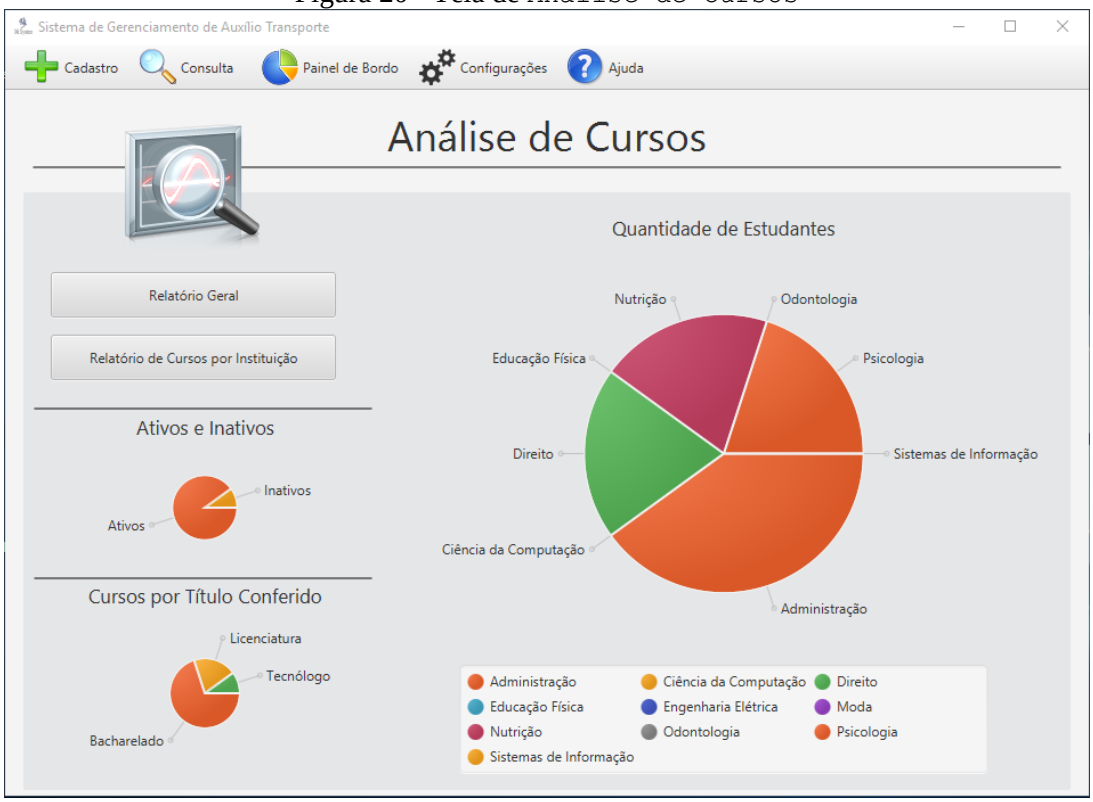
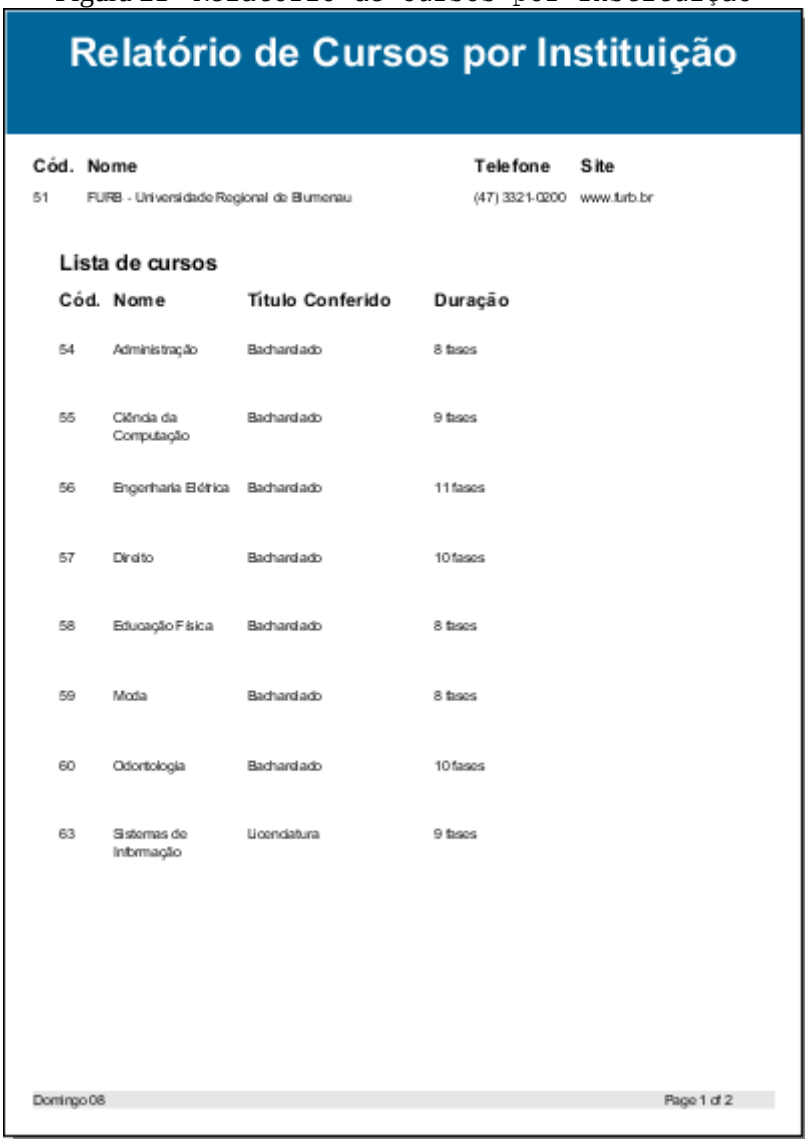


Figura 21 - `Relatório de Cursos por Instituição`



3.4 RESULTADOS E DISCUSSÕES

O presente trabalho apresentou o desenvolvimento de um sistema para administrar e gerenciar a distribuição do auxílio transporte para estudantes. Conforme descrito na seção 3.3, a linguagem utilizada foi o JavaFX em conjunto com alguns *frameworks* e algumas bibliotecas. A linguagem se demostrou bastante robusta e com grande potencial para desenvolvimento de interfaces agradáveis e dinâmicas conforme descrito na seção 2.4. No desenvolvimento do sistema se observou alguns detalhes que o JavaFX precisa amadurecer, como é o caso das máscaras para os campos de texto que a linguagem ainda não traz. Outro caso foi a distribuição das classes no pacote MVC, na qual a classe `controller` responsável por administrar o arquivo FXML conforme descrito na seção 3.3.1 precisou permanecer na mesma pasta do arquivo FXML caso contrário não seria reconhecido pelo mesmo.

No Quadro 20 são apresentadas algumas características, estabelecendo um comparativo entre o presente trabalho e os trabalhos correlatos apresentados na seção 2.6. O foco do trabalho realizado por Schaefer (2004) foi desenvolver um aplicativo para coleta e transmissão de dados de um dispositivo móvel (celular) para uma base local onde os dados serão armazenados e tratados, aplicado a uma empresa transportadora. No caso do trabalho desenvolvido por Boeck (2012) o foco foi a especificação e implementação de um aplicativo *Web* para a consulta de itinerários e horários de transporte público urbano.

Quadro 20 - Comparativo entre os trabalhos

Características	**Trabalhos desenvolvidos**		
	Software deste trabalho	Software de Boeck (2012)	Software de Schaefer (2004)
Linguagem de desenvolvimento	JavaFX	C#	Java
SGBD utilizado	MySQL	SQL Server Express 2008 R2 Edition Free	_
Persistência de dados	EclipseLink	Entity Framework	_
Ferramentas utilizadas	Netbeans 8.0.2 Scene Buider 2.0 JasperSoft Studio 6.1	ASP NET Visual Studio 2010	Sun ONE Studio 5 ME J2ME Wireless Toolkit v 2.1 Default Gray Phone
Ambiente (*web/desktop/mobile*)	*Desktop*	*Web*	*Desktop/Mobile*
Atuação	Gerenciamento da Distribuição do Auxílio Transporte para Estudantes	Consulta de itinerários de transporte público com visualização no Google Maps	Transmissão de dados a partir de dispositivos móveis aplicado a uma empresa de transportes
Análise de informações gerenciais	SIM	NÃO	SIM
Relatórios gerenciais	SIM	NÃO	NÃO

Os trabalhos correlatos possuem em comum o foco na questão dos transportes, sendo que cada qual é direcionado a uma área específica dos transportes, nesse caso de pessoas ou de materiais. O trabalho de Schaefer (2004) foi desenvolvido em Java, utilizando ferramentas como o Sun ONE Studio e J2ME Wireless Toolkit, além de ser desenvolvido para ambiente *Desktop* e *Mobile*. O trabalho de Boeck (2012) foi desenvolvido em C#, utilizando ferramentas como o ASP NET e o Visual Studio, além de ser desenvolvido para ambiente *Web*.

Para o desenvolvimento deste trabalho foram utilizadas ferramentas como o Scene Builder e o Netbeans, os quais possibilitaram a implementação do sistema para ambiente *Desktop*. Além disso, o sistema desenvolvido possibilita análise de informações gerenciais, como no trabalho de Schaefer (2004). Através da ferramenta JasperSoft Studio foi possível a

construção de relatórios gerenciais que o sistema pode emitir, sendo que essa funcionalidade se difere dos trabalhos correlatos apresentados, pois os mesmos não os possuem.

4 CONCLUSÕES

Neste trabalho é apresentada uma solução para apoiar na gestão da distribuição do auxílio transporte oferecido por diversos municípios através da gestão pública. A solução da distribuição do auxílio transporte permite que alguns processos que eventualmente são realizados a mão possam ser executados automaticamente, como por exemplo, realizar um levantamento dos estudantes que realizam determinado curso ou levantamento dos veículos utilizados por determinada empresa.

Os objetivos propostos neste trabalho foram alcançados. Foi desenvolvido um sistema de gerenciamento da distribuição do auxílio transporte, no qual é possível cadastrar os cursos, as instituições, os estudantes, as empresas, os veículos, os motoristas e os trajetos disponíveis. Além disso, é possível realizar consultas dos registros inseridos, podendo ser realizadas alterações nos dados destes registros. O sistema também disponibiliza um painel de bordo, no qual é possível realizar verificações da quantidade de registros ativos e inativos e algumas verificações específicas, como por exemplo, verificar a quantidade de estudantes por curso ou por trajeto. Contudo o sistema ainda disponibiliza a geração de relatórios, no qual o usuário, através de do painel de bordo, pode emitir um relatório gerencial ou um relatório específico, como por exemplo, um relatório de estudantes por curso e instituição de ensino.

As tecnologias utilizadas mostraram-se apropriadas para desenvolvimento da solução. Mesmo o JavaFX, que surgiu há menos de 10 anos e passou por grandes mudanças nesse período conforme descrito na seção 2.4. O JavaFX, através da ferramenta Scene Builder, possibilitou o desenvolvimento ágil e alinhado das telas do sistema, tornando-se uma opção válida para trabalhos futuros. Sendo assim, o JavaFX possivelmente substituirá a antiga API Swing utilizada por muito tempo para desenvolvimento de aplicações *Desktop*. Porém, pode-se destacar que o JavaFX necessita de algumas melhorias conforme descrito na seção 3.4, que poderão ser acrescentadas em futuras atualizações da API. Em relação à compatibilidade das tecnologias utilizadas, nenhuma delas demonstrou-se incompatível em relação à outra, tanto o JavaFX quanto o EclipseLink relacionaram-se perfeitamente para a solução desenvolvida.

Diante dos resultados obtidos no desenvolvimento deste trabalho é possível afirmar que o sistema desenvolvido atende o atual cenário no qual está inserido. A implementação propiciou ao autor a oportunidade de adquirir novas habilidades e aprimorar os conhecimentos obtidos durante o curso de graduação. Nesse caso, o aprimoramento do levantamento de requisitos e análise do sistema a ser desenvolvido, a especificação detalhada do projeto, a elaboração e a construção da base de dados, e a utilização de alguns *frameworks*.

Além disso, o trabalho proporcionou a obtenção de novos conhecimento e habilidades, como é o caso do JavaFX e o Jasper Reports que ainda não haviam sidos utilizados pelo autor.

4.1 EXTENSÕES

Como sugestões de extensões para melhorar o sistema desenvolvido, tem-se:

a) criar a funcionalidade de emitir carteirinhas para os estudantes;
b) criar a funcionalidade de emitir o vale transporte para ser apresentado à empresa que realiza o serviço;
c) disponibilizar um módulo web para consultas de trajetos e horários;
d) disponibilizar um módulo web para transparência pública;
e) disponibilizar um módulo web para que o estudante possa emitir o vale transporte.

REFERÊNCIAS

ALMEIDA, Rodrigo. **Oracle lança Java SE 7 Update 4 e JavaFX 2.1**. 2012. Disponível em: <http://www.rodrigoalmeida.net/oracle-lanca-java-se-7-update-4-javafx-2-1/>. Acesso em: 19 set. 2015.

BOECK, Kelly Cristina. **Aplicação web para consulta de itinerários de transporte público com visualização no Google Maps**. 2012. 81 f. TCC (Graduação) - Curso de Curso de Ciências da Computação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2012.

BRASIL. ANTT. **Agência Nacional de Transportes Terrestres**. 2015. Disponível em: <http://www.antt.gov.br/index.php/content/view/4890/Apresentacao.html>. Acesso em: 10 mar. 2015.

BRASIL. ASSESSORIA DE COMUNICAÇÃO SOCIAL DO INEP. **Brasil teve mais de 7 milhões de matrículas no ano passado**. 2013a. Disponível em: <http://portal.inep.gov.br/visualizar/-/asset_publisher/6AhJ/content/brasil-teve-mais-de-7-milhoes-de-matriculas-no-ano-passado>. Acesso em: 04 abr. 2015.

BRASIL. INSTITUTO NACIONAL DE ESTUDOS E PESQUISAS EDUCACIONAIS ANÍSIO TEIXEIRA. **Censo da Educação Superior**. 2013b. Disponível em: <http://download.inep.gov.br/educacao_superior/censo_superior/apresentacao/2014/coletiva_censo_superior_2013.pdf>. Acesso em: 16 maio 2015.

CAELUM. **Uma introdução prática ao JPA com Hibernate**. 2012. Disponível em: <http://www.caelum.com.br/apostila-java-web/uma-introducao-pratica-ao-jpa-com-hibernate/#14-1-mapeamento-objeto-relacional>. Acesso em: 20 set. 2015.

DEVELOPER.COM. **Combining JavaFX and Java Enterprise Modules**. 2014. Disponível em: <http://www.developer.com/java/ent/combining-javafx-and-java-enterprise-modules.html>. Acesso em: 14 nov. 2015.

DOEDERLEIN, Osvaldo Pinali. **JavaFX 1.3 - Java Magazine 81**. 2012. Disponível em: <http://www.devmedia.com.br/javafx-1-3-java-magazine-81/17435>. Acesso em: 19 set. 2015.

GARCIA, Francisco A. **Java JPA**: A evolução da persistência em Java. 2014. Disponível em: <http://www.devmedia.com.br/java-jpa-a-evolucao-da-persistencia-em-java/29694>. Acesso em: 20 set. 2015.

GESTOR EFICAZ. **Planejamento Estratégico**. 2012. Disponível em: <http://gestoreficaz.blogspot.com.br/2012/12/planejamento-estrategico.html>. Acesso em: 14 mar. 2015

GIONGO, Scheila; ARAÚJO, Everton Coimbra de; RODRIGUES, Daniel. **Introdução ao JavaFX 2.0 - Revista Easy Java Magazine 23**. 2013. Disponível em: <http://www.devmedia.com.br/introducao-ao-javafx-2-0-revista-easy-java-magazine-23/26035>. Acesso em: 19 set. 2015.

ITATIBA. PREFEITURA DO MUNICIPIO. **Auxílio Transporte**. 2015. Disponível em: <http://www.itatiba.sp.gov.br/Educacao/auxilio-transporte.html>. Acesso em: 04 abr. 2015.

ITUPORANGA. **Estão abertas as inscrições para estudantes solicitarem auxílio-transporte para o 2º semestre**. 2013. Disponível em: <http://www.ituporanga.sc.gov.br/noticias/estao-abertas-as-inscricoes-para-estudantes-solicitarem-auxilio-transporte-para-o-2-semestre-2634.html>. Acesso em: 13 set. 2015.

JACUPIRANGA. Lei nº 1.184, de 06 de fevereiro de 2015. **Dispõe Sobre Programa de Transporte Universitário e Técnico Profissionalizante no Município de Jacupiranga, e de Outras Providências**. Jacupiranga, SP.

LEME (Município). Lei Nº 3.284, de 02 de abril de 2013. **Dispõe Sobre Instituição do Programa Municipal de Auxílio Transporte Para Estudantes Universitários "PAE" e Dá Outras Providências**. Leme, SP.

LOCAWEB. **MVC, o que que é isso?**. 2008. Disponível em: <http://blog.locaweb.com.br/tecnologia/aspnet-mvc-o-que-e-isso/>. Acesso em: 18 out. 2015.

MASSARANDUBA (Município). Lei nº 1285, de 17 de maio de 2011. **Concede Auxílio Financeiro Para Transporte Escolar de Estudantes Universitários e de Cursos Técnicos de Nível Médio Profissionalizante do Município de Massaranduba**. Massaranduba, SC.

MASSARANDUBA (Município). Lei nº 1610, de 15 de agosto de 2014. **Altera Artigos da Lei Nº. 1285/2011 de 17 de Maio de 2011**. Massaranduba, SC.

MELO, Ivo Soares. **Administração de sistemas de informação**. 3. ed. São Paulo: Pioneira Thomson Learning, 2002.

OLIVEIRA, Bruno. **JavaFX**: Interfaces com qualidade para aplicações desktop. São Paulo: Casa do Código, 2013.

OLIVEIRA, Djalma de Pinho Rebouças. **Sistemas de informação gerenciais:** estratégias, táticas, operacionais. 15. ed. São Paulo: Atlas, 2012.

ORACLE. **JavaFX 1.1 SDK Release Notes**. 2009. Disponível em: <http://www.oracle.com/technetwork/java/javafx/javafx-sdk-release-notes-1-1-139569.html>. Acesso em: 19 set. 2015.

ORACLE. **JavaFX Scene Builder**: A Visual Layout Tool for JavaFX Applications. 2012a. Disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html&prev=search>. Acesso em: 19 set. 2015.

ORACLE. **JavaFX Scene Builder User Guide**. 2012b. Disponível em: <https://docs.oracle.com/javafx/scenebuilder/1/user_guide/main-window.htm>. Acesso em: 19 set. 2015.

ORACLE. **Using JavaFX Scene Builder with Java IDEs**. 2013. Disponível em: <http://docs.oracle.com/javafx/scenebuilder/1/use_java_ides/sb-with-nb.htm>. Acesso em: 19 set. 2015.

PATI, Camila. **As melhores universidades do Brasil, segundo o MEC**. Disponível em: <http://exame.abril.com.br/carreira/noticias/as-melhores-universidades-do-brasil-segundo-o-mec>. Acesso em: 06 abr. 2015.

PENTEADO, Luciano de Camargo. **Responsabilidade civil no transporte de pessoas:** diálogo entre o CDC e o CC. 2006. Disponível em: <http://www.ambito-juridico.com.br/site/?n_link=revista_artigos_leitura&artigo_id=1332>. Acesso em: 14 mar. 2015.

QUERINO FILHO, Luiz Carlos. **Por dentro do JavaFX 2 - Revista Java Magazine 105**. 2013. Disponível em: <http://www.devmedia.com.br/por-dentro-do-javafx-2-revista-java-magazine-105/25041>. Acesso em: 19 set. 2015.

RIO DO OESTE (Município). Lei nº 2002, de 05 de abril de 2013. **Dispõe Sobre O Programa Municipal de Auxílio Transporte Para Estudantes e DÁ Outras Providências**. Rio do Oeste, SC.

SCHAEFER, Carine. **Protótipo de aplicativo para transmissão de dados a partir de dispositivos móveis aplicado a uma empresa de transportes**. 2004. 53 f. TCC (Graduação) - Curso de Curso de Ciências da Computação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2004.

SCHULZ, Bennet. **Bye Bye JavaFX Scene Builder, Welcome Scene Builder 8.0.0 Gluon**. 2013. Disponível em: <https://dzone.com/articles/bye-bye-javafx-scene-builder&prev=search>. Acesso em: 19 set. 2015.

TAMAN, Mohamed. **JavaFX Essentials**. Birmingham, Uk: Packt Publishing, 2015.

APÊNDICE A – Descrição dos Casos de Uso

Este Apêndice apresenta a descrição dos principais casos de uso. No Quadro 21 apresenta-se o caso de uso `Efetuar login`.

Quadro 21 - Descrição do caso de uso UC01

UC01	O sistema deverá permitir o Usuário, Administrador efetuar o *login* no sistema.
Descrição	Permite que o Usuário, Administrador através de identificação de *login* e senha conecte-se ao sistema.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados.
Fluxo Principal	Usuário, Administrador preenche seu *login* e senha. Sistema valida o *login* e a senha. Sistema direciona o Usuário, Administrador para a tela inicial do sistema.
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem "*login* e/ou senha inválidos" é apresentada;
Pós-condição	O Usuário, Administrador conecta-se ao sistema.

No Quadro 22 apresenta-se o caso de uso `Alterar Senha`.

Quadro 22 - Descrição do caso de uso UC02

UC02	O sistema deverá permitir o Usuário, Administrador alterar senha do *login*.
Descrição	Permite que o Usuário, Administrador altere sua senha de *login*, informando a senha antiga, a nova senha e a nova senha novamente para garantia.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema mostra tela de alteração de senha; Usuário, Administrador informa a senha antiga, a nova senha e repete a nova senha; Sistema valida a nova senha e apresenta a mensagem "Senha alterada com sucesso!".
Fluxo Alternativo	Nome de usuário e/ou senha inválido(s). Alerta com mensagem "usuário e/ou senha inválidos" é apresentada. Nova senha diferente de confirmação nova senha. Alerta com mensagem "nova senha é diferente da confirmação" é apresentada.
Pós-condição	O Usuário, Administrador alterou a senha de acesso.

No Quadro 23 apresenta-se o caso de uso `Manter Estudante`.

Quadro 23 - Descrição do caso de uso UC03

UC03	O sistema deverá permitir o Usuário, Administrador manter estudantes.
Descrição	Permite que o Usuário, Administrador cadastre alunos informando os dados cadastrais. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa os estudantes cadastrados; Usuário, Administrador opta por alterar, modificar status ou cadastrar um estudante;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem “Favor preencher todos os campos obrigatórios” é apresentada; Alerta com a mensagem “Estudante já cadastrado” é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de estudante; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem “Estudante cadastrado com sucesso”.
Cenário - Edição	Usuário, Administrador seleciona um estudante para edição; Sistema mostra tela para edição; Usuário, Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem “Estudante alterado com sucesso”.
Cenário - Status	Usuário, Administrador seleciona um estudante para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de um estudante.

No Quadro 24 apresenta-se o caso de uso `Manter empresa`.

Quadro 24 - Descrição do caso de uso UC04

UC04	O sistema deverá permitir o Usuário, Administrador manter empresas de transporte
Descrição	Permite que o Usuário, Administrador cadastre empresas de transporte informando os dados cadastrais. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa as empresas cadastradas; Usuário, Administrador opta por alterar, modificar status ou cadastrar uma empresa;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem "Favor preencher todos os campos obrigatórios" é apresentada; Alerta com a mensagem "Empresa já cadastrada" é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de empresas; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem "Empresa cadastrada com sucesso".
Cenário - Edição	Usuário, Administrador seleciona uma empresa para edição; Sistema mostra tela para edição; Usuário alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem "Empresa alterada com sucesso".
Cenário - Status	Usuário, Administrador seleciona uma empresa para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de uma empresa de transporte.

No Quadro 25 apresenta-se o caso de uso `Manter veículo`.

Quadro 25 - Descrição do caso de uso UC05

UC05	O sistema deverá permitir o Usuário, Administrador manter veículos para transporte
Descrição	Permite que o Usuário, Administrador cadastre veículos para transporte informando marca, modelo, ano, placa, renavan, chassi, quantidade de lugares, cor. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa os veículos cadastrados; Usuário, Administrador opta por alterar, modificar status ou cadastrar um veículo;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem “Favor preencher todos os campos obrigatórios” é apresentada; Alerta com a mensagem “Veículo já cadastrado” é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de veículo; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem “Veículo cadastrado com sucesso”.
Cenário - Edição	Usuário, Administrador seleciona um veículo para edição; Sistema mostra tela para edição; Usuário, Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem “Veículo alterado com sucesso”.
Cenário - Status	Usuário, Administrador seleciona um veículo para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de um veículo de transporte.

No Quadro 26 apresenta-se o caso de uso `Manter motoristas`.

Quadro 26 - Descrição do caso de uso UC06

UC06	O sistema deverá permitir o Usuário, Administrador manter motoristas
Descrição	Permite que o Usuário, Administrador cadastre motoristas informando os dados cadastrais. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa os motoristas cadastrados; Usuário, Administrador opta por alterar, modificar status ou cadastrar um motorista;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem “Favor preencher todos os campos obrigatórios” é apresentada; Alerta com a mensagem “Motorista já cadastrado” é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de motorista; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem “Motorista cadastrado com sucesso”.
Cenário - Edição	Usuário, Administrador seleciona um motorista para edição; Sistema mostra tela para edição; Usuário, Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem “Motorista alterado com sucesso”.
Cenário - Status	Usuário, Administrador seleciona um motorista para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de um motorista.

No Quadro 27 apresenta-se o caso de uso `Manter instituição`.

Quadro 27 - Descrição do caso de uso UC07

UC07	O sistema deverá permitir o Usuário, Administrador manter instituições de ensino
Descrição	Permite que o Usuário, Administrador cadastre instituições de ensino informando os dados cadastrais. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa as instituições cadastradas; Usuário, Administrador opta por alterar, modificar status ou cadastrar uma instituição;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem "Favor preencher todos os campos obrigatórios" é apresentada; Alerta com a mensagem "Instituição já cadastrada" é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de instituição; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem "Instituição cadastrada com sucesso".
Cenário - Edição	Usuário, Administrador seleciona uma instituição para edição; Sistema mostra tela para edição; Usuário, Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem "Instituição alterada com sucesso".
Cenário - Status	Usuário, Administrador seleciona uma instituição para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de uma instituição de ensino

No Quadro 28 apresenta-se o caso de uso `Manter curso`.

Quadro 28 - Descrição do caso de uso UC08

UC08	O sistema deverá permitir o Usuário, Administrador manter cursos
Descrição	Permite que o Usuário, Administrador cadastre cursos informando os dados cadastrais. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa os cursos cadastrados; Usuário, Administrador opta por alterar, modificar status ou cadastrar um curso;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem “Favor preencher todos os campos obrigatórios” é apresentada; Alerta com a mensagem “Curso já cadastrado” é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de curso; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem “Curso cadastrado com sucesso”.
Cenário - Edição	Usuário, Administrador seleciona um curso para edição; Sistema mostra tela para edição; Usuário, Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem “Curso alterado com sucesso”.
Cenário - Status	Usuário, Administrador seleciona um curso para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de um curso.

No Quadro 29 apresenta-se o caso de uso `Manter trajeto`.

Quadro 29 - Descrição do caso de uso UC09

UC09	O sistema deverá permitir o Usuário, Administrador manter trajetos para o transporte
Descrição	Permite que o Usuário, Administrador cadastre trajetos informando descrição, valor, período, ano, local de saída, horário, instituição, empresa, veículo e motorista. Altere o cadastro, assim como efetuar consultas.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estar cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa os trajetos cadastrados; Usuário, Administrador opta por alterar, modificar status ou cadastrar um trajeto;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem “Favor preencher todos os campos obrigatórios” é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de trajeto; Usuário, Administrador informa todos os campos obrigatórios; Usuário, Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem “Trajeto cadastrado com sucesso”.
Cenário - Edição	Usuário, Administrador seleciona um trajeto para edição; Sistema mostra tela para edição; Usuário, Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem “Trajeto alterado com sucesso”.
Cenário - Status	Usuário, Administrador seleciona um trajeto para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Usuário, Administrador cadastrou, editou ou modificou status de um trajeto.

No Quadro 30 apresenta-se o caso de uso `Manter usuário`.

Quadro 30 - Descrição do caso de uso UC10

UC10	O sistema deverá permitir o administrador manter usuários no sistema
Descrição	Permite que o administrador cadastre usuário no sistema informando nome de usuário, senha, telefone, privilégio. Altere o cadastro, assim como efetuar consultas.
Ator	Administrador
Pré-condição	Administrador deve estar cadastrado no banco de dados. Administrador deve efetuar *login* no sistema.
Fluxo Principal	Sistema informa os motoristas cadastrados; Administrador opta por alterar, modificar status ou cadastrar um usuário;
Fluxo Alternativo	Campo(s) obrigatório(s) não preenchido(s); Alerta com a mensagem "Favor preencher todos os campos obrigatórios" é apresentada; Alerta com a mensagem "Usuário já cadastrado" é apresentada;
Cenário - Inclusão	Sistema mostra tela de cadastro de usuário; A informa todos os campos obrigatórios; Administrador seleciona a opção salvar; Sistema valida o cadastro e apresenta a mensagem "Usuário cadastrado com sucesso".
Cenário - Edição	Administrador seleciona um usuário para edição; Sistema mostra tela para edição; Administrador alterar os dados necessários e seleciona a opção salvar; Sistema valida a alteração e apresenta a mensagem "Usuário alterado com sucesso".
Cenário - Status	Usuário seleciona um usuário para alterar o status; Sistema altera o status e atualiza a lista exibida
Pós-condição	Administrador cadastrou, editou ou modificou status de um usuário.

No Quadro 31 apresenta-se o caso de uso `Relatório de empresas de transporte`.

Quadro 31 - Descrição do caso de uso UC11

UC11	O sistema deverá emitir relatório de empresas de transporte
Descrição	Permite que o Usuário, Administrador emita relatórios referente as empresas que prestam o serviço de transporte.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estra cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador seleciona a opção no painel de bordo. Sistema apresenta tela de análise de empresas de transporte. Usuário, Administrador seleciona "relatório geral". Sistema gera o relatório.
Fluxo Alternativo	Nenhum registro inserido. Alerta com a mensagem "Relatório sem páginas".
Pós-condição	Usuário, Administrador emitiu um relatório de empresas.

No Quadro 32 apresenta-se o caso de uso `Relatório de veículos por empresa`.

Quadro 32 - Descrição do caso de uso UC12

UC12	O sistema deverá emitir relatório de veículos por empresa
Descrição	Permite que o Usuário, Administrador emita relatórios referentes aos veículos de cada empresa que realizam o transporte dos estudantes.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estra cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador seleciona a opção no painel de bordo. Sistema apresenta tela de análise de veículos. Usuário, Administrador seleciona "relatório de veículos por empresa". Sistema gera o relatório.
Fluxo Alternativo	Nenhum registro inserido. Alerta com a mensagem "Relatório sem páginas".
Pós-condição	Usuário, Administrador emitiu um relatório de veículos.

No Quadro 33 apresenta-se o caso de uso `Relatório de cursos por instituição de ensino`.

Quadro 33 - Descrição do caso de uso UC13

UC13	O sistema deverá emitir relatório de cursos por instituição de ensino
Descrição	Permite que o Usuário, Administrador emita relatórios dos cursos por instituição de ensino.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estra cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador seleciona a opção no painel de bordo. Sistema apresenta tela de análise de cursos. Usuário, Administrador seleciona "relatório de cursos por instituição". Sistema gera o relatório.
Fluxo Alternativo	Nenhum registro inserido. Alerta com a mensagem "Relatório sem páginas".
Pós-condição	Usuário, Administrador emitiu um relatório de instituições de ensino.

No Quadro 34 apresenta-se o caso de `uso Relatório de estudantes por curso e instituição de ensino`.

Quadro 34 - Descrição do caso de uso UC14

UC14	O sistema deverá emitir relatório de estudantes por curso e instituição de ensino
Descrição	Permite que o Usuário, Administrador emita relatórios de estudantes separando por curso e também pela instituição de ensino.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estra cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador seleciona a opção no painel de bordo. Sistema apresenta tela de análise de estudantes. Usuário, Administrador seleciona "relatório de estudantes por cursos e instituição". Sistema gera o relatório.
Fluxo Alternativo	Nenhum registro inserido. Alerta com a mensagem "Relatório sem páginas".
Pós-condição	Usuário, Administrador emitiu um relatório de estudantes.

No Quadro 35 apresenta-se o caso de uso `Relatório de usuários`.

Quadro 35 - Descrição do caso de uso UC15

UC15	O sistema deverá permitir o administrador emitir relatório de usuários
Descrição	Permite que o administrador emita relatórios dos usuários que utilizam o sistema, listando o *login* e as datas de acesso.
Ator	Administrador
Pré-condição	Administrador deve estra cadastrado no banco de dados. Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador seleciona a opção no painel de bordo. Sistema apresenta tela de análise de usuários. Administrador seleciona "relatório geral". Sistema gera o relatório.
Fluxo Alternativo	Nenhum registro inserido. Alerta com a mensagem "Relatório sem páginas".
Pós-condição	Administrador emitiu um relatório de usuários.

No Quadro 36 apresenta-se o caso de uso `Análise de informações gerenciais`.

Quadro 36 - Descrição do caso de uso UC16

UC16	O sistema deverá permitir o usuário analisar as informações gerenciais.
Descrição	Permite que o usuário analise as informações cadastradas através de gráficos apresentados em um painel de bordo.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estra cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador escolhe uma opção do painel de bordo. Sistema apresenta a tela com gráficos referentes a escolha.
Pós-condição	Usuário, Administrador analisou informações do sistema.

No Quadro 37 apresenta-se o caso de uso `Relatórios gerenciais`.

Quadro 37 - Descrição do caso de uso UC17

UC17	O sistema deverá permitir o Usuário, Administrador emitir relatórios gerenciais.
Descrição	Permite que o Usuário, Administrador emita relatórios gerenciais contendo a lista geral de registros inseridos.
Ator	Usuário, Administrador
Pré-condição	Usuário, Administrador deve estra cadastrado no banco de dados. Usuário, Administrador deve efetuar *login* no sistema.
Fluxo Principal	Usuário, Administrador escolhe uma opção do painel de bordo. Sistema apresenta a tela referente a escolha. Usuário, Administrador opta por gerar um relatório geral. Sistema gera relatório que pode ser impresso.
Fluxo Alternativo	Nenhum registro inserido. Alerta com a mensagem "Relatório sem páginas".
Pós-condição	Usuário, Administrador gerou um relatório do sistema.

APÊNDICE B – Dicionário de Dados

Este Apêndice apresenta a descrição das entidades propostas para o sistema.

O Quadro 38 apresenta a tabela `usuário`.

Quadro 38 - Tabela usuario

`Usuario` – A tabela `usuario` serve para armazenar os usuários do sistema		
Campo	**Tipo**	**Descrição**
id_usuario	INT	Identificação do usuário
ds_usuario	VARCHAR(45)	Login do usuário
ds_senha	VARCHAR(45)	Senha do usuário
is_admin	BOOLEAN	Identificação de privilégios
sg_status	CHAR(1)	Status do usuário
nr_telefone	VARCHAR(20)	Telefone do usuário

O Quadro 39 apresenta a tabela `localidade`.

Quadro 39 - Tabela localidade

`Localidade` – A tabela `localidade` serve para armazenar as localidades do sistema		
Campo	**Tipo**	**Descrição**
id_localidade	INT	Identificação da localidade
nm_cidade	VARCHAR(45)	Nome da cidade
sg_estado	CHAR(2)	Sigla do estado

O Quadro 40 apresenta a tabela `empresa`.

Quadro 40 - Tabela empresa

`Empresa` – A tabela `empresa` serve para armazenar as empresas do sistema		
Campo	**Tipo**	**Descrição**
id_empresa	INT	Identificação do usuário
id_localidade	INT	Identificação da localidade
nm_fantasia	VARCHAR(100)	Nome fantasia da empresa
ds_razao_social	VARCHAR(100)	Razão social da empresa
nr_cnpj	VARCHAR(20)	CNPJ da empresa
nr_inscricao_estadual	VARCHAR(45)	Inscrição estadual da empresa
nr_inscricao_municipal	VARCHAR(45)	Inscrição municipal da empresa
nr_tel_comercial	VARCHAR(20)	Telefone comercial da empresa
nr_tel_complementar	VARCHAR(20)	Telefone complementar da empresa
ds_email	VARCHAR(45)	Email da empresa
ds_site	VARCHAR(45)	Site da empresa
ds_bairro	VARCHAR(100)	Bairro da empresa
nr_cep	VARCHAR(10)	Cep da empresa
ds_logradouro	VARCHAR(100)	Logradouro da empresa
sg_status	CHAR(1)	Status da empresa

O Quadro 41 apresenta a tabela `veiculo`.

Quadro 41 - Tabela veiculo

Veiculo – A tabela veiculo serve para armazenar os veículos do sistema		
Campo	**Tipo**	**Descrição**
id_veiculo	INT	Identificação do veiculo
id_empresa	INT	Identificação da empresa
ds_modelo	VARCHAR(45)	Modelo do veículo
ds_marca	VARCHAR(45)	Marca do veículo
ds_placa	VARCHAR(10)	Placa do veículo
ds_ano_fab	VARCHAR(4)	Ano de fabricação
ds_ano_mod	VARCHAR(4)	Ano do modelo
ds_cor	VARCHAR(20)	Cor do veículo
nr_renavam	VARCHAR(20)	Renavam do veículo
nr_chassi	VARCHAR(45)	Chassi do veículo
nr_passageiros	INT	Quantidade máxima de passageiros
sg_status	CHAR(1)	Status do veículo

O Quadro 42 apresenta a tabela estudante.

Quadro 42 - Tabela estudante

Estudante – A tabela estudante serve para armazenar os estudantes do sistema		
Campo	**Tipo**	**Descrição**
id_estudante	INT	Identificação do estudante
id_localidade	INT	Identificação da localidade
id_curso	INT	Identificação do curso
id_trajeto	INT	Identificação do trajeto
nm_estudante	VARCHAR(45)	Nome do estudante
ds_cpf	VARCHAR(15)	CPF do estudante
sg_sexo	CHAR(1)	Gênero do estudante
dt_nascimento	DATE	Data de nascimento
nr_tel_celular	VARCHAR(20)	Telefone celular do estudante
nr_tel_residencial	VARCHAR(20)	Telefone residencial do estudante
ds_email	VARCHAR(45)	Email do estudante
ds_periodo	VARCHAR(10)	Período de estudos
ds_bairro	VARCHAR(100)	Bairro do estudante
nr_cep	VARCHAR(10)	Cep do estudante
ds_logradouro	VARCHAR(100)	Logradouro do estudante
sg_status	CHAR(1)	Status do estudante

O Quadro 43 apresenta a tabela trajeto.

Quadro 43 - Tabela trajeto

Trajeto – A tabela trajeto serve para armazenar os trajetos do sistema		
Campo	**Tipo**	**Descrição**
id_trajeto	INT	Identificação do trajeto
id_instituição	INT	Identificação da instituição
id_veiculo	INT	Identificação do veiculo
id_motorista	INT	Identificação do motorista
ds_trajeto	VARCHAR(100)	Descrição do trajeto
ds_periodo	VARCHAR(10)	Descrição do período
ds_ano_semestre	VARCHAR(10)	Descrição do ano e semestre
ds_local_saida_ida	VARCHAR(100)	Descrição do local de saída da ida
ds_horario_saida_ida	TIME(4)	Horário de saída da ida
ds_local_saida_retorno	VARCHAR(100)	Descrição do local de saída do retorno
ds_horario_saida_retorno	TIME(4)	Horário de saída do retorno
nr_valor	FLOAT	Valor do trajeto
sg_status	CHAR(1)	Status do trajeto

O Quadro 44 apresenta a tabela motorista.

Quadro 44 - Tabela motorista

Motorista – A tabela motorista serve para armazenar os motoristas do sistema		
Campo	**Tipo**	**Descrição**
id_motorista	INT	Identificação do motorista
id_empresa	INT	Identificação da empresa
id_localidade	INT	Identificação da localidade
nm_motorista	VARCHAR(45)	Nome do motorista
ds_cpf	VARCHAR(15)	CPF do motorista
sg_sexo	CHAR(1)	Gênero do motorista
dt_nascimento	DATE	Data de nascimento
nr_tel_celular	VARCHAR(20)	Telefone celular do motorista
nr_tel_complementar	VARCHAR(20)	Telefone complementar do motorista
ds_renach	VARCHAR(45)	Renach do motorista
ds_categoria	VARCHAR(5)	Categoria da carteira de motorista
ds_bairro	VARCHAR(100)	Bairro do motorista
nr_cep	VARCHAR(10)	Cep do motorista
ds_logradouro	VARCHAR(100)	Logradouro do motorista
sg_status	CHAR(1)	Status do motorista

O Quadro 45 apresenta a tabela curso.

Quadro 45 - Tabela curso

Curso – A tabela curso serve para armazenar os cursos do sistema		
Campo	**Tipo**	**Descrição**
id_curso	INT	Identificação do curso
id_instituicao	INT	Identificação da instituição
nm_curso	VARCHAR(45)	Nome do curso
ds_titulo_conferido	VARCHAR(20)	Título conferido ao curso
ds_duracao	VARCHAR(10)	Duração do curso
ds_periodo	VARCHAR(45)	Período do curso
ds_observacao	VARCHAR(100)	Observação sobre o curso
sg_status	CHAR(1)	Status do estudante

O Quadro 46 apresenta a tabela instituição.

Quadro 46 - Tabela instituicao

Instituicao – A tabela instituicao serve para armazenar as instituições do sistema		
Campo	**Tipo**	**Descrição**
id_instituicao	INT	Identificação da instituição
id_localidade	INT	Identificação da localidade
nm_instituicao	VARCHAR(100)	Nome da instituição
nr_cnpj	VARCHAR(20)	CNPJ da instituição
nr_inscricao_estadual	VARCHAR(45)	Inscrição estadual da instituição
nr_inscricao_municipal	VARCHAR(45)	Inscrição municipal da instituição
ds_portaria_ministerial	VARCHAR(45)	Descrição da portaria ministerial
nr_tel_comercial	VARCHAR(20)	Telefone comercial da instituição
nr_tel_complementar	VARCHAR(20)	Telefone complementar da instituição
ds_site	VARCHAR(45)	Site da instituição
ds_bairro	VARCHAR(100)	Bairro da instituição
nr_cep	VARCHAR(10)	Cep da instituição
ds_logradouro	VARCHAR(100)	Logradouro da instituição
sg_status	CHAR(1)	Status da instituição