UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ANDRÉ LUIZ RABELLO DA SILVA

IMPLEMENTAÇÃO DE UM SISTEMA PARA GESTÃO DE BARES INTEGRANDO TECNOLOGIAS WEB E MOBILE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2014

ANDRÉ LUIZ RABELLO DA SILVA

IMPLEMENTAÇÃO DE UM SISTEMA PARA GESTÃO DE BARES INTEGRANDO TECNOLOGIAS WEB E MOBILE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Profa. Eliane De Bortoli Fávero.

PATO BRANCO
2014

ATA Nº: **249**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **ANDRé LUIZ RABELLO DA SILVA.**

Às 15:30 hrs do dia 16 de dezembro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Eliane Maria de Bortoli Fávero (Orientadora), Robison Cris Brito (Convidado) e Fábio Favarim (Convidado), para avaliar o Trabalho de Diplomação do aluno André Luiz Rabello da Silva, matrícula 01031350, sob o título **Implementação de um Sistema para Gestão de Bares Integrando Tecnologias Web e Mobile**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 15:35 hrs foi encerrada a sessão.

Profa. Eliane Maria de Bortoli Fávero, M.Sc.
Orientadora

Prof. Robison Cris Brito, M.Sc.
Convidado

Prof. Fábio Favarim, Dr.
Convidado

Profa. Eliane Maria de Bortoli Fávero, M.Sc
Coordenadora do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

SILVA, André Luiz Rabello da. Implementação de um Sistema para Gestão de Bares Integrando Tecnologias *Web* e *Mobile*. 2014. 103f. Monografia (Trabalho de Conclusão de Curso). Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2014.

Com o atual crescimento no uso de computadores para a automatização de processos, muitas áreas ainda estão carentes desta facilidade, possuem aplicações defasadas ou feitas de forma artesanal. Criar projetos de sistemas de informação bem elaborados, visando evitar possíveis falhas em sua implementação tem sido uma exigência do mercado. A atividade desenvolvida por bares apresenta uma demanda considerável por sistemas de software que ofereçam suporte à realização de seus processos. Dentro desse contexto, esse trabalho visa seguir essa tendência e realizar a análise, projeto e implementação de um sistema para a gestão de bares em geral, seguida da implementação desse sistema em plataforma *Web* integrada à *mobile*. O sistema deve automatizar o processo de venda, recebimentos e controle de estoque do estabelecimento, assim como fornecer suporte à gerência na tomada de decisões, apresentando interfaces simples, porém funcionais. Foram utilizadas as tecnologias de *Java* para *Web* e *Android*, sendo apresentadas técnicas utilizadas para a implementação e integração das tecnologias, de forma a contribuir com futuros usuários dessas tecnologias.

Palavras-chave: Aplicação *Web.* Aplicação *mobile.* Sistema para Gestão de bares. Integração *Web* e *mobile*.

ABSTRACT

SILVA, André Luiz Rabello da. Implementation of a System for Bar Management Integrating Web Technologies and Mobile. 2014. 103f. Monografia (Trabalho de Conclusão de Curso). Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2014.

With the current growth in the use of computers for process automation many areas are still deprived of this facility and have lagged or handmade applications. Creating projects for well-designed information systems in order to avoid possible failures in its implementation has been a market's requirement. The activity developed by bars shows a considerable demand for software systems that support the realization of its processes. Within this context, this work aims to follow this trend and perform the analysis, design and implementation of a system for the management of bars in general, proceeded by the implementation of this system in integrated Web platform to mobile. The system should automate the processes of sale, receipts and inventory control category, as well as providing support to management in decision-making, with simple but functional interfaces. Java technologies were used for Web and Android, and included techniques used for implementation and integration of technologies in order to contribute to future users of these technologies.

Palavras-chave: Web application. Application mobile. System for bars Management. Integration Web and mobile.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, dando uma visão geral do trabalho, seus objetivos e como o mesmo se organiza.

1.1 Considerações iniciais

Estabelecimentos comerciais que oferecem bebidas de diversos tipos e petiscos em geral têm sido denominados bares. A disseminação desse tipo de estabelecimento iniciou-se desde a. C até os dias de hoje, sempre inovando na forma de atendimento.

A forma como o proprietário ou atendente controla as vendas muitas vezes é manual, na qual é anotado em alguma folha de papel, formatada ou não, o pedido e a mesa a qual o pedido foi requisitado. Como as informações são descritas manualmente e em um meio nada seguro para armazenar as informações, torna-se difícil o gerenciamento das vendas deixando quase que impossível a elaboração de relatórios sobre as mesmas. Essa realidade é encontrada principalmente em bares de pequeno porte.

Com o mercado cada vez mais informatizado, em que processos automatizados geram menos riscos que processos manuais, além do barateamento de investimentos para a aquisição de equipamentos, torna viável a informatização de negócios, sejam eles grandes ou pequenos.

Com isso, a busca de melhores resultados nos rotineiros processos das empresas e uma fundamental busca pelo gerenciamento das informações de forma eficaz e organizada, dá-se início a uma nova era, a era dos sistemas de informação. Segundo Bezerra (2007), um sistema de informação é uma combinação de pessoas, dados, processos, interfaces, redes de comunicações e tecnologia que interagem com o objetivo de dar suporte e melhorar o processo de negócio de uma organização empresarial com relação às informações que nela fluem.

Tendo em vista que a automatização de processos de negócio se tornou mais acessível atualmente, há a necessidade de criar projetos de sistemas de informação bem elaborados, visando evitar possíveis falhas ou possíveis equívocos na implementação de suas funcionalidades, gerando assim informações confiáveis.

Boa parte do mercado de negócios tem observado essa mudança e estão cientes de que a indústria alavancou produções severas aos aparelhos de telas sensíveis ao toque. Com essa forte tendência, grandes marcas e algumas empresas passaram a investir em tecnologia *mobile*. Elas reconhecem que todas as iniciativas que focam o crescente alvo dos consumidores *high-tech*, em pouco tempo compensarão os investimentos, ou seja, o incentivo para que sejam desenvolvidas ferramentas básicas e aplicativos para mobile está aí (ARAÚJO, 2014). Além disso, houve um aumento da capacidade, tanto de processamento como o de armazenamento de *smartphones* e *tablets.*

Dentro desse contexto, esse trabalho visa seguir essa tendência e realizar a análise, projeto e implementação de um sistema para a gestão de bares em geral, seguida da implementação desse sistema em plataforma Web integrada à *mobile*. O sistema deve automatizar o processo de venda, recebimentos e controle de estoque do estabelecimento, assim como fornecer suporte à gerência na tomada de decisões.

1.2 Objetivos

1.2.1 Objetivo Geral

- Implementar um sistema para gestão de bares, integrando tecnologias para Web e *mobile,* a fim *de* automatizar o processo de venda, recebimentos e controle de estoque, assim como fornecer suporte à gerência do estabelecimento.

1.2.2 Objetivos Específicos

- Facilitar o controle das vendas realizadas em bares em geral;
- Facilitar o trabalho dos garçons no registro dos pedidos por meio do uso de tecnologias móveis;
- Agilizar a tomada de decisão por parte da gerência de estabelecimentos dessa natureza.

1.3 Justificativa

O desenvolvimento do sistema de gestão de bares justifica-se pela necessidade de se ter um melhor controle sobre os pedidos realizados pelos clientes de estabelecimentos dessa natureza ou mesmo de uma melhor gestão do ponto de vista administrativo, em que relatórios sobre o estoque e vendas podem ser gerados de forma rápida e simplificada. Outro fato que justifica o seu desenvolvimento é o fato de que os dados sobre as vendas ficarão armazenados por um período de prazo maior e com uma elevada garantia de segurança sem que isso custe maior espaço físico no estabelecimento, tendo em vista que os dados gerados ficaram armazenados no meio eletrônico, o que facilita tanto na busca como manipulação desses dados.

Há uma crescente demanda por sistemas para essa finalidade, mesmo existindo algumas soluções já em uso. O fato é que muitas delas apresentam um grande número de recursos, muitas vezes não utilizados, ou mesmo são difíceis de usar. Outras, por sua vez, deixam a desejar no atendimento às funcionalidades necessárias à esses estabelecimentos, especialmente os de pequeno porte.

Sendo assim, a proposta desse software tem como objetivo auxiliar na organização dos dados de estoque e pedidos de um bar, bem como a tomada de decisão por parte de sua gerência. O software também deverá apresentar um controle de estoque eficiente em tempo real. Desta forma, o software

deverá agilizar o acesso a esses dados de forma íntegra, possibilitando um atendimento de melhor qualidade e propiciando aos clientes maior agilidade em seu atendimento.

A escolha da tecnologia *Android* para o desenvolvimento *Mobile* se justifica pelo fato de ser uma tecnologia atual, ser de código aberto e também pelo fato de ser o sistema operacional móvel mais utilizado no planeta. Já a escolha da tecnologia Web se justifica pela sua flexibilidade de acesso, não havendo a necessidade de instalar o sistema, basta acessá-lo a partir de um *browser* convencional.

1.4 Estrutura do trabalho

O trabalho está estruturado em capítulos, sendo que o Capítulo 1 é o atual, no qual estão contidas as considerações iniciais para o sistema, objetivos e justificativa. O Capítulo 2 contém o referencial teórico que fundamenta a proposta do sistema proposto. No Capítulo 3 são apresentados os métodos utilizados na elaboração do projeto. O Capítulo 4 apresenta os resultados do projeto, desde a sua análise até o seu desenvolvimento, como escopo, requisitos, modelagem, apresentação do sistema e finalizando com a implementação do sistema. No Capítulo 5 são apresentadas as conclusões do trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico utilizado para fundamentar o trabalho. O referencial apresenta e descreve as tecnologias utilizadas no desenvolvimento da aplicação proposta.

2.1 Visão de um sistema de informação

Com a busca de melhores resultados nos rotineiros processos das empresas e uma fundamental busca pelo gerenciamento das informações de forma eficaz e organizada, dá-se início a uma nova era, a era dos sistemas de informação.

Segundo Bezerra (2007), um sistema de informação é uma combinação de pessoas, dados, processos, interfaces, redes de comunicações e tecnologia que interagem com o objetivo de dar suporte e melhorar o processo de negócio de uma organização empresarial com relação às informações que nela fluem.

Esses sistemas de informações são elaborados e implementados de forma a atender de forma eficaz e eficiente os usuários que o utilizarem, facilitando e agilizando suas tarefas na empresa a qual colaborem. O gasto no desenvolvimento deste sistema só será justificado caso ele consiga atender os pontos citados.

O desenvolvimento de um sistema deste tipo, não é uma tarefa fácil. Quanto mais funções o sistema possuir, maior será a sua complexidade, tanto no gerenciamento de suas atividades, como na elaboração do seu projeto.

2.2 Linguagem de Modelagem Unificada (UML)

No início dos anos 50, os sistemas eram significativamente simples. Seu desenvolvimento era de forma artesanal ou com uso de alguns fluxogramas para esboçar como o sistema iria ser.

Com o passar dos anos, os computadores foram evoluindo sua performance e seu custo começaram a se tornar acessíveis. Sistemas mais complexos começaram a surgir. Em virtude dessa emergente evolução, surge no final da década de 70 a análise e projeto estruturado.

No decorrer da década de 80 os computadores continuaram a melhorar seu custo e desempenho e surge a necessidade de interfaces amigáveis entre o homem-máquina. No início da década de 90 o mercado segue a tendência dos anos anteriores. Com o surgimento das linguagens de programação orientadas a objeto e de novas tecnologias, os sistemas ficando cada vez mais complexos, surge então a análise orientada a objetos, que visava dar resposta à algumas dificuldades encontradas na análise estruturada e melhorar o processo de desenvolvimento de um sistema.

Então a década de 90 tornou-se a década em que a Orientação a Objetos ganhou vida, mas com o decorrer dos anos diversas técnicas de notações gráficas foram surgindo. Tendo em vista que isso poderia ser tornar um grande problema, percebe-se a necessidade de padronizar a modelagem de sistemas orientados a objetos. Então, em 1996, surge a *Unified Modeling Language* (UML ou Linguagem de Modelagem Unificada) como o padrão na forma de representar sistemas orientados a objetos.

No ano de 1997, a UML é aprovada como padrão para modelagem orientada a objetos. Com o passar dos anos, seu padrão foi se ajustando conforme as necessidades e atualizações foram feitas para torná-la mais clara. Atualmente a UML encontra se na versão 2.0.

A UML é uma linguagem visual para realizar modelagem de sistemas orientados a objetos. Através desses elementos visuais é possível construir diagramas que representam diferentes perspectivas de um sistema (estática, dinâmica e física).

Cada elemento possui sua própria sintaxe e sua própria semântica, sendo que a sintaxe descreve a forma pré-definida de como o elemento visual deve ser desenhado e a semântica o que o elemento significa e com que outros objetos ele poderá se relacionar.

A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas complexos de software. A UML proporciona uma forma-padrão para a preparação de planos de arquitetura de projetos de sistemas, incluindo aspectos conceituais tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de software reutilizáveis (BOOCH, RUMBAUGH E JACOBSON, 2005).

Por fim, a UML é independente tanto de linguagem de programação quanto de processos de desenvolvimento, isto é, ela pode servir para modelar qualquer sistema, não importando a linguagem de programação que será utilizada, nem o processo de desenvolvimento adotado pelos desenvolvedores.

2.3 Diagramas da UML

Assim como é possível projetar uma casa e visualizar como ela vai ser antes mesmo do início de sua construção, os diagramas da UML tem a mesma função. Esses diagramas demonstram os requisitos de usuário e como os processos serão realizados para atender à esses requisitos, o tipo de resposta que o sistema retorna ao usuário e quais os possíveis erros que podem ocorrer em seu tempo de execução, além dos modelos de projeto, tanto de código como de banco de dados.

Cada diagrama da UML analisa o sistema, ou parte dele, sob uma determinada ótica. É como se o sistema fosse modelado em camadas, sendo que alguns diagramas enfocam o sistema de forma mais geral, apresentando uma visão externa do sistema, como é o objetivo do Diagrama de Casos de Uso, enquanto outros fornecem uma visão de uma camada mais profunda do

software, apresentando um enfoque mais técnico ou ainda visualizando apenas uma característica específica do sistema ou um determinado processo (GUEDES, 2008). A Figura 1 apresenta as perspectivas de um sistema.

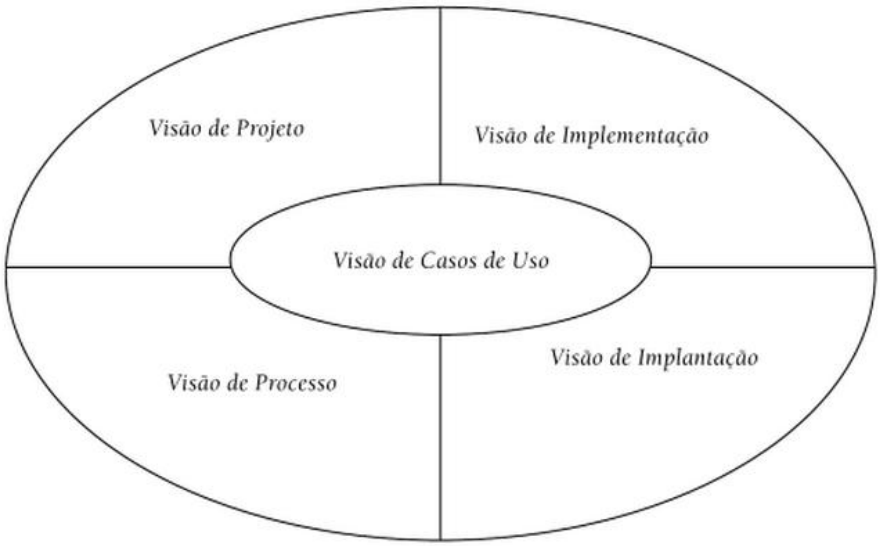

Figura 1 - **Visões (perspectivas) de um sistema de software**
Fonte: Bezerra (2007)

Os diagramas podem ser divididos em duas categorias, os diagramas estruturais e os diagramas comportamentais. Cada categoria documenta uma parte específica do sistema. Os diagramas estruturais são utilizados para visualizar, especificar, construir e documentar os aspectos estáticos do sistema, abrangem também a existência de itens como classes, interfaces, colaborações, componentes e nós. (BOOCH, RUMBAUGH E JACOBSON, 2005).

Os diagramas comportamentais são utilizados para especificar e documentar os processos dinâmicos do sistema. A seguir, a Figura 2 apresenta os diagramas definidos pela UML.

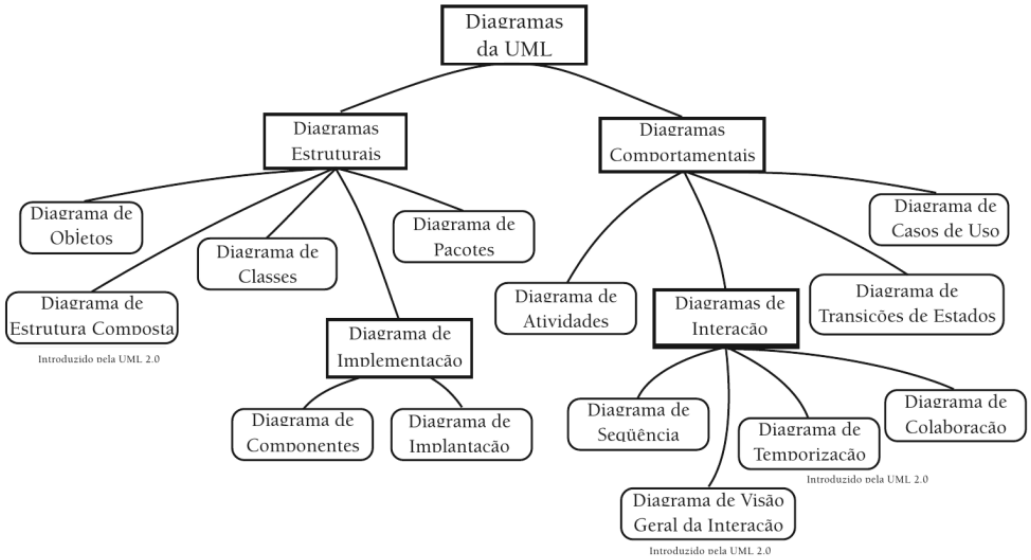

Figura 2 - Diagramas definidos pela UML

Fonte: Bezerra (2007)

2.4 Processos de Desenvolvimento de Software

O desenvolvimento de software é uma tarefa difícil, complexa e que devido a sua complexidade leva um considerável tempo para ser concluído. Esta complexidade está ligada ao fato de que sistemas são a compilação de diversos processos corriqueiros ou não de uma empresa ou negócio, o qual o sistema se propõe a servir. Formas e tentativas de se lidar e contornar essa complexidade, reduzir os problemas e conflitos são parte da definição de processos de desenvolvimento de software.

Segundo Bezerra (2007), um processo de desenvolvimento de software compreende todas as atividades necessárias para definir, desenvolver, testar e manter um produto de software.

Os principais objetivos de um processo de desenvolvimento é definir as etapas e as atividades que serão executadas em cada etapa, em qual momento e como a atividade será executada, promovendo assim uma melhor gestão e um padrão no desenvolvimento do software.

As etapas às quais um processo de desenvolvimento de software é submetido são variadas. Ao longo dos anos diversos modelos foram propostos, cada um com seus pontos fortes e fracos, mas há consenso entre a comunidade de desenvolvimento de software, que não existe um modelo que seja possível aplicar em todas as situações. No entanto há algumas etapas que são comuns entre grande parte dos modelos propostos. Os processos comuns entre os modelos são (BEZERRA, 2007):

- Levantamento de requisitos
- Análise
- Projeto
- Implementação
- Teste
- Implantação

2.4.1 Levantamento de requisitos

O levantamento de requisitos, ou elicitação de requisitos, é a etapa inicial em que os analistas do sistema começam a ver qual é o objetivo do sistema e quais os seus requisitos para que ele atenda às necessidades dos usuários que o irão utilizar. Geralmente o analista junto com o usuário que utilizará o sistema, tem uma breve conversa visando compreender quais as reais necessidades do usuário e de que forma ela deve ser resolvida. Existem outras técnicas de levantamento de requisitos, tais como questionários ou a inserção e observação do ambiente de trabalho, mas a técnica mais utilizada para realizar o levantamento de requisitos é a entrevista com o usuário.

Segundo Maciaszek (2001), um requisito é uma condição ou capacidade que deve ser alcançada ou possuída por um sistema ou componente deste para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos.

Os requisitos de um sistema são descrições dos serviços fornecidos pelo sistema e as suas restrições operacionais (SOMMERVILLE, 2007).

O analista tenta levantar o máximo de requisitos possíveis nessa conversa para poder sanar todas as necessidades levantadas pelo cliente, afim de não precisar fazer futuras alterações nos requisitos, tornando assim o desenvolvimento do sistema mais rápido e barato. Cada requisito levantado é na verdade uma funcionalidade e suas restrições de como deverá ser essa funcionalidade.

O resultado do levantamento de requisitos é o documento de requisitos, que contém todos os requisitos identificados pelo analista, mas de forma alguma ele deve informar soluções de desenvolvimento do sistema. O foco principal do documento de requisitos é especificar da forma mais clara possível as necessidades do usuário final antes de iniciar o desenvolvimento, permitindo a rastreabilidade dos requisitos, pois caso haja a necessidade de alterar algum requisito ou implementar outro, que se possa ver qual o seu impacto sobre os demais requisitos.

Os requisitos são definidos em três categorias (BEZERRA, 2007):

- Requisitos funcionais
- Requisitos não funcionais
- Requisitos normativos

2.4.1.1 Requisitos funcionais

Os requisitos funcionais descrevem basicamente o que o sistema deve fazer, contendo a descrição de suas funcionalidades, a origem proveniente da chamada da função e quem irá atendê-la, os dados que serão enviados à controladora do sistema.

Cada requisito corresponde a uma função específica para atender a demanda do sistema, seja ela uma função de entrada para dar persistência aos dados informados ao sistema, ou uma função de saída como, por exemplo, um relatório sobre dados armazenados em sua base de dados.

Segundo Sommerville (2007), requisitos funcionais são as declarações de serviços que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como o sistema deve se comportar em determinadas situações.

Os requisitos funcionais podem ser classificados de duas maneiras (BEZERRA, 2007):

- As funcionalidades podem ser ocultas, que são funções que executam de forma autônoma sem a interação com o usuário e sem que o mesmo tenha conhecimento sobre elas;
- As funcionalidades evidentes, que são aquelas em que só serão executadas quando houver uma ação do usuário, como por exemplo o clique de um mouse sobre um botão.

2.4.1.2 Requisitos não-funcionais

Os requisitos não-funcionais estão relacionados com as características referentes às funcionalidades. Cada requisito não-funcional está diretamente ligado a um requisito funcional, definindo suas restrições lógicas, suas regras de negócio e restrições tecnológicas.

Segundo Sommerville (2007), os requisitos não-funcionais são aqueles não diretamente relacionados às funções específicas fornecidas pelo sistema. Eles podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, tempo de resposta e espaço de armazenamento. Como alternativa, eles podem definir restrições, como a capacidade dos dispositivos de E/S (entrada e saída) e as representações de dados usadas nas interfaces do sistema.

Os requisitos não-funcionais podem ser classificados segundo algumas categorias, tais como (BEZERRA, 2007):

- Confiabilidade: requisitos que correspondem ao poder do sistema de evitar, tratar e se recuperar de falhas

- Desempenho: requisitos que correspondem ao poder de resposta do sistema ao usuário
- Portabilidade: requisitos que correspondem a compatibilidade do sistema com o hardware ou demais sistemas
- Segurança: requisitos que correspondem a segurança dos dados trafegados no sistema e de acessos não autorizados
- Usabilidade: requisitos que correspondem a facilidade e utilidade do sistema

A Figura 3 apresenta como as categorias dos requisitos não-funcionais se dividem.

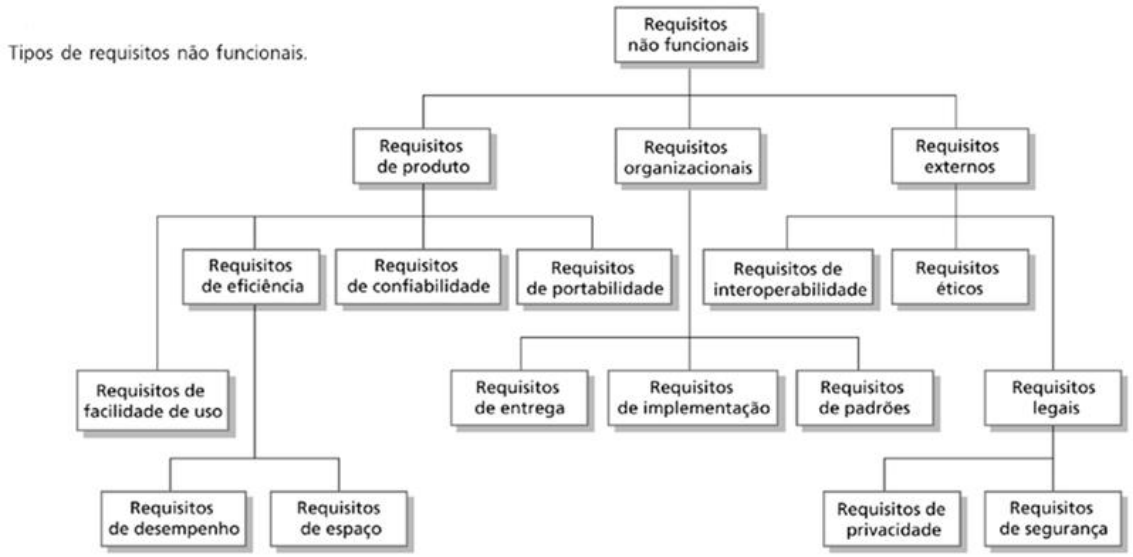

Figura 3 - Tipos de requisitos não funcionais

Fonte: Sommerville (2007)

Ainda tratando das características dos requisitos não-funcionais, eles podem também ser classificados como essenciais ou desejáveis. Os classificados como essenciais são aqueles requisitos indispensáveis para que o sistema funcione, tais como as regras do negócio. Os classificados como desejáveis são aqueles o qual o sistema atenderia as necessidades do usuário mas que se tivessem sido incrementados, melhorariam o desempenho e eficácia do sistema.

Ainda em relação aos requisitos não-funcionais, Waslawick (2004) afirma que existem aqueles diretamente associados a uma função e outros que são gerais para o sistema, os quais são denominados suplementares. Normalmente será útil ter dois quadros: um que relacione os requisitos funcionais e suas restrições associadas (Quadro 1), e outro que relacione os requisitos suplementares (Quadro 2).

F1 Registrar empréstimos		**Oculto ( )**		
Descrição: O sistema deve registrar empréstimos de fitas, indicando o cliente e as fitas que foram emprestadas, bem como a data do empréstimo e o valor previsto para pagamento na devolução				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF1.1 Controle de Acesso	A função só pode ser acessada por usuário com perfil de operador ou superior	Segurança	( )	(X)
NF1.2 Identificação de Fitas	Os filmes devem ser identificadas por um código de barras.	Interface	( )	(X)
NF1.3 Tempo de Registro	O tempo para registro de cada fita deve ser inferior a um segundo.	Performance	(X)	( )

Quadro 1 - Quadro para especificação de requisitos funcionais e não-funcionais
Fonte: WAZLAWICK (2004)

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
S1 Tipo de Interface	As interfaces do sistema devem ser implementadas como formulários acessíveis em um browser HTML.	Interface	( )	( )
S2 Armazenamento de Dados	A camada de persistência deve ser implementada de forma que diferentes tecnologias de banco de dados possam vir a ser utilizadas no futuro.	Persistência	( )	(X)
S3 Perfis de Usuário	Os perfis de usuário para acesso ao sistema são: 3. Administrador – pode efetuar todas as operações. 2. Operador – pode efetuar as operações de empréstimo, devolução,	Segurança	( )	( )

	pagamento e cadastramento. 1. Convidado – pode efetuar apenas consultas nos próprios dados (cliente)			

Quadro 2 - Exemplo de tabela que especifica requisitos suplementares
Fonte: WAZLAWICK (2004)

2.4.1.3 Requisitos normativos

Os requisitos normativos são aqueles que definem o desenvolvimento do sistema. São eles que definem prazo e custo do desenvolvimento assim como a plataforma e tecnologias utilizadas no desenvolvimento do sistema.

Segundo Bezerra (2007), requisitos normativos são declarações de restrições impostas sobre o desenvolvimento do sistema. Restrições definem, por exemplo, a adequação a custos e prazos, a plataforma tecnológica, aspectos legais, limitações sobre a interface com o usuário, componentes de software e hardware a serem adquiridos, eventuais necessidades de comunicação do novo sistema com sistemas legados etc.

Normalmente os requisitos normativos são definidos no documento de plano de projeto.

2.4.2 Análise e projeto

A etapa da análise corresponde ao processo de analisar os requisitos levantados pelo analista, em sua entrevista com o cliente, entender o que é o sistema e elaborar um modelo de solução para poder suprir as necessidades do cliente. O objetivo é elaborar modelos lógicos para todas as funcionalidades do sistema, a afim de obter uma visão de como o sistema irá se comportar.

Segundo Bezerra (2007), nesta atividade, o foco de interesse é tentar construir uma estratégia de solução sem se preocupar com a maneira como

essa estratégia será realizada. A razão desta prática é tentar obter a melhor solução para o problema sem se preocupar com os detalhes da tecnologia a ser utilizada.

A análise dedica-se inteiramente a propor modelos e soluções para as funcionalidades do sistema, deixando completamente de lado as tecnologias que o sistema irá utilizar, assim como a forma como elas serão utilizadas.

Os modelos obtidos nessa fase devem ser rigorosamente revisados, validando seus dados, a afim de que eles correspondam aos requisitos previamente levantados. Nessa etapa, o analista apresenta o modelo ao cliente para que o mesmo possa avaliar se aquela solução proposta atenderá as suas necessidades. Essa atividade é realizada para que não haja contradições entre o que o cliente quer e o que o analista compreende a respeito.

Essa etapa do processo de desenvolvimento de software reduz consideravelmente os riscos de que o sistema não atenda às necessidades do cliente, evitando assim maiores custos no desenvolvimento do sistema.

Ao termino dessa etapa se obtém um modelo funcional de todo o sistema a ser desenvolvido.

2.4.2.1 Análise Orientada a Objetos

A análise orientada a objetos é uma forma de modelar um sistema, ela trata todo e qualquer elemento como um objeto, o qual possui seus atributos e métodos, para que possa interagir com o restante do sistema.

Segundo Sommerville (2007), a análise orientada a objetos concentra-se no desenvolvimento de um modelo orientado a objetos do domínio da aplicação. Os objetos nesse modelo refletem as entidades e as operações associadas ao problema a ser resolvido.

O objetivo da análise orientada a objetos é formular modelos de análise das funcionalidades do sistema, descrevendo o seu comportamento e a sua estrutura.

Como a orientação a objetos trata cada elemento independentemente, caso haja a necessidade de fazer alterações na análise, o processo torna-se muito mais fácil, tendo em vista que apenas uma parte do modelo necessita ser alterado, e ela não irá interferir nas demais partes do sistema.

2.4.2.1.1 Objetos

Objetos são a abstração computacional de uma entidade tangível, seja ela física ou abstrata, identificando, formalizando os seus atributos e suas funcionalidades (BOOCH, RUMBAUGH E JACOBSON, 2005).

Segundo Sommerville (2007), um objeto é uma entidade que possui um estado e um conjunto definido de operações definidas para funcionar nesse estado. O estado é representado como um conjunto de atributos de objeto. As operações associadas ao objeto fornecem serviços a outros objetos que solicitam esses serviços quando alguma computação é necessária.

Na Figura 4, pode-se ter a perspectiva de um objeto:

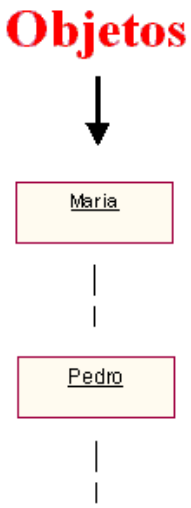

Figura 4 - Exemplo de um objeto
Fonte: Macoratti (2013)

2.4.2.1.2 Classes

Uma classe é uma abstração das características de um grupo de coisas do mundo real. É a descrição dos atributos e serviços comuns a um grupo de objetos. Sendo assim, pode-se entender uma classe como sendo um molde a partir do qual objetos são construídos. Ainda sobre terminologia, diz-se que um objeto é uma instância de uma classe (BEZERRA, 2007).

As classes de programação são projetos de um objeto, aonde têm características e comportamentos, ou seja, permite armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração. Uma classe geralmente representa um substantivo, por exemplo: uma pessoa, um lugar, algo que seja “abstrato”(PALMEIRA, 2013).

Na Figura 5, pode-se observar um modelo de classe:

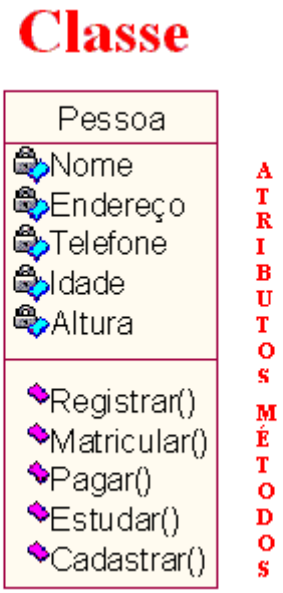

Figura 5 - Exemplo de uma classe
Fonte: Macoratti (2013)

2.4.2.1.3 Herança

A herança diz respeito à extensibilidade de classes no modelo orientado a objetos. Quando se diz estender determinada classe, entende-se que uma nova classe será criada, contendo suas próprias propriedades e características e, agregando a esta nova classe as propriedades e características de outra já existente a qual é conhecida também como uma classe Genérica (ou superclasse). Já a nova classe é conhecida como classe especializada (ou subclasse) (ARAÚJO, 2013).

A seguir, a Figura 6 apresenta um exemplo de herança:

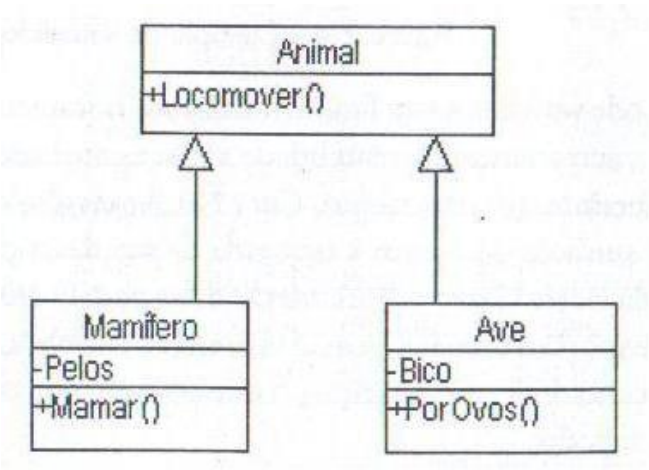

Figura 6 - Exemplo de herança
Fonte: Guedes (2008)

Em alguns casos, uma classe pode herdar características de duas ou mais superclasses.

A Figura 7 demonstra a múltipla herança:

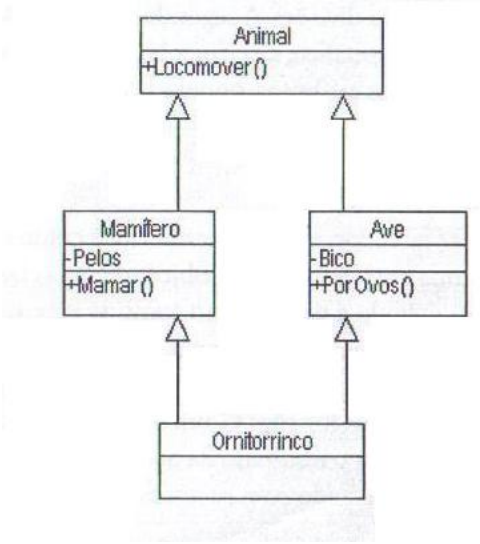

Figura 7 - Exemplo de múltiplas heranças
Fonte: Guedes (2008)

2.4.2.1.4 Encapsulamento

O mecanismo de encapsulamento é uma forma de restringir o acesso ao comportamento interno de um objeto. Um objeto que precise da colaboração de outro objeto para realizar alguma tarefa simplesmente envia uma mensagem a este último. O método que o objeto usa para realizar a tarefa não é conhecido dos objetos requisitantes.

Portanto, através do encapsulamento, a única coisa que um objeto precisa saber para pedir a colaboração de outro objeto é conhecer a sua interface (BEZERRA, 2007).

O uso da técnica do encapsulamento torna o sistema mais seguro e flexível, tendo em vista que a forma como as tarefas são executadas são de cunho pessoal de um objeto e qualquer elemento externo a ele não tem conhecimento dos métodos utilizados para executar a requerida tarefa.

2.4.2.1.5 Polimorfismo

O Polimorfismo permite, em uma de suas metodologias de aplicação, que diferentes classes tenham métodos com a mesma assinatura (mesmo contrato), porém estes métodos (em suas respectivas classes) podem possuir comportamentos diferentes, de acordo à necessidade de cada classe que o implementa.

A implementação do polimorfismo pode ser realizada fazendo uso de interfaces, ou classes abstratas, em que ocorrem apenas a implementação das assinaturas dos métodos, ou seja, do contrato. Desta forma o comportamento deve ser implementado nas classes concretas que implementam as interfaces ou estendem as classes abstratas (ARAÚJO, 2013).

Resumindo, a técnica do polimorfismo permite autonomia à classe filha, podendo ela sobre escrever métodos herdados da classe pai ou mesmo instanciar novos métodos que ela necessite. Figura 8 ilustra um exemplo de polimorfismo.

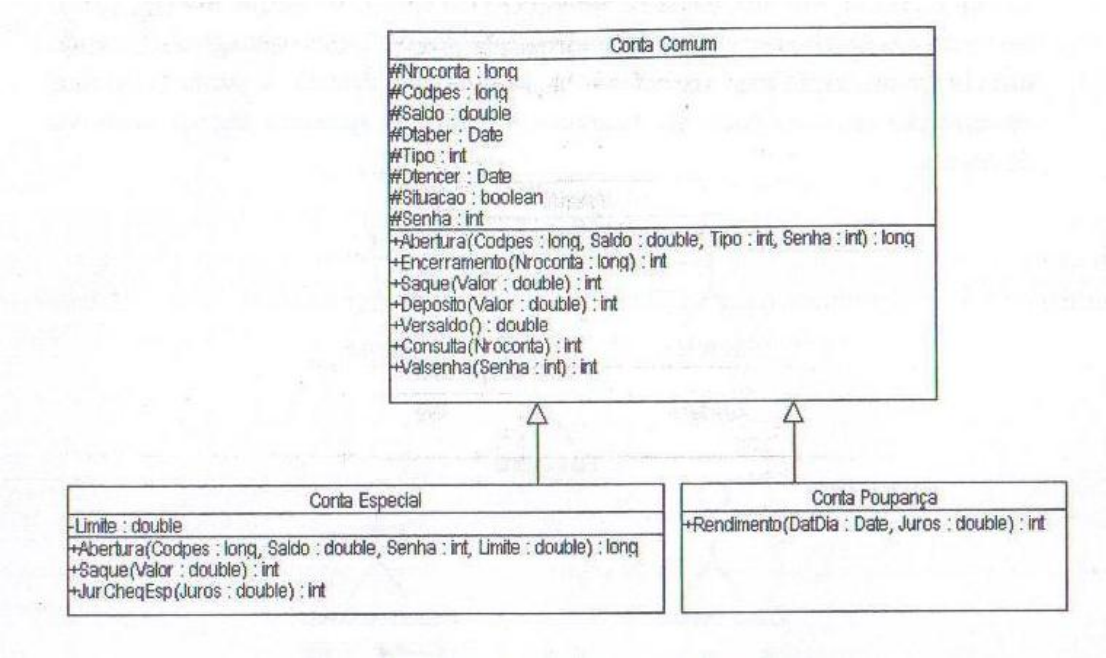

Figura 8 - Exemplo de polimorfismo
Fonte: Guedes (2008)

A Figura 8 demonstra duas classes distintas que herdaram de uma classe pai seus métodos e atributos, mas com o emprego do polimorfismo tornaram-se independentes, tendo ambas criado novos métodos e em particular, a classe Conta Especial sobre escreve o método herdado *Abertura*, ajustando-se a sua necessidade.

2.5 Sistemas de Gestão

Um sistema de gestão, nada mais é que um sistema de informação voltado ao gerenciamento de processos e procedimentos de uma determinada instituição ou pessoa ao qual ele é destinado. É importante que as instituições, sejam ela comerciais, industriais ou pessoais, manter controle sobre seus eventos, para poder ter uma melhor visão de como seus processos estão acontecendo, como torná-los melhores, como reduzir seu custo e como aumentar sua lucratividade.

Segundo Matievicz (2013), a necessidade de modernizar e ter as informações da empresa na ponta dos dedos para impulsioná-la no mercado vai muito além de apenas controlar o estoque ou o contas a pagar/receber. Informações relacionadas aos custos, produção, evolução financeira, orçamentos tanto quanto as questões tributárias e de análises garantem aos gestores perceber a importância da contabilidade gerencial no dia-a-dia e no processo de tomada de decisões, tudo isso aliado à tecnologia de um bom sistema de informação.

Enfim, pode-se afirmar que a utilização de um sistema de informação gerencial é uma das mais importantes ferramentas para o processo decisório e de gestão nas empresas, uma vez que conhecer as informações que envolvem o dia-a-dia da empresa permite ao gestor planejar e projetar o futuro almejado sem perder o controle da organização.

3 MATERIAIS E MÉTODO

Este capítulo, trata os materiais e ferramentas utilizadas na modelagem do sistema de gestão de bares, bem como o método adotado para o desenvolvimento do trabalho.

3.1 Materiais

O Quadro 3 apresenta as ferramentas e as tecnologias que foram utilizadas para modelar e implementar o sistema.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
UML	2.4.1	http://www.uml.org/	Tecnologia de modelagem e arquitetura de software
Case Studio 2	2.25 (Demo)	http://www.casestudio.com	Modelagem do modelo relacional do banco de dados.
Visual Paradigm	8.1 Community Edition	http://www.visual-paradigm.com/	Modelagem dos diagramas de atividades, seqüência, casos de uso, classes e modelo conceitual do sistema.
Eclipse Juno	4.3.2 Standard	https://www.eclipse.org/	Desenvolvimento de aplicativos
JDK	7.3	http://www.oracle.com/technetwork/java/javase/downloads/index.html	Kit de Desenvolvimento Java
ADT Plugin	22.2	http://developer.android.com/tools/sdk/eclipse-adt.html	Plugin para a construção de aplicativos
Android SDK	21.1	http://developer.android.com/sdk/index.html	Kit de Desenvolvimento Android

Netbeans IDE	7.3.1	https://netbeans.org/	Desenvolvimento de aplicações
Apache Tomcat	7.0.3	http://tomcat.apache.org/	Servidor Web Java
MySQL	5.6.22	http://dev.mysql.com/	Sistema de Gerenciamento de Banco de Dados (SGBD)
Jaspersoft Studio	5.6.2 Final	http://community.jaspersoft.com/	Criação dos relatórios

Quadro 3 - Ferramentas e tecnologias utilizadas

3.2 Método

Esse tópico apresenta o ciclo de vida que será utilizado no desenvolvimento do sistema proposto. O ciclo de vida utilizado é o iterativo e incremental.

A abordagem do desenvolvimento interativo e incremental divide o desenvolvimento do sistema em ciclos, em que cada ciclo, apenas um conjunto de requisitos do sistema é trabalhado. Em cada ciclo do desenvolvimento do sistema, um novo conjunto de requisitos é submetido às fases padrão do desenvolvimento de software. As etapas são: levantamento de requisitos, análise e projeto, implementação e testes.

Desta forma o sistema vai tomando forma através de versões geradas a cada iteração, sendo que a cada ciclo, o sistema ganha novas funcionalidades e isso irá prosseguir até que o sistema esteja completo.

A Figura 9 demonstra o ciclo de vida Interativo e Incremental:

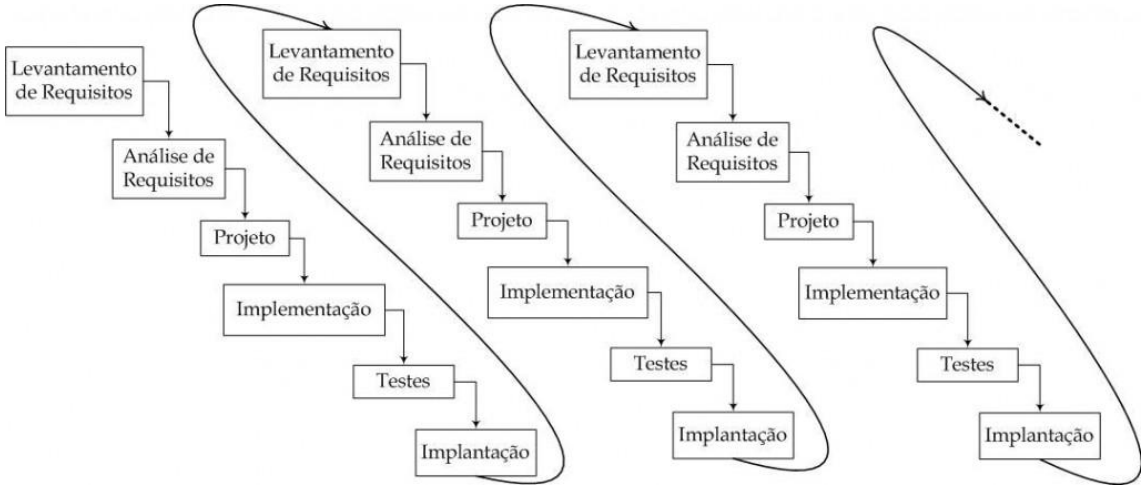

Figura 9 - Ciclo de vida Interativo e Incremental
Fonte: Bezerra (2007)

Segundo Bezerra (2007), a abordagem incremental e interativa somente é possível se existir um mecanismo para dividir os requisitos do sistema em partes, para que cada parte seja alocada a um ciclo de desenvolvimento. Essa alocação é realizada em função do grau de importância atribuída a cada requisito. Os fatores considerados no particionamento são a prioridade e o risco de cada requisito. É função do gerente de projeto alocar os requisitos aos ciclos de desenvolvimento.

O Quadro 4 apresenta os processos e as interações desenvolvidas:

Iterações / Processos	1ª iteração	2ª iteração	3ª iteração	4ª Interação
Requisitos	Definição dos requisitos funcionais e não funcionais. Modelagem de negócios para o sistema.	Definição dos requisitos Complementares. Modelagem Final.	Revisão dos requisitos.	
Análise e projeto	Modelagem das classes e tabelas.	Revisão da modelagem das classes e tabelas. Ajustes de campos e	Ajuste do modelo de entidade e relacionamento	

		atributos.		
Implementação	Estudo das tecnologias e testes de implementação visando identificar a melhor forma de fazê-lo.	Desenvolvimento do módulo *Web* do sistema.	Desenvolvimento do módulo *mobile* e da integração do sistema.	Desenvolvimento dos relatórios e complementações necessárias nos módulos *Web* e *mobile*.
Testes		Testes de interface e das funcionalidades do sistema desenvolvidas para o módulo *Web*.	Testes do módulo *mobile*.	Teste de todo o sistema.

Quadro 4 - Interações definidas

As etapas informadas no Quadro 4, que definem o processo de desenvolvimento do sistema e as atividades realizadas em cada ciclo, são descritas a seguir:

a) Levantamento de Requisitos

O levantamento de requisitos iniciou com a pesquisa das atividades corriqueiras de um bar através da Internet e de conversas com alguns proprietários de estabelecimento do tipo proposto. Após a elaboração prévia de um escopo do sistema e em conversa com professores da área, foram descobertas novas funcionalidade para o sistema, sendo escopo concluído.

A partir do escopo do sistema foram extraídos os requisitos do sistema, os quais foram classificados em funcionais e não funcionais. Algumas alterações foram realizadas para melhor se adequar ao sistema e manter a unicidade e integridade do sistema.

Com base nos requisitos levantados foi realizada a modelagem de negócios para o sistema.

b) Análise e Projeto

Com base nos requisitos levantados, foram definidos os casos de uso do sistema. Esses casos de uso foram documentados gerando informações para a definição do relacionamento dos objetos e classes reconhecidos.

Com base nas classes de entidade identificadas foram definidos os seus campos, como tipo e tamanho do dado. Com isso foi elaborado o diagrama entidade relacionamento para o sistema.

c) Implementação

Com base no diagrama de entidade relacionamento gerado na etapa anterior, foi possível criar o banco de dados, fazendo uso do MySQL. A partir do banco de dados criado, foi possível dar inicio ao módulo Web do sistema, a implementação foi realizada utilizando a ferramenta Netbeans para o desenvolvimento do módulo *Web* do sistema e para o servidor de serviços .

Para a implementação do módulo *mobile* do sistema, foi feito uso do Eclipse Juno com o complemento do *Android* SDK e do ADT *Plugin*.

Por fim, para a criação dos relatórios do sistema, foi feito uso do Jaspersoft Studio.

d) Testes

Os testes foram sendo realizados na medida em que as funcionalidades do sistema iam sendo implementadas, todos os testes foram informais e realizados pelo próprio autor deste trabalho sem nenhum plano de testes previamente elaborado. Os testes consistiam em verificar o código e testar se a funcionalidade estava de acordo e não apresentava erros ao usuário.

Ao final da implementação foi realizado mais testes em todo o sistema, visando buscar falhas na implementação e corrigi-las, tornando o sistema totalmente funcional aos futuros usuários.

4 RESULTADOS

Esse capítulo tem como objetivo apresentar os resultados das etapas de levantamento de requisitos, análise, projeto e implementação do sistema. Inicialmente é apresentada a visão do sistema, em que é possível se ter uma real noção de quais funções o sistema contém, posteriormente é apresentada a modelagem do sistema implementado, contendo os requisitos funcionais e não funcionais identificados assim como a modelagem com diagramas baseados na UML. Após os diagramas, é apresentado o sistema e trechos de códigos utilizados na implementação.

4.1 Escopo do Sistema

O sistema recebe pedidos dos clientes de um bar, estes realizados pelos garçons, armazenando a mesa em que o mesmo se encontra, os itens e a quantidade solicitados.

O garçom realiza o pedido via um aparelho *smartphone*, devendo fazer a conexão com o servidor remotamente via rede sem fio. A cada inclusão deve se mostrar uma mensagem de confirmação.

Para poder adicionar um pedido o garçom deve verificar se a mesa está iniciada, ou se está zerada, caso esteja iniciada basta fazer a adição do pedido, caso contrário o garçom deve iniciar a mesa. O sistema só aceita pedidos de mesas em aberto.

O sistema é capaz de incluir produtos, assim como novos usuários apenas em uma estação de mesa. O controle de caixa também é realizado na estação de mesa.

O pagamento da conta pode ser a vista em dinheiro ou no cartão de débito ou ainda no prazo pelo cartão de crédito, a opção é informada no ato do pagamento. Caso seja no cartão de crédito, o sistema salva a venda com seus dados e gerar uma conta a receber. Caso o pagamento seja a vista ou cartão

de débito, o sistema adiciona ao caixa o valor referente à venda. Ao encerrar o caixa o sistema informa ao operador o saldo disponível.

Para que atenda a demanda dos garçons, é utilizada a tecnologia *mobile*, já para os demais usuários do sistema é utilizada a tecnologia *Web*.

4.2 Modelagem do sistema

As Figuras 10 e 11 apresentam diagramas de atividade dando uma visão geral do processo Efetuar Venda e Finalizar Venda de um bar, conforme considerado para o sistema proposto. Os requisitos para o sistema foram obtidos a partir do conhecimento próprio do acadêmico sobre o estabelecimento que pretende informatizar.

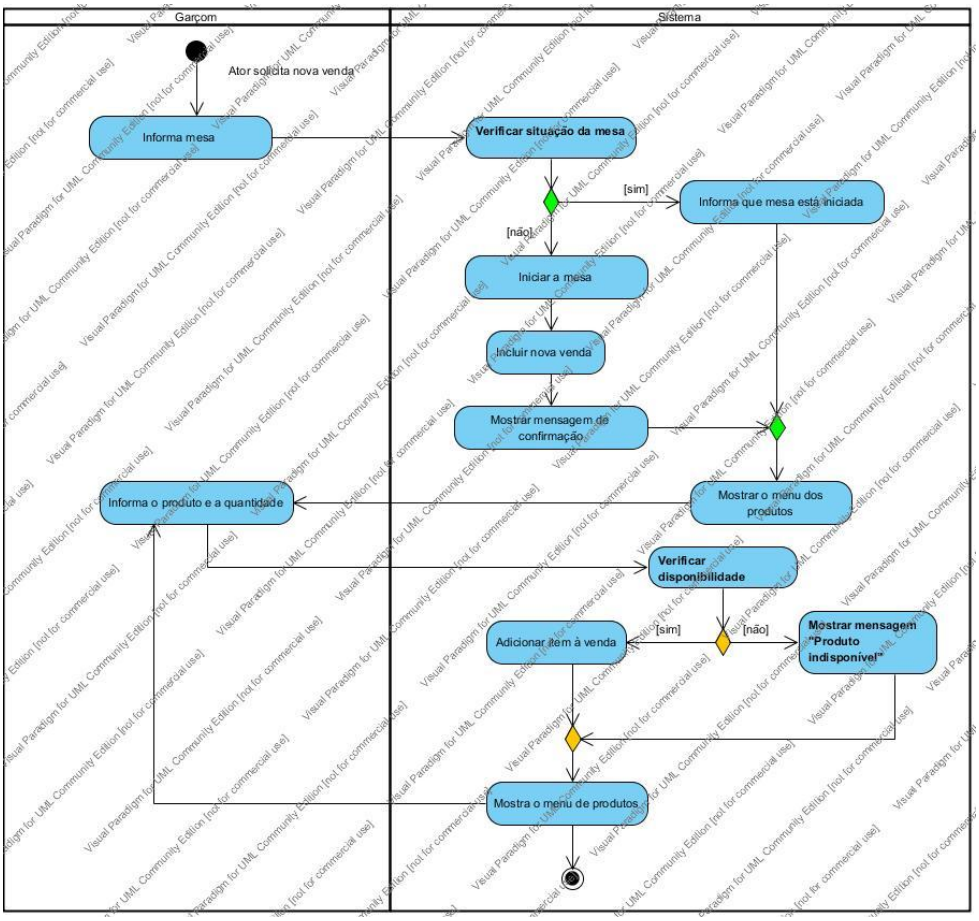

Figura 10 - Processo Efetuar Venda

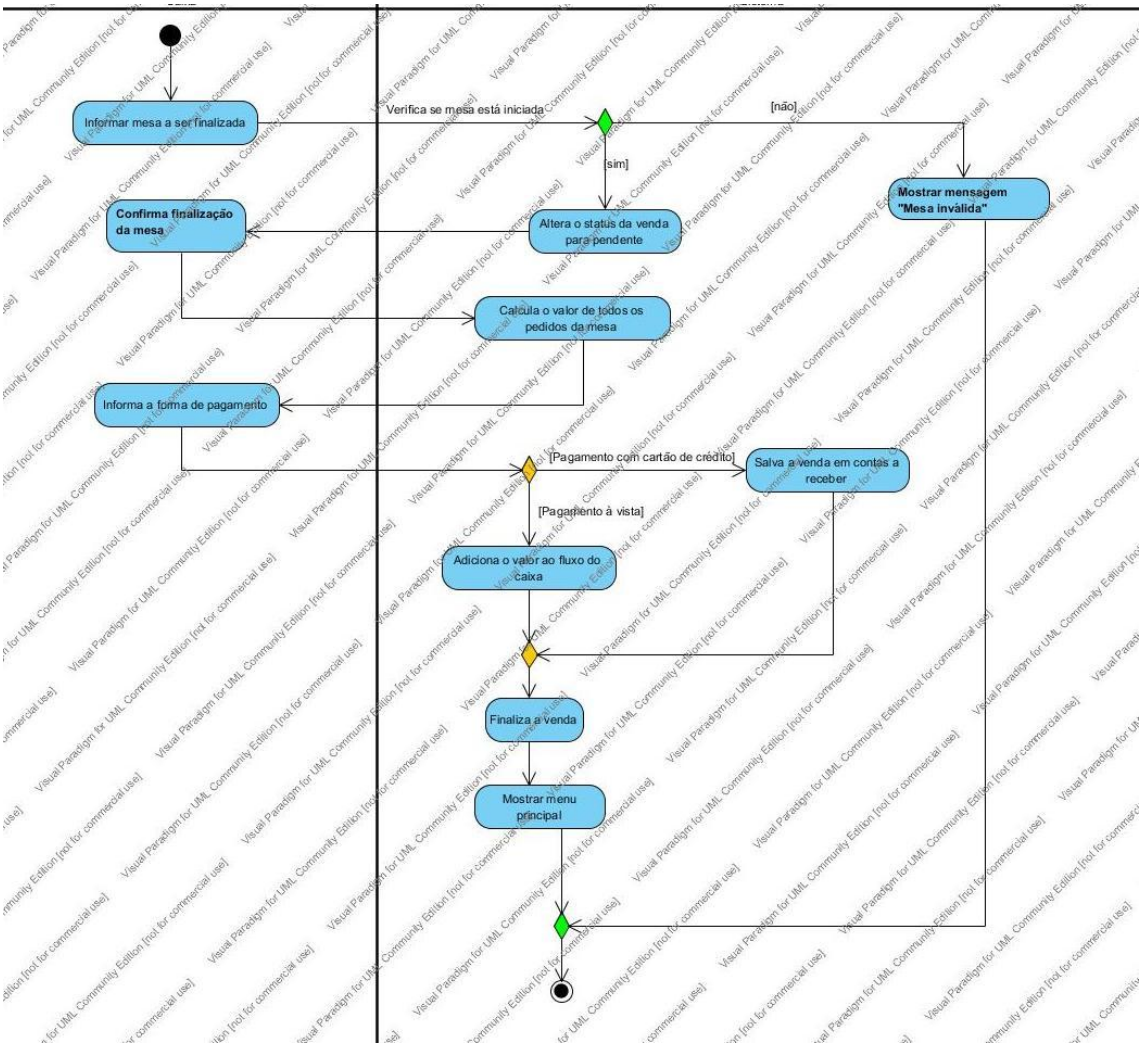

Figura 11 - Processo Finalizar Venda

Com base na identificação dos requisitos de usuário e visão geral do sistema apresentada anteriormente, foram identificados os requisitos funcionais e não funcionais para o sistema proposto, conforme os Quadros 5 a 14.

F1 Cadastrar funcionário	**Oculto ( )**
Descrição: O sistema é capaz de aceitar o registro de novos funcionários. Armazenando em seu cadastro pelo menos um código, nome, CPF, endereço, bairro, cidade, CEP, telefone, data de nascimento, sexo, cargo, e-mail.	
Requisitos Não-Funcionais	

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF1.1 Controle de acesso	A função de cadastro de funcionário só pode ser acessado por usuários do tipo gerente ou administrador.	Segurança	( )	(x)
NF1.2 Controle de função	O funcionário é classificado como ativo ou inativo, permitindo ações por parte de funcionários ativos.	Segurança	( )	(x)
NF1.3 Janela Única	Todas as funções relacionadas a registros e alterações são efetuadas em uma única janela.	Interface	(X)	( )
NF1.4 Acesso	Esse requisito do Sistema só é disponível para a parte Web do sistema.	Usabilidade	( )	(x)

Quadro 5 - Requisitos Funcionais - Cadastrar funcionário

F2 Cadastrar produto	**Oculto ( )**
Descrição: O sistema aceita o registro de novos produtos.	
Requisitos Não-Funcionais	

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF2.1 Controle de acesso	O formulário de cadastro de produtos pode ser acessado apenas por pessoas do tipo gerente ou administrador.	Segurança	( )	(x)
NF2.2 Unicidade dos dados	O Sistema não permite que um mesmo produto seja cadastrado duas vezes.	Segurança	( )	(x)
NF2.3 Janela Única	Todas as funções relacionadas a registros e alterações são efetuadas em uma única janela.	Interface	(X)	( )
NF2.4 Integridade dos dados	Alteração de dados só podem ser alterados por um usuário do tipo administrador.	Segurança	(X)	( )
NF2.5 Acesso	Esse requisito do Sistema só estará disponível para a parte Web do sistema	Usabilidade	( )	(x)

Quadro 6 - Requisitos Funcionais - Cadastrar produto

F3 Cadastrar venda	**Oculto ( )**
Descrição: O sistema receber pedido dos clientes, estes realizados por um funcionário, armazenando a mesa em que o mesmo se encontra, os itens e a quantidade solicitados. Junto ao pedido do cliente o sistema armazena o funcionário responsável pelo pedido.	
Requisitos Não-Funcionais	

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**

NF3.1 Controle de acesso	Para poder cadastrar uma venda, o funcionário deve estar logado com uma conta ativa.	Segurança	( )	(x)
NF3.2 Verificação da mesa	O Sistema verifica se a mesa indicada pelo funcionário está zerada, caso não esteja ela deve mostrar o menu para adicionar um novo pedido, caso contrário, ela deve emitir uma mensagem se o usuário quer iniciar a mesa.	Segurança	( )	(x)
NF3.3 Adicionar produto	Ao clicar em adicionar produto, um menu deve aparecer, mostrando os produtos. O produto selecionado deverá ser instanciado no novo item de venda. Também deverá ser indicada a quantidade do produto. Uma mensagem de aviso deve ser informada ao usuário se a operação obteve êxito ou não. Retornando ao menu principal ao término da operação.	Interface	(X)	( )
N3.4 Alteração de dados	Alteração de dados da venda só poderão ser realizados por um usuário do tipo gerente ou superior a partir de um estação.	Segurança	(X)	( )
NF3.5 Disponibilidade da função	Esse requisito do Sistema estará disponível para ambas as plataformas utilizadas.	Interface	( )	(x)

Quadro 7 - Requisitos Funcionais - Cadastrar venda

F4 Finalizar venda	**Oculto ( )**			
Descrição: O sistema deve finalizar a conta de uma mesa, calculando o valor final com base nos itens do pedido. O sistema deve fornecer formas de pagamento diferenciadas, como cartão de crédito, cartão de débito e pagamento em espécies.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF4.1 Controle de acesso	Para poder finalizar uma venda, o nível de usuário deverá ser gerente ou atendente de caixa com o privilégio para realizar a função.	Segurança	( )	(x)
NF4.2 Alteração de dados	Os dados da venda, não podem ser alterados após ela ser finalizada.	Segurança	(X)	( )
NF4.3 Disponibilidade da função	Esse requisito do Sistema estará disponível apenas na parte Web do sistema.	Interface	( )	(X)

Quadro 8 - Requisitos Funcionais - Finalizar venda

F5 Emitir relatório de vendas	**Oculto ( )**			
Descrição: O sistema deve fornecer ao usuário a opção de emissão de vendas por período.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**

NF5.1 Controle de acesso	Para poder emitir um relatório, o nível do usuário deve ser gerente ou administrador.	Segurança	( )	(x)
NF5.2 Filtros	O sistema deve dar a opção do usuário de utilizar um ou mais filtros na busca.	Regra de negócio	( )	(x)
NF5.3 Armazenamento e impressão	O sistema deve permitir que o usuário possa optar em fazer a impressão do relatório gerado ou mesmo salvá-lo em disco.	Segurança	(X)	( )
NF5.4 Disponibilidade da função	Esse requisito do sistema estará disponível apenas na parte Web do sistema.	Interface	( )	(X)

Quadro 9 - Requisitos Funcionais - Emitir relatório

F6 Operar Caixa	**Oculto ( )**			
Descrição: O sistema deve fornecer ao usuário a opção de operar o caixa, realizando o recebimento das vendas realizadas				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF6.1 Contabilidade do caixa	Caso o pagamento seja efetuado à vista, o caixa deve ser alterado, adicionando o valor total da venda. Caso seja pagamento no cartão, o sistema deve finalizar a venda e fazer um lançamento no contas a receber. Ao salvar os dados, o sistema deve informar que a venda foi finalizada.	Regra de negócio / Interface	( )	(x)
NF6.2 Controle de acesso	Para poder operar o caixa, o usuário deve possuir a permissão para poder realizar a ação ou possuir nível administrador.	Segurança	( )	(x)
NF6.3 Identificação	O movimento deve ser identificado de modo sequencial pelo sistemas, armazenando data, hora, valor e funcionário responsável.	Segurança	( )	(x)
NF6.4 Disponibilidade da função	Esse requisito do Sistema estará disponível apenas na parte Web do sistema.	Interface	( )	(X)

Quadro 10 - Requisitos Funcionais – Operar caixa

F7 Emitir relatório de caixa	**Oculto ( )**			
Descrição: O sistema deve fornecer ao usuário a opção de emitir relatórios de caixa por período de tempo.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF7.1 Controle de acesso	Para poder operar o caixa, o usuário deve possuir a permissão para poder realizar a ação ou possuir nível administrador.	Segurança	( )	(x)

NF7.2 Identificação	O movimento deve ser identificado de modo seqüencial pelo sistemas, armazenando data, hora, valor e funcionário responsável.	Segurança	( )	(x)
NF7.3 Disponibilidade da função	Esse requisito do Sistema estará disponível apenas na parte Web do sistema.	Interface	( )	(X)

Quadro 11 - Requisitos Funcionais – Emitir relatório de caixa

F8 Cadastrar categoria		**Oculto ( )**		
Descrição: Permite ao usuário cadastrar categorias de produtos.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF8.1 Controle de acesso	O formulário de cadastro de categorias pode ser acessado apenas por pessoas do tipo gerente ou administrador.	Segurança	( )	(x)
NF8.2 Unicidade dos dados	O Sistema não deve permitir que um mesmo produto seja cadastrado duas vezes.	Segurança	( )	(x)
NF8.3 Janela Única	Todas as funções relacionadas a registros e alterações devem ser efetuadas em uma única janela.	Interface	(X)	( )
NF8.4 Integridade dos dados	Alteração de dados só podem ser alterados por um usuário do tipo administrador.	Segurança	(X)	( )
NF8.5 Acesso	Esse requisito do Sistema só estará disponível para a parte Web do sistema	Usabilidade	( )	(x)

Quadro 12 - Requisitos Funcionais – Cadastrar categoria

F9 Cadastrar mesas		**Oculto ( )**		
Descrição: Permite ao usuário cadastrar as mesas disponíveis no bar.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF9.1 Controle de acesso	O formulário de cadastro de mesas pode ser acessado apenas por pessoas do tipo gerente ou administrador.	Segurança	( )	(x)
NF9.2 Unicidade dos dados	O sistema não deve permitir que um mesmo produto seja cadastrado duas vezes.	Segurança	( )	(x)
NF9.3 Janela Única	Todas as funções relacionadas a registros e alterações devem ser efetuadas em uma única janela.	Interface	(X)	( )
NF9.4 Integridade dos dados	Alteração de dados só podem ser alterados por um usuário do tipo administrador.	Segurança	(X)	( )
NF9.5 Acesso	Esse requisito do Sistema só estará disponível para a parte Web do sistema	Usabilidade	( )	(x)

Quadro 13 - Requisitos Funcionais – Cadastrar mesas

F10 Adicionar item de produto	**Oculto ( )**
Descrição: O sistema deverá permitir ao garçom a adição de itens de produtos e suas quantidades às vendas cadastradas. Ao clicar em adicionar produto, um menu deve aparecer, mostrando os produtos. O produto selecionado deverá ser instanciado no novo item de venda. Também deverá ser indicada a quantidade do produto	

Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF10.1 Controle de acesso	Para poder cadastrar uma venda, o funcionário deverá estar logado com uma conta ativa.	Segurança	( )	(x)
NF10.2 Status da mesa	Somente será possível adicionar itens de produtos às vendas que estiverem com status aberta.	Segurança	( )	(x)
NF10.3 Mensagem de retorno	Uma mensagem de aviso deve ser informada ao usuário se a operação obteve êxito ou não. Retornando ao menu principal ao término da operação.	Interface	(X)	( )
NF10.5 Disponibilidade da função	Esse requisito do sistema estará disponível para ambas as plataformas utilizadas.	Interface	( )	(x)

Quadro 14 - Requisitos Funcionais – Adicionar item de produto

Os requisitos suplementares são aqueles que se aplicam ao sistema todo, não somente a um requisito funcional. Sendo assim, o Quadro 15 apresenta os requisitos suplementares para o sistema.

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
S1	O sistema deve possuir uma interface leve e simplificada, atendendo as regras de usabilidade propostas pelo estudo da interação homem-máquina.	Usabilidade	( )	(x)
S2	O sistema como trabalha de forma distribuída, deve tratar exceções para garantir a integridade do banco de dados.	Confiabilidade	(x)	( )

Quadro 14 - Requisitos Suplementares

A Figura 12 apresenta o modelo de casos de uso para o sistema proposto, o qual apresenta as funcionalidades completas, ou seja, aquelas que além de possuir algum tipo de interação com um ator, podem ser realizadas de forma isolada.

O ator Administrador pode realizar todas as funcionalidades do

sistema, já o ator gerente não tem privilégios para manipular a classe funcionários. O ator Caixa fica limitado as funcionalidades do caixa no módulo *Web* e o ator garçom tem permissão para uso das funcionalidades do módulo *mobile*, embora consiga logar no sistema no módulo *Web*, o garçom não possui privilégios neste módulo.

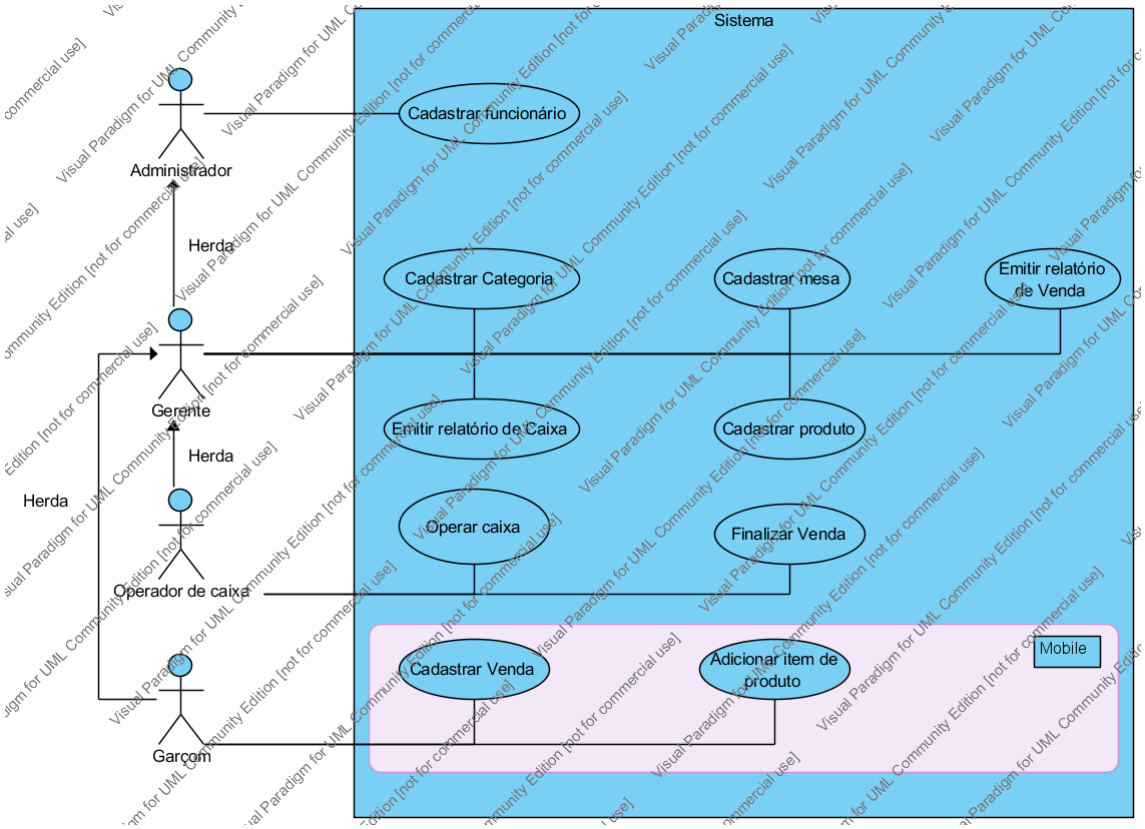

Figura 12 - Diagrama de casos de uso

A expansão de casos de uso tem como finalidade especificar os processos de um sistema de forma a facilitar a descoberta dos objetos envolvidos em cada processo como também servir de base para testes do sistema.

O Quadro 16 apresenta a expansão do caso de uso Cadastrar venda.

Caso de Uso: Cadastrar venda
Atores: Garçom, Gerente, Administrador
Precondições: Ator logado no sistema. Produtos cadastrados em estoque.
Pós-condições: Venda realizada com sucesso. Venda cancelada.
Sequência típica de eventos (Fluxo Principal): Este caso de uso inicia quando o ator solicita o cadastro de uma nova venda no sistema. 1. [OUT] O sistema busca na base de dados e apresenta todas as mesas vazias. 2. [IN] O ator informa a mesa à qual a venda se refere. 3. [IN] O ator inicia a mesa. 4. [OUT] O sistema informa que a venda está apta a aceitar novos pedidos e apresenta o menu de produtos.
Tratamento de Exceções e Variantes: **Exceção 2a: Mesa já iniciada** **Variante 2a1: Mensagem de aviso** 2a1.1 [OUT] O sistema avisa que a mesa já está iniciada. 2a1.2 [OUT] O sistema mostra o menu de produtos. **Variante 2a2: Finaliza o caso de uso.** 2a2.1 [OUT] O sistema finaliza o caso de uso. Exceção 4a: Venda cancelada pelo Gerente ou Administrador 4a.1 [IN] O gerente/administrador solicitam o cancelamento da venda por algum motivo. 4a.2 [OUT] O sistema cancela a venda.

Quadro 15 - Expansão de caso de uso - Cadastrar venda

O Quadro 17 apresenta a expansão do caso de uso Adicionar produto.

Caso de Uso: Adicionar produto
Atores: Garçom ou Gerente ou Administrador
Precondições: O ator deve estar logado no sistema. Uma mesa já deve estar iniciada.
Pós-condições: Ator apto a adicionar novos produtos.
Seqüência típica de eventos (Fluxo Principal): Este caso de uso inicia quando o ator solicita adicionar um novo pedido de produto à mesa ou após a mesa ser iniciada. 1. [OUT] O sistema busca na base de dados e apresenta todas as mesas iniciadas, assim como todas as categorias de produtos. 2. [IN] O ator informa a mesa e a categoria de produtos. 3. [OUT] O sistema carrega todos os produtos da categoria selecionada pelo ator. 4. [IN] O ator informa o produto solicitado pelo cliente. 5. [OUT] O sistema apresenta uma nova tela com dados do pedido e solicitando a

quantidade requerida pelo cliente. 6. [IN] O ator informa a quantidade solicitada pelo cliente. 7. [OUT] O sistema informa que o pedido foi efetuado com sucesso. 8. [OUT] O sistema mostra o menu de vendas.
Tratamento de Exceções e Variantes: **Exceção 3a: Produto em falta** 3a.1 [OUT] O sistema avisa que o produto está em falta. 3a.2 [IN] O sistema mostra o menu de produtos.

Quadro 16 - Expansão de caso de uso - Adicionar produto

O Quadro 17 apresenta a expansão do caso de uso Finalizar venda.

Caso de Uso: Finalizar venda
Atores: Garçom ou Gerente ou Administrador
Precondições: O ator deve estar logado no sistema. Uma mesa já deve estar iniciada.
Pós-condições: Venda finalizada.
Sequência típica de eventos (Fluxo Principal): Este caso de uso inicia quando o cliente solicita a um funcionário para finalizar sua mesa. 1. [OUT] O sistema verifica se o usuário possui permissão. 2. [OUT] O sistema verifica todos os pedidos da mesa e gera o valor da conta. 3. [OUT] O sistema informa ao caixa o fechamento da conta. 4. [IN] O ator informa a forma de pagamento. 5. [OUT] O sistema altera o status da mesa para zerada.
Tratamento de Exceções e Variantes:

Quadro 17 - Expansão de caso de uso - Finalizar venda

O Quadro 18 apresenta a expansão do caso de uso Operar caixa.

Caso de Uso: Operar caixa
Atores: Operador de caixa ou Gerente ou Administrador
Precondições: O ator deve estar logado no sistema. Uma venda já ter sido finalizada.
Pós-condições: Venda recebida. Lista de todas as operações realizadas.
Sequência típica de eventos (Fluxo Principal): Este caso de uso inicia quando o ator solicita abrir o caixa. 1. [OUT] O sistema verifica se o usuário possui permissão. 2. [OUT] O sistema verifica se há vendas pendentes e as mostra em tela.

3. [IN] O ator seleciona a venda pendente e informa a forma de pagamento. 4. [OUT] O sistema adiciona o valor da venda ao fluxo do caixa. 5. [OUT] Volta a tela de caixa principal.
Tratamento de Exceções e Variantes: **Exceção 2a: Não há nenhuma venda pendente.** 2a.1 [OUT[O sistema avisa que não há nenhuma venda pendente. 2a.2 [OUT] O sistema mostra a tela de caixa principal.

Quadro 18 - Expansão de caso de uso - Operar caixa

A Figura 13 apresenta o diagrama de classes para o sistema proposto, o qual apresenta as classes com seus atributos e relacionamentos.

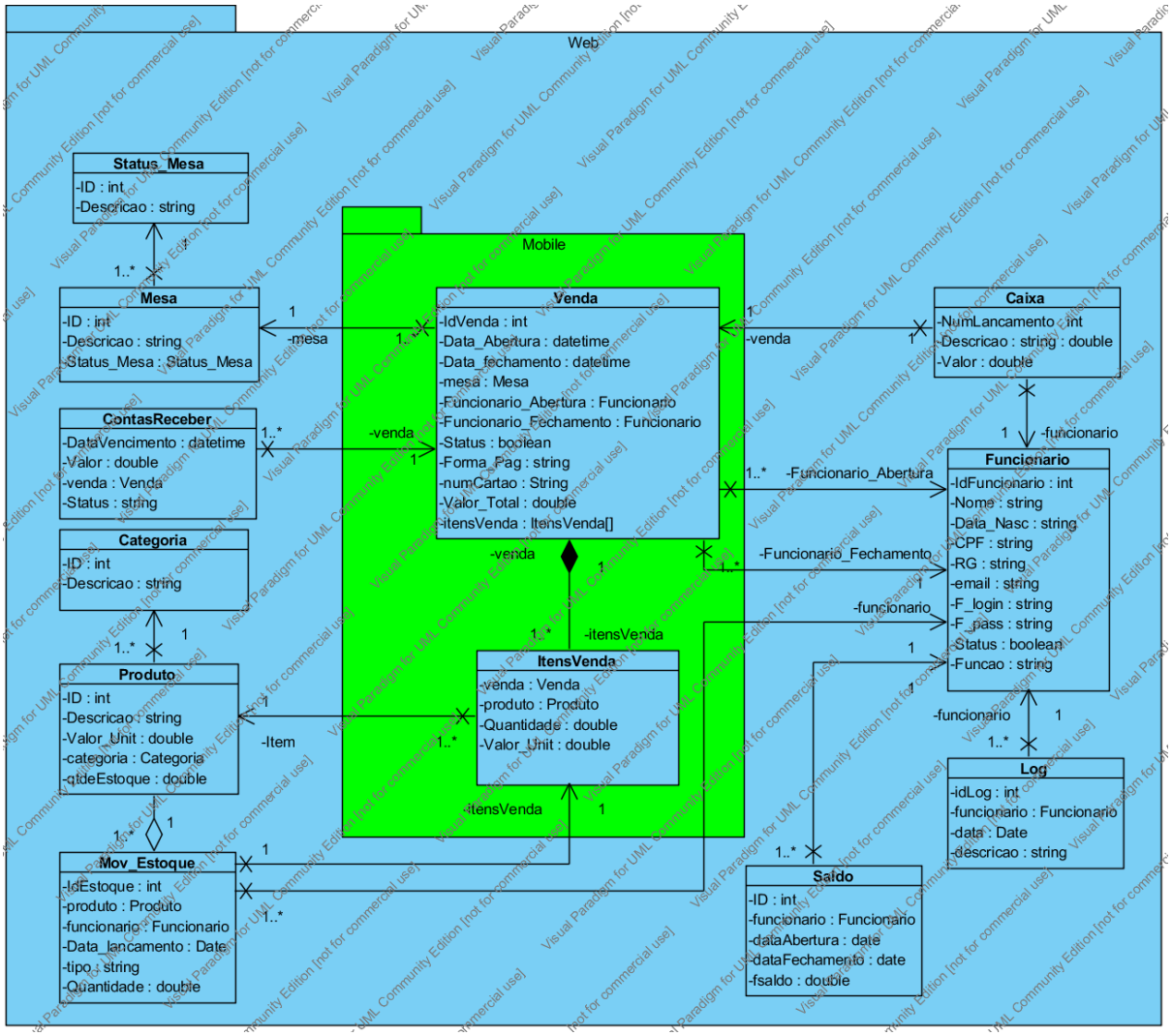

Figura 13 – Diagrama de classes

As classes apresentadas no diagrama da Figura 13 estão documentadas nos Quadros 20 a 31.

Identificação:	Caixa
Descrição:	Registros de entrada no caixa.
Requisitos:	F6
Atributos:	numlancamento (int): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. idfuncionario (int): chave estrangeira do funcionário que realizou a operação. Idvenda (int): chave estrangeira da venda correspondente a movimentação. Descricao (string): campo para armazenar complementos da operação. valor (double): Valor da movimentação. tipo (string): Armazena o tipo do pagamento efetuado.
Métodos da classe *DAO*:	void **salvar**(Caixa caixa); void **atualizar**(Caixa caixa); void **excluir**(Caixa caixa); List<Caixa> **listar**(); Caixa **buscaCaixa** (int id);

Quadro 19 - Descrição da classe Caixa

Identificação:	Categoria
Descrição:	Categoria de produtos.
Requisitos:	F8
Atributos:	idcategoria (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. descricao (string): descrição da categoria.
Métodos da classe *DAO*:	void **salvar**(Categoria categoria); void **atualizar**(Categoria categoria); void **excluir**(Categoria categoria); List<Categoria> **listar**(); Categoria **buscaCategoria** (int id); Categoria **buscaCategoriaS** (String descricao);

Quadro 20 - Descrição da classe Categoria

Identificação:	Contas_a_Receber
Descrição:	Registros das contas a receber, fruto das vendas realizadas com pagamento no cartão de Crédito.
Requisitos:	F6
Atributos:	idconta (int): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. idvenda (int): chave estrangeira da venda correspondente a movimentação. Data_vencimento(date): data do vencimento da conta. Status_conta (string): Estado o qual ela se encontra, paga ou não.
Métodos da classe *DAO*:	void **salvar**(Contas_a_Receber conta); void **atualizar**(Contas_a_Receber conta); void **excluir**(Contas_a_Receber conta); List< Contas_a_Receber > **listar**(); Contas_a_Receber **buscaConta** (int id);

Quadro 21 - Descrição da classe Contas_a_Receber

Identificação:	Funcionario
Descrição:	Tabela de cadastro dos funcionários da empresa, também utilizado como login do sistema.
Requisitos:	F1
Atributos:	idfuncionario (int): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nome (string): Nome do funcionário. telefone (string): telefone do funcionário. cpf (string): cpf do funcionário. rg (string): rg do funcionario. data_nascimento(date): data de nascimento do funcionario. atividade (string): se o funcionário está apto a logar no sistema. f_login (string): login do funcionario no sistema. f_pass (string): senha do funcionario no sistema. funcao (string): classe do usuário no sistema. data_cadastro (date): data de cadastro do funcionário no sistema. ultimoacesso (date): data do ultimo acesso do usuário no sistema.
Métodos da classe *DAO*:	void **salvar**(Funcionario funcionario); void **atualizar**(Funcionario funcionario); void **excluir**(Funcionario funcionario); List<Funcionario> **listar**(); Funcionario **buscaConta** (int id);

Quadro 22 - Descrição da classe Funcionario

Identificação:	Itens_Venda
Descrição:	Tabela em que os pedidos da mesa são armazenados.
Requisitos:	F2, F3
Atributos:	iditem (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. idvenda (int): chave estrangeira da venda correspondente a movimentação. idproduto (int): chave estrangeira do produto solicitado pelo cliente. quantidade (double): quantidade do produto requerida. valor_unitario (double): custo unitário do produto na ocasião da venda.
Métodos da classe *DAO*:	void **salvar**(Itens_Venda itens); void **atualizar**(Itens_Venda itens); void **excluir**(Itens_Venda itens); List< Itens_Venda > **listarItensVenda** (int idvenda);

Quadro 23 - Descrição da classe Itens_Venda

Identificação:	Mesa
Descrição:	Tabela de registro das mesas do estabelecimento.
Requisitos:	F9
Atributos:	idmesa (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. status_mesa (int): chave estrangeira dos possíveis estados possíveis da mesa. descricao (string): nome ou descricao da mesa, o que irá ser apresentado ao usuário..
Métodos da classe *DAO*:	void **salvar**(Mesa mesa); void **atualizar**(Mesa mesa); void **excluir**(Mesa mesa); List< Mesa > **listar**(); Mesa **buscaMesa** (int id);

Quadro 24 - Descrição da classe Mesa

Identificação:	Mov_Estoque
Descrição:	Tabela de registro de movimentação do estoque.
Requisitos:	F2, F3, F10
Atributos:	idestoque (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. idfuncionario (int): chave estrangeira do funcionário que realizou a operação. idvenda (int): chave estrangeira da venda caso movimentação seja gerada por um pedido. idproduto (int): chave estrangeira do produto. data_lancamento (date): data do registro. qtde_lancada (double): quantidade movimentada. tipo (string): define a operação a ser tomada, caso seja entrada adiciona ao estoque caso seja saida subtrai .
Métodos da classe DAO:	void **salvar**(Mov_Estoque mov_estoque); void **atualizar**(Mov_Estoque estoque); void **excluir**(Mov_Estoque mov_estoque); List<Mov_Estoque> **listar**(); Mov_Estoque **buscaMov_Estoque** (int id);

Quadro 25 - Descrição da classe Mov_Estoque

Identificação:	Produto
Descrição:	Tabela de registro dos produtos cadastrados.
Requisitos:	F2
Atributos:	idproduto (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. descricao (string): descrição do produto valor_unitario (double): custo unitário para o produto. unidade_metrica (string): unidade métrica usada para o produto. qtde_estoque (double): quantidade em estoque do produto. imagem (string): url em que a imagem do produto se encontra. Idcategoria (int): chave estrangeira da categoria a que o produto pertence.
Métodos da classe *DAO*:	void **salvar**(Produto produto); void **atualizar**(Produto produto); void **excluir**(Produto produto); List< Produto > **listar**(); List< Produto > **listarPorCategoria**(String categoria); Produto **buscaProduto** (int id);

Quadro 26 - Descrição da classe Produto

Identificação:	Log
Descrição:	Tabela que registra o log do sistema, com as ações realizadas pelos usuários.
Requisitos:	Relaciona-se a todos os requisitos
Atributos:	idlog (int): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. idfuncionario (int): chave estrangeira do funcionário que realizou a ação. descricao (string): descrição da ação. data (date): data da ação.
Métodos da classe *DAO*:	void **salvar**(SYS_LOG log); List< SYS_LOG > **listar**();

Quadro 27 - Descrição da classe SYS_LOG

Identificação:	Saldo
Descrição:	Tabela que armazena os dados do caixa ativo.
Requisitos:	F6
Atributos:	idsaldo (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. idfuncionario (int): chave estrangeira do funcionário que opera o caixa. data_abertura (Date): data de abertura do caixa. data_fechamento (Date): data de fechamento do caixa. fsaldo (double): saldo do caixa.
Métodos da classe *DAO*:	void **salvar**(Saldo saldo); void **atualizar**(Saldo saldo); Saldo **buscaSaldo** (int id); Saldo **buscaSaldoIniciadoFuncionario** (int id);

Quadro 28 - Descrição da classe Saldo

Identificação:	Status_Mesa
Descrição:	Estados possíveis em que uma mesa pode estar.
Requisitos:	F3, F4, F6
Atributos:	idstatus (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. descricao (string): descrição do status.
Métodos da classe *DAO*:	List< Status_Mesa> **listarStatus**(); Status_Mesa **buscaEstado** (int id);

Quadro 29 - Descrição da classe Status_Mesa

Identificação:	Venda
Descrição:	Classe que representa as vendas realizadas.
Requisitos:	F3
Atributos:	idvenda (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. Idmesa (int): mesa da venda. data_abertura (Date): data de abertura da venda. data_fechamento (Date): data de fechamento da venda. valor_total (double): valor total da venda. forma_pagamento (string): forma de pagamento utilizada pelo cliente. num_cartao (string): armazena numero do cartão caso pagamento seja feito via Crédito ou Débito. status_pedido (string): status da venda, se esta aberta ou pendente ou fechada.
Métodos da classe *DAO*:	void **salvar**(Venda venda); void **atualizar**(Venda venda); void **excluir**(Venda venda); List< Venda > **listar**(); List< Venda > **listarPersonalizada**(String status); Venda **buscaVenda** (int id);

Quadro 30 - Descrição da classe Venda

A Figura 14 mostra o Diagrama Entidade e Relacionamento (DER) que apresenta as tabelas e seus relacionamentos identificados, os quais representam o banco de dados da aplicação.

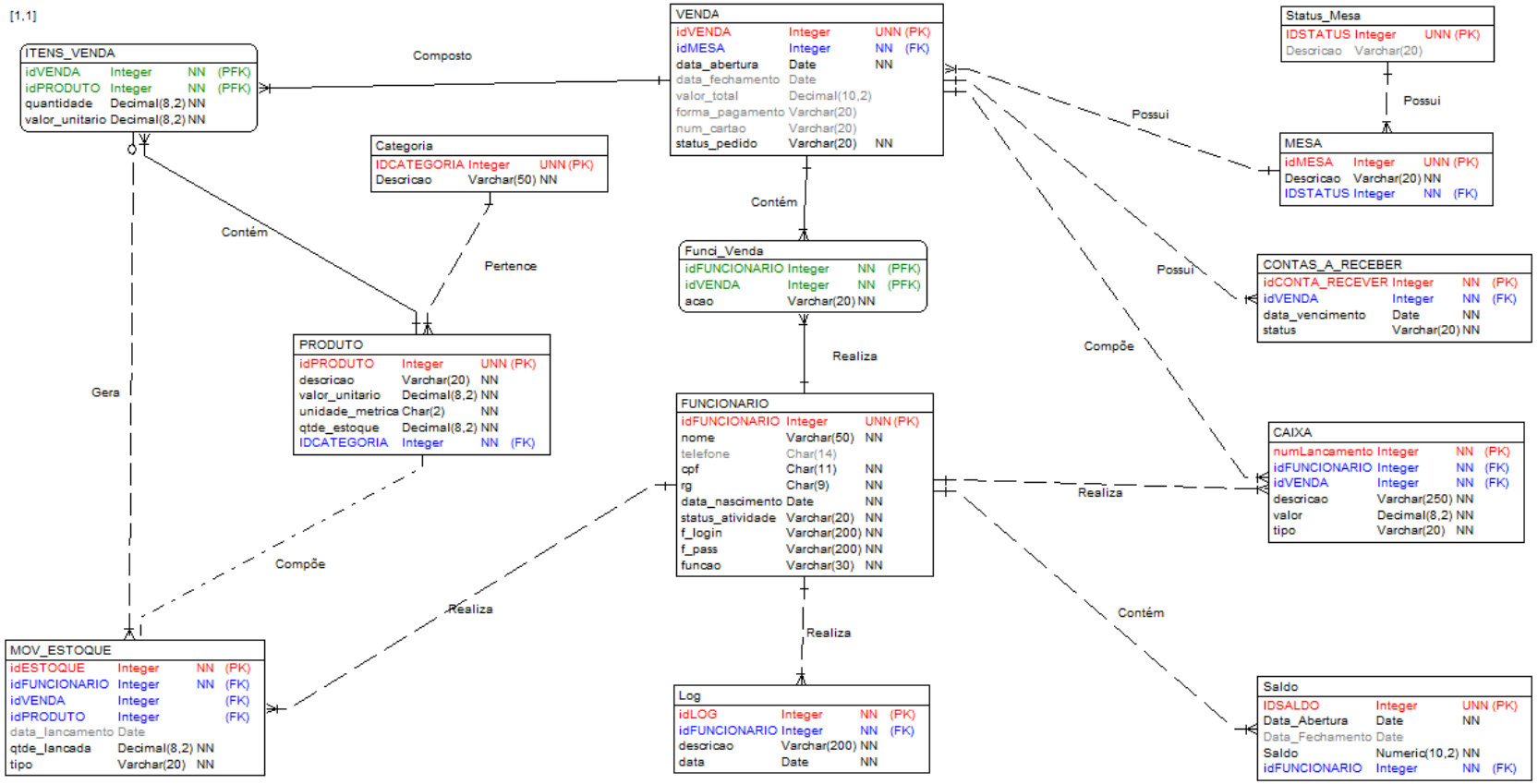

Figura 14 – Diagrama Entidade e Relacionamento para o sistema.

A seguir (Quadros 31 a 42) a descrição das tabelas que compõem o banco de dados, conforme expõem a Figura 14.

Produto

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idPRODUTO	Inteiro	Não	Sim	Não	
descricao	Texto (20)	Não	Não	Não	
valor_unitario	Numérico	Não	Não	Não	
unidade_metrica	Texto (2)	Não	Não	Não	
qtde_estoque	Numérico	Não	Não	Não	
idCATEGORIA	Inteiro	Não	Não	Sim	

Quadro 31 - Mapeamento entidade - Estoque

Mesa

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idMESA	Inteiro	Não	Sim	Não	
status_mesa	Texto	Não	Não	Não	

Quadro 32 - Mapeamento entidade - Mesa

Contas a receber

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idCONTA	Inteiro	Não	Sim	Não	
idVENDA	Inteiro	Não	Não	Sim	
data_vencimento	Data	Não	Não	Não	
status	Texto (20)	Não	Não	Não	

Quadro 33 - Mapeamento entidade - Contas a receber

Venda

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idVENDA	Inteiro	Não	Sim	Não	

idMESA	Inteiro	Não	Não	Sim	
data_abertura	Data	Não	Não	Não	
data_fechamento	Data	Sim	Não	Não	
valor_total	Numérico	Sim	Não	Não	
forma_pagamento	Texto (20)	Sim	Não	Não	
num_cartao	Texto (20)	Sim	Não	Não	Só será preenchido quando a forma de pagamento for via cartão de crédito.
status	Texto (20)	Não	Não	Não	

Quadro 34 - Mapeamento entidade - Venda

Funci_Venda

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idVENDA	Inteiro	Não	Sim	Sim	
idFuncionário	Inteiro	Não	Sim	Sim	
acao	Texto (20)	Não	Não	Não	

Quadro 35 - Mapeamento entidade - Funci_Venda

Itens venda

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idVENDA	Inteiro	Não	Sim	Sim	
idPRODUTO	Inteiro	Não	Sim	Sim	
quantidade	Numérico	Não	Não	Não	
valor_unitario	Numérico	Não	Não	Não	

Quadro 36 - Mapeamento entidade - Itens_venda

Caixa

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
numLancamento	Inteiro	Não	Sim	Não	
idFUNCIONARIO	Inteiro	Não	Não	Sim	

idVENDA	Inteiro	Não	Não	Sim	
descricao	Texto (250)	Não	Não	Não	
valor	Numérico	Não	Não	Não	
tipo	Texto (20)	Não	Não	Não	

Quadro 37 - Mapeamento entidade - Caixa

Funcionario

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idFUNCIONARIO	Inteiro	Não	Sim	Não	
nome	Texto (50)	Não	Não	Sim	
telefone	Texto (14)	Sim	Não	Não	
cpf	Texto (11)	Não	Não	Não	
rg	Texto (9)	Não	Não	Não	
data_nascimento	Data	Não	Não	Não	
status_atividade	Texto (20)	Não	Não	Não	
f_login	Texto (200)	Não	Não	Não	
f_pass	Texto (200)	Não	Não	Não	
funcao	Texto (30)	Não	Não	Não	

Quadro 38 - Mapeamento entidade - Funcionario

Log

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idLOG	Inteiro	Não	Sim	Não	
idFUNCIONARIO	Inteiro	Não	Não	Sim	
descricao	Texto	Não	Não	Não	
data	Data	Não	Não	Não	

Quadro 39 - Mapeamento entidade - Log

Mov_Estoque

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idESTOQUE	Inteiro	Não	Sim	Não	
idFUNCIONARIO	Inteiro	Não	Não	Sim	
idPRODUTO	Inteiro	Não	Não	Sim	
Data_lancamento	Data	Não	Não	Não	
Qtde_lancada	Numérico	Não	Não	Não	
tipo	Texto (20)	Não	Não	Não	

Quadro 40 - Mapeamento entidade - Estoque

Categoria

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idCATEGORIA	Inteiro	Não	Sim	Não	
Descricao	Texto(20)	Não	Não	Não	

Quadro 41 - Mapeamento entidade - Categoria

Saldo

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idSALDO	Inteiro	Não	Sim	Não	
idFUNCIONARIO	Inteiro	Não	Não	Sim	
data_abertura	Data	Não	Não	Não	
data_fechamento	Data	Sim	Não	Não	
FSALDO	Numérico	Não	Não	Não	

Quadro 42 - Mapeamento entidade - Saldo

4.3 Apresentação do Sistema

Nesta seção são apresentadas algumas capturas de telas (*Print Screens)* que demonstram a interface do sistema desenvolvido. Para o módulo *Web* as capturas foram feitas utilizando um navegador (*browser*) comum (*Google Chrome*), já para a parte *mobile* do sistema, foi utilizado um emulador *Android* (*AVD Manager*) para que fosse possível a captura das interfaces.

A seguir, a Figura 15 apresenta a tela de *login* para o sistema na plataforma *Web*. Nesta tela o usuário informará seus dados, login e senha, e o sistema irá verificar se os dados informados conferem com algum dos registros do banco de dados. A Figura 16 apresenta a mensagem de erro ao usuário caso ocorra falha na validação desses dados.

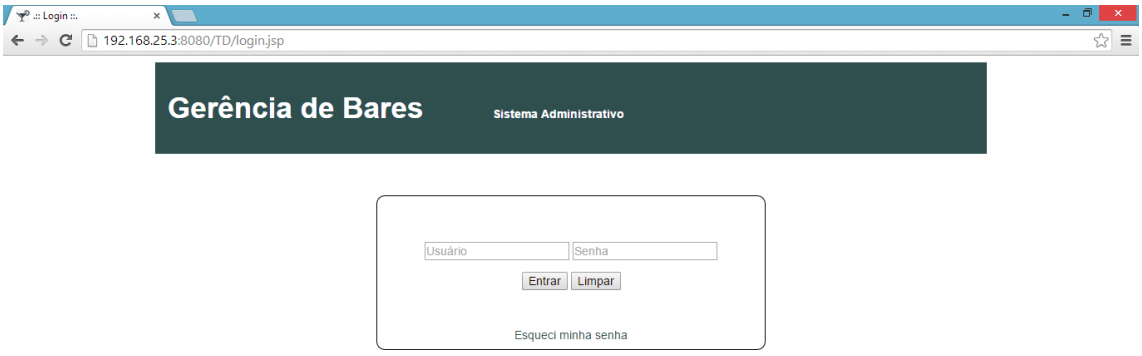

Figura 15 - Tela de Login do módulo *Web*

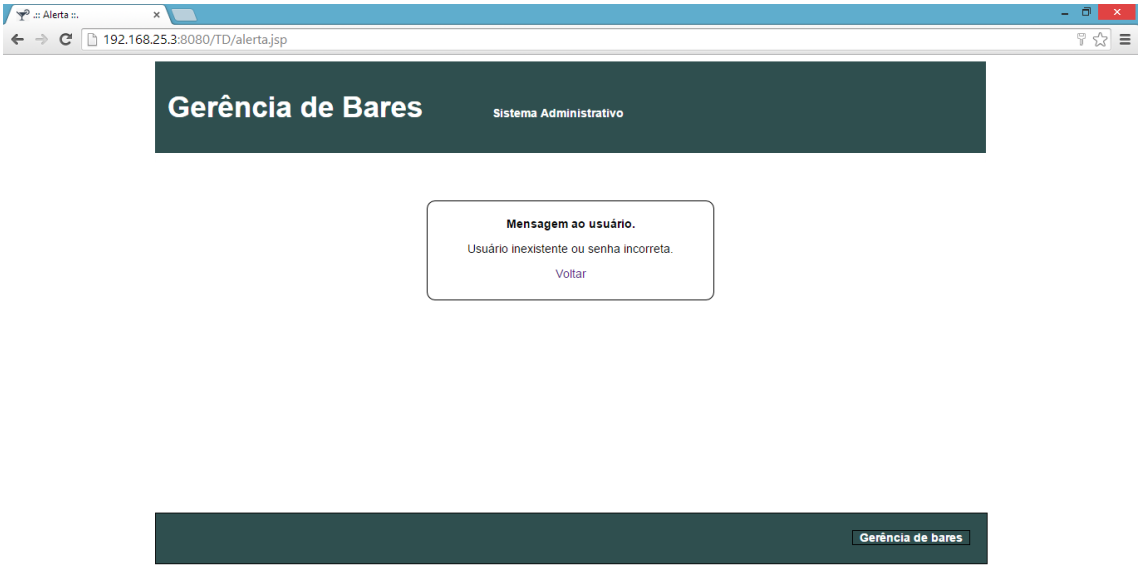

Figura 16 - Mensagem de falha no login do módulo *Web*

Após ter validado com êxito o *login*, o sistema irá redirecionar o usuário até a página *index* (Figura 17), que é a página principal do módulo *Web* do sistema.

Figura 17 - Página principal do módulo *Web*

Se por algum motivo o usuário fazer o *logoff* do sistema ou sua sessão expirar e o mesmo tentar acessar alguma página que não seja a de login, o

sistema irá redirecionar o usuário à página de alerta e lhe mostrar um aviso que ele precisa logar no sistema para poder visualizar o conteúdo. A Figura 18 apresenta esse aviso.

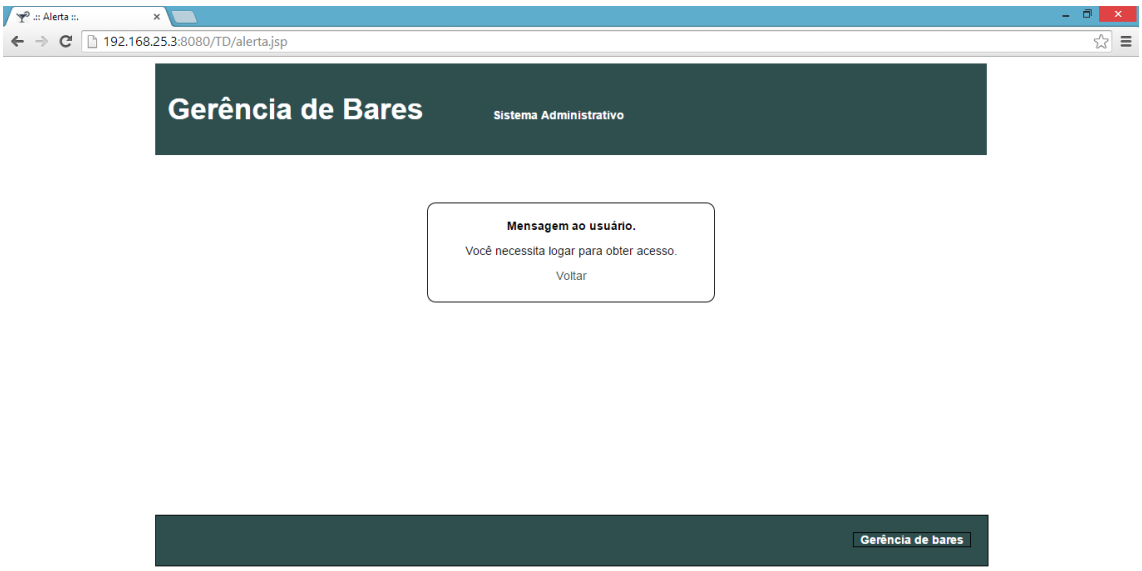

Figura 18 - Alerta de login

O *layout* da página é composta por quatro elementos, o cabeçalho que contém um *banner* com o título "Gerência de Bares" e "Sistema Administrativo", o qual é estático assim como o rodapé. O menu de navegação que é dinâmico e mostra ao usuário apenas as funcionalidades do sistema ao qual ele está habilitado a utilizar. É também no menu de navegação que se encontra a opção de *logoff* do sistema, que fica ao lado do nome de *login* do usuário. No restante do corpo da página são apresentados os resultados como formulários de cadastros e outras funcionalidades como, por exemplo, a lista de registros, opções de alteração e exclusão de registros entre outras.

A Figura 19 apresenta a tela que mostra as informações do usuário logado no sistema, esta tela também possui a opção do usuário alterar sua senha.

Figura 19 - Minha Conta do módulo *Web*

Na Figura 20 é apresentada a tela de cadastro de novos Funcionários, que são os usuários do sistema. Já a Figura 21 mostra a tela de cadastro de Produtos.

Figura 20 - Tela de Cadastro de Funcionário

Figura 21 - Tela de Cadastro de Produto

.

As ações realizadas pelos usuários do sistema são armazenadas, gerando assim um *log*, para que se possa ter um controle efetivo sobre o que os usuários fazem no sistema. A Figura 22 mostra o *log* do sistema.

Figura 22 - Log do Sistema

A Figura 23 mostra o menu do Caixa, sendo que o usuário responsável pode utilizar das funcionalidades, conforme apresentadas na figura.

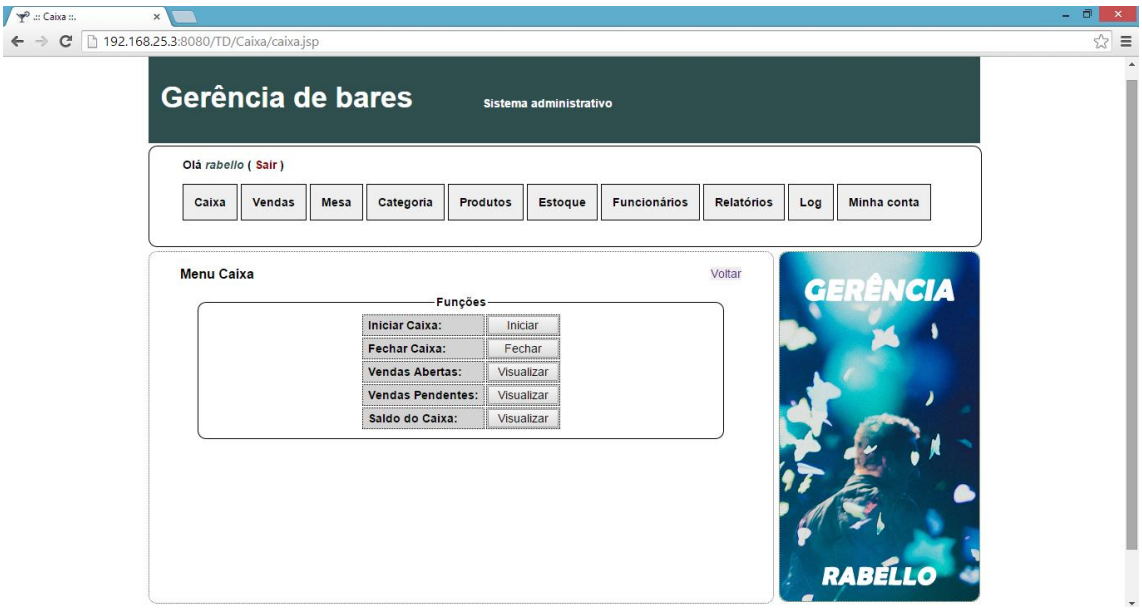

Figura 23 - Tela do Caixa

O usuário que atende ao caixa, deve dar prioridade às vendas pendentes, elas estão disponíveis no menu do Caixa, essa tela lista todas as vendas com situação pendente e as mostra em uma tabela, assim como é possível ver na Figura 24. É nesta tela que o operador do caixa possui a opção de iniciar o processo de finalizar a venda.

Figura 24 - Tela de Vendas Pendentes

Na tela da Figura 25 o operador do caixa irá finalizar a venda, escolhendo a forma de pagamento e preenchendo os dados adicionais.

Figura 25 - Finalizar Venda

Após finalizada a venda, é possível ver os detalhes da venda conforme demonstra a Figura 26.

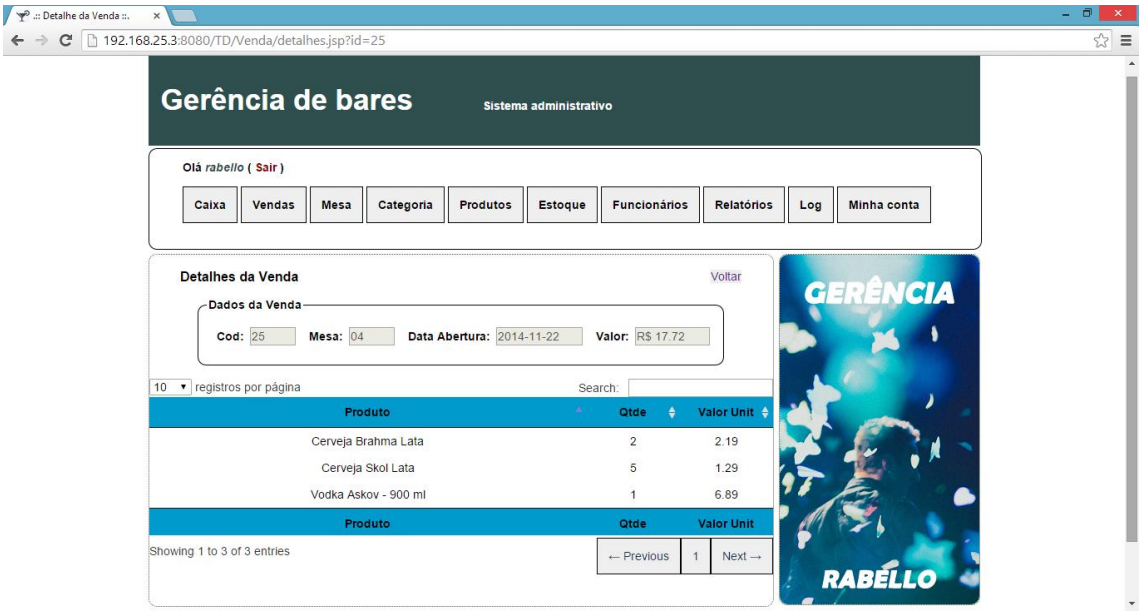

Figura 26 - Detalhes da Venda

O estoque funciona em tempo real, toda vez que é realizada uma venda a quantidade vendida é debitada instantaneamente do estoque. A Figura 27 mostra a lista de produtos e suas respectivas quantias em estoque.

Figura 27 - Tela de Estoque

Os usuários com papel de *admin*, podem gerar relatórios com base nas vendas realizadas, filtrando os dados como melhor desejarem. A Figura 28 demonstra a tela com as opções de relatórios, os quais o usuário pode gerar.

Figura 28 - Tela de menu de relatórios

O resultado obtido através da filtragem feita nas opções de relatórios, é de um relatório gerado pelo *Jasper Report*, como a Figura 29 apresenta o resultado.

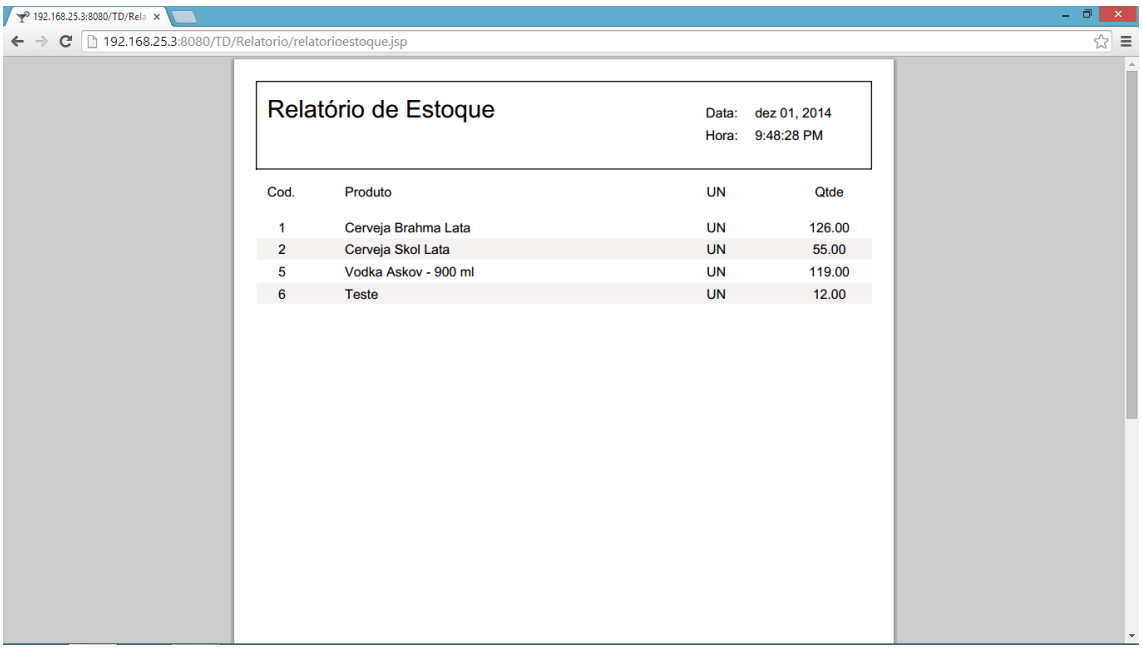

Figura 29 - Relatório

A Figura 30 apresenta a tela de *login* para o sistema na plataforma *mobile*. Nesta tela, assim como no módulo *Web* do sistema, o usuário informará seus dados, *login* e senha, e o aplicativo irá verificar se os dados informados conferem com algum dos registros do banco de dados. A Figura 31 apresenta a mensagem de erro ao usuário caso ocorra falha na validação desses dados.

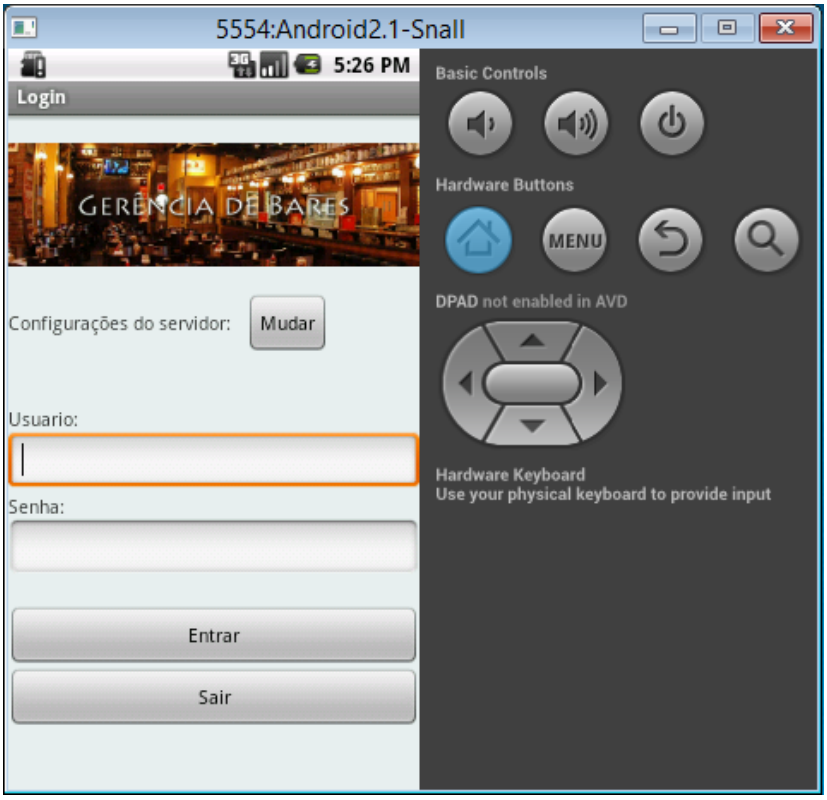

Figura 30 - Tela de Login do módulo *mobile*

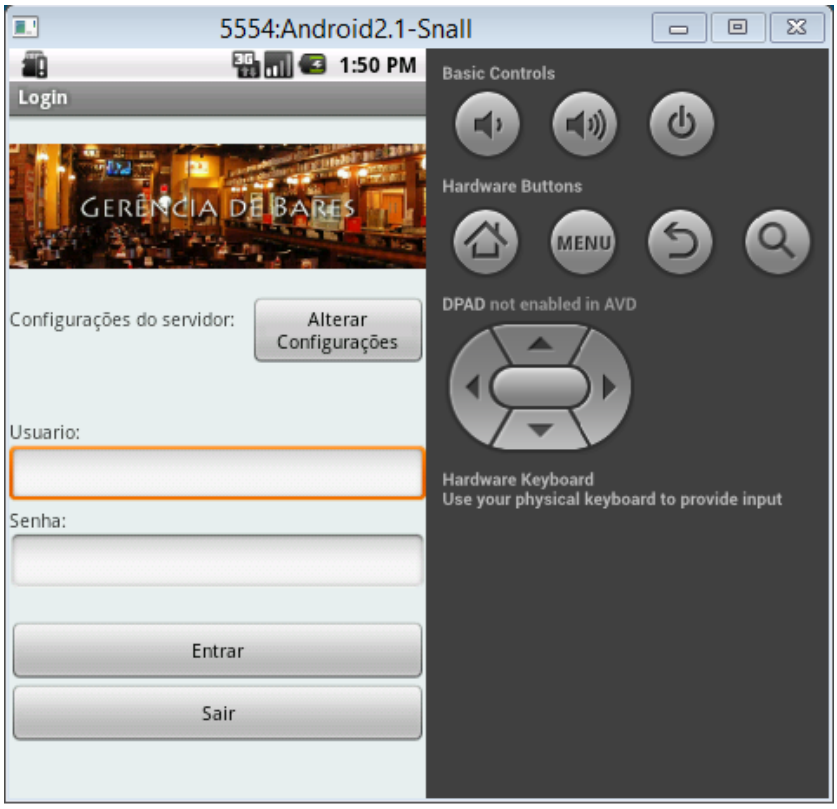

Figura 31 - Mensagem de falha no login do módulo *mobile*

Ainda na tela *Login* existe o botão "Alterar Configurações", ao clicar neste botão o aplicativo mostrará ao usuário a tela de configurações do servidor, é nela que o usuário informa os dados, *IP* e porta de destino, do *servidor*, salvando os dados informados em *Shared Preference* em *MODE_APPEND*, para que seja possível acessá-los em qualquer lugar da aplicação. A Figura 32 mostra a tela para alteração das configurações.

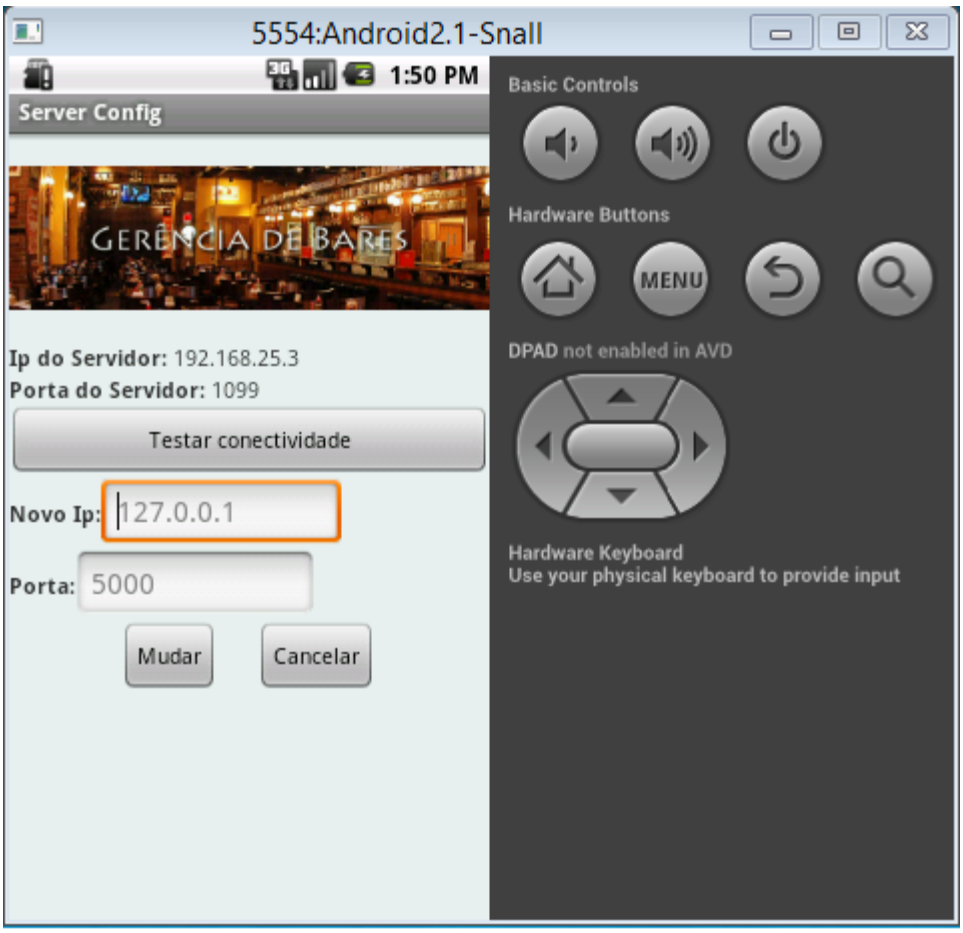

Figura 32 - Tela Server Config

Após ter validado o *login* com êxito, o sistema irá direcionar o usuário até o menu principal do aplicativo (Figura 33), local onde se apresentam as funcionalidades em que o usuário pode submeter o aplicativo.

Figura 33 - Menu principal do módulo mobile

A primeira opção do menu é a "Usuário", essa opção irá invocar e mostrar outra tela no aplicativo, assim como mostra a Figura 34. A tela Minha Conta, a qual é igual a tela Minha Conta do módulo *Web*, contém as mesmas informações e opção de alteração da senha.

Figura 34 - Minha Conta do módulo *mobile*

A segunda opção do menu principal é a Venda, ela chama uma outra tela que possui um menu semelhante, a diferença são suas opções, as três funções disponíveis nessa nova tela são "Iniciar Venda, "Adicionar Produto" e "Finalizar Venda". Cada uma dessas opções irá chamar a sua tela respectiva.

A opção "Iniciar Venda" mostra a tela em que é possível escolher uma das mesas cadastradas no banco de dados e com o status de "Vazia" e iniciar uma venda atribuída a ela. A Figura 35 apresenta a tela.

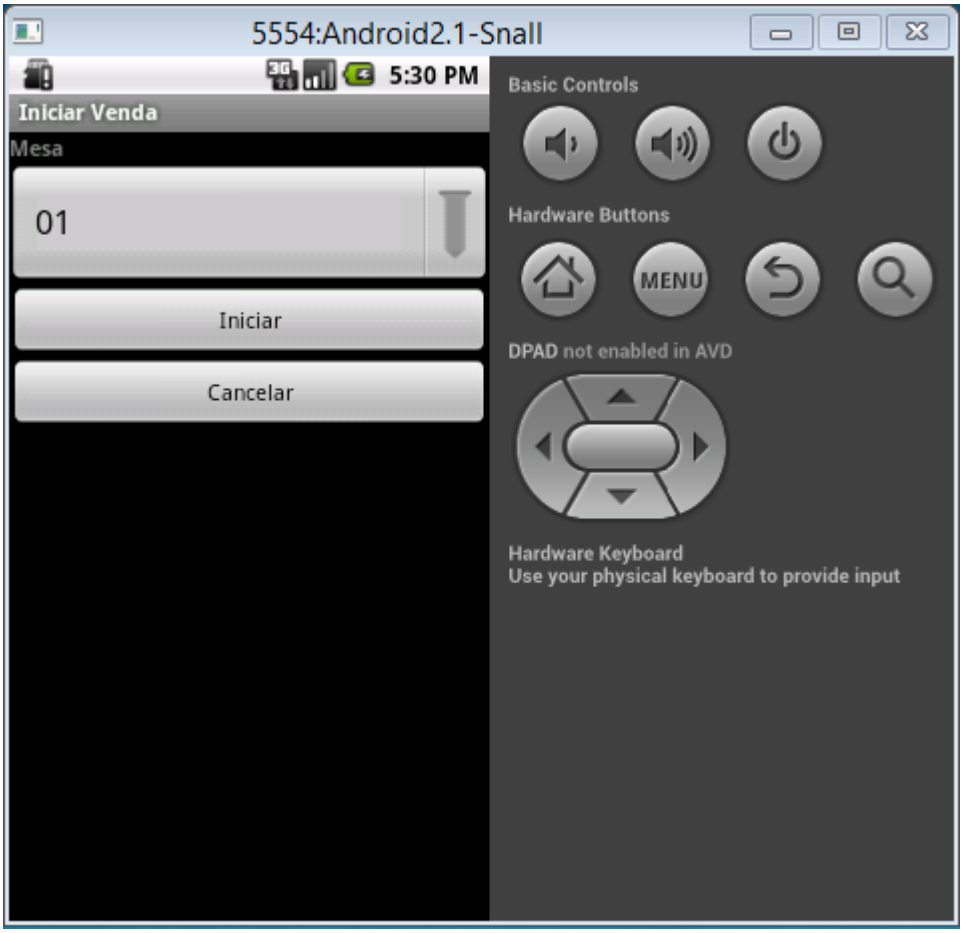

Figura 35 - Tela Iniciar Venda

Após iniciar uma venda para uma mesa, o aplicativo invoca a tela Adicionar Produto (Figura 36), essa tela mostra componentes que torna possível escolher a mesa a qual se deseja atribuir este novo pedido, as mesas que aparecem nessa tela são apenas mesas que possuam seu status definido como "Aberta". Também apresenta a categoria de produtos a ser selecionada e em seguida será carregada na mesma tela a lista de produtos e suas respectivas imagens que correspondam a categoria escolhida. As imagens são todas carregadas do servidor a partir da url salva no banco de dados, todas as imagens do sistema, incluindo as do produto são armazenadas no próprio servidor.

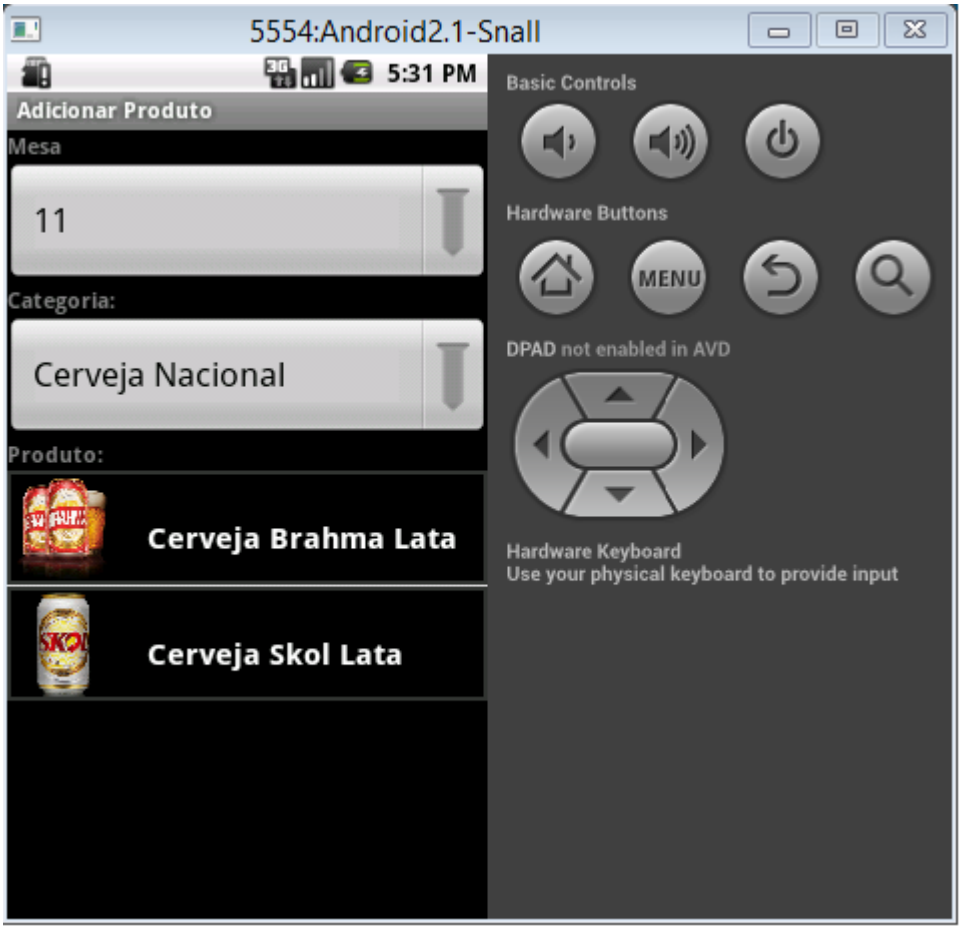

Figura 36 - Tela Adicionar Produto

Ao ser apresentada a lista de produtos pertencentes à categoria selecionada, o usuário deverá selecionar um dos produtos clicando sobre o mesmo, o que irá carregar uma outra tela (Figura 37), o usuário definirá a quantidade requerida pelo cliente para o produto. É nessa tela que é feita a verificação se há estoque suficiente para atender o cliente.

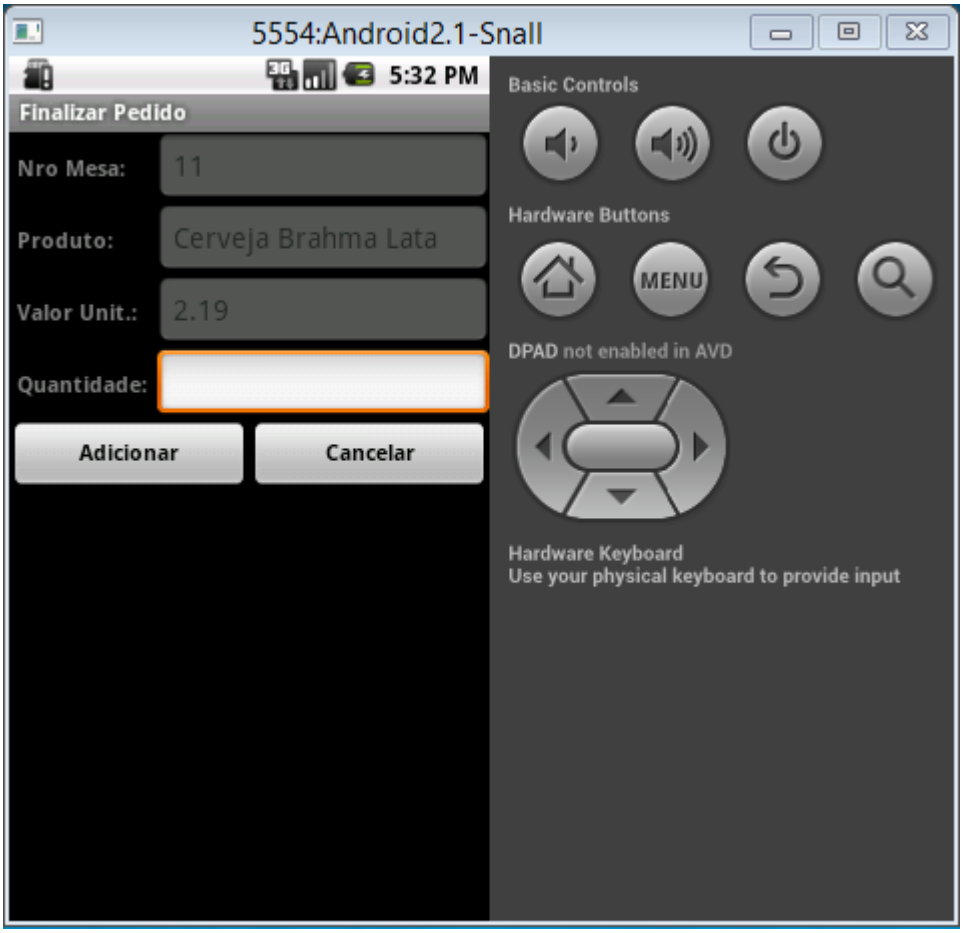

Figura 37 - Finalizar Pedido

Outra forma de chamar a tela Adicionar Produto é usando a segunda opção do menu Venda, ela mostrará diretamente a tela deixando disponível para o usuário à escolha da mesa e da categoria que deseja.

A terceira opção do menu Venda é "Finalizar Venda", esta opção assim como as demais chamará uma outra tela, assim como é possível observar na Figura 38, essa tela é semelhante a tela de Iniciar Venda, com a diferença que as mesas apresentadas são as mesma que se apresentam na tela Adicionar Produto. Tratam-se de mesas que possuam seu *status* definido como "Aberta".

Ao selecionar a mesa e clicar em finalizar, o aplicativo irá requisitar confirmação do usuário. Caso ele confirme, o aplicativo irá alterar o *status* da mesa de "Aberta" para "Pendente", fazendo com que essa venda apareça na lista de pendências no caixa.

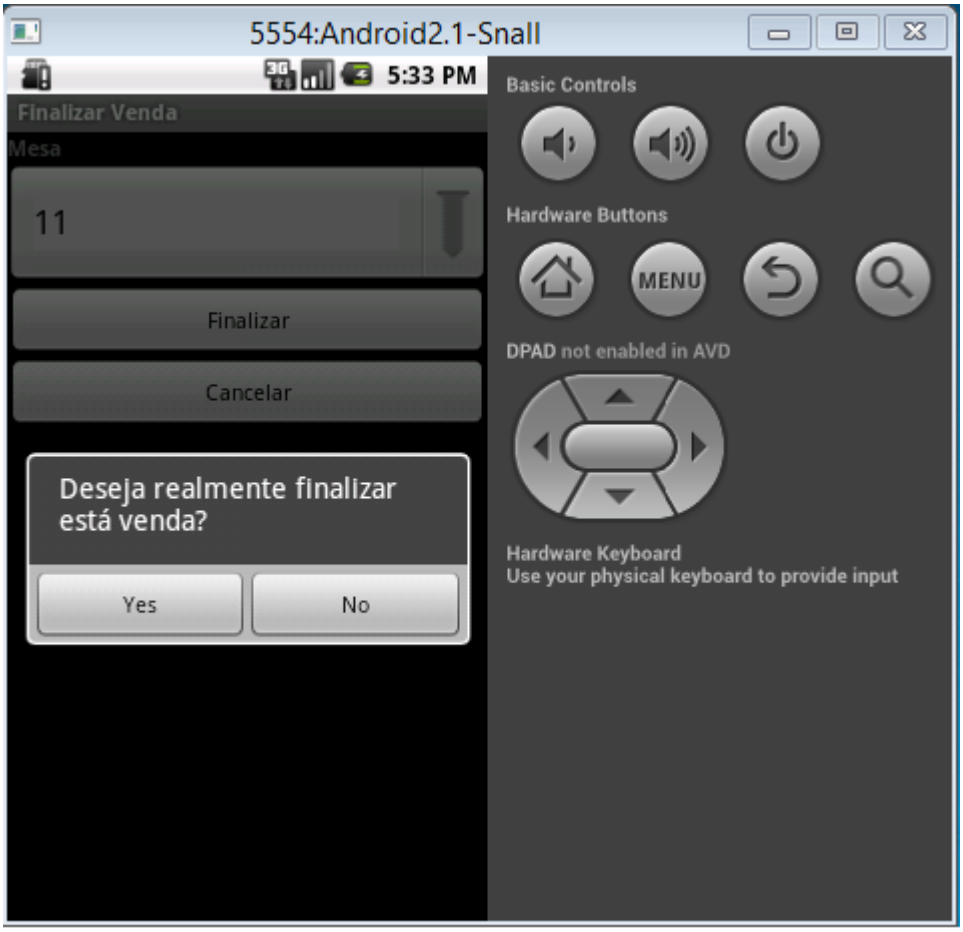

Figura 38 - Tela Finalizar Venda

Para que fosse possível iniciar o banco de dados e o servidor *socket*, que é quem se comunica com a parte *mobile* do sistema, foi criada uma aplicação *java*. Essa aplicação possui uma interface simples, como mostra a Figura 39, pois seu único objetivo é iniciar os serviços e pará-los quando solicitado.

Figura 39 - Servidor de Serviços

4.4 Implementação do Sistema

A implementação desse sistema, iniciou-se pela criação do banco de dados com base no Diagrama de Entidade e Relacionamento, gerado na Análise e Projeto desse sistema.

A partir da criação do banco de dados, foi possível criar a camada de negócios (*Model)* do sistema, com base nas tabelas, gerando classes de objetos com os seus devidos atributos, mapeando-os com a ajuda do *Framework JPA.* Na Listagem 1 temos o exemplo de uma das classes mapeadas.

```
package Model;
import java.io.Serializable;
import java.util.Date;
import javax.persistence.*;
@Entity
@Table(name = "funcionario")
public class Funcionario implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    @Column(name = "idfuncionario")
    private int idfuncionario;

    @Column(name="nome", length = 50,nullable = false)
    private String nome;

    @Column(name="telefone", length = 20,nullable = true)
    private String telefone;

    @Column(name="cpf", length = 14,nullable = false)
    private String cpf;

    @Column(name="rg", length = 12,nullable = false)
    private String rg;

    @Column(name="data_nascimento")
    @Temporal(TemporalType.DATE)
    private Date data_nascimento;

    @Column(name="atividade", length = 20,nullable = false)
    private String atividade;

    @Column(name="f_login", length = 200,nullable = false)
    private String f_login;

    @Column(name="f_pass", length = 200,nullable = false)
    private String f_pass;

    @Column(name="funcao", length = 30,nullable = false)
    private String funcao;

    @Column(name="datacadastro", nullable = false)
    @Temporal(TemporalType.DATE)
    private Date datacadastro;

    @Column(name="ultimoacesso")
    @Temporal(TemporalType.DATE)
    private Date ultimoacesso;
    Getters & Setters
}
```

Listagem 1 - Código *Model* Funcionario

Após completada a etapa de mapeamento das tabelas do banco, tornou-se possível a criação das classes *Data Access Object* (*DAO*) do sistema, essas classes são responsáveis por todas as ações do sistema que terão interação com o banco de dados, tal como incluir, excluir, alterar, buscar. As funções dessas classes foram criadas com o auxilio do *Framework Hibernate,* que auxiliou para que a complexidade do código fosse diminuída, deixando todas as chamadas de *Structured Query Language* (*SQL*) a seu cargo. Outra utilidade do *Hibernate* é a facilidade para a portabilidade da base de dados, já que é necessário apenas modificar a classe de configuração do *Hibernate*, a *hibernate.cfg.xml*, essa classe será abordada logo abaixo. Na Listagem 2, é apresentado um exemplo das funções presentes nas classes DAO.

```
package Dao;

import DB.HibernateUtil;
import Model.Funcionario;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class FuncionarioDao {


    public void salvar(Funcionario funcionario) {
        Session sessao = null;
        Transaction transacao = null;

        try {
                sessao = HibernateUtil.getSessionFactory().openSession();
                transacao = sessao.beginTransaction();
                sessao.save(funcionario);
                transacao.commit();
        } catch (HibernateException ex) {
                System.out.println("Erro - erro ao salvar!" + ex.getMessage());
        } finally {
                sessao.close();
        }
    }

    public void atualizar(Funcionario funcionario);
    public void excluir(Funcionario funcionario);
    public Funcionario buscar(int id);
    public ArrayList<Funcionario> listar();
```

```
}
```

Listagem 2 - Código *DAO* Funcionario

Para conseguir conectar-se ao banco de dados, uma classe *DAO* necessita do auxílio de uma classe que gere uma nova instância da *SessionFactory* do *Hibernate*. Essa instância é gerada baseada nos dados dispostos no arquivo *Hibernate.cfg.xml* (Listagem 4). A Listagem 3 mostra a instanciação de uma nova sessão de conexão.

```
package DB;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;

public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
                // Create the SessionFactory from standard (hibernate.cfg.xml)
                 // config file.
                sessionFactory                          =                          new
Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
                // Log the exception.
                System.err.println("Initial SessionFactory creation failed." + ex);
                throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

}
```

Listagem 3 - Código HibernateUtil

O arquivo *Hibernate.cfg.xml* é o local onde ficam armazenadas as informações referentes ao banco de dados, como por exemplo, o *Driver JDBC*,

local em que o banco está hospedado e o mapeamento das tabelas com base nas classes *Model* geradas anteriormente. A Listagem 4 apresenta o arquivo *XML* com as informações do sistema dispostas.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Configurações -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/barestd?zeroDateTimeBehavior=convertToNull</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password"></property>
    <property name="hibernate.show_sql">true</property>
    <mapping class="Model.Caixa"/>
    <mapping class="Model.Categoria"/>
    <mapping class="Model.Contas_a_Receber"/>
    <mapping class="Model.Funcionario"/>
    <mapping class="Model.FuncionarioVenda"/>
    <mapping class="Model.Itens_Venda"/>
    <mapping class="Model.Mesa"/>
    <mapping class="Model.Mov_Estoque"/>
    <mapping class="Model.Produto"/>
    <mapping class="Model.Saldo"/>
    <mapping class="Model.Status_Mesa"/>
    <mapping class="Model.SYS_LOG"/>
    <mapping class="Model.Venda"/>
  </session-factory>
</hibernate-configuration>
```

Listagem 4 - Hibernate.cfg.xml

As telas do módulo *Web*, são todas apresentadas em *JavaServer Pages* (*JSP*), mas são desenvolvidas utilizando a linguagem de marcação *HTML* com o auxílio de *Frameworks* de *JavaScripts*, como por exemplo, *AJAX*, *JQuery* e *JSON*.

Os campos que necessitam preenchimento por parte do usuário, são todos validados com uso das propriedades do *HTML5*. A Listagem 5 mostra um exemplo de formulário em que é feita a validação desses campos.

```
<form method="POST" action="FuncionarioServlet?acao=cadastrar">
    <div id="formCad">
        <fieldset>
            <legend>Dados Pessoais</legend>
            <p><label for="nome">Nome:  </label>
<input  type="text"  name="nome"  placeholder="Nome"  title="Preencha  o  campo
corretamente." maxlength="50" size="63" required /></p>
<p><label for="cpf">CPF:</label>
<input  type="text"  name="cpf"  pattern="¥d{3}.¥d{3}.¥d{3}-¥d{2}"  maxlength="14"
placeholder="000.000.000-00" title="Preencha o campo corretamente."  required />
<label for="rg">RG:</label>
<input type="text" name="rg" pattern="¥d{2}.¥d{3}.¥d{3}-¥d{1}" maxlength="12"
placeholder="00.000.000-0" title="Preencha o campo corretamente." required /></p>
<p><label for="DataNasc">Data Nascimento:</label>
<input type="text" name="DataNasc" id="txtDataNasc" maxlength="10" title="Preencha o
campo corretamente."
placeholder="00/00/0000"  pattern="[0-9]{2}¥/[0-9]{2}¥/[0-9]{4}"  min="1900-01-01"
required />
<label for="tel">Telefone:</label>
<input type="text" name="tel" pattern="¥([0-9]{2}¥) [0-9]{4}-[0-9]{4}" maxlength="20"
placeholder="(00) 0000-0000" title="Preencha o campo corretamente." required /></p>
        </fieldset>

        <fieldset>
            <legend>Dados Operacionais</legend>
                <p><label for="login">Login:</label>
<input  type="text"  name="login"  placeholder="Login"  title="Preencha  o  campo
corretamente." size="20" required />
                <label for="f_pass">Senha:</label>
<input  type="password"  name="f_pass"  placeholder="Senha"  title="Preencha  o  campo
corretamente." size="20" required /></p>
                <p><label for="funcao">Função:</label>
                    <select name="funcao" id="funcao" required>
                        <option value="" selected="selected">Selecione</option>
                        <option value="Admin">Admin</option>
                        <option value="Caixa">Caixa</option>
                        <option value="Gerente">Gerente</option>
                        <option value="Garcom">Garçom</option>
                    </select>
```

```
                    <label for="status">Status:</label>
                    <select name="status" id="status" required>
                        <option value="" selected="selected">Selecione</option>
                        <option value="Ativo">Ativo</option>
                        <option value="Inativo">Inativo</option>
                    </select>
                </p>
        </fieldset>

    </div>

    <div id="botoes">
        <p>
            <button type="submit">Cadastrar</button>
            <button type="reset">Limpar</button>
        </p>
    </div>

</form>
```

Listagem 5 - Código Formulário de Cadastro de Funcionário

Todas as ações realizadas pelo usuário que necessite interagir com informações do banco de dados, seja ela cadastrar um novo registro, editar um registro ou simplesmente fazer a listagem de registros de alguma tabela, o formulário da página irá requisitar ao seu *servlet* correspondente, que irá receber a solicitação, tratar e responder a requisição gerada pelo usuário do sistema. É dentro dos *servlets* que são chamadas as funções que realizam as operações com interação com o banco.

A Listagem 6 apresenta a função de inserção de um novo registro na tabela "Funcionario" do banco de dados, ela é a principal função realizada na requisição do formulário mostrado na Listagem 5.

```
private boolean cadastrarFuncionario(HttpServletRequest request) {
        boolean retorno = false;
        Funcionario funcionario = new Funcionario();
        FuncionarioDao fdo = new FuncionarioDao();

        funcionario.setNome(request.getParameter("nome"));
        funcionario.setCpf(request.getParameter("cpf"));
        funcionario.setRg(request.getParameter("rg"));
funcionario.setData_nascimento(Date.valueOf(Data.padronizaData(request.getParameter("
DataNasc"))));
```

```
        funcionario.setTelefone(request.getParameter("tel"));
        funcionario.setF_login(request.getParameter("login"));
        funcionario.setF_pass(Criptografia.encripta(request.getParameter("f_pass")));
        funcionario.setFuncao(request.getParameter("funcao"));
        funcionario.setAtividade(request.getParameter("status"));
funcionario.setDatacadastro(Date.valueOf(Data.padronizaData(Data.dateToString(Data.re
tornaDataAtual()))));
        funcionario.setUltimoacesso(null);

        fdo.salvar(funcionario);
        retorno = true;
        return retorno;
    }
```

Listagem 6 - Código cadastrarFuncionário do *servlet* FuncionarioServlet

A geração de um relatório é feita após a requisição por parte do usuário em gerá-lo, o *Servlet* irá interpretar a ação requerida pelo usuário e irá chamar o relatório correspondente. A chamada do relatório é feita no corpo da página JSP, a seguir a Listagem 7 apresenta os códigos referentes a chamada do relatório.

```
try {
                //Connecting to the MySQL database
                con = ConexaoFactory.getConnection();

                File                reportFile                =                new
File(application.getRealPath("/Relatorio/Models/RelEstoque.jasper"));
                Map parameters = new HashMap();
                 byte[] bytes = JasperRunManager.runReportToPdf(reportFile.getPath(),
parameters, con);

                response.setContentType("application/pdf");
                response.setContentLength(bytes.length);
                ServletOutputStream outS = response.getOutputStream();
                outS.write(bytes, 0, bytes.length);
                outS.flush();
                outS.close();
            } catch (Exception ex) {
                System.out.println("Erro: " + ex.getMessage());
            } finally {
                con = null;
            }
```

Listagem 7 - Código Chamada de Relatório

Todo o módulo Web respeita a estrutura apresentada acima, sendo que o usuário interage exclusivamente com as páginas *JSP*, sendo o *servlets* os seus *controler*, os quais fazem a relação entre os eventos e os dados gerados pela interação do usuário com o sistema e a camada de negócios (*Models*) criados pelo mapeamento das tabelas.

Para que fosse possível o uso do mesmo banco de dados pelo módulo *mobile* do sistema, foi criado um novo projeto, visando dar suporte a conexão via *Socket* de dispositivos móveis.

Este novo projeto reutiliza códigos do módulo *Web*. Os códigos reaproveitados são das classes *Model* e das classes *DAO,* mas alterando os *controlers* da aplicação, pois agora o modo de visualização é *mobile*.

Para que fosse possível o sistema aceitar conexões de dispositivos móveis, foi criada uma classe que cria um servidor de *Socket*, aceitando assim as requisições de conexão dos dispositivos. A Listagem 8 apresenta códigos referentes a criação de um servidor *Socket* para múltiplos acessos.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Servidor implements Runnable {

    private DataInputStream in;
    private DataOutputStream out;
    private ServerSocket server;
    private Socket cliente;
    private static final int port = 1099;
    private String erro = "";

    private volatile boolean stopped = false;

    @Override
    public void run(){
        try {
            //Cria servidor
            server = new ServerSocket(port);
            System.out.println("Servidor Criado.");
            while(!stopped){
                if (server != null) {
```

```
                    cliente = server.accept();
                    new Thread(new ListenerSocket(cliente)).start();
                }
            }
        } catch (IOException ex){
            System.out.println("Erro: " + ex.getMessage());
        }
    }

    private class ListenerSocket implements Runnable {

        public ListenerSocket(Socket cliente){
            try{
                in = new DataInputStream(cliente.getInputStream());
                System.out.println("IP: " + cliente.getInetAddress().getHostAddress()
+ " conectou-se!");

                System.out.println("Aguardando");
                String acao = in.readUTF();

                acaoRequerida(acao);
            } catch (IOException | ClassNotFoundException ex){
                System.out.println("Erro: " + ex.getMessage());
            }
        }

        @Override
        public void run() {
            System.out.println("Requisição atendida.");
        }

    }

}
```

Listagem 8 - Código Servidor Socket

Após ter sido executado, o servidor irá aceitar novas requisições até o momento em que for finalizado.

A interface do módulo *mobile*, é toda desenvolvida em *XML*, cabendo ao compilador interpretá-las de modo que seja possível manipulá-las em tempo de execução, para isso o *Android* gera uma classe chamada de arquivo *R.java*, atualizando-a toda vez que é recompilada, sendo que todos os elementos do projeto estarão indexados, tornando possível que o *Android* ao executar o aplicativo possa interpretar corretamente as informações.

A Listagem 9 mostra a criação da tela de *login* do módulo *mobile*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="#E0EEEE"
    tools:context=".MainActivity" >


 <ImageView
         android:id="@+id/banner"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:src="@drawable/banner"
 />

 <LinearLayout
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:orientation="horizontal"
         android:background="#E0EEEE" >

    <TextView
         android:id="@+id/tvConf"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Configura y ш‌es do servidor:    " />

    <Button
        android:id="@+id/btServer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="serverOnClick"
        android:text="Alterar Configura y ш‌es" />

 </LinearLayout>


 <TextView
         android:id="@+id/tvUsuario"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_alignParentTop="true"
         android:layout_marginTop="38dp"
         android:text="Usuario:" />

 <EditText
         android:id="@+id/etUsuario"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
```

```
        android:inputType="text" >

    <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/tvSenha"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Senha:"
     />

    <EditText
        android:id="@+id/etSenha"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />

    <Button
        android:id="@+id/btEntrar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Entrar"
        android:onClick="entrarOnClick"
     />

    <Button
        android:id="@+id/btSair"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Sair"
        android:onClick="sairOnClick"
     />

    </LinearLayout>
```

Listagem 9 - Activity_login.xml

A seguir, a Listagem 10 apresenta o arquivo *R.java* gerado para o módulo *mobile* do sistema. Esta classe possui subprogramas que indexam os recursos da aplicação, tornando possível referenciá-los e utilizá-los. Esses recursos são todos os elementos contidos dentro do diretório *res* da aplicação, ou seja, imagens, telas da aplicação, entre outros.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.rabello.estagio;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        /**  Default screen margins, per the Android Design guidelines.

             Customize dimensions originally defined in res/values/dimens.xml (such as
             screen margins) for sw720dp devices (e.g. 10" tablets) in landscape here.

         */
        public static final int activity_horizontal_margin=0x7f040000;
        public static final int activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        public static final int adicionarproduto=0x7f020000;
        public static final int banner=0x7f020001;
        public static final int finalizarvenda=0x7f020002;
        public static final int ic_launcher=0x7f020003;
        public static final int iniciarvenda=0x7f020004;
        public static final int produto=0x7f020005;
        public static final int removerproduto=0x7f020006;
        public static final int sair=0x7f020007;
        public static final int usuario=0x7f020008;
        public static final int venda=0x7f020009;
    }
    public static final class id {...}
    public static final class layout {...}
    public static final class menu {...}
    public static final class string {...}
    public static final class style {...}
}
```

Listagem 10 - Arquivo *R.java*

O arquivo *AndroidManifest.xml* é o local onde estão contidas algumas das mais primordiais informações do projeto. É nele em que ficam armazenados informações como o nome do pacote da aplicação, versão da aplicação, serviços e permissões de uso da aplicação assim como a restrição de qual o requisito mínimo do sistema para executá-lo.

A Listagem 11 apresenta o *AndroidManifest.xml* gerado para o módulo *mobile* do sistema.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rabello.estagio"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="7"
        android:targetSdkVersion="7" />

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.rabello.estagio.MainActivity"
            android:label="Gerência de Bares" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.rabello.estagio.LoginActivity"
            android:label="@string/title_activity_login" >
        </activity>
        <activity
            android:name="com.rabello.estagio.Server2Activity"
            android:label="@string/title_activity_server2" >
        </activity>
        <activity
            android:name="com.rabello.estagio.MyInfoActivity"
            android:label="@string/title_activity_my_info" >
        </activity>
        <activity
```

```
            android:name="com.rabello.estagio.PasswordActivity"
            android:label="@string/title_activity_password" >
        </activity>
        <activity
            android:name="com.rabello.estagio.MenuVendaActivity"
            android:label="@string/title_activity_menu_venda" >
        </activity>
        <activity
            android:name="com.rabello.estagio.IniciarVendaActivity"
            android:label="@string/title_activity_iniciar_venda" >
        </activity>
        <activity
            android:name="com.rabello.estagio.AdicionarProdutoActivity"
            android:label="@string/title_activity_adicionar_produto" >
        </activity>
        <activity
            android:name="com.rabello.estagio.FinalizaVendaActivity"
            android:label="@string/title_activity_finaliza_venda" >
        </activity>
        <activity
            android:name="com.rabello.estagio.AdicionarProdutoCompActivity"
            android:label="@string/title_activity_adicionar_produto_comp" >
        </activity>
    </application>

</manifest>
```

Listagem 11 - AndroidManifest.xml

Para toda tela criada, o *Android* cria além do arquivo *XML*, que contem os elementos gráficos da tela, também uma classe *Java* com o mesmo nome, onde é possível manipular e gerenciar a interface. É nessa classe que se trata e processa as requisições do usuário. Essa classe é estendida de *Activity*, herdando todas as suas funcionalidades, podendo assim sobre escrevê-las caso seja necessário.

A Listagem 12 mostra a classe MainActivity.java que é a classe do menu principal da aplicação (Figura 33).

```
package com.rabello.estagio;

import java.util.ArrayList;
import java.util.List;

import Adapter.CustomListViewAdapter;
import RowDetails.RowItem;
import android.app.Activity;
import android.app.AlertDialog;
```

```
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;

public class MainActivity extends Activity {

         private static final String[] menu = {"Usuário", "Venda", "Sair"};
         private static final int[] imagens = {R.drawable.usuario, R.drawable.venda,
R.drawable.sair};
         private ListView lvPrincipal;
         List<RowItem> rowItems;

         @Override
         protected void onCreate(Bundle savedInstanceState) {

                  super.onCreate(savedInstanceState);
                  setContentView(R.layout.activity_main);

                  rowItems = new ArrayList<RowItem>();
        for (int i = 0; i < menu.length; i++) {
            RowItem item = new RowItem(imagens[i], menu[i]);
            rowItems.add(item);
        }

        lvPrincipal = (ListView) findViewById(R.id.lvPrincipal);
        CustomListViewAdapter     adapter     =     new     CustomListViewAdapter(this,
R.layout.elemento, rowItems);
        lvPrincipal.setAdapter(adapter);

                  lvPrincipal.setOnItemClickListener(                              new
AdapterView.OnItemClickListener() {

                           @Override
                           public void onItemClick(AdapterView<?> arg0, View arg1,
int posicao, long arg3) {

                                    switch (posicao) {
                                    case 0:
                                             myInfoOnClick();
                                             break;
                                    case 1:
                                             vendaOnClick();
                                             break;
                                    case 2:
                                             sairOnClick();
                                             break;
                                    default:
                                             break;
                                    }
```

```
                        }
                });

                if(!verificaLogin()){
                        chamaLogin();
                }

        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
                // TODO Auto-generated method stub
                super.onActivityResult(requestCode, resultCode, data);

                String usuario = "";
                String senha = "";
                boolean flag = false;

                if ( resultCode == RESULT_OK ) {
                        usuario = data.getStringExtra("usuario");
                        senha = data.getStringExtra("senha");

                        armazenaDados(usuario, senha);

                } else if ( resultCode == 0 ){
                        this.finish();
                        flag = true;
                }

                if(!flag){
                        if(!verificaLogin()){
                                chamaLogin();
                        }
                }

        }

        @Override
        public void onResume(){
                super.onResume();
        }


        private boolean verificaLogin();
        private void armazenaDados(String usuario, String senha);
        private void chamaLogin();
        private void myInfoOnClick();

        private void sairOnClick(){
```

```
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setMessage("Deseja realmente sair?")
                  .setIcon(android.R.drawable.ic_dialog_alert)
                  .setPositiveButton("Yes",new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {
                                loggout();
                                finish();
                        }
                    })
                .setNegativeButton("No", null)
                .show();
        }

        private void vendaOnClick();
        private void loggout();

}
```

Listagem 102 - MainActivity.java

Uma conexão via *Socket* com o servidor é realizada quando se tem a necessidade de buscar informações, seja ela sobre uma mesa ou um produto ou mesmo sobre o usuário logado, salvar dados de uma venda ou alterar dados de vendas.

A seguir, a Listagem 13 demonstra o uso de *Socket* em uma das *Activity* da aplicação.

```
@SuppressWarnings("unchecked")
private void carregaMesas(){
        SharedPreferences prefs = getSharedPreferences("dadosServer", MODE_APPEND);
        ip = prefs.getString("ipServer", "0");
        port = Integer.parseInt(prefs.getString("portaServer", "0"));

        String resposta = "";

        try {

                if (socket == null){
                        socket = new Socket(ip, port);
                }

                out = new DataOutputStream(socket.getOutputStream());
                in = new DataInputStream(socket.getInputStream());

                out.writeUTF("mesasIniciadas");
```

```
                    out.flush();

                    resposta = in.readUTF();

                    if (resposta.equals("OK")){

                            if (socket != null){
                                    ObjectInputStream        ino        =        new
ObjectInputStream(socket.getInputStream());
                                    Object object = ino.readObject();

                                    if (object instanceof ArrayList){
                                            lista = (ArrayList<Mesa>) object;
                                    } else {
                                            lista = new ArrayList<Mesa>();
                                    }

                                    if (!lista.isEmpty()){
                                            itens = new String[lista.size()];
                                    } else {
                                            itens = new String[1];
                                            itens[0] = "Nenhuma venda encontrada.";
                                            btFinalizarMesa.setEnabled(false);
                                    }

                                    for (int i = 0; i < lista.size(); i++){
                                            itens[i] = lista.get(i).getDescricao();
                                    }

                                    ino.close();
                            } else {
                                    Toast.makeText(this,  "Falha  em  conectar-se  ao
servidor.", Toast.LENGTH_SHORT).show();
                            }

                    }

                    in.close();
                    out.close();
                    socket.close();

            } catch (IOException ex){
                    ex.printStackTrace();
            } catch (ClassNotFoundException ex)  {
                    ex.printStackTrace();
            } finally {
                    socket = null;
            }
}
```

Listagem 113 - Código carregarMesas

Todas as telas da aplicação respeitam as determinações descritas acima, com algumas exceções pois, algumas telas não necessitam buscar dados ou inserir dados no banco.

As telas que necessitam interagir com o banco dependem que o servidor *Socket*, descrito acima na Listagem 7, esteja funcionando para que possam funcionar perfeitamente.

4.5 Estrutura do sistema

O sistema apresentado, é composto por três aplicações distintas, em que cada uma das aplicações atende à um módulo do sistema. As aplicações dos módulos *Web* e do Servidor de Serviços foram desenvolvidos utilizando o padrão de desenvolvimento *Model-View-Controller* (MVC).

O módulo *Web* é dividido em três partes, a primeira é constituída por páginas JSP, arquivos de estilização (CSS), arquivo de *Java Script* para validação de usuário e suas permissões no sistema e as estruturas modelo dos relatórios, é o *View* do módulo.

A segunda parte da aplicação do módulo Web, é composta por classes Java, como por exemplo os *Models*, *HibernateUtil* e arquivos *JSON*, *Servlets* e arquivo de configuração do *Hibernate* também encontram-se nesta parte, é aqui que se encontram os *Model* e *Controllers* do módulo. Por fim, a terceira parte é onde ficam as bibliotecas utilizadas pela aplicação, como por exemplo as bibliotecas de uso do *Hibernate* e do *JasperReports*. A Figura 40 mostra a estrutura da aplicação do módulo *Web*.

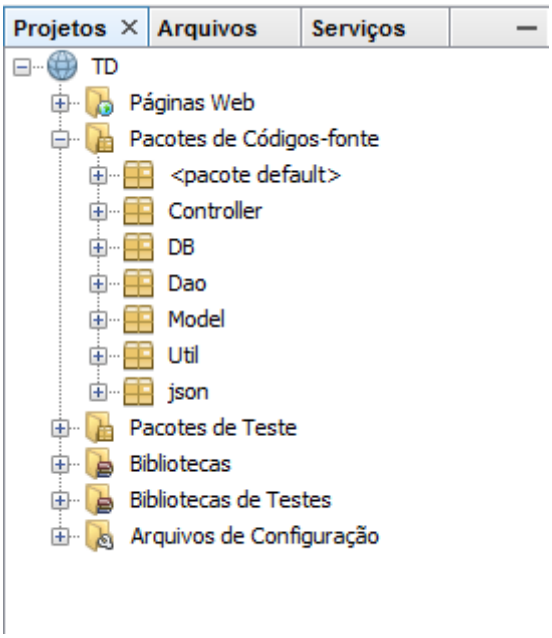

Figura 40 - Estrutura da aplicação do módulo *Web*

O módulo do Servidor de Serviços é semelhante ao Web pois reutiliza códigos do módulo *Web* para atender requisições do módulo *Mobile*, diferenciando-se por possuir tela única, sendo que ela fica visível apenas no servidor do sistema. É composto apenas por classes *Java* e arquivos *xml*.

Sua estrutura também é utiliza o padrão MVC, sendo que o pacote *gui* possui a tela única, *View* do módulo, e o restante da estrutura é idêntica ao módulo Web, se diferenciando apenas por não possuir as mesmas funções, pois o módulo *Mobile* é diferente do *Web*. A Figura 41 mostra a estrutura do módulo.

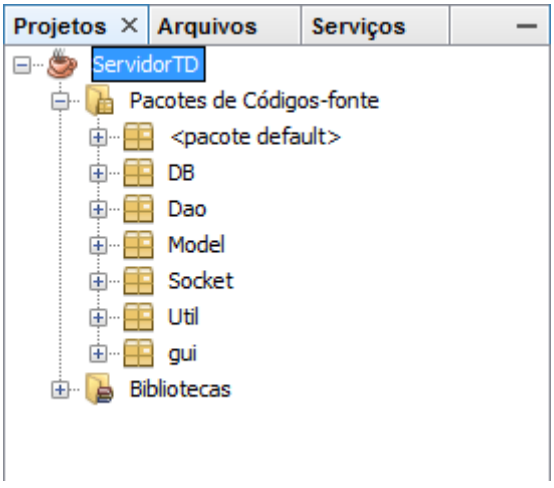

Figura 41 - Estrutura da aplicação do Servidor de Serviços

O módulo Mobile não segue um padrão de desenvolvimento, é apenas dividido em pacotes. Sendo que no pacote *res/layout* é onde os arquivos *xml* das telas ficam dispostos e o pacote *src* as classes Java, como por exemplo as classes estendidas de *Activity* e *Adapters*. A Figura 42 monstra a estrutura do módulo.

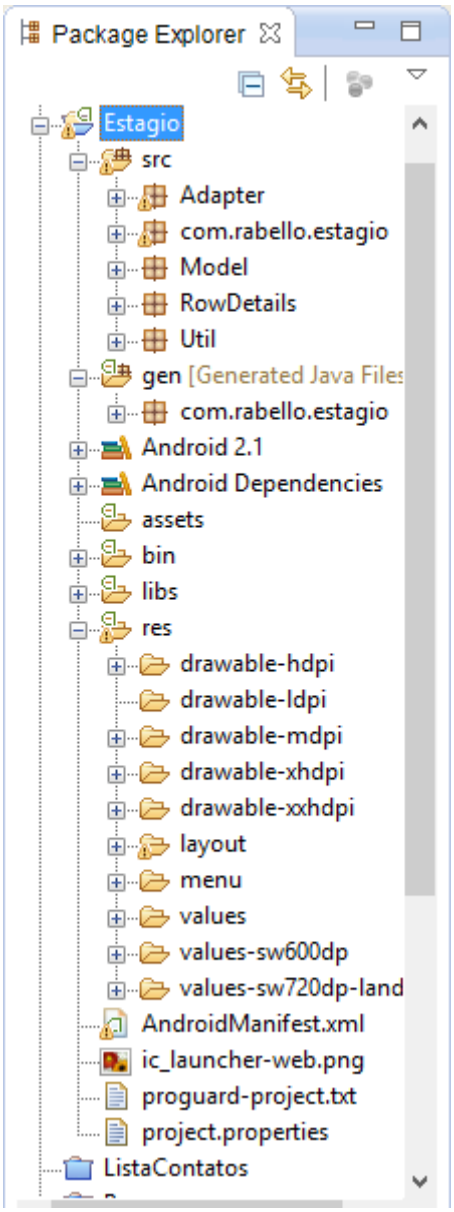

Figura 42 - Estrutura da aplicação do módulo *Mobile*

4.6 Implantação do Sistema

Para que o sistema possa ser implantado em um estabelecimento, o mesmo precisa possuir ao menos uma estação de mesa e um dispositivo móvel que utilize o Sistema Operacional *Android* com versão superior a 2.1 e um sistema de rede local com roteamento sem fio (*Wi-fi*). O ideal é que o estabelecimento possua ao menos duas estações de mesa, uma sendo exclusiva para o Servidor de Serviços, que é quem executa o Banco de Dados (MySQL), *Apache TomCat* e Servidor de *Sockets* e outra que irá servir como cliente. Nada impede que um mesmo computador execute as funções de Cliente e Servidor ao mesmo tempo. O sistema não possui um limite máximo de clientes utilizando o sistema, já que o limitador é a velocidade da rede local e do servidor ou o seu limite desempenho de *hardware*.

O sistema também permite que o Servidor esteja em outra localidade sem ser o estabelecimento, mas assim tornando necessário que o mesmo possua acesso a Internet. O sistema também pode ser implantado em qualquer Sistema Operacional de computador, já que por sua vez o sistema é todo desenvolvido com linguagens *Open Source*, grande parte em Java que é multiplataforma, e os serviços *MySQL* e *Apache TomCat* também possuem distribuições para todos os S.O.

5 CONCLUSÃO

O objetivo desse trabalho foi desenvolver uma aplicação para a gestão de bares, que fizesse a integração das tecnologias *Web* e *mobile*. Por meio desse sistema, os usuários podem gerar vendas aos clientes do estabelecimento, agilizando assim o processo de atendimento, tanto na hora de realizar um pedido como na hora de finalizar sua compra.

Para que fosse possível o desenvolvimento desse sistema, foi adotado como base para esse sistema linguagens que fossem *Open Source*, afim de que não houve-se a necessidade de aquisição de licenças. O outro fator que foi determinante para do uso de linguagens desse tipo, foi a adoção para o módulo *mobile* do sistema o uso do sistema operacional (SO) *Android*, já que por sua vez ele é o SO mais utilizado no mundo para essa plataforma.

Foi feito uso da linguagem de programação *Java*, por ela ser *Open Source*, compatível com o *Android* e oferecer recursos muito bons para a implementação do módulo *Web* do sistema, como classes de comunicação específicas (ex. *Servlets*) e compatibilidade com servidores acessíveis como o TomCat.

O desenvolvimento deste trabalho possibilitou desafios, como uso e aprendizado de novas tecnologias, que trouxeram como frutos novos conhecimentos e diferentes experiências, assim como o reforço das tecnologias aprendidas ao longo das disciplina destinadas à área, demonstrando a importância de uma boa análise e projeto e do emprego da orientação a objetos na implementação de um sistema, além da devida importância nas técnicas de programação.

Este trabalho proporcionou também novas expectativas, por ser de natureza complexa, é um trabalho que necessita de total força criativa e de vontade em completar os desafios por ele proposto, como a busca, o aprendizado e o emprego de novas soluções técnicas, mesmo nunca tendo as vista em disciplinas, traz a vontade de aplicar esse novo conhecimento

adquirido em novos projetos futuros, demonstrando que o período investido na instituição, foi bem aproveitado e de grande valia.

Como trabalhos futuros pretende-se aprimorar o sistema administrativo, implementando novas funcionalidades ao sistema, tais como a melhoria no controle da entrada de estoque, menu de opções para contas a pagar e receber, serviço de mensagens entre os usuários do sistema e o aperfeiçoamento do módulo *mobile*, de forma a dar suporte mais efetivo aos usuários do sistema. Ainda como possíveis trabalhos futuros, pretende-se fazer um estudo e posteriormente ampliação do sistema para que possa atender restaurantes e lanchonetes também.

6 REFERÊNCIAS

ABLESON, Frank. **Desenvolver Aplicações Android com o Eclipse.** Disponível em: <http://www.ibm.com/developerworks/br/library/os-eclipse-android/>. Acesso em: 27 set. 2014.

ARAÚJO, Everton Coimbrade. **Sobrecarga, Herança, Polimorfismo e Exceção em C#.** Disponível em: <http://www.linhadecodigo.com.br/artigo/2622/sobrecarga-heranca-polimorfismo-e-excecao-em-csharp.aspx**>**. Acesso em 17 fev. 2014.

BEZERRA, Eduardo. **Princípios de Analise e Projeto de Sistemas com UML - 2ª Edição.** São Paulo: Campus, 2007.

BOOCH, Grady. RUMBAUGH, James. JACOBSON, Ivar. **UML - Guia do usuário- 2ª Edição.** Rio de Janeiro: Elsevier, 2005.

GUEDES, Gilleanes Thorwald Araújo. **UML Uma abordagem prática - 3ª Edição.** São Paulo: Novatec, 2008.

IBM. **Unified Modeling Language.** Disponível em: <http://www-01.ibm.com/software/rational/uml/>. Acesso em: 01 out. 2013.

MACIASZEK, Leszek. **Requirements Analysis and System Design: Developing Information Systens with UML - 1ª Edição.** Sydney: Addison Wesley, 2001.

MACORATTI, José Carlos. **UML - Conceitos Básicos.** Disponível em: <http://www.macoratti.net/vb_uml2.htm>. Acesso em :17 nov. 2013.

MATIEVICZ, Tainá Roberta S. **Sistema de Informação Aliado à Gestão Empresarial.** Disponível em: http://www.consisanet.com/sistema-de-informacao-gestao-empresarial/. Acessado em 30/03/2014.

OMG - **Introdução à UML.** Disponível em: <http://www.uml.org>. Acesso em: 01 out. 2013.

PALMEIRA, Thiago Vinícius Varallo. **Introdução à Programação Orientada a Objetos em Java.** Disponível em: <http://www.devmedia.com.br/introducao-a-programacao-orientada-a-objetos-em-java/26452>. Acesso em 08 dez. 2013.

SOMMERVILLE, Ian. **Engenharia de Software - 8ª Edição.** São Paulo: Pearson, 2007.

WAZLAWICK, Raul Sidnei. **Análise e Projeto de Sistemas de Informação Orientados a Objetos.** Rio de Janeiro: Elsevier, 2004.