UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

CONTROLE DE IMUNIZAÇÕES E VACINAS UTILIZANDO *SMART CARDS*

EDUARDO PANIZ MALLMANN

BLUMENAU
2010

2010/1-08

EDUARDO PANIZ MALLMANN

CONTROLE DE IMUNIZAÇÕES E VACINAS UTILIZANDO *SMART CARDS*

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Marcel Hugo, Mestre - Orientador

BLUMENAU
2010

2010/1-08

CONTROLE DE IMUNIZAÇÕES E VACINAS UTILIZANDO *SMART CARDS*

Por

EDUARDO PANIZ MALLMANN

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

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Adilson Vahldick, Mestre – FURB

Blumenau, 06 de julho de 2010

Dedico este trabalho a todos que me apoiaram e acreditaram nele desde o seu início.

AGRADECIMENTOS

A força superior que rege o Universo.

A minha família que esteve sempre presente em minha vida.

A minha namorada Luana, que esteve presente nos melhores momentos de minha vida.

Aos meus amigos, pelas cobranças, empurrões e apoios quando necessário.

Ao meu orientador, Marcel Hugo, pelos conselhos e credibilidade na conclusão deste trabalho.

“Se sabemos exatamente o que vamos fazer, para quê fazê-lo?”

Pablo Picasso

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação para controle de vacinas e imunizações presente na caderneta de saúde do cidadão utilizando *smart cards*. Para a implementação foi utilizado a tecnologia Java Card para comunicação direta com o *smart card* e Java para desenvolver um applet para camada de interface ao usuário que permite ao mesmo realizar os controles necessários via web. Os dados são salvos no *smart card* e em uma base de dados externa como forma de garantia.

Palavras-chave: *Smart card*. Java card. Controle de vacinas.

ABSTRACT

This work describes development of an application to control vaccines and immunizations presented at citizen health handbook using smart cards. To implement have been used Java Card technology to communicate directly to smart card and Java to develop an applet for user interface layer allowing to make necessaries control over the web. Data are saved at smart card and external database by way of security.

Key-words: Smart card. Java card. Vaccines and immunizations control.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

APDU – *Application Protocolo Data Unit*

API – *Application Programming Interface*

CAD – *Card Acceptance Device*

CE – *Classic Edition*

CNH – Carteira Nacional de Habilitação

ConE – *Connected Edition*

CPF – Cadastro de Pessoa Física

CSC – Caderna de Saúde da Criança

DES – *Data Encryption Standard*

EEPROM – *Electrical Erasable Programmable Read-Only Memory*

GSM – *Global System for Mobile Communications*

HTML – *HyperText Markup Language*

JCRE – Java *Card Runtime Environmnet*

JCVM – Java *Card Virtual Machine*

MER – Modelo de Entidade de Relacionamento

RAM – *Random Access Memory*

RG – Registro Geral

RIC – Número Único de Registro de Identidade Civil

ROM – *Read-Only Memory*

SIM – *Subscriber Identity Module*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, vive-se em uma sociedade que precisa conviver com vários documentos pessoais no dia a dia, sendo que a grande maioria é baseada em modelos antigos, como papel ou um cartão de plástico com uma fita magnética.

Este modelo está em desuso e o próprio governo federal reconhece isso, ao aprovar a Lei 9454/1997, concebida para integrar em um único documento, os bancos de dados de diversos órgãos de sistemas de identificação do Brasil. Esta lei determina que seja criado o Número Único de Registro de Identidade Civil (RIC), onde contemplará documentos como o Registro Geral (RG), Cadastro de Pessoa Física (CPF), Carteira Nacional de Habilitação (CNH) e título do eleitor, além da impressão digital do cidadão. Para permitir esta unificação de documentos, será utilizado um *smart card*, um cartão de plástico do tamanho de um cartão de crédito, com um microchip (CERTISIGN, 2008).

Analisando esta perspectiva do governo federal, pode-se observar a atual caderneta de saúde. Ela é constituída por uma série de cadastros pertinentes a identificação e saúde pessoal do cidadão, conforme instituído pela Portaria nº 964/GM de 23 de junho de 2005 (PORTAL DA SAÚDE, 2009). Segundo Alves *et al* (2009), "A Caderneta de Saúde da Criança (CSC) é um documento imprescindível para a promoção da saúde infantil". Um dos conteúdos da mesma é o controle de imunizações e vacinas, onde ficam registradas todas as imunizações e vacinas efetuadas pelo portador da caderneta de saúde. A cada visita do cidadão ao serviço de saúde, a caderneta é atualizada e em casos de vacinas ou imunizações, carimbada com a determinada vacina ou imunização. Porém, ela é comumente esquecida pelos cidadãos pelo fato de que ela não é prática para ser transportada o tempo todo, onde muitos esquecem onde ela está guardada. Isso ocasiona em uma caderneta desatualizada e muitas vezes com informações inválidas.

Verificando-se estes fatos, torna-se necessário atualizar o formato deste documento, facilitando o seu transporte e o armazenamento das informações contidas na mesma.

A solução proposta é utilizar *smart cards* para substituir este documento, visto que o *smart card* é do tamanho de um cartão de crédito, o que facilita o seu transporte, pois pode facilmente ser carregado em uma carteira pessoal e também pode armazenar todas as informações necessárias, visto que possui um microchip para conter as informações. Propõe-se também utilizar a *Application Programming Interface* (API) Java *Card* para fazer acesso ao *smart card*, tanto para leitura quanto para gravação das informações, por ser uma

tecnologia que facilita estas operações e trabalha de forma simples com o protocolo *Application Protocolo Data Unit* (APDU), responsável por fazer a comunicação de mensagens entre a aplicação e o *smart card*.

Pelo fato de ser um documento pessoal, faz-se necessário aplicar uma criptografia sobre os dados, deste modo evitando que as informações fiquem acessíveis a qualquer pessoa, garantindo uma segurança e integridade sobre as informações.

Outro fato a ser considerado, é que o cidadão tem o direito de visualizar as informações no cartão, afinal diz respeito a ele. Para isto, caso o cidadão possua um leitor de *smart cards*, ele poderá acessar um sistema visualizador do controle de imunizações e vacinas, através de um portal eletrônico.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é informatizar o atual controle de imunizações e vacinas, presente na caderneta de saúde de uma pessoa física, através de *smart card* utilizando a tecnologia Java *Card*.

Os objetivos específicos do trabalho são:

a) utilizar métodos atuais de criptografia para garantir a segurança e integridade das informações;
b) desenvolver um sistema para registrar novas imunizações e vacinas no *smart card* que será utilizado pelos profissionais dos postos de saúde;
c) desenvolver um sistema para visualização das informações contidas no *smart card*, para ser utilizado pelo cidadão dono do cartão.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão.

O capítulo 2 apresenta a fundamentação teórica sobre *smart cards*, a tecnologia Java *Card*, o protocolo APDU, a caderneta de saúde e trabalhos correlatos a este.

Os principais requisitos deste trabalho, assim como a diagramação do mesmo através da *Unified Modeling Language* (UML) e detalhes de desenvolvimento estão abordados no capítulo 3.

No capítulo 4 são apresentadas conclusões finais com os resultados obtidos bem como sugestões para implementações futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Com relação à fundamentação teórica serão apresentados assuntos que são de grande importância para o entendimento geral do trabalho. Os assuntos que serão abordados são:

a) caderneta de saúde;
b) *smart cards*;
c) Java Card;
d) APDU;
e) trabalhos correlatos.

2.1 CADERNETA DE SAÚDE

A caderneta de saúde é um documento que acompanha o cidadão desde a fase infantil até a fase idosa.

Ela é dividida em Caderneta de Saúde da Criança, Caderneta de Saúde do Adolescente e Caderneta de Saúde da Pessoa Idosa. Cada caderneta possui uma legislação própria, com seus próprios conteúdos, porém todas apresentam o controle de imunizações e vacinas.

São constituídas por uma série de cadastros pertinentes a identificação e saúde pessoal do cidadão, conforme instituído pela Portaria nº 964/GM de 23 de junho de 2005 (PORTAL DA SAÚDE, 2009).

É distribuída em todos os postos de saúde do Brasil e em casos de recém nascidos, é distribuída nas maternidades dos hospitais, já devidamente preenchida (ALVES *et al*, 2009).

Um exemplo da seção de vacinas e imunizações de uma caderneta de saúde pode ser visto na Figura 1.

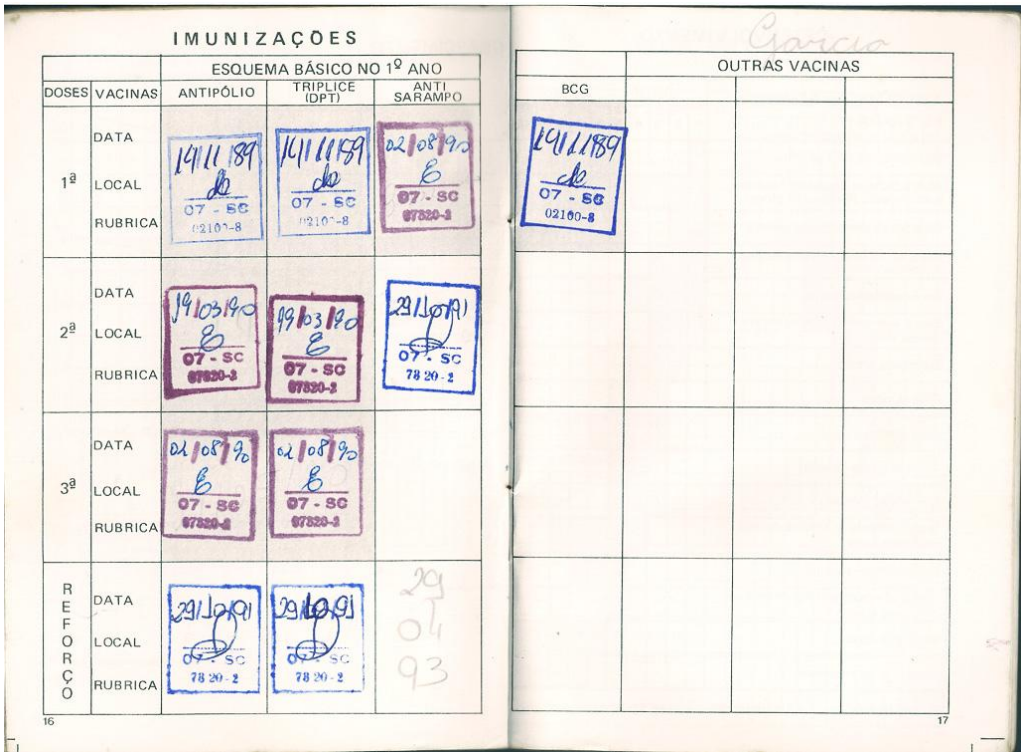


Figura 1 – Exemplo da seção de vacinas e imunizações de uma caderneta de saúde

2.2 SMART CARDS

Smart cards (cartões inteligentes) são dispositivos móveis com tamanho e dimensão semelhante à de cartões com tarja magnética, porém com a diferença de possuírem um chip com capacidade para armazenagem e processamento (CHEN, 2000).

Começaram a ser desenvolvidos em 1970 e nos anos 80 já eram utilizados como cartões pré pagos para celulares e cartões de crédito. O exemplo de um *smart card* pode ser visto na Figura 2.

Fonte: Sonsun (2010).

Figura 2 – Exemplo de um *smart card*

Segundo Chen (2000, p. 14-16), os *smart cards* possuem oito pontos de contatos e uma unidade central de processamento. Alguns tipos especiais de *smart cards* que são utilizados para aplicações seguras, possuem um co-processador de criptografia, especializado em operações criptográficas. Possuem ainda três tipos de memórias: *Read-Only Memory* (ROM), *Electrical Erasable Programmable Read-Only Memory* (EEPROM) e *Random Access Memory* (RAM).

A memória ROM não pode ser escrita, apenas lida. Ela é utilizada para gravar informações básicas do cartão, como rotinas de sistema e dados do fabricante. Ela guarda informações sem precisar de uma fonte de energia externa. A EEPROM também não precisa de uma fonte de energia externa para guardar informações, porém pode ser regravada. Nela é que ficam os dados e aplicativos dos usuários. Estima-se que a sua vida útil é de cem mil ciclos de escrita e pode guardar informações por até dez anos. A RAM é uma memória temporária de acesso rápido, utilizada pelos aplicativos e sem restrições, como a EEPROM e não armazena dados sem uma fonte de energia externa (CHEN, 2000).

O acesso às informações contidas no *smart card* são feitas através de um *Card Acceptance Device* (CAD). CADs podem ser do tipo leitores ou terminal. Leitores são dispositivos que possuem uma entrada para o *smart card* e estão conectados a um computador. Terminais são computadores com leitores integrados a si, como um caixa eletrônico.

Cada *smart card*, define como será feita a comunicação com o seu dispositivo, de forma que o desenvolvedor deve ter esta informação para criar suas aplicações. Para padronizar este acesso, foi criada a *Global Platform* que padroniza o acesso a estes dispositivos, independente do fabricante. Porém, o fabricante pode escolher em utilizar ou não este padrão.

Atualmente os *smart cards* estão sendo utilizados principalmente por instituições financeiras em forma de cartões de crédito, débito e acesso geral a caixas de auto-atendimento. Estes cartões utilizam o padrão ISO 7816, porém geralmente não implementam a *Global Platform* por uma questão de segurança. Desta forma somente desenvolvedores que possuem o algoritmo próprio do cartão conseguem acessar o mesmo.

São utilizados em larga escala também em celulares com a tecnologia *Global System for Mobile Communications* (GSM). Porém conforme Suavi (2005, p. 31), o formato não está no padrão ISO 7816 e são mais conhecidos por *Subscriber Identity Module* (SIM).

Outra forma de utilização cada vez mais crescente é por empresas para controle de seus funcionários, como por exemplo, cartões pontos e vales refeições.

No exterior é possível encontrar estes *smart cards* já implementando a última versão do Java *Card*, a 3.0, porém no Brasil somente foi encontrado *smart cards* implementando a versão 2.2.1 do Java *Card*, o que mostra um certo atraso tecnológico do nosso país em relação a esta tecnologia.

2.3 JAVA CARD

Java *Card* é uma tecnologia utilizada em *smart cards* e outros dispositivos que possuem memória e processamento limitados. Com esta tecnologia é possível criar pequenas aplicações, chamadas de *applets*, compatíveis com os padrões da ISO 7816 partes 1, 2 e 3 (SUN MICROSYSTEMS, 2004).

Atualmente encontra-se na especificação 3.0 e é divida em duas edições: *Classic Edition* (CE) e *Connected Edition* (ConE). CE consiste em uma evolução da especificação 2.2.2, com correções de erros e novos algoritmos de segurança. ConE apresenta uma nova abordagem, com uma nova máquina virtual e um ambiente de execução mais robusto. Apresenta também suporte à aplicações web, incluindo a API Java Servlet. Todas as funcionalidades presentes na especificação CE estão presentes na ConE (SUN MICROSYSTEMS, 2008).

Possui uma máquina virtual própria, a Java *Card Virtual Machine* (JCVM). Segundo Suavi (2005, p. 21), a JCVM é divida em duas partes, sendo a primeira o conversor responsável por efetuar um pré-processamento das classes e gerar um *applet* e a segunda sendo responsável pelo registro e instalação do *applet.*

Possui também uma *Runtime Environment* própria, a Java *Card Runtime Environment* (JCRE), ambiente que é executado dentro um *smart card.* A JCRE é responsável pelo gerenciamento de recursos, execução dos *applets*, segurança dos *applets*, e comunicações de rede. Segundo Chen (2000, p. 36), a JCRE é o sistema operacional de um *smart card.*

2.4 APDU

O protocolo APDU, conforme definido pela ISO 7816-4, é um protocolo a nível de

aplicação para troca de mensagens entre um *smart card* e uma aplicação local. As mensagens APDU compreendem duas estruturas: uma usada pela aplicação local para enviar dados ao *smart card* através do CAD: *command* APDU; e a outra é usada pelo cartão para enviar uma resposta a aplicação local: *response* APDU (CHEN, p. 18).

A estrutura dos APDUs é apresentada nos Quadros 1 e 2.

Cabeçalho obrigatório				Corpo opcional		
CLA	INS	P1	P2	Lc	Data field	Le

Quadro 1 - Estrutura *command* APDU

Corpo opcional	Trailer	
Data field	SW1	SW2

Quadro 2 – Estrutura *response* APDU

O *command* APDU pode ser subdividido em cabeçalho e corpo. O cabeçalho consiste de quatro bytes, sendo um byte para cada componente, sendo eles:

a) CLA: classe;
b) INS: código de instrução a ser executada no *smart card*;
c) P1 e P2: representam dois parâmetros que podem ser passados para complementar a instrução a ser executada.

O corpo do *command* APDU pode variar de tamanho e apresenta os seguintes componentes:

a) Lc: contém um byte identificando o tamanho da mensagem que será enviada ao *smart card;*
b) *Data field*: contém a informação que será enviada ao *smart card* para ser executada conforme a instrução passada no cabeçalho;
c) Le: especifica o número de bytes esperados pela aplicação na resposta do *smart card*.

O *response* APDU pode ser subdividido em um corpo opcional e o trailer. No corpo opcional virá a informação de resposta do *command* APDU, com o tamanho informado no componente Le.

O trailer é composto de dois componentes, o SW1 e o SW2, que são chamados de *status word*, indicando o estado do processo realizado, como por exemplo retornar o código 0x9000, indicando que o comando foi bem executado.

2.5 TRABALHOS CORRELATOS

A seguir são apresentados dois trabalhos com características semelhantes aos objetivos deste trabalho: “Documentos e Dinheiro Eletrônico com *Smart Card* Utilizando Tecnologia Java *Card*” (SUAVI, 2005) e “Smart Interface: Ferramenta de Auxílio ao Desenvolvimento de Aplicações Java Card” (MINORA; ALEIXO; DIOLINO, 2007).

O Trabalho de Conclusão de Curso (TCC) de Cleber Giovani Suavi com o tema Documentos e Dinheiro Eletrônico com *Smart Card* Utilizando Tecnologia Java *Card* desempenha um papel bem semelhante ao proposto neste trabalho.

Em seu projeto, Suavi (2005) apresenta aspectos gerais sobre o uso de *smart cards* e como eles podem ser utilizados em formas de documento e dinheiro eletrônico, garantindo sua segurança e integridade, através de algoritmos de criptografia e certificados digitais. O trabalho preocupa-se mais com a especificação de *smart cards,* Java *Cards e applets,* e não com o desenvolvimento de uma aplicação a ser utilizada de forma real.

Segundo Suavi (2005, p. 86), foi possível armazenar as informações e criptografar as mesmas, porém sem o certificado digital, visto que a API Java *Card* não oferece esta funcionalidade.

Minora, Aleixo e Diolino (2007, p. 80), propõem uma ferramenta para auxiliar o desenvolvedor que utiliza a tecnologia Java *Card*. Esta ferramenta tem como finalidade ser um *upgrade* da ferramenta SMART SHELL, utilizada para trabalhar com comandos APDU de forma mais abstrata, abstraindo-se as cadeias de *bytes* por comandos em alto nível como AUTH ou INSTALL, porém esta ferramenta é textual, dificultando o trabalho com a mesma.

A solução proposta por Minora, Aleixo e Diolino (2007) é uma ferramenta visual e um aperfeiçoamento dos comandos já existentes no SMART SHELL. Esta ferramenta executa comandos com o *smart card* da mesma forma proposta neste trabalho, porém de uma forma mais abstrata.

3 DESENVOLVIMENTO

Neste capítulo são abordados os principais pontos no desenvolvimento do presente trabalho. Primeiramente são apresentados os requisitos do problema a ser trabalhado e a sua especificação. Na seqüencia a implementação, onde são comentadas as técnicas e ferramentas utilizadas, o desenvolvimento e a operacionalidade da ferramenta. Por fim são discutidos os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos apresentados a seguir são divididos em Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF).

3.1.1 Disponibilizar uma interface para seleção do leitor de cartão (RF)

O *applet* deve permitir ao usuário selecionar a leitora que deseja utilizar para realizar as operações caso possua mais de uma leitora instalada e configurada.

3.1.2 Disponibilizar uma interface para cadastrar ou alterar os dados pessoais de um portador do cartão (RF)

O *applet* deve permitir ao agente de saúde, cadastrar ou alterar informações pessoais do portador do cartão, sendo que estas informações devem ficar salvas no cartão e em uma base externa.

As informações necessárias são: nome do portador, sexo do portador, tipo sanguíneo do portador, data de nascimento do portador, RG do portador, CPF do portador, nome da mãe do portador, data de nascimento da mãe do portador, nome do pai do portador e data de nascimento do pai do portador.

3.1.3 Disponibilizar uma interface para cadastro de imunizações e vacinas para serem utilizadas no sistema (RF)

O *applet* deve permitir ao agente de saúde, cadastrar vacinas e imunizações que possam ser utilizadas posteriormente no sistema para registrar vacinas e imunizações no cartão do portador.

Estes registros ficarão salvos em uma base externa, com um código, descrição da vacina ou imunização e data de atualização do registro.

3.1.4 Disponibilizar uma interface para registrar imunizações e vacinas de um determinador portador (RF)

O *applet* deve permitir ao agente de saúde registrar uma vacina ou imunização a um determinado portador do cartão. O valor registrado da vacina ou da imunização será o código da mesma, conforme RF 3.1.3.

Estas informações ficarão salvas tanto no cartão quanto em uma base externa.

3.1.5 Disponibilizar um sistema visualizador das informações contidas no *smart card* (RF)

As informações contidas no *smart card* podem ser visualizadas tanto pelo portador do mesmo quanto por um agente de saúde. Desta forma faze-se necessário haver um sistema para visualizar estas informações que será um *applet* executado via web.

Para que o *applet* possa obter as informações é necessário que na máquina da pessoa haja pelo menos um CAD instalado e configurado. Com o CAD instalado e configurado é também necessário que a pessoa tenha o *smart card* onde estão salvas as informações.

3.1.6 Implementar rotina de criptografia nas informações para garantir sua integridade e segurança (RNF)

As informações enviadas para o *smart card* devem passar por uma rotina de criptografia previamente antes de serem salvas. Este processo é necessário para que haja um

mínimo de segurança nas informações contidas no *smart card*, havendo a possibilidade de extravio do *smart card.*

A rotina de criptografia utilizada deve ser reversível, ou seja, deve ser possível descriptografar as informações posteriormente.

3.1.7 Ser implementado na linguagem de programação Java SE 6 (RNF)

O *applet* que irá realizar as comunicações com o *smart card* deve ser implementado utilizado a linguagem de programação Java SE 6, por ser a versão mais atual no mercado.

A partir desta versão há uma API específica para trabalhar com este tipo de dispositivo, facilitando a codificação.

3.1.8 Utilizar banco de dados MySql em sua versão 5 (RNF)

Como base de dados externa ao *applet*,deve ser utilizado o banco de dados MySql em sua versão 5, por ser uma solução *free* e também por ser a sua versão mais moderna no mercado atual.

Esta base de dados externa será utilizada em partes do sistema como uma forma de backup das informações registradas no mesmo.

3.2 ESPECIFICAÇÃO

Esta seção descreve a especificação do trabalho através da UML. São apresentados os diagramas de casos de uso, atividades, classes e modelo de entidade de relacionamento (MER).

A modelagem dos diagramas de casos de uso, atividades e classes foi feita utilizando a ferramenta *Enterprise Architect.* Para o MER foi utilizada a ferramenta *DBDesigner*.

3.2.1 Diagrama de casos de uso

O diagrama apresentado na Figura 3 apresenta as principais ações que os usuários terão ao utilizar o sistema.

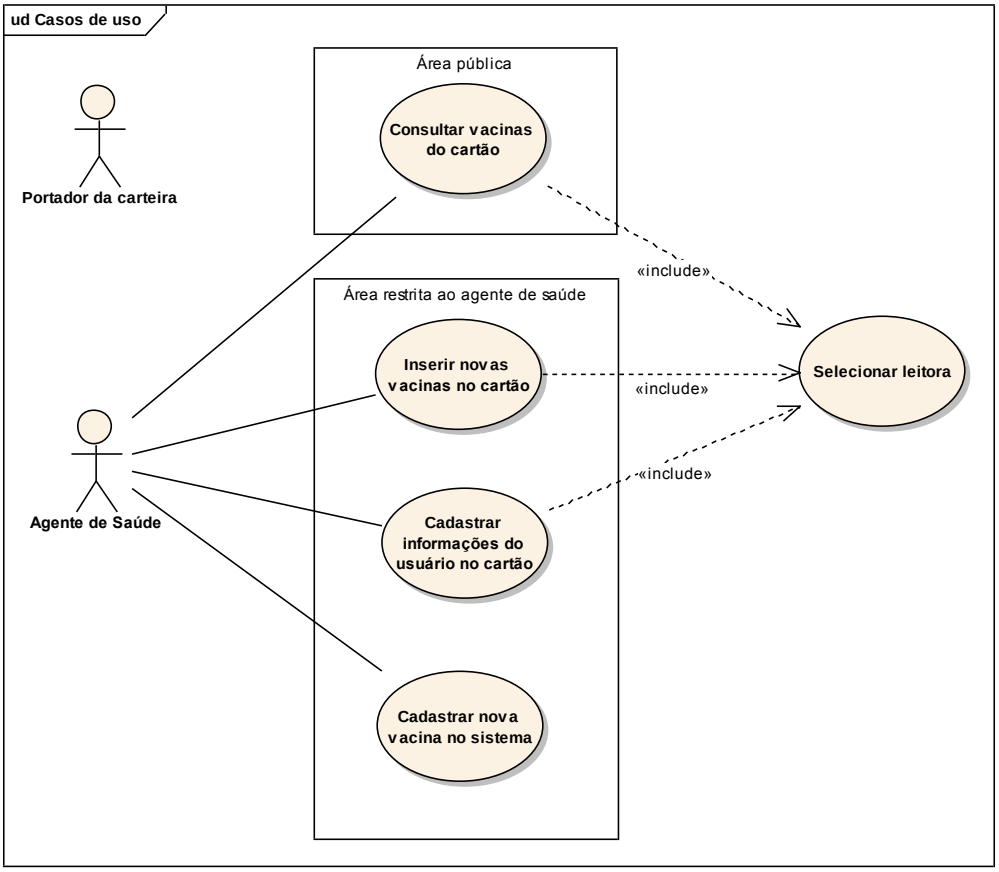


Figura 3 – Diagrama de casos de uso

O diagrama foi divido em duas áreas, sendo elas área pública e área restrita ao agente de saúde.

A área pública contém o caso de uso `Consultar vacinas do cartão`, que será utilizado tanto pelo portador da carteira quanto pelo agente de saúde. Este caso de uso é utilizado para que tanto o portador da carteira quanto o agente de saúde possam visualizar as vacinas que estão armazenadas no cartão. Esta consulta de vacinas é feita sem a intervenção de uma base de dados externas, acessando somente os dados gravados diretamente no *smart card*. Realiza *include* do caso de uso `Selecionar leitora`, pois para visualizar as informações é necessário que haja um CAD instalado e configurado na máquina a ser utilizada para realizar a consulta.

Na área restrita ao agente de saúde, estão os casos de uso que somente um agente de saúde poderá realizar no sistema e no cartão.

No caso de uso `Cadastrar nova vacina no sistema`, o agente de saúde poderá cadastrar novas vacinas no sistema, que serão utilizadas posteriormente para registrar vacinas que já foram aplicadas no portador da carteira. Este cadastro utiliza uma base de dados externa e não necessita que haja um CAD instalado e configurado à máquina.

O caso de uso `Inserir novas vacinas no cartão` trata do agente de saúde registrar as vacinas que o portador da carteira já recebeu. As vacinas que podem ser registradas são as que já estão cadastradas no sistema. Para registrar a vacina, o prestador deve selecionar a vacina e informar a dose da mesma, caso não seja dose única. A data de aplicação da mesma será registrada pelo próprio sistema. Ao registrar uma vacina, a mesma é salva tanto no cartão do portador quanto em uma base externa de dados, por questões de segurança. Este caso de uso realiza *include* do caso de uso `Selecionar leitora`, pois necessita que haja um CAD instalado e configurado na máquina.

Por último, o caso de uso `Cadastrar informações do usuário no cartão` é utilizado para realizar o cadastro das informações do usuário. As informações que devem ser cadastradas são: nome do portador, sexo, tipo sanguíneo, data de nascimento, RG, CPF, nome da mãe, data de nascimento da mãe, nome do pai e data de nascimento do pai. Ao gravar estas informações as mesmas são salvas no cartão e em uma base de dados externa, por questões de segurança. Este caso de uso faz *include* do caso de uso `Selecionar leitora`, pois necessita que haja um CAD instalado e configurado na máquina.

No Quadro 3 são apresentados os cenários de cada caso de uso para o portador da carteira.

CASO DE USO	**CENÁRIOS**
Consultar vacinas do cartão	**Consultar vacinas do cartão (Principal)** 01 – O usuário realiza login no sistema. 02 – O usuário seleciona a opção Consultar vacinas do cartão. 03 – O sistema apresenta as vacinas cadastradas no cartão. **Cartão não presente (Exceção)** No passo 01, caso não haja um cartão presente no CAD o sistema não permite realizar o login e informa o usuário do ocorrido.

Quadro 3 – Cenário do caso de uso do portador da carteira

No Quadro 4 são apresentados os casos de uso de um agente de saúde.

CASOS DE USO	**CENÁRIOS**
Cadastrar nova vacina no sistema	**Cadastrar nova vacina no sistema (Principal)** 01 – Agente de saúde faz login no sistema. 02 – Agente de saúde seleciona opção para cadastrar nova vacina no sistema. 03 – Sistema apresenta tela para informar o nome da vacina. 04 – Agente informa o nome da vacina e salva a mesma. 05 – Sistema salva a vacina em uma base de dados externa.
Cadastrar informações do usuário no cartão	**Cadastrar informações do usuário no cartão (Principal)** 01 – Agente de saúde faz login no sistema. 02 – Agente de saúde seleciona opção para cadastrar informações pessoais. 03 – Agente de saúde informa os dados pessoais do portador e salva as informações. 04 – Sistema envia as informações para o *smart card*. 05 – *Smart card* recebe as informações, as criptografa e salva as mesmas no cartão. **Cartão inválido (Exceção)** No passo 03, caso o cartão seja um inválido, como por exemplo o cartão de crédito do portador, o sistema aborta a gravação e informa ao usuário o ocorrido.
Inserir novas vacinas no cartão	**Inserir novas vacinas no cartão (Principal)** 01 – Agente de saúde faz login no sistema. 02 – Agente de saúde seleciona opção para inserir nova vacina no cartão. 03 – Sistema apresenta tela para escolher vacina e a dose para inserir.

	04 – Sistema salva a vacina no cartão.

Quadro 4 – Cenários dos casos de uso do agente de saúde

3.2.2 Diagramas de atividades

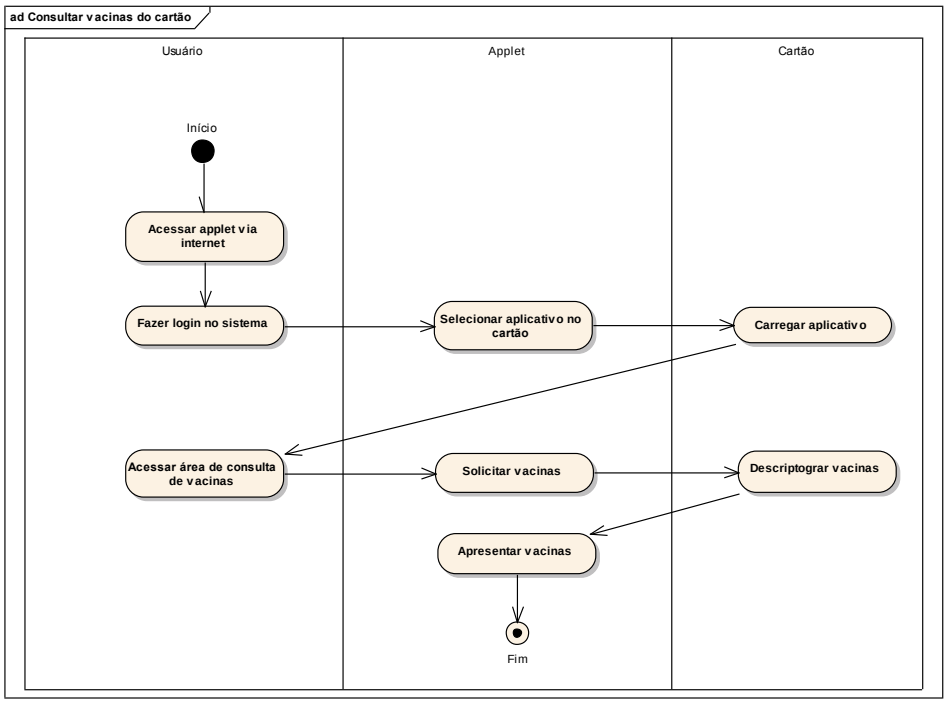


Figura 4 – Diagrama de atividades para consulta de vacinas registradas no cartão

Na Figura 4 encontra-se o diagrama de atividades para consulta de vacinas registradas no cartão.

Neste processo o usuário, tanto portador quanto agente de saúde, acessa o *applet* pela internet e realiza o seu *login* no sistema. Após realizar o login, o *applet* acessa o CAD da máquina do usuário e o seu respectivo cartão, solicitando que o aplicativo do cartão seja inicializado. Na sequência, o usuário deve acessar a área de consulta de vacinas, fazendo com que o *applet* solicite as vacinas que estão cadastradas no cartão. Estas informações estão criptografadas no cartão, portanto o aplicativo do mesmo deve descriptografar estas informações antes de enviá-las para o *applet*. Ao término deste processo, o *applet* apresenta as vacinas para o usuário.

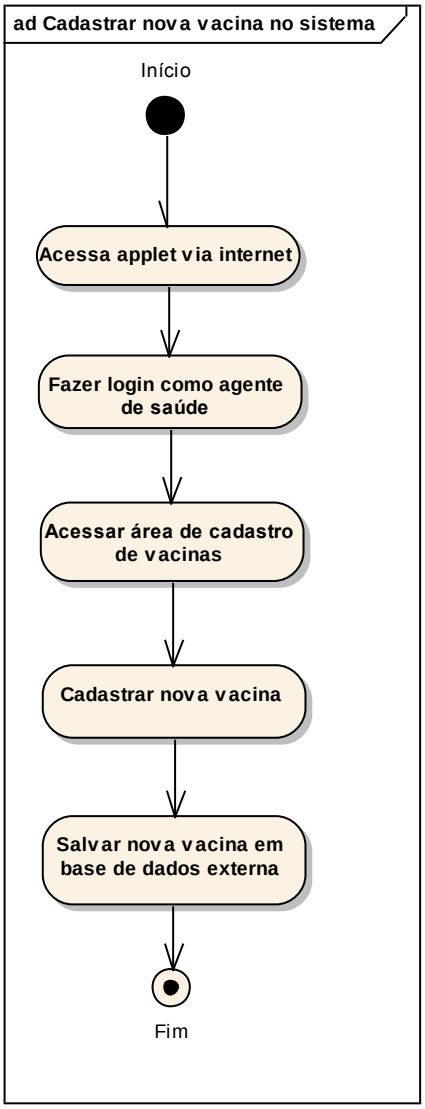


Figura 5 – Diagrama de atividades para cadastro de novas vacinas

Na Figura 5 encontra-se o diagrama de atividades para cadastro de novas vacinas no sistema. O fluxo representado é feito por um agente de saúde, acessando primeiramente o *applet* via internet e realizando seu devido *login*. Deve acessar a área de cadastro de vacinas e adicionar uma nova vacina ao sistema. Esta nova vacina fica salva em uma base externa de dados.

O diagrama de atividades para inserir nova vacina no cartão, conforme figura 6, demonstra o processo feito pelo agente de saúde para registrar uma nova vacina no cartão do portador.

O agente de saúde deve acessar o *applet* via internet e fazer o seu *login*. Nesse momento o *applet* acessa o CAD da máquina do usuário e o seu respectivo cartão, solicitando que o aplicativo do cartão seja inicializado. O agente deve acessar a área para registrar vacinas e escolher uma vacina disponível. Ao registrar esta vacina, o *applet* irá enviar esta vacina para o cartão, onde o mesmo irá criptografar a vacina recebida e então irá salvar a

mesma no cartão. Ao receber o retorno do cartão de que a vacina foi salva, o *applet* salva a mesma em uma base externa de dados.

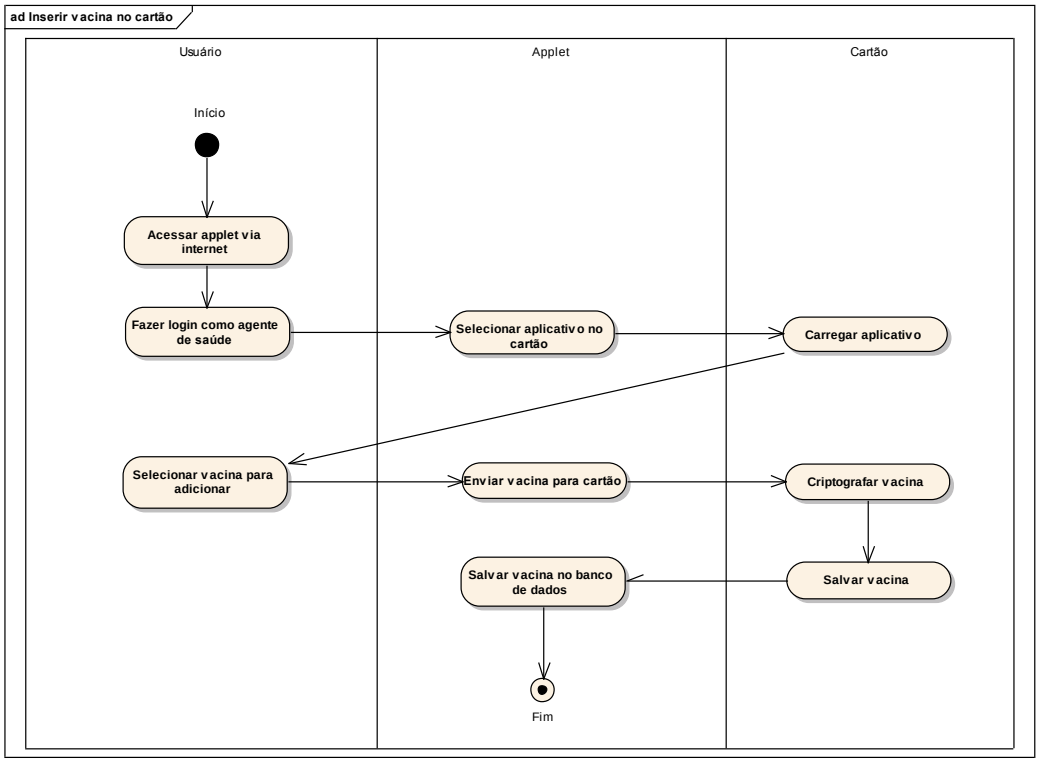


Figura 6 – Diagrama de atividades para inserir nova vacina no cartão

O processo de cadastrar as informações do portador da carteira, conforme diagrama de atividades de cadastro de informações do portador da Figura 7, demonstra o fluxo para um agente de saúde cadastrar as informações pessoais do portador.

O agente acessa o *applet* via internet e faz o seu *login*. Nesse momento o *applet* acessa o CAD da máquina do usuário e o seu respectivo cartão, solicitando que o aplicativo do cartão seja inicializado. Ao acessar a área de cadastro de informações do portador, o *applet* deve verificar no cartão se já existem informações cadastradas. Caso já existam, deve solicitar ao aplicativo do cartão estas informações. Para isto, o aplicativo do cartão deve descriptografar as mesmas e enviá-las ao *applet* para que sejam visualizadas.

Após essa verificação o agente pode então cadastrar as informações do portador ou alterar as informações já previamente cadastradas, salvando-as posteriormente, fazendo com que o *applet* envie estas informações para o aplicativo do cartão, que irá criptografar as informações, salvá-las e retornar ao *applet* a mensagem de que foram salvas. Nesse momento, o *applet* irá salvar as mesmas informações em uma base externa de dados.

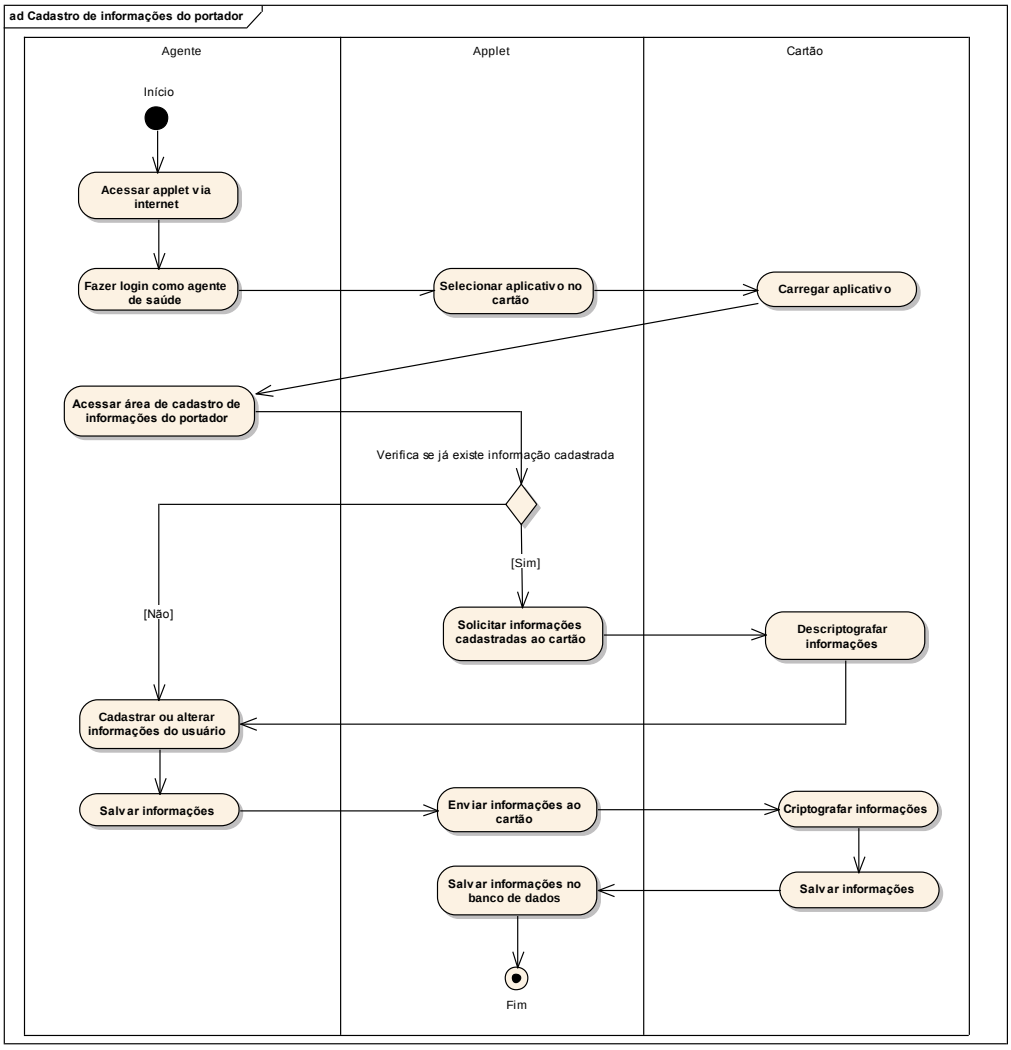


Figura 7 – Diagrama de atividades para cadastro de informações do portador

3.2.3 Diagramas de classes

A Figura 8 mostra o diagrama de classes do *applet* para internet.

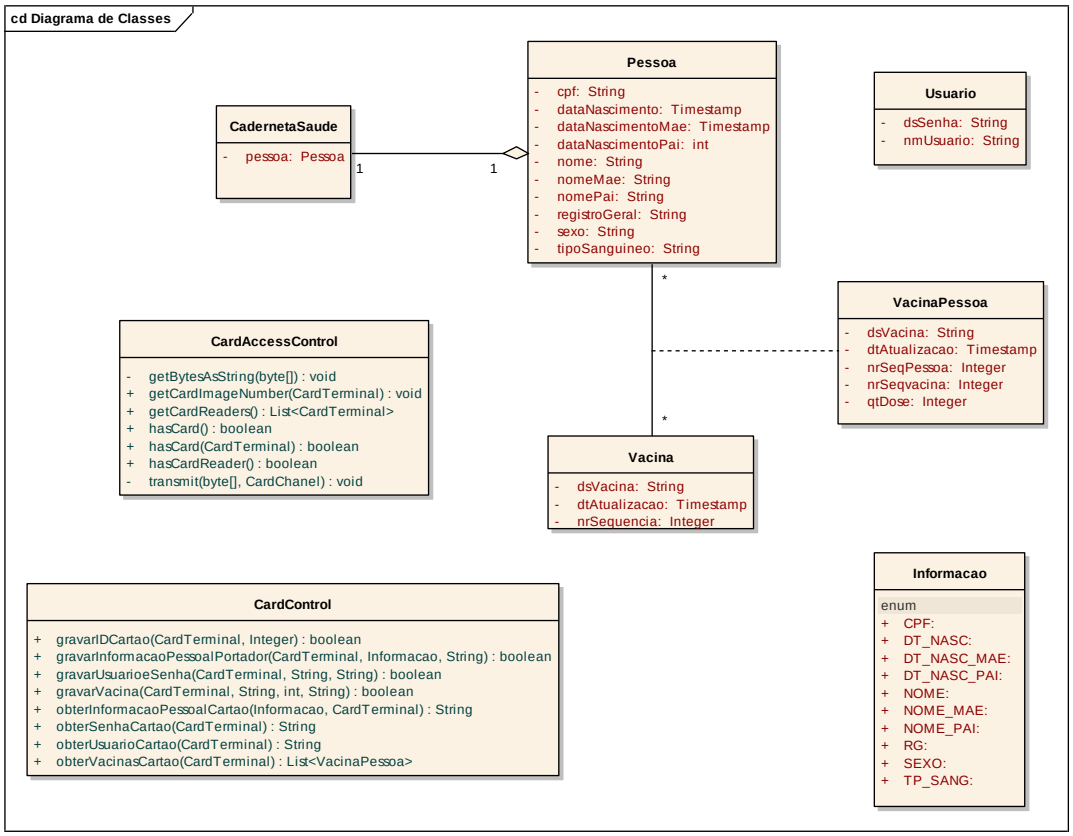


Figura 8 – Diagrama de classes do *applet* para internet

Para a aplicação a ser carregada no *smart card* foi divido em *packages*, sendo elas `br.com.card`, `br.com.card.control`, `br.com.card.model` e `br.com.card.crypto`. As figuras 9, 10, 11 e 12 apresentam as classes respectivamente.

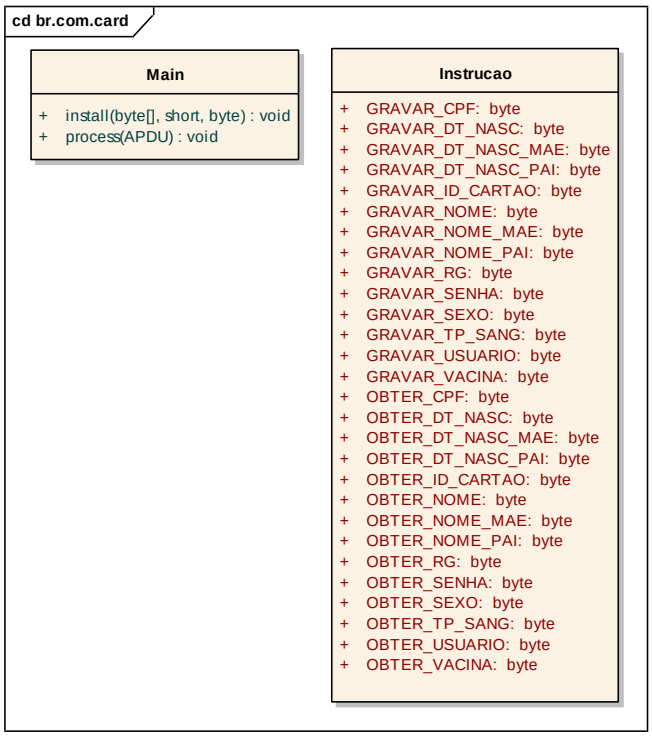


Figura 9 – Diagrama de classes da *package* `br.com.card`

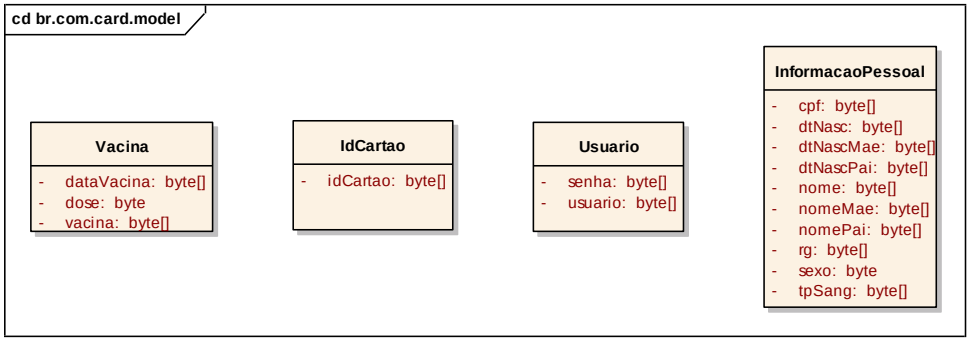


Figura 10 – Diagrama de classes da *package* `br.com.card.model`

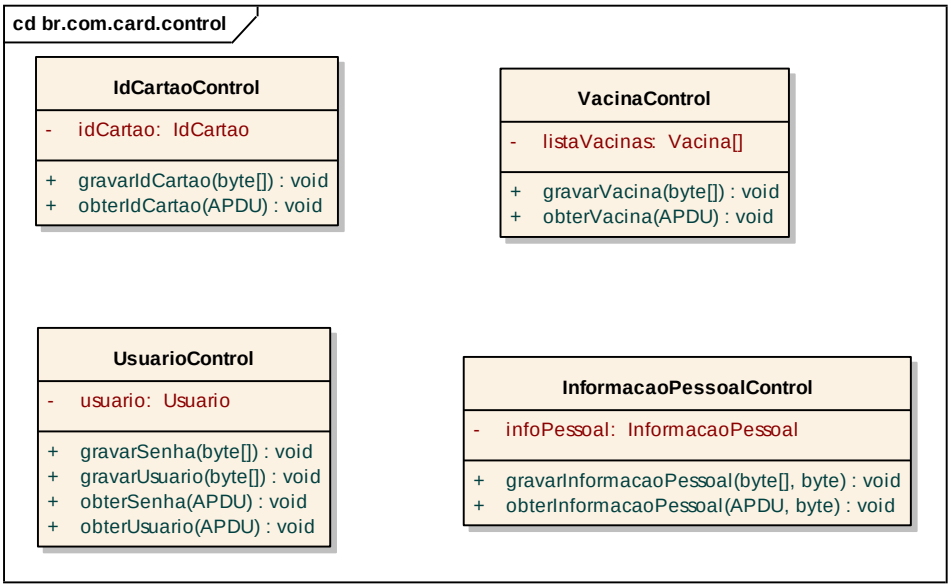


Figura 11 – Diagrama de classes da *package* `br.com.card.control`

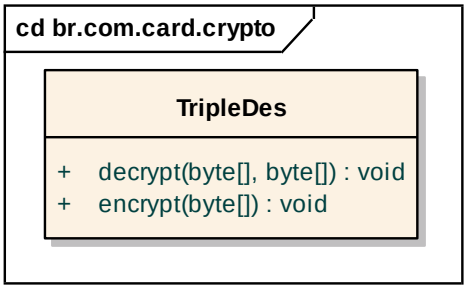


Figura 12 – Diagrama de classes da *package* `br.com.control.crypto`

3.2.4 MER

Na Figura 13 encontra-se o modelo de entidade de relacionamento.

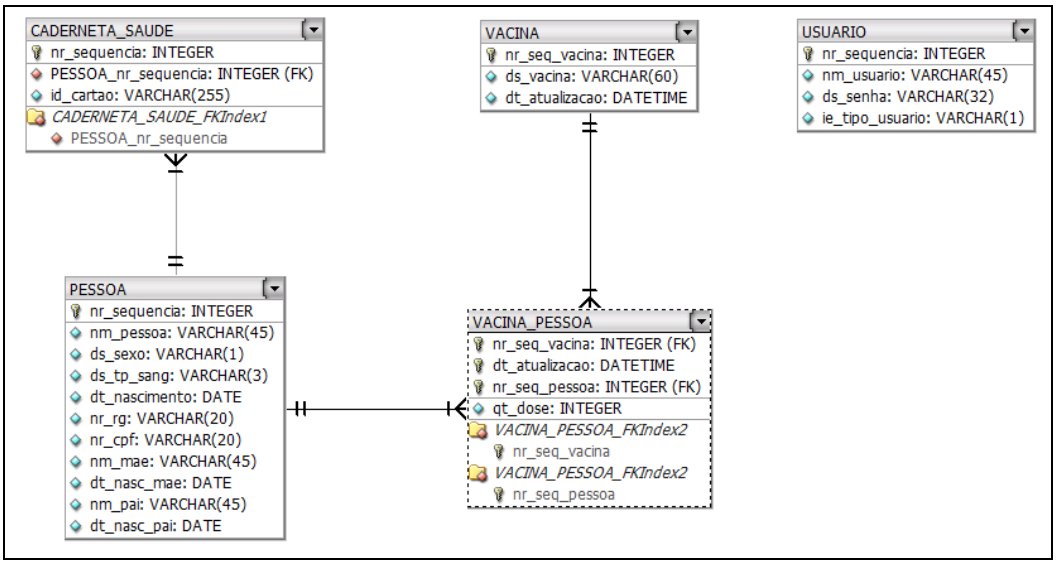


Figura 13 – Modelo de entidade de relacionamento

Para este trabalho foi utilizado um banco de dados externo por uma questão de segurança, de possuir as informações que estão no *smart card* salvas em uma base de dados externa. Desta forma, quando um agente de saúde realiza os seus devidos processos, as informações são tanto salvas no banco de dados quanto no cartão, garantindo uma maior segurança sobre os dados da aplicação. Para os processos do portador da carteira, o banco de dados não é utilizado, utilizando-se somente do *smart card* do mesmo.

Nos Quadros 5, 6, 7, 8 e 9 é apresentado o dicionário de dados das entidades.

Entidade: CADERNETA_SAUDE Descrição: Entidade responsável por armazenar o identificado único de cada cartão.			
Atributo	Domínio	Tamanho	Descrição
Nr_sequencia	Numérico		
Id_cartao	Texto	255	Id único do cartão

Quadro 5 – Entidade `CADERNETA_SAUDE`

Entidade: VACINA_PESSOA Descrição: Entidade responsável por armazenar todas as vacinas de um portador de carteira.			
Atributo	Domínio	Tamanho	Descrição
Nr_seq_vacina	Numérico		Referência a vacina
Nr_seq_pessoa	Numérico		Referência a pessoa
Dt_atualizacao	Data		Data em que o registro foi alterado
Qt_dose	Numérico		Dose da vacina

Quadro 6 – Entidade `VACINA_PESSOA`

Entidade: PESSOA Descrição: Entidade responsável por armazenar as informações pessoais do portador da carteira.			
Atributo	Domínio	Tamanho	Descrição
Nr_sequencia	Numérico		
Nm_pessoa	Texto	45	Nome da pessoa
Ds_sexo	Texto	1	M ou F
Ds_tp_sang	Texto	3	Tipo sanguíneo
Dt_nascimento	Data		Data de nascimento
Nr_rg	Texto	20	RG com máscara
Nr_cpf	Texto	20	CPF com máscara
Nm_mae	Texto	45	Nome da mãe
Dt_nasc_mae	Data		Data de nascimento da mãe
Nm_pai	Texto	45	Nome do pai
Dt_nasc_pai	Data		Data de nascimento do pai

Quadro 7 – Entidade PESSOA

Entidade: VACINA Descrição: Entidade responsável por armazenar as vacinas que serão utilizadas no sistema.			
Atributo	Domínio	Tamanho	Descrição
Nr_seq_vacina	Inteiro		
Ds_vacina	Texto	60	Descrição da vacina
Dt_atualizacao	Data		Data em que o registro foi alterado

Quadro 8 – Entidade VACINA

Entidade: USUARIO Descrição: Entidade responsável por armazenar as informações de login do portador da carteira.			
Atributo	Domínio	Tamanho	Descrição
Nr_sequencia	Numérico		
Nm_usuario	Texto	45	Usuário do sistema
Ds_senha	Texto	32	HASH da senha gerada em MD5
Ie_tipo_usuario	Texto	1	A ou C

Quadro 9 – Entidade USUARIO

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas, implementação e a operacionalidade da implementação.

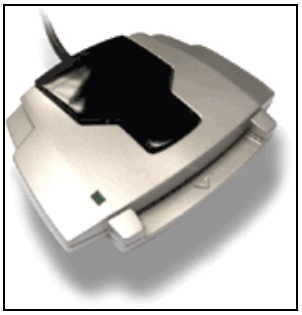
3.3.1 Técnicas e ferramentas utilizadas

A implementação do *applet* para internet foi realizada utilizando-se a linguagem de programação Java 6, assim como a implementação do módulo para que fosse possível visualizar o *applet* a partir uma página *HyperText Markup Language* (HTML). Para o aplicativo que é interpretado pela JCVM do *smart card* foi utilizada a linguagem de programação Java *Card* 2.2.1.

Para a codificação do *applet* para internet e do módulo foi utilizado o ambiente de programação NetBeans 6.8. Para o aplicativo *smart card* foi utilizado o ambiente de programação Eclipse Galileo. Para carregar o aplicativo no *smart card* foi utilizada a ferramenta JCManager. Como servidor de aplicação para o projeto Web foi utilizado o *Apache Tomcat* 6.0.20.

Foram utilizados dois ambientes de programação pelo fato de que é necessário gerar os arquivos compilados para o *smart card* utilizando o ambiente Java *Card*, o qual não estava sendo respeitado no NetBeans.

Para realizar os testes e os acessos foi utilizado um CAD modelo ACR 38, conforme Figura 14 e o *smart card* JCOP21 V2.3.1 36KB. O *smart card* implementa a versão 2.2.1 do Java *Card* e a especificação 2.1.1 da *Global Platform*. Possui como algoritmos de criptografia *Single*, *Dual* e *Triple Data Encryption Standard* (DES). Trabalha com os protocolos de comunicação T=0 e T=1 (SONSUN, 2010).



Fonte: Sonsun (2010).

Figura 14 – CAD ACR 38

3.3.2 Desenvolvimento da aplicação

O desenvolvimento da aplicação foi separado em três partes, o desenvolvimento do *applet* para internet, o desenvolvimento da aplicação para o *smart card* e o desenvolvimento da aplicação Web para rodar o *applet*.

3.3.2.1 Desenvolvimento do *applet* para internet

Foi criado um projeto Java *Application* no NetBeans para trabalhar com um *applet* para internet. Para criar um *applet* deve-se possuir uma classe que será a principal do *applet*, que será instanciada no momento de criação do mesmo e carregará as características do mesmo.

Esta classe deve extender de `java.applet.Applet` ou `javax.swing.JApplet`. A classe `java.applet.Applet` é mais básica e trabalha somente com componentes visuais da arquitetura *Abstract Windowing Toolkit* (AWT), porém esta arquitetura foi descontinuada pela *Sun Microsystems* e não é recomendado utilizar a mesma para trabalhar com componentes visuais de tela. Para solucionar esta situação, foi criada a classe `javax.swing.JApplet` que utiliza a arquitetura *Swing* para componentes visuais de tela.

Para este trabalho foi criada a classe `Main`, que extende da classe `javax.swing.JApplet`, conforme Quadro 10.

```
 1 package br.com;
 2
 3 import javax.swing.JApplet;
 4 import javax.swing.SwingUtilities;
 5 import javax.swing.UIManager;
 6
 7 /**
 8  * @author Eduardo Paniz Mallmann
 9  */
10 public class Main extends JApplet {
11
12     @Override
13     public void init() {
14         try {
15             SwingUtilities.invokeAndWait(new Runnable() {
16
17                 public void run() {
18                     try {
19
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
20                     } catch (Exception e) {
21                     }
22                     setContentPane(new MainPanel());
23                 }
24             });
25         } catch (Exception e) {
26             System.err.println("Erro ao criar GUI");
27         }
28     }
29 }
```

Quadro 10 – Estrutura da classe `Main`

Esta classe sobrescreve o método `init` da classe `javax.swing.JApplet`. Isto se faz necessário pois um *applet* possui ciclos de vida. Cada ciclo pode ser programado conforme sua necessidade, sendo eles: `init`, `start`, `stop` e `destroy`. Para este trabalho foi necessário apenas sobrescrever o método `init`.

A classe `Main` instancia a classe `MainPanel`, que representa o painel que será utilizado em toda a aplicação. Todos os outros painéis utilizados na aplicação serão instanciados a partir da classe `MainPanel`. A troca dos painéis é realizada com o método `trocarPanel`, conforme Quadro 11.

```
32     public void trocarPanel(JPanel novoPanel) {
33         if (panelAtual != null) {
34             jPanelLayout.remove(panelAtual);
35         }
36         jPanelLayout.add(novoPanel, BorderLayout.CENTER);
37         panelAtual = novoPanel;
38         jPanelLayout.revalidate();
39         jPanelLayout.repaint();
40     }
```

Quadro 11 – Estrutura do método `trocarPanel`

Para trabalhar com o CAD e com o *smart card*, deve-se utilizar as classes presentes na *package* `javax.smartcardio` do Java. Esta package só está disponível na versão 6 do Java. Esta nova *package* facilita o trabalho com o *smart card* e o CAD pois adiciona classes com métodos diretos e fáceis de utilizar.

Para obter uma lista de todas as leitoras conectadas em uma máquina, utiliza-se a classe `TerminalFactory`, conforme foi utilizado na classe `CardAccessControl` no Quadro

12. Este método irá retornar uma lista de `CardTerminal`, que representa cada CAD conectado ao computador.

```
38 public static List<CardTerminal> getCardReaders() {
39         try {
40             return TerminalFactory.getDefault().terminals().list();
41         } catch (CardException e) {
42             e.printStackTrace();
43             return null;
44         }
45 }
```

Quadro 12 – Obter lista de CAD conectados ao computador

Para verificar se um CAD possui cartão presente, podemos acessar o método `isCardPresent()` da classe `CardTerminal`. Este método irá retornar um booleano indicando o resultado, conforme podemos ver no Quadro 13.

```
63     public static boolean hasCard(CardTerminal cardTerminal) {
64         try {
65             return cardTerminal.isCardPresent();
66         } catch (CardException ex) {
67             ex.printStackTrace();
68             return false;
69         }
70     }
```

Quadro 13 – Verificar presença de cartão em um CAD

Isto tudo permite acesso a classe `Card`, que representa um *smart card*. Esta classe será a utilizada para buscar o `CardChannel`, ou seja, o canal de comunicação com o *smart card*. Este canal de comunicação será responsável por enviar os comandos APDU ao *smart card* e receber os comandos de reposta APDU. Um exemplo deste funcionamento pode ser visto no Quadro 14.

```
24 private static byte[] GET_CARD_IMAGE_NUMBER = {0x00, 0x1B, 0x00, 0x00, 0x10};

72 public static String getCardImageNumber(CardTerminal cardTerminal) {
73         try {
74             if (cardTerminal.isCardPresent()) {
75                 Card card = cardTerminal.connect("*");
76                 CardChannel cc = card.getBasicChannel();
77                 ResponseAPDU response = transmit(cc, GET_CARD_IMAGE_NUMBER);
78                 String imageCardNumber = getBytesAsString(response.getBytes());
79                 card.disconnect(true);
80                 return imageCardNumber;
81             }
82             throw new IllegalArgumentException("Cartão não presente!");
83         } catch (CardException e) {
84             e.printStackTrace();
85             throw new IllegalArgumentException("Não foi possível obter image number do
cartão informado!", e);
86         }
87     }

89     private static ResponseAPDU transmit(CardChannel cc, byte[] command) throws
CardException {
90         CommandAPDU apdu = new CommandAPDU(command);
91         return cc.transmit(apdu);
92     }
93
94     private static String getBytesAsString(byte[] bytes) {
95         HexBinaryAdapter hex = new HexBinaryAdapter();
96         return hex.marshal(bytes);
97     }
```

Quadro 14 – Exemplo de envio de comando ao canal de comunicação do *smart card*

Na linha 75, obtêm-se o cartão que está conectado ao CAD. Para obter o mesmo, deve-

se utilizar o método `connect` da classe `CardTerminal`, passando como parâmetro o protocolo que será utilizado para realizar a comunicação. Este protocolo pode ser T=0, T=1 ou *. Neste caso está sendo utilizado *, onde o *smart card* irá decidir qual protocolo utilizar, desta forma sendo compatível com qualquer protocolo que o *smart card* trabalhe. É solicitado então ao *smart card* o seu canal de comunicação padrão, conforme linha 76. O comando em *bytes* que será enviado pode ser visto na linha 24, e será convertido a um comando APDU na linha 90, antes de ser enviado ao *smart card*, conforme linha 91. Este processo irá retornar um `ResponseAPDU`, onde pode-se buscar o valor resultante da operação, em forma de String conforme linhas 95 e 96. Ao término do processo, deve-se encerrar o canal de comunicação, para que não fique aguardando demais comandos, conforme linha 79.

Todo cartão utilizado no sistema, recebe um identificador único quando os dados do portador são gravados no cartão. Este identificador único é utilizado para relacionar o cartão ao portador. O comando utilizado para gravar esta informação no cartão, pode ser visto no Quadro 15.

Função	**CLA**	**INS**	**P1**	**P2**	**LC**	**Dados**	**LE**
Gravar id cartão	0x00	0x1C	0x00	0x00	16 *bytes*	Id cartão	-

Quadro 15 – Comando para gravar identificador único do cartão

Um exemplo da utilização deste comando pode ser visto no Quadro 16.

```
32      public static boolean gravarIDCartao(CardTerminal cardTerminal, Integer idCartao) {
33          try {
34              byte[] idCartaoAsBytes = Util.completarNumeroComZero(idCartao, 16).getBytes();
35              byte[] comandoGravarIDCartao = new byte[21];
36              comandoGravarIDCartao[0] = 0x00;
37              comandoGravarIDCartao[1] = 0x1C;
38              comandoGravarIDCartao[2] = 0x00;
39              comandoGravarIDCartao[3] = 0x00;
40              comandoGravarIDCartao[4] = 0x10;
41              int pos = 5;
42              for (int i = 0; i < idCartaoAsBytes.length; i++) {
43                  comandoGravarIDCartao[pos++] = idCartaoAsBytes[i];
44              }
45              Card card = cardTerminal.connect("*");
46              CardChannel cc = card.getBasicChannel();
47              transmitirComando(cc, comandoGravarIDCartao);
48              return true;
49          } catch (Exception e) {
50              e.printStackTrace();
51              return false;
52          }
53      }
```

Quadro 16 – Exemplo de um comando para gravar identificador único do cartão

O comando para obter o identificado único do cartão é composto de um comando APDU para solicitar e uma resposta APDU, conforme Quadros 17 e 18 respectivamente.

Função	CLA	INS	P1	P2	LC	Dados	LE
Obter id cartão	0x00	0x1B	0x00	0x00	-	-	16 *bytes*

Quadro 17 – Comando APDU para obter identificador único do cartão

Função	Dados retornados
Obter id cartão	ID do cartão

Quadro 18 – Comando de resposta APDU para obter identificador único

Para que o portador possa fazer o seu *login* no sistema, ele deve possuir um usuário e senha cadastrados no *smart card*. Estas informações ficam gravadas no cartão, desta forma é possível acessar o sistema sem a necessidade de validação em uma base de dados externas. Para gravar estas informações, são utilizados os comandos do Quadro 19.

Função	CLA	INS	P1	P2	LC	Dados	LE
Gravar usuário	0x00	0x0E	0x00	0x00	Tamanho usuário	Usuário	-
Gravar senha	0x00	0x0F	0x00	0x00	32 *bytes*	Senha	-

Quadro 19 – Comandos APDU para gravar usuário e senha

O Quadro 20 exemplifica a utilização deste comando no sistema.

```
55     public static boolean gravarUsuarioeSenha(CardTerminal cardTerminal, String usuario,
String senha) {
56         try {
57             byte[] comandoGravarUsuarioCartao = new byte[5 + usuario.length()];
58             comandoGravarUsuarioCartao[0] = 0x00;
59             comandoGravarUsuarioCartao[1] = 0x0E;
60             comandoGravarUsuarioCartao[2] = 0x00;
61             comandoGravarUsuarioCartao[3] = 0x00;
62             comandoGravarUsuarioCartao[4] = (byte) usuario.length();
63             int pos = 5;
64             for (int i = 0; i < usuario.getBytes().length; i++) {
65                 comandoGravarUsuarioCartao[pos++] = usuario.getBytes()[i];
66             }
67             Card card = cardTerminal.connect("*");
68             CardChannel cc = card.getBasicChannel();
69             transmitirComando(cc, comandoGravarUsuarioCartao);
70 
71             byte[] comandoGravarSenhaCartao = new byte[37];
72             comandoGravarSenhaCartao[0] = 0x00;
73             comandoGravarSenhaCartao[1] = 0x0F;
74             comandoGravarSenhaCartao[2] = 0x00;
75             comandoGravarSenhaCartao[3] = 0x00;
76             comandoGravarSenhaCartao[4] = 0x20;
77             pos = 5;
78             for (int i = 0; i < senha.getBytes().length; i++) {
79                 comandoGravarSenhaCartao[pos++] = senha.getBytes()[i];
80             }
81 
82             transmitirComando(cc, comandoGravarSenhaCartao);
83 
84             card.disconnect(true);
85             return true;
86         } catch (Exception e) {
87             e.printStackTrace();
88             return false;
89         }
90     }
```

Quadro 20 – Exemplo de utilização dos comandos para gravar usuário e senha

Os comandos APDU para obter o usuário e senha são demonstrados no Quadro 21 e as respostas APDU são demonstradas no Quadro 22.

Função	**CLA**	**INS**	**P1**	**P2**	**LC**	**Dados**	**LE**
Obter usuário	0x00	0x01	0x00	0x00	-	-	16 *bytes*
Obter senha	0x00	0x02	0x00	0x00	-	-	32 *bytes*

Quadro 21 – Comandos APDU para obter usuário e senha

Função	**Dados retornados**
Obter usuário	Usuário do cartão
Obter senha	Senha do cartão

Quadro 22 – Comando de resposta APDU para obter usuário e senha

A utilização destes comandos pode ser vista no Quadro 23, onde se busca o usuário e senha que estão gravados no cartão.

```
92      public String obterUsuarioCartao(CardTerminal cardTerminal) {
93          try {
94              byte[] comandoObterUsuarioCartao = {0x00, 0x01, 0x00, 0x00, 0x10};
95              Card card = cardTerminal.connect("*");
96              CardChannel cc = card.getBasicChannel();
97              ResponseAPDU resposta = transmitirComando(cc, comandoObterUsuarioCartao);
98              String usuarioCartao = getBytesAsString(resposta.getBytes());
99              card.disconnect(true);
100             return usuarioCartao;
101         } catch (Exception e) {
102             e.printStackTrace();
103             return null;
104         }
105     }
106
107     public String obterSenhaCartao(CardTerminal cardTerminal) {
108         try {
109             byte[] comandoObterSenhaCartao = {0x00, 0x02, 0x00, 0x00, 0x20};
110             Card card = cardTerminal.connect("*");
111             CardChannel cc = card.getBasicChannel();
112             ResponseAPDU resposta = transmitirComando(cc, comandoObterSenhaCartao);
113             String senhaCartao = getBytesAsString(resposta.getBytes());
114             card.disconnect(true);
115             return senhaCartao;
116         } catch (Exception e) {
117             e.printStackTrace();
118             return null;
119         }
120     }
```

Quadro 23 – Exemplo de utilização dos comandos para obter usuário e senha do cartão

Cada informação pessoal do portador equivale a um comando enviado para o cartão. A lista completa de comandos para gravar estas informações pode ser vista no Quadro 24.

Função	CLA	INS	P1	P2	LC	Dados	LE
Gravar nome	0x00	0x11	0x00	0x00	Tamanho nome	Nome	-
Gravar sexo	0x00	0x12	0x00	0x00	1 *byte*	Sexo	-
Gravar tipo sanguíneo	0x00	0x13	0x00	0x00	3 *bytes*	Tipo sanguíneo	-
Gravar data de nascimento	0x00	0x14	0x00	0x00	10 *bytes*	Data de nascimento	-
Gravar RG	0x00	0x15	0x00	0x00	8 *bytes*	RG	-
Gravar CPF	0x00	0x16	0x00	0x00	11 *bytes*	CPF	-
Gravar nome da mãe	0x00	0x17	0x00	0x00	Tamanho nome mãe	Nome da mãe	-
Gravar data de nascimento da mãe	0x00	0x18	0x00	0x00	10 *bytes*	Data de nascimento da mãe	-
Gravar nome do pai	0x00	0x19	0x00	0x00	Tamanho nome do	Nome do pai	-

					pai		
Gravar data de nascimento do pai	0x00	0x1A	0x00	0x00	10 *bytes*	Data de nascimento do pai	-

Quadro 24 – Relação de comandos APDU para gravar dados pessoais do portador

No Quadro 25 é possível ver o código para gravar as informações pessoais do portador, junto da enumeração para controle da informação passada.

```
 7 public enum Informacao {
 8
 9     NOME, SEXO, TP SANG, DT NASC, RG, CPF,
10     NOME_MAE, DT_NASC_MAE,
11     NOME_PAI, DT_NASC_PAI
12 }

122     public boolean gravarInformacaoPessoalPortador(CardTerminal cardTerminal,
123             Informacao tpInformacao, String informacao) {
124         try {
125             byte[] comando = null;
126             switch (tpInformacao) {
127                 case NOME: {
128                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x11,
129                             (byte) 0x00, (byte) 0x00, (byte) informacao.length(),
130                             informacao);
131                     break;
132                 }
133                 case SEXO: {
134                     byte sexo = informacao.getBytes()[0];
135                     comando = new byte[]{0x00, 0x12, 0x00, 0x00, 0x01, sexo};
136                     break;
137                 }
138                 case TP_SANG: {
139                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x13,
140                             (byte) 0x00, (byte) 0x00, (byte) 0x03, informacao);
141                     break;
142                 }
143                 case DT_NASC: {
144                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x14,
145                             (byte) 0x00, (byte) 0x00, (byte) 0x0A, informacao);
146                     break;
147                 }
148                 case RG: {
149                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x15,
150                             (byte) 0x00, (byte) 0x00, (byte) 0x08, informacao);
151                     break;
152                 }
153                 case CPF: {
154                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x16,
155                             (byte) 0x00, (byte) 0x00, (byte) 0x0B, informacao);
156                     break;
157                 }
158                 case NOME_MAE: {
159                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x17,
160                             (byte) 0x00, (byte) 0x00, (byte) informacao.length(),
161                             informacao);
162                     break;
163                 }
164                 case DT_NASC_MAE: {
165                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x18,
166                             (byte) 0x00, (byte) 0x00, (byte) 0x0B, informacao);
167                     break;
168                 }
169                 case NOME_PAI: {
170                     comando = montarComandoAPDU((byte) 0x00, (byte) 0x19,
171                             (byte) 0x00, (byte) 0x00, (byte) informacao.length(),
172                             informacao);
173                     break;
174                 }
175                 case DT_NASC_PAI: {
```

```
176                         comando = montarComandoAPDU((byte) 0x00, (byte) 0x1A,
177                                 (byte) 0x00, (byte) 0x00, (byte) 0x0B, informacao);
178                         break;
179                     }
180                 }
181                 Card card = cardTerminal.connect("*");
182                 CardChannel cc = card.getBasicChannel();
183                 transmitirComando(cc, comando);
184                 card.disconnect(true);
185                 return true;
186             } catch (Exception e) {
187                 e.printStackTrace();
188                 return false;
189             }
190         }

192         private byte[] montarComandoAPDU(byte cla, byte ins, byte p1, byte p2, byte length,
String dados) {
193             byte[] comandoAPDU = new byte[5 + length];
194             comandoAPDU[0] = cla;
195             comandoAPDU[1] = ins;
196             comandoAPDU[2] = p1;
197             comandoAPDU[3] = p2;
198             comandoAPDU[4] = length;
199             int pos = 5;
200             for (int i = 0; i < dados.getBytes().length; i++) {
201                 comandoAPDU[pos++] = dados.getBytes()[i];
202             }
203             return comandoAPDU;
204         }
```

Quadro 25 – Exemplo para gravar informações pessoais do portador no cartão

O Quadro 26 apresenta a relação de comandos para buscar as informações pessoais do portador e no Quadro 27 as repostas de cada comando.

Função	CLA	INS	P1	P2	LC	Dados	LE
Obter nome	0x00	0x04	0x00	0x00	-	-	45 bytes
Obter sexo	0x00	0x05	0x00	0x00	-	-	1 byte
Obter tipo sanguíneo	0x00	0x06	0x00	0x00	-	-	3 bytes
Obter data de nascimento	0x00	0x07	0x00	0x00	-	-	10 bytes
Obter RG	0x00	0x08	0x00	0x00	-	-	8 bytes
Obter CPF	0x00	0x09	0x00	0x00	-	-	11 bytes
Obter nome da mãe	0x00	0x0A	0x00	0x00	-	-	45 bytes
Obter data de nascimento da mãe	0x00	0x0B	0x00	0x00	-	-	10 bytes
Obter nome do pai	0x00	0x0C	0x00	0x00	-	-	45 bytes
Obter data de nascimento do pai	0x00	0x0D	0x00	0x00	-	-	10 bytes

Quadro 26 – Comandos APDU para obter informações pessoais do portador

Função	Dados retornados
Obter nome	Nome do portador
Obter sexo	Sexo do portador
Obter tipo sanguíneo	Tipo sanguíneo
Obter data de nascimento	Data de nascimento
Obter RG	RG
Obter CPF	CPF
Obter nome da mãe	Nome da mãe
Obter data de nascimento da mãe	Data de nascimento da mãe
Obter nome do pai	Nome do pai
Obter data de nascimento do pai	Data de nascimento do pai

Quadro 27 – Respostas dos comandos para obter informações do portador

Para gravar as vacinas no cartão o comando é listado no Quadro 28.

Função	CLA	INS	P1	P2	LC	Dados	LE
Gravar vacina	0x00	0x10	0x00	0x00	Tamanho dados	Vacina 0x00 dose 0x00 data	-

Quadro 28 – Comando APDU para gravar uma vacina no cartão

Este comando engloba alguns valores em um mesmo, pelo fato de a vacina conter a descrição da mesma, a dose e a data em que foi registrada. Todos estes dados podem ser passados em somente um comando pois não ultrapassam o valor de 255 *bytes*. Os valores são separados pelo *byte* de controle 0x00, desta forma sendo possível separar o comando no cartão e buscar cada valor separado.

Um exemplo de como este comando é gerado pode ser visto no Quadro 29.

```
206     public boolean gravarVacina(CardTerminal cardTerminal, String vacina, int dose,
String data) {
207         try {
208             if (dose == -1) {
209                 dose = 0;
210             }
211             int tamanhoDados = vacina.getBytes().length + 13;
212             int tamanhoArray = 5 + tamanhoDados;
213             byte[] comandoGravarVacina = new byte[tamanhoArray];
214             comandoGravarVacina[0] = 0x00;
215             comandoGravarVacina[1] = 0x10;
216             comandoGravarVacina[2] = 0x00;
217             comandoGravarVacina[3] = 0x00;
218             comandoGravarVacina[4] = (byte) tamanhoDados;
219             int pos = 5;
220             for (int i = 0; i < vacina.getBytes().length; i++) {
221                 comandoGravarVacina[pos++] = vacina.getBytes()[i];
222             }
223             comandoGravarVacina[pos++] = 0x00;
224             comandoGravarVacina[pos++] = (byte) dose;
225             comandoGravarVacina[pos++] = 0x00;
226             for (int i = 0; i < data.getBytes().length; i++) {
227                 comandoGravarVacina[pos++] = data.getBytes()[i];
228             }
229             Card card = cardTerminal.connect("*");
230             CardChannel cc = card.getBasicChannel();
231             transmitirComando(cc, comandoGravarVacina);
232             card.disconnect(true);
233             return true;
234         } catch (Exception e) {
235             e.printStackTrace();
236             return false;
237         }
238     }
```

Quadro 29 – Exemplo de criação do comando APDU para gravar vacina

Para obter as vacinas que estão registradas no cartão, é necessário utilizar o comando do Quadro 30, tendo como resposta o Quadro 31.

Função	**CLA**	**INS**	**P1**	**P2**	**LC**	**Dados**	**LE**
Obter vacina	0x00	0x03	0x00 ou 0x01	0x00	-	-	*59 bytes*

Quadro 30 – Comando APDU para obter uma vacina do cartão

Função	**Dados retornados**
Obter vacina	Vacina 0x00 dose 0x00 data 0x00 ou 0x01

Quadro 31 – Resposta do comando APDU para obter vacina do cartão

O comando para obter uma vacina difere dos outros por ser necessário passar um valor ao P1 do comando APDU, sendo possível passar dois parâmetros: 0x00 ou 0x01. Caso o valor informado seja 0x00, significa que está sendo solicitado ao cartão a primeira vacina registrada, caso seja passado 0x01, está sendo solicitado a próxima vacina. Este comando irá retornar uma resposta contendo a vacina, um byte de controle 0x00, a dose, outro *byte* de controle 0x00, a data e um *byte* final de controle podendo ser 0x00 ou 0x01.

Este byte final de controle indica se há mais vacinas registradas ou se é a última vacina do cartão. Caso retorne 0x00, não há mais vacinas a serem buscadas, caso retorne 0x01, ainda há vacinas para serem carregadas.

Um exemplo desta aplicação pode ser visto no Quadro 32.

```
244     public List<VacinaPessoa> obterVacinasCartao(CardTerminal cardTerminal) {
245         List<VacinaPessoa> listaVacinas = new ArrayList<VacinaPessoa>();
246         try {
247             byte[] comandoObterVacina = new byte[]{0x00, 0x03, 0x00, 0x00, 0x3A};
248 
249             Card card = cardTerminal.connect("*");
250             CardChannel cc = card.getBasicChannel();
251 
252             byte temMaisVacina = 0x01;
253             while (temMaisVacina == 0x01) {
254                 ResponseAPDU apdu = transmitirComando(cc, comandoObterVacina);
255                 byte[] resposta = apdu.getBytes();
256 
257                 String vacina = String.valueOf(Arrays.copyOfRange(resposta, 0, 44));
258                 int dose = Integer.parseInt(String.valueOf(Arrays.copyOfRange(resposta,
45, 46)));
259                 String data = String.valueOf(Arrays.copyOfRange(resposta, 48, 57));
260                 temMaisVacina = resposta[58];
261 
262                 VacinaPessoa vacinaPessoa = new VacinaPessoa();
263                 vacinaPessoa.setDsVacina(vacina);
264                 vacinaPessoa.setQtDose(dose);
265                 vacinaPessoa.setDtAtualizacao(Util.getStringAsTimestamp(data));
266 
267                 listaVacinas.add(vacinaPessoa);
268 
269                 if (comandoObterVacina[2] == 0x00) {
270                     comandoObterVacina[2] = 0x01;
271                 }
272             }
273 
274             card.disconnect(true);
275         } catch (Exception e) {
276             e.printStackTrace();
277         }
278         return listaVacinas;
279     }
```

Quadro 32 – Exemplo de comando APDU para obter vacina do cartão

Na linha 247 o comando é montado pela primeira vez, informando que deseja obter a primeira vacina registrada no cartão. Uma variável de controle é criada na linha 252 para verificar se há mais vacinas no cartão, a qual é alterada na linha 260. Caso ainda tenha mais vacinas, o comando é alterado para buscar a próxima vacina, conforme linhas 269 e 270.

3.3.2.2 Desenvolvimento da aplicação para o *smart card*

Para o desenvolvimento da aplicação para o *smart card* foi utilizada a tecnologia Java *Card* na versão 2.2.1, por ser compatível com o *smart card* utilizado para os testes. Esta tecnologia não está acoplada ao Java *Standard Development Kit* (JSDK) do Java, por tanto deve ser feito o *download* da mesma pelo programador para ser utilizada como uma biblioteca

de projeto.

A *package* onde ficam as classes do Java *Card* é a `javacard.framework`. Nesse *package* encontra-se a classe `Applet`, responsável pelo controle da aplicação que será carregada no *smart card*. Comumente uma aplicação para *smart card* é chamada de um *applet* em função desta classe. Para utilizar a mesma, deve-se criar uma classe e estender da mesma. Por esta classe ser abstrata, faz-se necessário sobrescrever o método `process`. Outros três métodos podem ser sobrescrevidos para realizar controles sobre a aplicação, sendo eles: `install`, `select` e `deselect`.

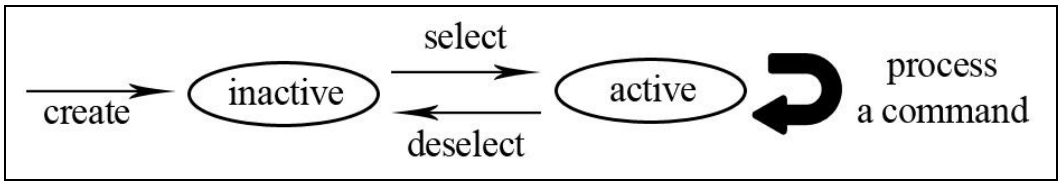
O método `process` é invocado sempre que um comando é enviado à aplicação, recebendo como parâmetro a classe `javacard.framework.APDU`, que representa o comando APDU enviado. O método `install` é chamado somente uma vez, quando a aplicação é instalada no *smart card*. Nesse método devem ser feitas as inicializações de variáveis necessárias e o registro da aplicação, invocando no final do método a chamada do método `register()`. O método `select` é invocado sempre que uma aplicação é requisitada para ser ativada. Este método retorna um booleano indicando se a aplicação pode ser inicializada ou não. Por padrão, este método retorna verdadeiro. O método `deselect` é invocado quando a aplicação é desativada. Costuma-se liberar algumas variáveis neste método, para que não ocupem espaço desnecessário. A estrutura básica de uma classe com estas configurações pode ser vista no quadro 33.

```
 1 package br.com.card;
 2
 3 import javacard.framework.APDU;
 4 import javacard.framework.Applet;
 5 import javacard.framework.ISOException;
 6
 7 public class ExemploEstruturaApplet extends Applet {
 8
 9     protected ExemploEstruturaApplet() {
10         register();
11     }
12
13     public static void install(byte[] bArray, short bOffset, byte bLength) {
14         new ExemploEstruturaApplet();
15     }
16
17     public boolean select() {
18         return true;
19     }
20
21     public void process(APDU arg0) throws ISOException {
22     }
23
24     public void deselect() {
25     }
26 }
```

Quadro 33 – Estrutura básica de um *applet* para *smart card*

Um *applet* de *smart card* possui estados de execução, sendo que o mesmo é criado e colocado no estado de inativo. Quando selecionado, o *applet* passa para o estado de ativo e

então processa os comandos. Quando encerrado, retorna ao estado de inativo. A Figura 15 exemplifica esta situação.



Fonte: CHEN (2000, p. 72).

Figura 15 – Estados de execução um *applet*

Todo comando recebido pelo *applet* é invocado o método `process`, como já foi dito. Este método recebe como parâmetro a classe `javacard.framework.APDU`, que representa o comando enviado. Para verificar o comando que foi enviado, utiliza-se o método `getBuffer()` que retorna um vetor de *bytes*. Por se tratar de um vetor, cada posição do vetor representa um campo do comando APDU. Por tanto a posição 0 representa o campo CLA, a posição 1 o campo INS e assim por diante.

Para facilitar a visualização destas informações no código, pode-se utilizar a classe `javacard.framework.ISO7816` e suas constantes `OFFSET_CLA`, `OFFSET_INS`, `OFFSET_P1` e `OFFSET_P2`. Um exemplo desta utilização pode ser visto no Quadro 27.

```
18      public void process(APDU apdu) throws ISOException {
19          byte[] buffer = apdu.getBuffer();
20
21          if (buffer[ISO7816.OFFSET_CLA] != 0x00) {
22              ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
23          }
24
25          switch (buffer[ISO7816.OFFSET_INS]) {
26              case Instrucao.OBTER_USUARIO:
27                  break;
28          }
29      }
```

Quadro 34 – Exemplo de utilização do comando APDU

No Quadro 34 pode-se ver que na linha 19 busca-se o comando APDU que foi enviado para o *smart card*. Na linha 21, é buscado a campo CLA do comando APDU e feito uma verificação com o mesmo e na linha 25 busca-se o campo INS para verificar qual evento será invocado para este comando APDU recebido.

Pode-se verificar também no Quadro 34, que uma exceção é lançada na linha 22. As exceções no *applet* são representadas pela classe `javacard.framework.ISOException` e devem ser lançadas a partir de seu método estático `throwIt` passando um *short* como parâmetro, indicando o motivo da exceção. Estes motivos são constantes da classe `ISOException`. No exemplo o motivo é que o campo CLA passado não é suportado pelo *applet*.

Para retornar valores através de uma resposta APDU, deve-se colocar o comando APDU recebido em modo de resposta. Isto é feito invocando-se o método `setOutgoing`, que

retornará um *short* com o tamanho previsto da resposta. Após isto invoca-se o método que define quantos *bytes* realmente serão retornados, sendo ele `setOutgoingLength` e por último invoca-se o método que efetivamente irá enviar os *bytes*, `sendBytesLong`. Estes métodos devem ser chamados na ordem correta, caso contrário irá gerar um `javacard.framework.APDUException`.

Um exemplo destes métodos pode ser visto no Quadro 35.

```
18      public void process(APDU apdu) throws ISOException {
19          short total_bytes = (short) 5;
20          byte[] name = {'t', 'e', 's', 't', 'e'};
21
22          apdu.setOutgoing();
23
24          apdu.setOutgoingLength(total_bytes);
25
26          apdu.sendBytesLong(name, (short) 0, (short) name.length);
27      }
```

Quadro 35 – Exemplo de resposta de APDU

Para cada instrução recebida em um comando APDU é invocado um método para realizar o seu tratamento. No Quadro 36 há a lista de instruções disponíveis representados pela classe `Instrucao`.

```
 1 package br.com.card;
 2
 3 public class Instrucao {
 4
 5     public static final byte OBTER_USUARIO      = 0x01;
 6     public static final byte OBTER_SENHA        = 0x02;
 7
 8     public static final byte OBTER_VACINA       = 0x03;
 9
10     public static final byte OBTER_NOME         = 0x04;
11     public static final byte OBTER_SEXO         = 0x05;
12     public static final byte OBTER_TP_SANG      = 0x06;
13     public static final byte OBTER_DT_NASC      = 0x07;
14     public static final byte OBTER_RG           = 0x08;
15     public static final byte OBTER_CPF          = 0x09;
16     public static final byte OBTER_NOME_MAE     = 0x0A;
17     public static final byte OBTER_DT_NASC_MAE  = 0x0B;
18     public static final byte OBTER_NOME_PAI     = 0x0C;
19     public static final byte OBTER_DT_NASC_PAI  = 0x0D;
20
21     public static final byte OBTER_ID_CARTAO    = 0x1B;
22
23     public static final byte GRAVAR_USUARIO     = 0x0E;
24     public static final byte GRAVAR_SENHA       = 0x0F;
25
26     public static final byte GRAVAR_VACINA      = 0x10;
27
28     public static final byte GRAVAR_NOME        = 0x11;
29     public static final byte GRAVAR_SEXO        = 0x12;
30     public static final byte GRAVAR_TP_SANG     = 0x13;
31     public static final byte GRAVAR_DT_NASC     = 0x14;
32     public static final byte GRAVAR_RG          = 0x15;
33     public static final byte GRAVAR_CPF         = 0x16;
34     public static final byte GRAVAR_NOME_MAE    = 0x17;
35     public static final byte GRAVAR_DT_NASC_MAE = 0x18;
36     public static final byte GRAVAR_NOME_PAI    = 0x19;
37     public static final byte GRAVAR_DT_NASC_PAI = 0x1A;
38
39     public static final byte GRAVAR_ID_CARTAO   = 0x1C;
40 }
```

Quadro 36 – Instruções do *applet*

Para trabalhar com cada instrução existem as classes pertencentes à *package*

`br.com.card.control`. `IdCartao` controla as operações referente ao identificador único do cartão, `UsuarioControl` controla as operações de dados do usuário como usuário e senha, `VacinaControl` trabalha com as operações de uma vacina e armazena todas a vacinas registradas no cartão e `InformacaoPessoalControl` cuida de todas as informações pessoais do portador da carteira.

Um exemplo de como as informações são registradas no cartão pode ser visto no Quadro 37.

```
1  package br.com.card.control;
2
3  import javacard.framework.APDU;
4  import javacard.framework.Util;
5  import br.com.card.crypto.TripleDes;
6  import br.com.card.model.Usuario;
7
8  public class UsuarioControl {
9
10     private Usuario usuario = new Usuario();
11
12     public void gravarUsuario(byte[] buffer) {
13         byte[] user = new byte[16];
14         Util.arrayCopy(buffer, (short) 5, user, (short) 0, (short) 16);
15         TripleDes.encrypt(user);
16         usuario.setUsuario(user);
17     }
18
19     public void gravarSenha(byte[] buffer) {
20         byte[] senha = new byte[32];
21         Util.arrayCopy(buffer, (short) 5, senha, (short) 0, (short) 32);
22         TripleDes.encrypt(senha);
23         usuario.setSenha(senha);
24     }
25
26     public void obterUsuario(APDU apdu) {
27         apdu.setOutgoing();
28         apdu.setOutgoingLength((short) 16);
29         byte[] user = new byte[16];
30         TripleDes.decrypt(usuario.getUsuario(), user);
31         apdu.sendBytesLong(user, (short) 0, (short) user.length);
32     }
33
34     public void obterSenha(APDU apdu) {
35         apdu.setOutgoing();
36         apdu.setOutgoingLength((short) 32);
37         byte[] senha = new byte[16];
38         TripleDes.decrypt(usuario.getSenha(), senha);
39         apdu.sendBytesLong(senha, (short) 0, (short) senha.length);
40     }
41 }
```

Quadro 37 – Exemplo de registro de informação no cartão

Na linha 10, cria-se a classe de modelo `Usuario` para guardar os valores. A linha 14 contempla o código de cópia do valor recebido no buffer do comando APDU para uma variável local, criada na linha 13. Após feita a cópia de valores, na linha 15 é aplicado o algoritmo de criptografia 3DES sobre o valor e armazenado o mesmo na classe modelo `Usuario`, conforme linha 16. Esse processo é realizado para gravar um valor no cartão.

O processo para buscar um valor do cartão começa invocando o método para colocar o cartão em modo resposta, conforme linha 27. Após isso, invoca-se o método para informar a quantidade de *bytes* que será enviada, linha 28. Como o valor está criptografado é necessário

aplicar o algoritmo de descriptografia conforme linha 30, onde é passado o valor e uma variável de retorno, que foi criada na linha 29. Na linha 31, é feito o envio do valor para o cartão.

Para este processo de criptografia, foi criada a classe `TripleDes`. Essa classe possui um método para criptografar as informações e outro para descriptografar. Essa classe trabalha com uma chave utilizada para realizar os processos. O código completo de como é feito esses processos pode ser visto no Quadro 38.

```
 1 package br.com.card.crypto;
 2
 3 import javacard.security.DESKey;
 4 import javacard.security.KeyBuilder;
 5 import javacardx.crypto.Cipher;
 6
 7 public class TripleDes {
 8
 9     static byte[] TripleDESKey = {(byte) 0x38, (byte) 0x12, //
10         (byte) 0xA4, (byte) 0x19, (byte) 0xC6, (byte) 0x3B, //
11         (byte) 0xE7, (byte) 0x71, (byte) 0x00, (byte) 0x12, //
12         (byte) 0x00, (byte) 0x19, (byte) 0x80, (byte) 0x3B, //
13         (byte) 0xE7, (byte) 0x71, (byte) 0x01, (byte) 0x12, //
14         (byte) 0x01, (byte) 0x01, (byte) 0x01, (byte) 0x03, //
15         (byte) 0xE7, (byte) 0x71};
16
17     private static DESKey deskey = (DESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
KeyBuilder.LENGTH_DES3_3KEY, false);
18     private static Cipher cipherCBC = Cipher.getInstance(Cipher.ALG_DES_CBC_NOPAD, false);
19
20     public static void encrypt(byte[] inputBuffer) {
21         deskey.setKey(TripleDESKey, (short) 0);
22         cipherCBC.init(deskey, Cipher.MODE_ENCRYPT);
23         cipherCBC.doFinal(inputBuffer, (short) 0, (short) inputBuffer.length, inputBuffer,
(short) 0);
24     }
25
26     public static void decrypt(byte[] inputBuffer, byte[] outputBuffer) {
27         deskey.setKey(TripleDESKey, (short) 0);
28         cipherCBC.init(deskey, Cipher.MODE_DECRYPT);
29         cipherCBC.doFinal(inputBuffer, (short) 0, (short) inputBuffer.length,
outputBuffer, (short) 0);
30     }
31 }
```

Quadro 38 – Classe `TripleDes`

Para trabalhar com criptografia, Java *Card* 2.2.1 fornece duas *packages*: `javacard.security` e `javacardx.crypto`. A criptografia mais atual disponível para esta versão do Java *Card* é a 3DES. Para gerar a chave, utiliza-se a classe `javacard.security.KeyBuilder` onde passa-se o algoritmo que será utilizado, o tamanho e um booleano indicando se esta chave poderá ser acessada externamente ao *applet* ou não e para aplicar a criptografia, utiliza-se a classe `javacardx.crypto.Cipher`.

Utiliza-se o método `setKey` da classe `javacard.security.KeyBuilder` para registrar a chave utilizada para a criptografia, conforme linhas 21 e 27. O método `init` da classe `javacardx.crypto.Cipher` é utilizado para passar a chave gerada e o modo, sendo ele criptografia ou descriptografia, conforme linhas 22 e 28. Por último invoca-se o método `doFinal` passando-se o valor para realizar a criptografia conforme linhas 23 e 29.

3.3.2.3 Desenvolvimento da aplicação Web

Para o desenvolvimento da aplicação foi criado um projeto Web a partir do NetBeans, o qual já cria uma estrutura básica com os arquivos mínimos e bibliotecas necessárias, com o nome de CardManager.

Para que o *applet* possa ser visualizado na internet, faz-se necessário adicioná-lo ao projeto Web junto de suas bibliotecas, na pasta Web Pages do projeto Web, conforme Figura 16.

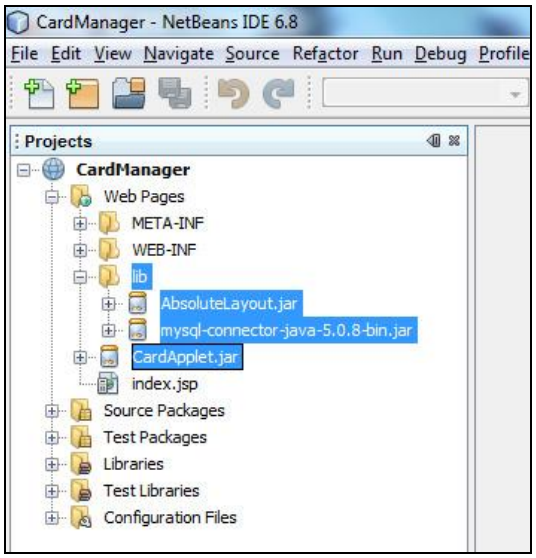


Figura 16 – *Applet* adicionado ao projeto Web

A configuração para visualização deste *applet* para internet deve ser feita em um arquivo de página para internet utilizando-se de marcações HTML. A *tag* utilizada para trabalhar com este tipo de objeto é a `<applet>`. Esta *tag* possui vários parâmetros, porém os utilizados para este trabalho foram: *width* para definir largura do *applet*, *height* para definir altura do *applet*, *archive* para definir qual o arquivo *Java Archive* (JAR) que representa o *applet* e *code* para definir a classe que será acessada para carregar o *applet*.

Esta configuração foi realizada no arquivo `index.jsp`, conforme Quadro 39.

```
1  <html>
2      <head>
3          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4      </head>
5      <body>
6          <applet
7              width="800"
8              height="600"
9              archive="CardApplet.jar"
10             code="br.com.Main.class"
11         </applet>
12     </body>
13 </html>
```

Quadro 39 – Estrutura do arquivo `index.jsp`

Este projeto deve ser carregado em um servidor de aplicação para que fique publicado. Para isto, deve ser gerado o *Web Application Archive* (WAR) do projeto e carregado o mesmo no servidor *Apache Tomcat*.

3.3.3 Operacionalidade da implementação

Nesta seção será apresentado o sistema através da execução de um estudo de caso. A demonstração será divida em duas partes, uma como agente de saúde e a outra como um portador da carteira, demonstrando um fluxo de utilização do sistema por ambas as partes. Será utilizado o navegador *Google Chrome* para demonstrar o *applet* via internet.

Para utilizar o sistema deve-se primeiramente acessar a página onde se encontra o *applet*.

Ao abrir a página o *applet* será carregado e aparecerá uma mensagem solicitando autorização do usuário para que o mesmo possa ser rodado, conforme Figura 17. Isto se faz necessário pois o *applet* necessita acessar um dispositivo que se encontra na máquina do usuário e não pode realizar esta ação sem a devida autorização do usuário.

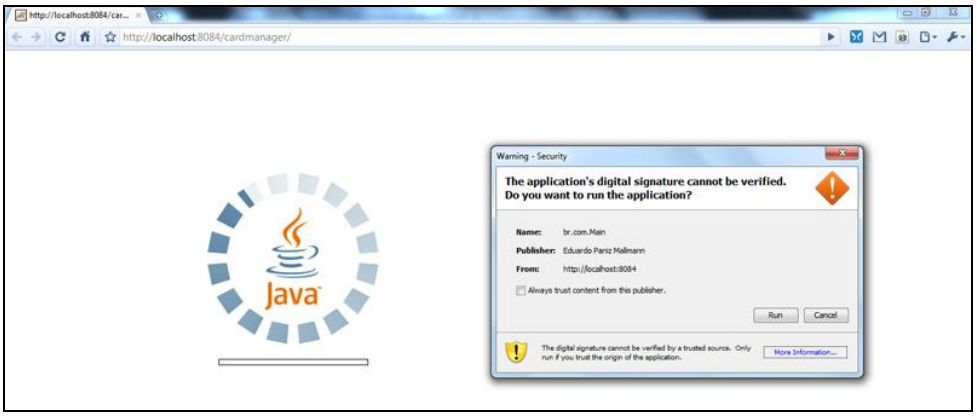


Figura 17 – Solicitação de autorização ao usuário

Após autorização concedida, o *applet* será carregado por completo, apresentando a tela de *login* do sistema, conforme Figura 18.

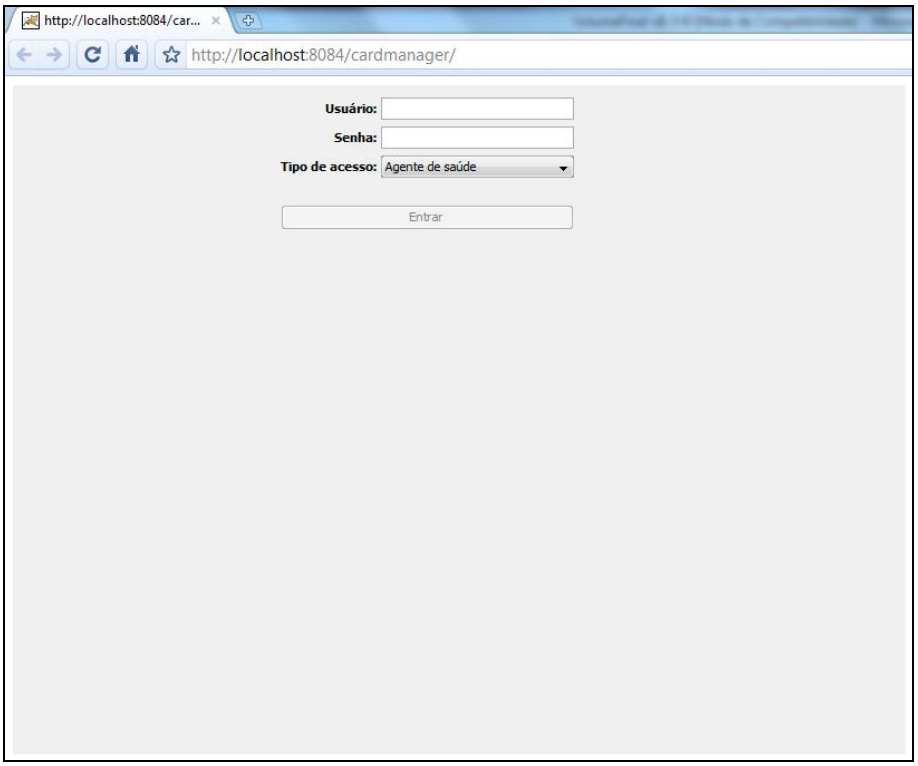


Figura 18 – Tela de *login* do sistema

3.3.3.1 Agente de Saúde

Na tela de *login* o agente de saúde informa seu usuário e senha e define que o tipo de acesso será como Agente de saúde e clica no botão Entrar. Caso os dados informados não estejam corretos será emitido uma mensagem ao usuário informando-o, conforme Figura 19, caso contrário irá acessar a seção do sistema referente as suas opções dentro do sistema, conforme Figura 20.

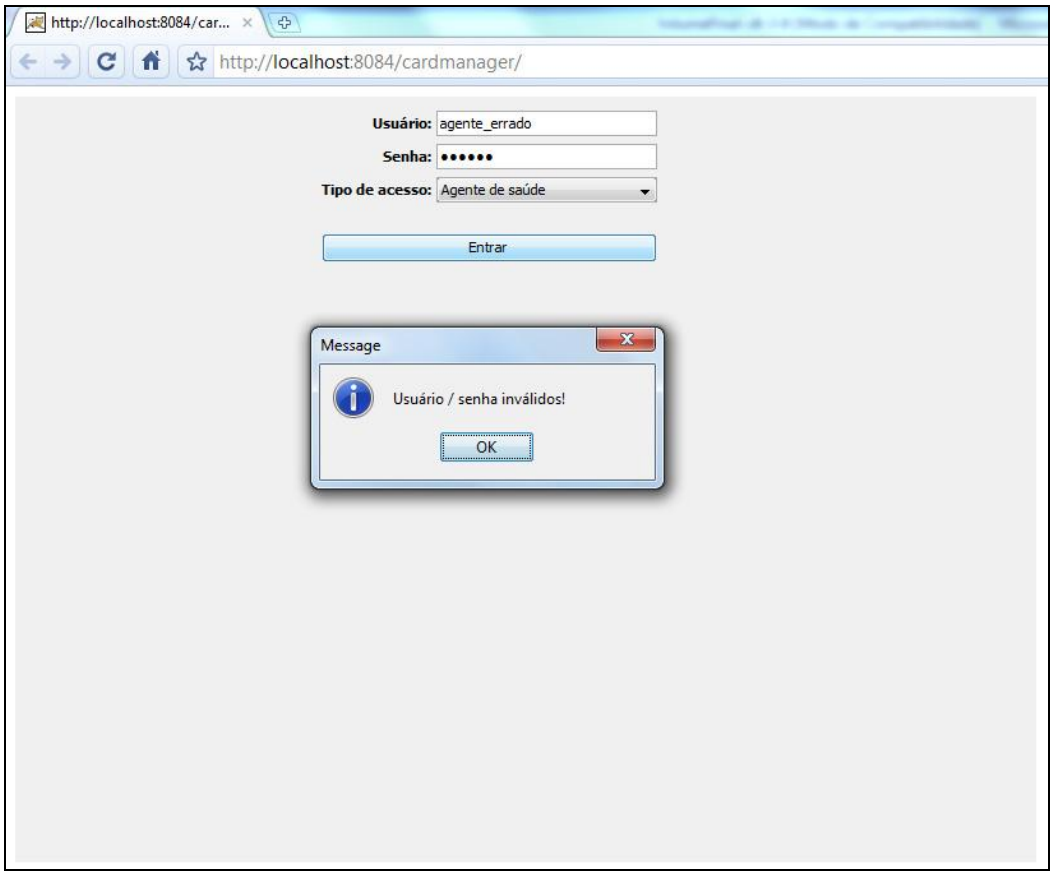


Figura 19 – Tela indicando erro ao fazer *login* no sistema

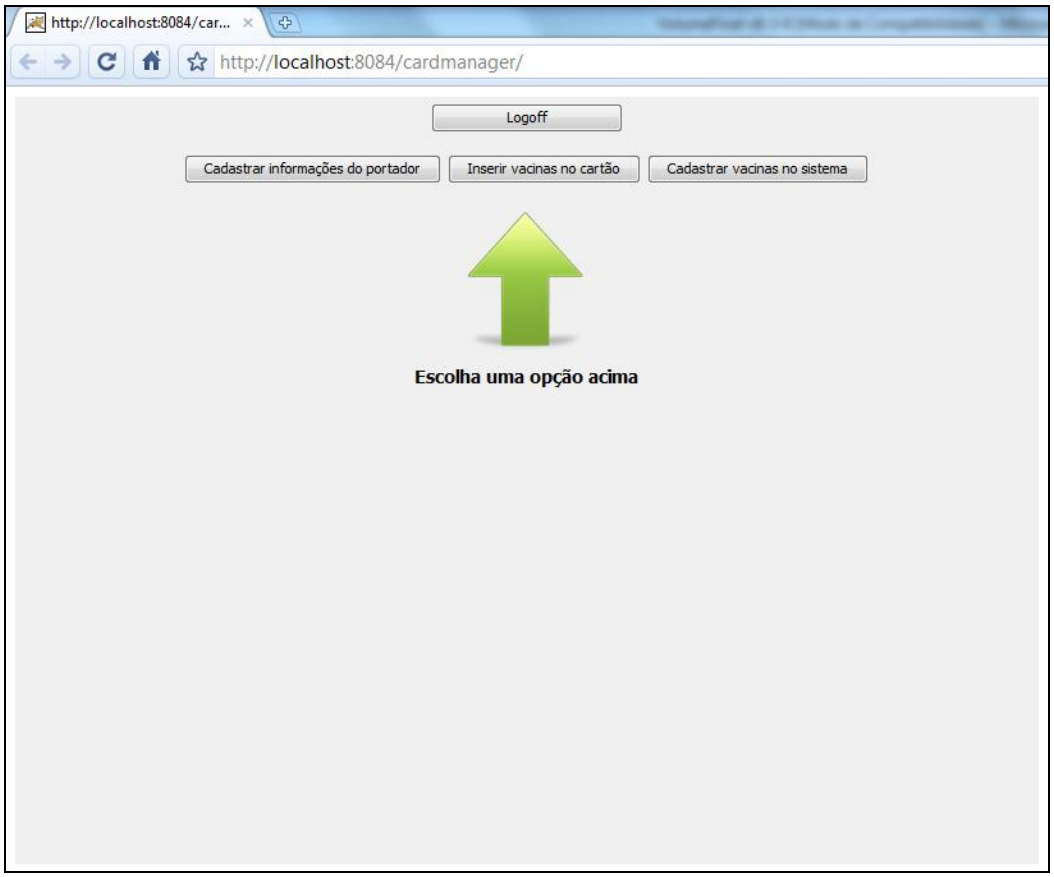


Figura 20 – Tela de opções do agente de saúde

O agente de saúde pode escolher quatro opções para realizar no sistema, sendo elas Cadastrar informações do portador, Inserir vacinas no cartão e Cadastrar vacinas no sistema e Logoff. Para ter acesso as opções Cadastrar informações do portador e Inserir vacinas no cartão é necessário que haja uma leitora e um cartão conectados à máquina, caso contrário será emitido um aviso ao usuário informando-o, conforme respectivamente Figura 21 e 22.

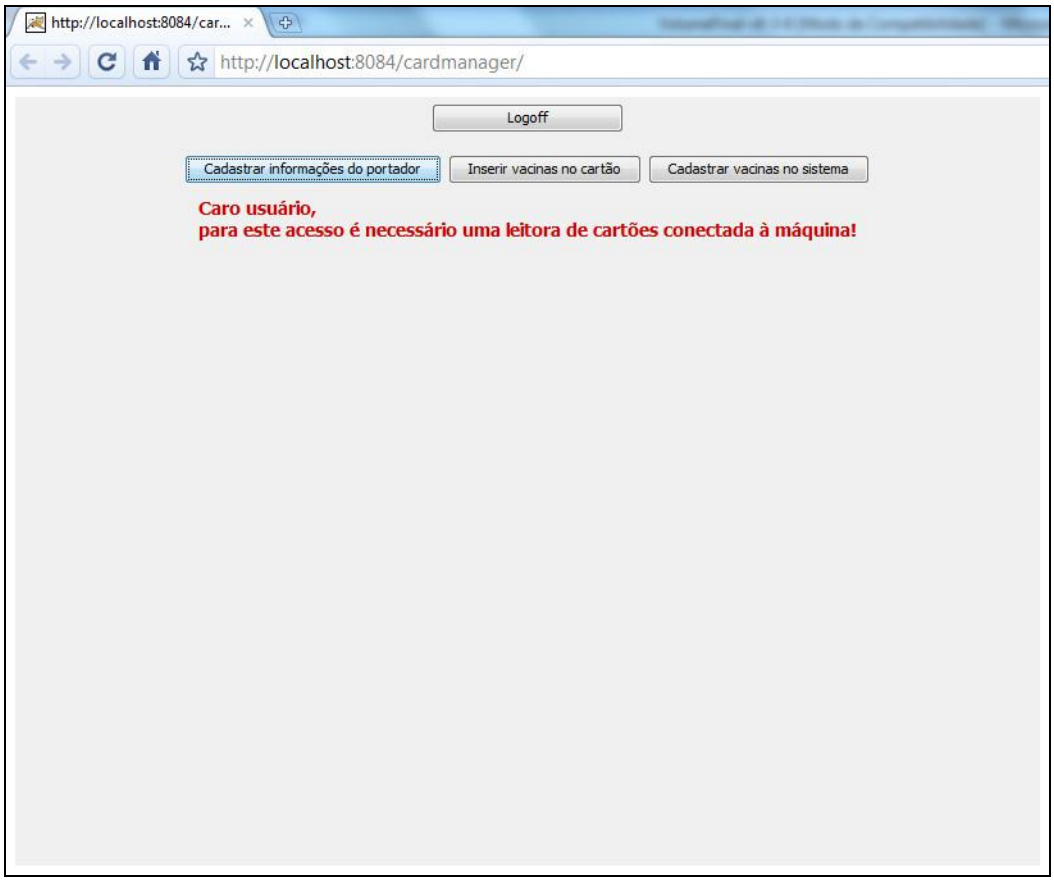


Figura 21 – Mensagem de falta de leitora

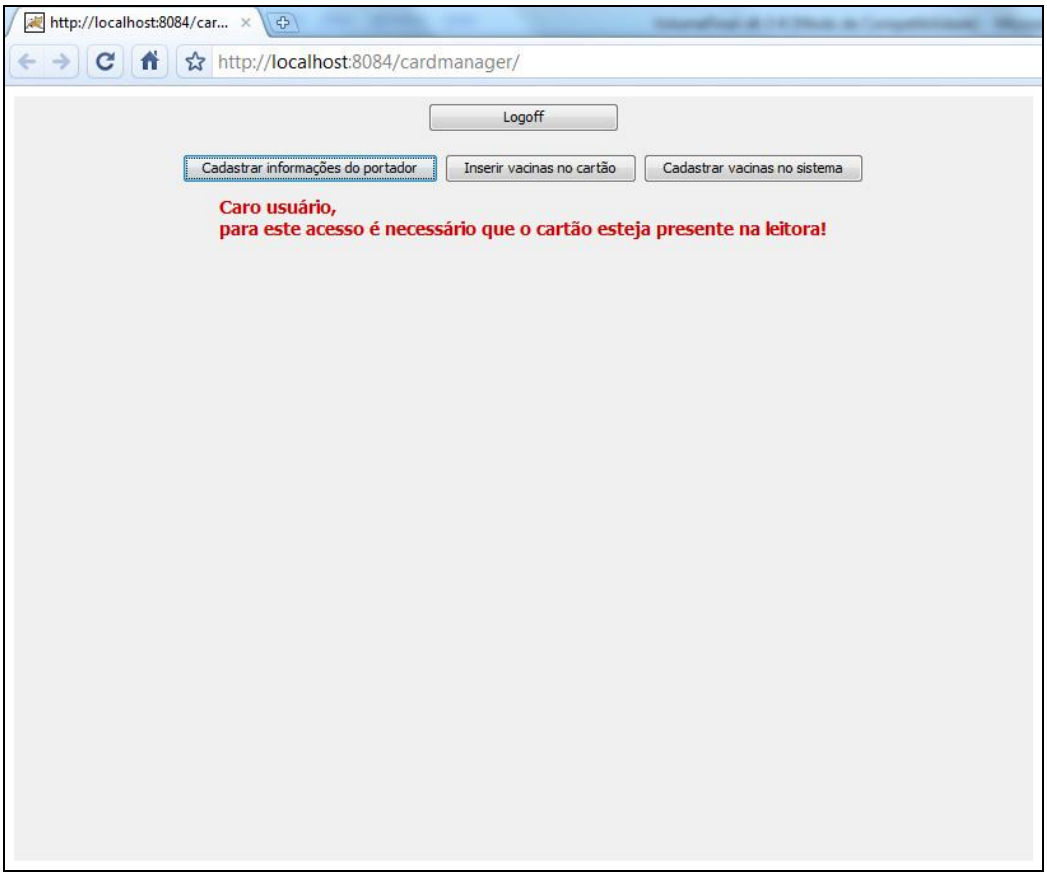


Figura 22 – Mensagem de falta de cartão na leitora

Na seção de cadastro de informações pessoais do portador, conforme Figura 23, o agente cadastra ou altera as informações necessárias do portador.

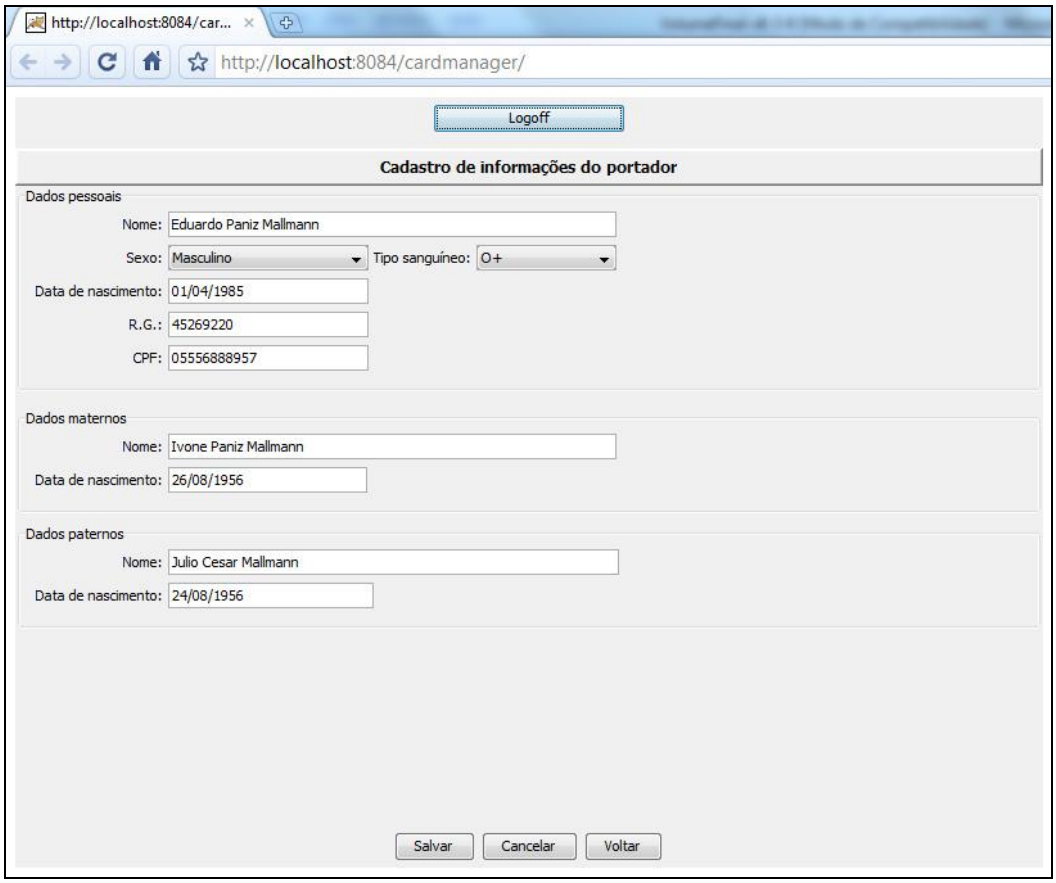


Figura 23 – Tela do cadastro de informações pessoais do portador

Para inserir novas vacinas na carteira do portador deve-se acessar a seção de Inserir vacinas no cartão, conforme Figura 24. Nesta tela são apresentadas todas as vacinas que já foram registradas no cartão do portador, para que o agente possa verificar o histórico de vacinas e verificar se necessita aplicar alguma que esteja faltando.

O agente registra uma nova vacina clicando no botão Inserir vacina no cartão, onde aparecerá uma tela solicitando a vacina que se deseja aplicar e a sua dose, conforme a Figura 25. As vacinas que aparecem na lista são as que já foram cadastradas previamente no sistema. Caso a vacina seja de dose única, o campo dose não deve ser preenchido.

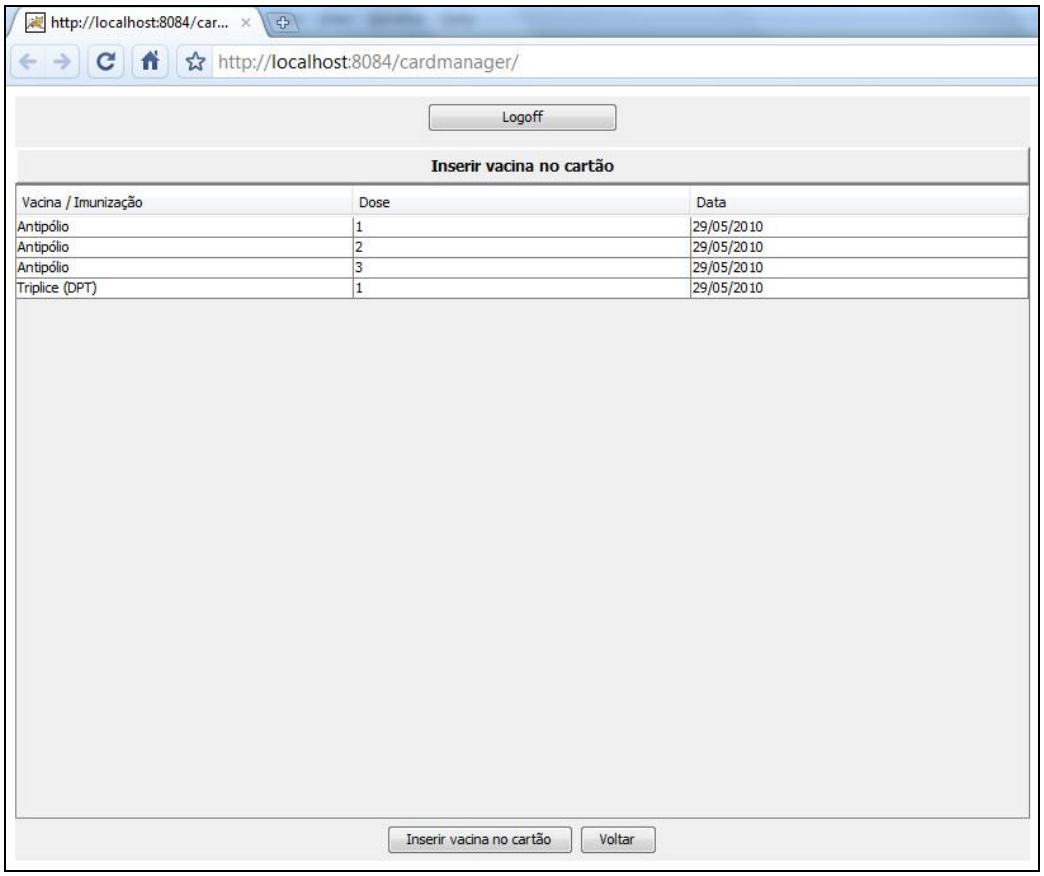


Figura 24 – Tela para inserir vacinas no cartão

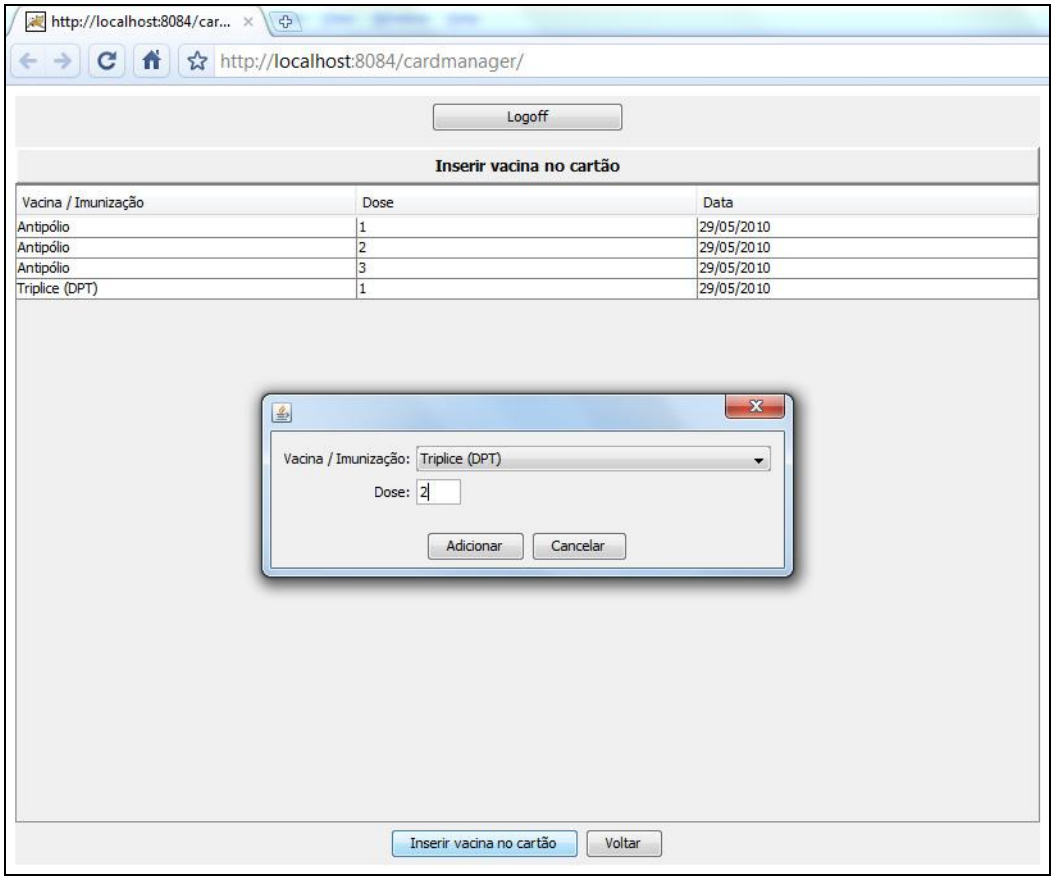


Figura 25 – Tela para inserir uma nova vacina ao cartão

Outra seção disponível ao agente de saúde é a de cadastro de vacinas no sistema, conforme Figura 26, onde as vacinas podem ser tanto cadastradas quanto editadas.

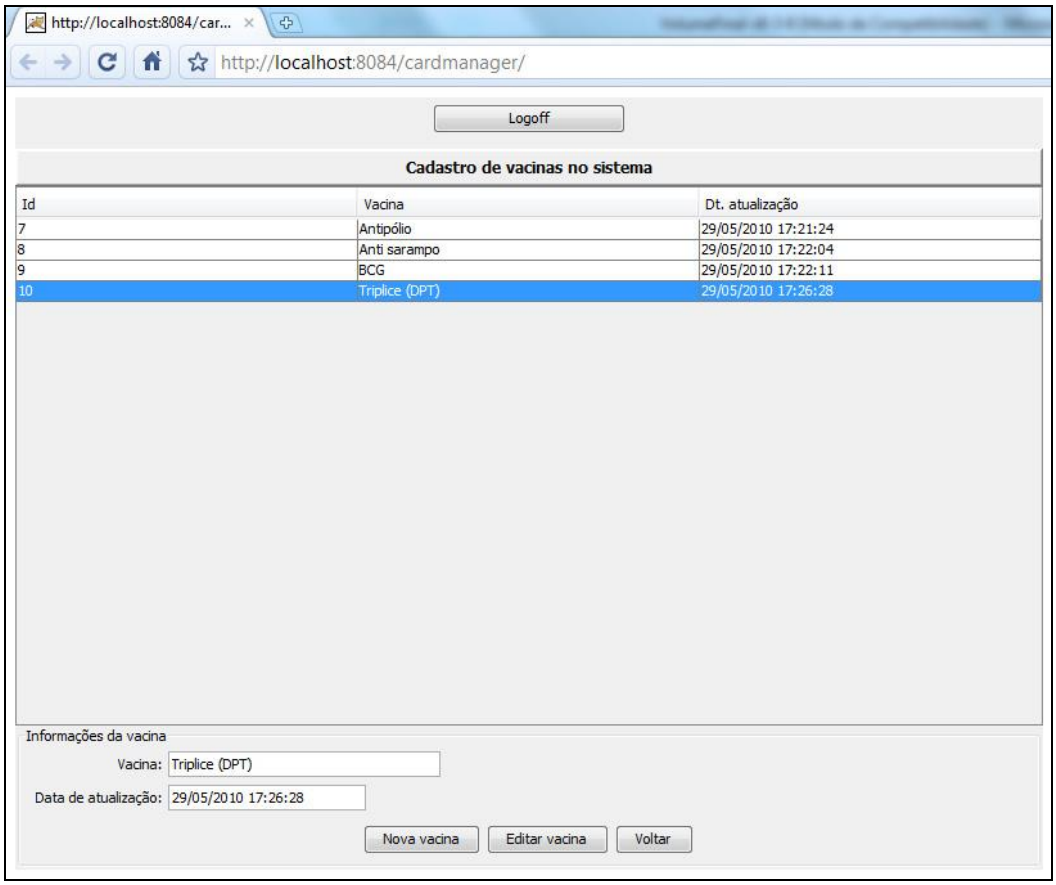


Figura 26 – Tela de cadastro de vacinas no sistema

Para cadastrar ou alterar uma vacina, deve-se clicar em Nova vacina ou Editar vacina e será aberta uma tela solicitando o nome da vacina, conforme Figura 27.

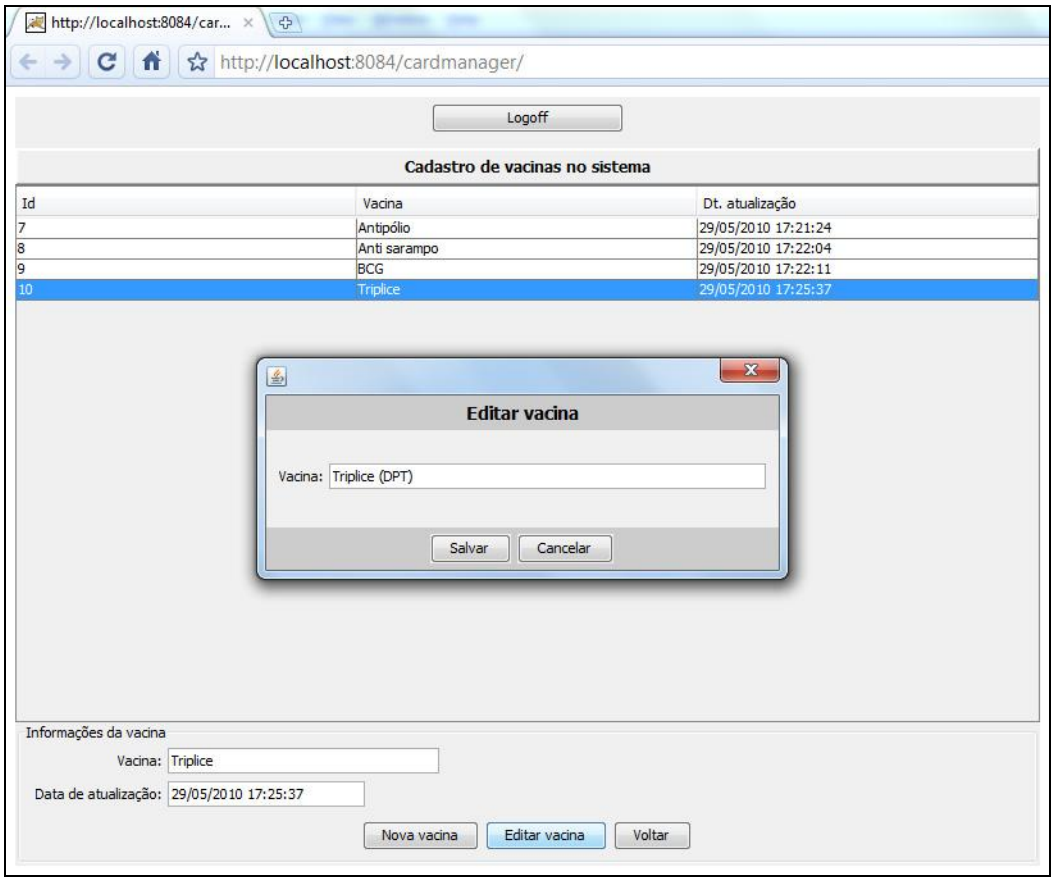


Figura 27 – Tela para cadastrar ou editar uma vacina no sistema

3.3.3.2 Portador da Carteira

Na tela de *login* deve-se informar o usuário e senha, e selecionar Portador como tipo de acesso. Ao selecionar este tipo de acesso, pode-se escolher a leitora que deseja utilizar para ler o cartão, conforme Figura 28.

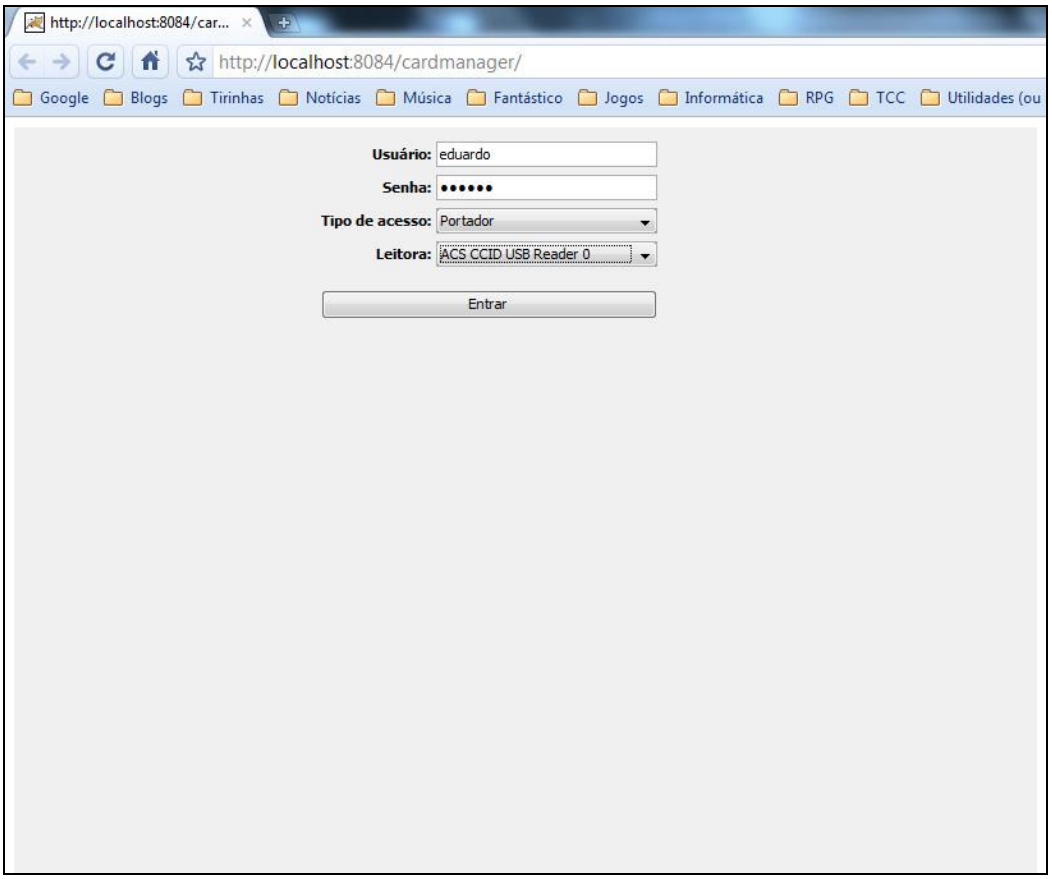


Figura 28 – Tela de *login* do portador de carteira

Caso o usuário não possua uma leitora conectada à máquina será emitido uma mensagem ao usuário informando-o do ocorrido, assim como se não houver um cartão na leitora, conforme Figuras 29 e 30.

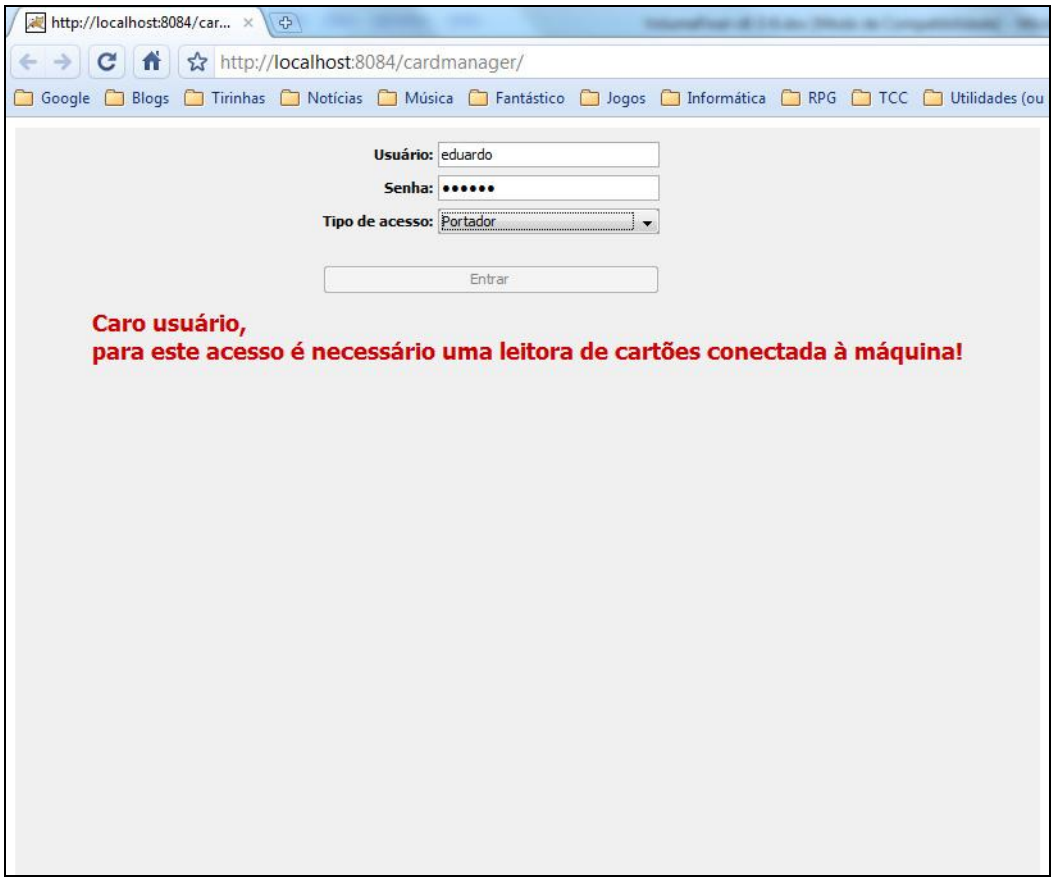


Figura 29 – Mensagem de aviso de falta de leitora

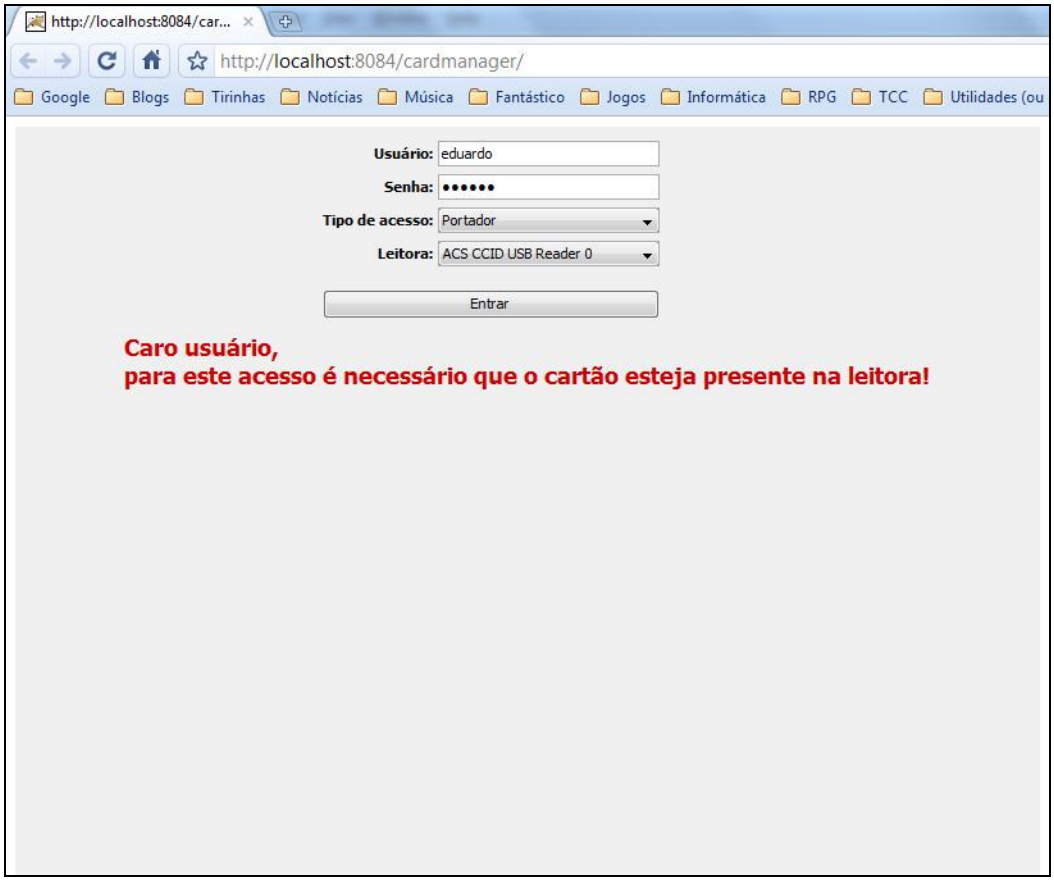


Figura 30 – Mensagem de falta de cartão na leitora

O portador pode escolher realizar três ações dentro do sistema, sendo elas: Consultar informações pessoais, Consultar vacinas registradas e *Logoff*, conforme Figura 31.

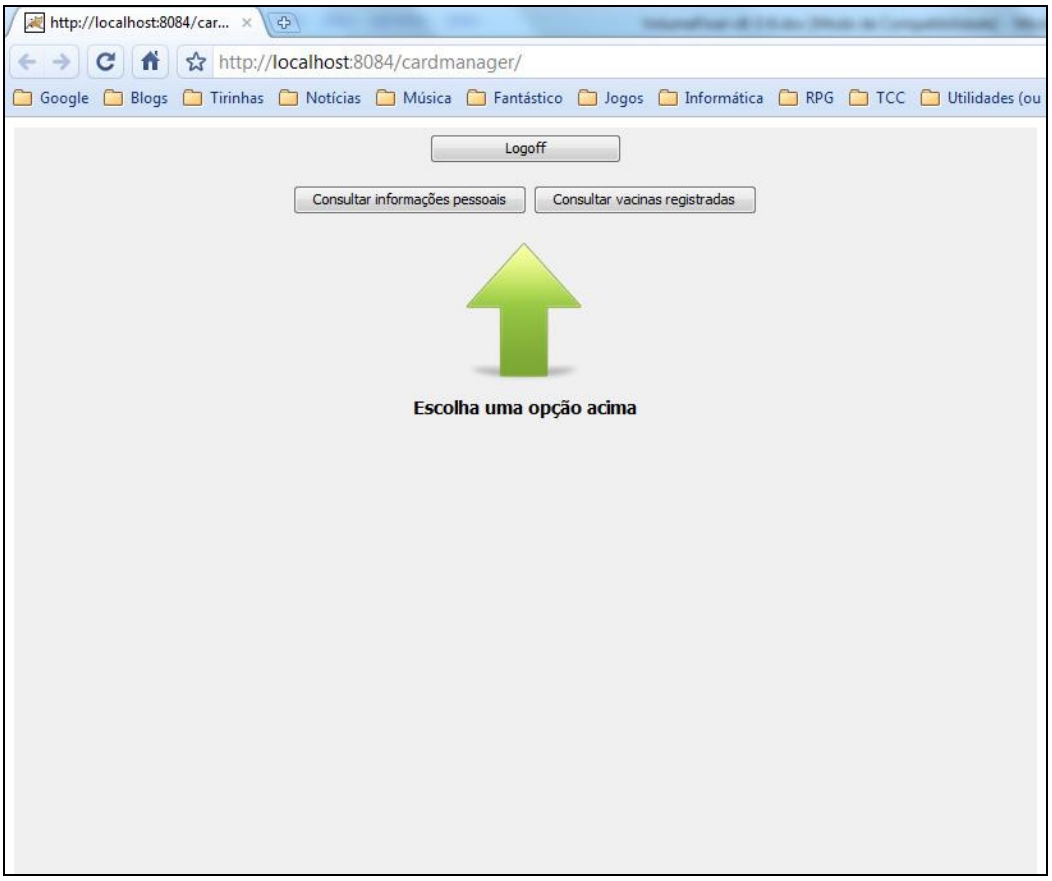


Figura 31 – Tela de ações do portador no sistema

Ao selecionar Consultar informações pessoais, será carregada a tela com as informações pessoais do portador que já estão cadastradas no cartão, conforme Figura 32. Estas informações não podem ser alteradas.

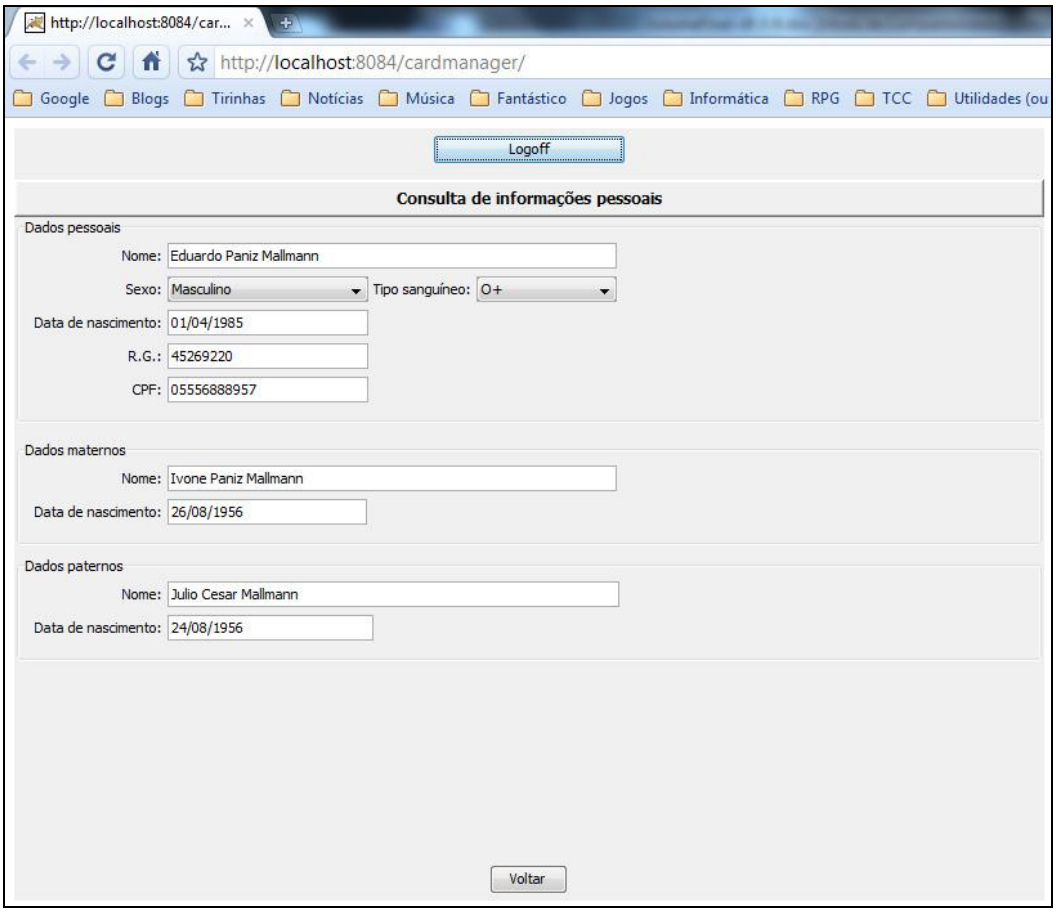


Figura 32 – Tela de consulta de informações pessoais

Ao selecionar a opção de Consulta de vacinas registradas, o portador pode visualizar todas as vacinas que estão registradas no cartão, conforme Figura 33.

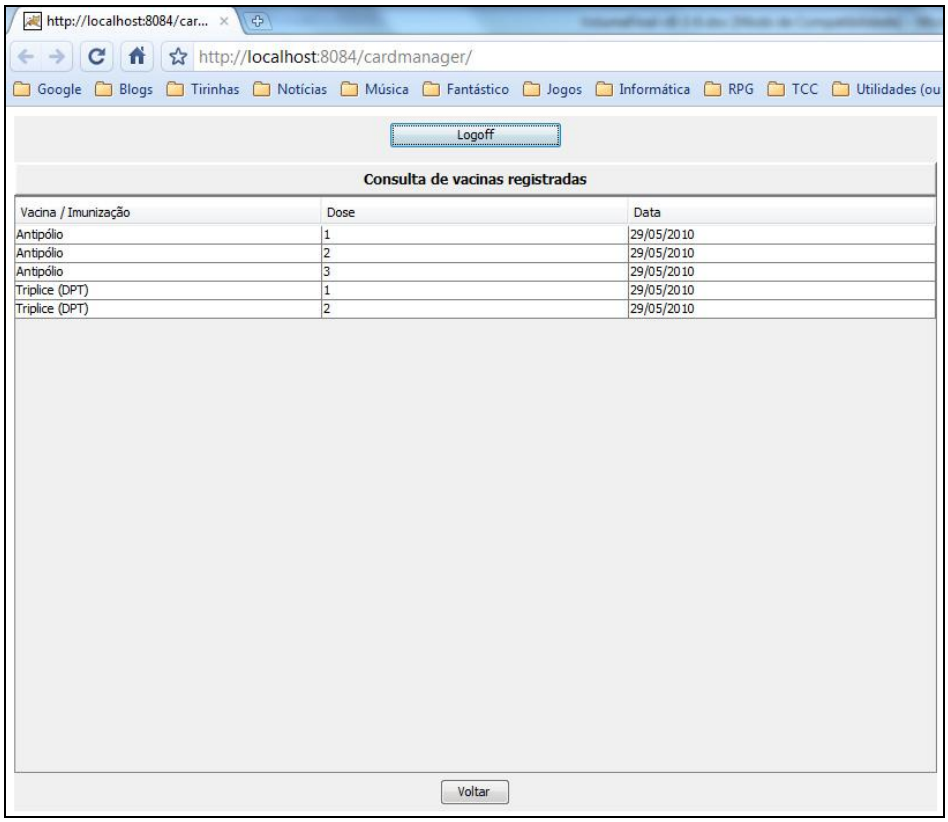


Figura 33 – Tela de consulta de vacinas registradas

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos ao final deste trabalho foram satisfatórios. Os objetivos do trabalho foram alcançados e o desenvolvimento do mesmo permitiu aumentar o conhecimento sobre a tecnologia e a mídia utilizada.

No Quadro 40 é apresentado um comparativo entre os trabalhos correlatos e o trabalho presente.

	SUAVI 2005	**MINORA; ALEIXO; DIOLINO 2007**	**MALLMANN 2010**
Utilização de *smart cards*	Sim	Sim	Sim
Persistência de dados no *smart card*	Não	Não	Sim
Envio de comandos ao *smart card*	Sim	Sim	Sim
Utilização do *smart card* pela internet	Não	Não	Sim
Utilização de rotina de criptografia	Sim	Não	Sim
Utilização de certificado digital	Não	Não	Não

Quadro 40 – Comparativo das ferramentas correlatas

Em relação aos trabalhos de Suavi (2005) e Minora, Aleixo e Diolino (2007), o presente trabalho se destaca por utilizar um *smart card* de forma real e prática, realizando persistência dos dados no cartão.

Destaca-se também o fato de a aplicação acessar o *smart card* pela internet através de um *applet*. Desta forma é possível trabalhar com o mesmo remotamente, desde que haja um CAD na máquina do usuário. Isto demonstra apenas uma fração de aplicações que podem ser criadas para esse tipo de mídia.

Os testes foram realizados utilizando dois *smart cards* e um CAD conforme o desenvolvimento da aplicação. Não foram feitos testes exaustivos com a aplicação ou com os *smart cards.*

4 CONCLUSÕES

Os requisitos propostos foram todos cumpridos, alcançando assim o objetivo principal do trabalho, o desenvolvimento de um controle de vacinas e imunizações utilizando Java *Card*.

O sistema permite a um agente de saúde cadastrar vacinas, registrar as informações pessoais de um portador e registrar vacinas ao mesmo, sendo essas informações salvas tanto no *smart card* quanto em uma base de dados externa. Permite ao portador do *smart card* visualizar as vacinas registradas e visualizar suas informações pessoais registradas no *smart card*, pela internet através de um *applet* carregado em um portal Web.

O uso da tecnologia Java *Card* foi de total importância para o trabalho e mostrou-se bastante completa para a realização deste trabalho, porém carente de documentação. Basicamente toda a documentação disponível encontra-se apenas no Javadoc das classes, muitas vezes faltando uma explicação mais detalhada sobre as classes e os métodos. Os exemplos fornecidos são úteis, mas como dito anteriormente, também carecem de uma documentação melhor e uma utilização mais completa de toda a tecnologia.

Uma grande dificuldade encontrada para a realização deste trabalho foi encontrar *smart cards* no Brasil. Foi encontrado somente em uma empresa, Sonsun, e mesmo assim o *smart card* disponibilizado está em uma versão atrasada da tecnologia Java *Card*. Isto acarreta não somente em um atraso tecnológico no país, mas também uma escassez de conhecimento e troca de informações sobre esta tecnologia. Apesar de o Brasil contar com uma das referências mundiais na plataforma Java *Card* (MEDEIROS, 2007), Igor Medeiros, durante a realização deste trabalho seu site encontrava-se fora do ar, assim como seu blog.

4.1 EXTENSÕES

Como extensão deste trabalho sugere-se a implementação de toda a caderneta de saúde, inclusive integrando as três disponíveis em uma só, sendo elas: Caderneta de Saúde da Criança, Caderneta de Saúde do Adolescente e Caderneta de Saúde da Pessoa Idosa.

Propõe-se como extensão deste trabalho também realizar testes exaustivos com a aplicação para verificar a durabilidade tanto da mídia quanto dos dados a longo prazo.

Sugere-se também fazer um estudo com a última versão da tecnologia Java *Card*, sendo ela 3.0.2 e um estudo mais detalhado entre suas versões CE e ConE.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, Claudia R. L. et al. Qualidade do preenchimento da caderneta de saúde da criança e fatores associados. **Cad. Saúde Pública**, Rio de Janeiro, v. 25, n. 3, mar. 2009. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-311X2009000300013&lng=pt&nrm=iso>. Acesso em: 10 abr. 2010.

CERTISIGN. **RIC está "travado" na agenda do Poder Executivo.** [S. l.], set. 2008. Disponível em: <https://www.certisign.com.br/certinews/banco_noticias/2008/09/ric-esta-travado-na-agenda-do-poder-executivo>. Acesso em: 10 abr. 2010.

CHEN, Zhiqun. **Java Card thecnology for smart cards:** architecture and programmer's guide. Massachusetts: Addison Wesley, 2000.

MEDEIROS, Igor. Smart Card I/O: Terminais Desktop Java para Smart Cards em 10 minutos. **MUNDOJAVA**, Rio de Janeiro, n. 25, p. 42, 2007.

MINORA, Leonardo A.; ALEIXO, Fellipe A.; DIOLINO, Gleison T.. **Smart interface:** ferramenta de auxílio ao desenvolvimento de aplicações Java Card. [Natal], 2007. Disponível em: <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/131/119>. Acesso em: 08 maio 2010.

PORTAL DA SAÚDE. **Portaria nº 964/GM de 23 de junho de 2005.** [S. l.], fev. 2009. Disponível em: <http://dtr2001.saude.gov.br/sas/PORTARIAS/Port2005/GM/GM-964.htm>. Acesso em: 10 abr. 2010.

SONSUN. **JCOP – Kit desenvolvimento javacard**. [S.l.], fev. 2010. Disponível em: <http://www.sonsun.com.br/sonsun_JCOP.php>. Acesso em: 09 jun. 2010.

SUAVI, Cleber G.. **Documentos e dinheiro eletrônico com *smart cards* utilizando tecnologia Java *Card.*** 2005. 104 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SUN MICROSYSTEMS. **Java Card technology overview.** [S. l.], abr. 2004. Disponível em: <http://java.sun.com/javacard/overview.jsp>. Acesso em: 26 mar. 2009.

_____. **Java Card 3.0 platform specifications.** [S. l.], mar. 2008. Disponível em: <http://java.sun.com/javacard/3.0/specs.jsp>. Acesso em: 28 mar. 2009.