UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Matteus Legat

SISTEMA DE CONTROLE DE ACESSO EM IOT

Florianópolis

2018

Matteus Legat

SISTEMA DE CONTROLE DE ACESSO EM IOT

Tese submetida ao Curso de Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Jean Everson Martina

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Legat, Matteus
Sistema de Controle de Acesso em IoT / Matteus Legat ; orientador, Jean Everson Martina, 2018.
189 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2018.

Inclui referências.

1. Ciências da Computação. 2. IoT. 3. Controle de Acesso. 4. RFID. I. Martina, Jean Everson. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Matteus Legat

SISTEMA DE CONTROLE DE ACESSO EM IOT

Esta Tese foi julgada aprovada para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovada em sua forma final pelo Curso de Ciências da Computação.

Florianópolis, 1 de junho 2018.

Prof. Dr. Rafael Luiz Cancian
Coordenador do Curso

Banca Examinadora:

Me. Lucas Pandolfo Perin

Prof. Dr. Jean Everson Martina
Orientador

Leonardo Meurer

RESUMO

O controle de acesso é essencial em ambientes onde se faz necessário restringir o acesso a apenas um certo grupo de pessoas. Em um sistema automático de acesso, utilizando um limitador de acesso, como portas com leitores ou catraca eletrônica, o acesso pode ser liberado somente após identificação do usuário que está efetuando a tentativa e a confirmação de sua permissão de acesso.
Atualmente, o controle de acesso na UFSC é realizado através de equipamentos terceirizados, com custo relativamente elevado para cada ponto controlado.
Para realizar uma contribuição científica para a sociedade buscando uma solução para o problema detalhado anteriormente, além de propor uma solução imediata de redução de custos para a implementação de áreas restritas na UFSC, este Trabalho de Conclusão de Curso propõe a construção de um sistema de Internet das Coisas para a realização do controle de acesso através de equipamentos com sensores RFID presentes nas portas de acesso aos Laboratórios, Áreas Restritas e ao Restaurante Universitário, capazes de identificar e controlar o acesso de indivíduos através de cartões de identificação compatíveis, como os utilizados atualmente na Universidade Federal de Santa Catarina.
Este sistema realizará o controle de acesso utilizando tecnologias de baixo custo e desenvolvidas na UFSC de forma eficiente.

Palavras-chave: IoT. Controle. Acesso. RFID. EPOS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

UFSC	Universidade Federal de Santa Catarina
EPOS	Embedded Parallel Operating System
IoT	Internet of Things
IoE	Internet of Everything
RFID	Radio-Frequency IDentification
LabSEC	Laboratório de Segurança em Computação
LISHA	Laboratório de Integração Hardware/Software
INE	Departamento de Informática e Estatística
ARM	Advanced RISC Machine
SoC	System-on-a-Chip
RISC	Reduced Instruction Set Computer
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
SeTIC	Superintendência de Governança Eletrônica e Tecnologia da Informação e Comunicação
GPIO	General Purpose Input/Output
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
CRC	Cyclic Redundancy Check
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
PICC	Proximity Integrated Circuit Card
PCD	Proximity Coupling Device
UID	Unique Identifier
VCC	Tensão em Corrente Contínua
TSTP	Trustful Space Time Protocol
SI	Sistema Internacional de Unidades
RU	Restaurante Universitário
LED	Light Emitting Diode
LCD	Liquid Crystal Display
CCA	Centro de Ciências Agrárias
DES	Data Encryption Standart

HTTP	Hypertext Transfer Protocol
MAC	Media Access Control

SUMÁRIO

1 INTRODUÇÃO

1.1 JUSTIFICATIVA

O controle de acesso de alunos, professores e servidores é uma tarefa essencial para que se possa garantir a segurança em áreas específicas da Universidade Federal de Santa Catarina. Até o presente momento, não existe um padrão definido sobre como este controle deve ser realizado, e a solução vigente consiste em um sistema de controle de acesso terceirizado, que possui um custo significativo para cada ponto de acesso.

Os estudantes, professores e servidores da Universidade Federal de Santa Catarina possuem um cartão de identificação que contém a tecnologia radio-frequency identification (RFID), que podem ser utilizados para identificação através de um sistema que seja compatível com a tecnologia. Utilizando esta tecnologia, é possível cadastrar os indivíduos em uma base de dados e identificá-los posteriormente.

Dispositivos de Internet das Coisas (IoT) estão sendo cada vez mais utilizados para se executar diferentes tipos de tarefas, principalmente quando se tratam de tarefas que não exigem muito poder computacional (EVANS, 2011). Um sistema de IoT possui a vantagem de não ficar concentrado em um único ponto, facilitando sua utilização em uma área de grandes dimensões, podendo ser acessado fisicamente em vários pontos, evitando-se assim o deslocamento até um ponto específico (VERMESAN; FRIESS, 2014).

Considerando esse contexto, é interessante o desenvolvimento de um sistema de IoT automatizado que controle a entrada e saída de alunos, professores e servidores à áreas restritas, utilizando-se de dispositivos presentes nas portas das salas capazes de identificar os indivíduos através da tecnologia RFID, controlando e registrando o acesso.

1.2 OBJETIVOS

1.2.1 Objetivos Gerais

Desenvolver um sistema de IoT automatizado que controle a entrada e saída de alunos, professores e servidores em áreas de acesso restrito, capaz de identificar alunos, professores e servidores através da

tecnologia RFID, controlando e registrando o acesso através de portas com trava ou catracas. Desenvolver um módulo Web para a realização do controle por parte do administrador, de modo a monitorar os registros e gerenciando permissões.

1.2.2 Objetivos Específicos

- Desenvolvimento de um Sistema IoT para o controle de acesso.
- Desenvolvimento de uma aplicação para o Sistema IoT.
- Montagem de um protótipo de catraca funcional.
- Montagem de um protótipo de porta funcional.
- Desenvolvimento de uma API REST.
- Desenvolvimento de um módulo de controle do administrador.
- Monografia do Trabalho.

1.3 ORGANIZAÇÃO

1.3.1 Métodos de Pesquisa

A pesquisa será realizada juntamente com o Laboratório de Segurança da Computação (LabSEC) do Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina (UFSC).

Inicialmente o trabalho possuirá caráter teórico, sendo realizado um levantamento de possíveis soluções para cada uma das etapas, quais os problemas podem ser apresentados para cada solução e maneiras de combater os problemas.

Quando terminada a etapa teórica, será feito desenvolvimento do sistema IoT. No início do desenvolvimento, é necessário fazer com que seja possível realizar as operações necessárias no hardware que será utilizado, implementando os componentes que ainda não existem no sistema a ser utilizado. O sistema deve ser capaz de controlar hardwares que realizam o controle de acesso, tais como portas com trava eletromagnética e catracas. Feito isso, será iniciado o desenvolvimento de uma aplicação capaz de identificar um indivíduo, armazenar as permissões de acesso de cada indivíduo e também os registros de acessos e tentativas.

A próxima etapa consiste em criar uma API Web capaz de conversar com a aplicação IoT, permitindo a modificação e atualização das políticas e permissões de indivíduos, a adição e remoção de usuários cadastrados e o recebimento dos registros de acesso e tentativas. Juntamente com a criação da API, será criado uma interface de controle para o administrador gerenciar as portas e catracas.

Após o término do desenvolvimento de todas as etapas, o sistema completo deve ser devidamente testado, e possíveis problemas que possam ocorrer na integração de suas partes devem ser corrigidos.

1.3.2 Tecnologias Adotadas

1.3.2.1 EPOS2

EPOS é um projeto que foca na automatização do desenvolvimento de sistemas para que os desenvolvedores foquem no que realmente importa: suas aplicações. EPOS conta com o método de Design de Sistema Embarcado Orientado a Aplicações (ADESD) para guiar o desenvolvimento de componentes de software e hardware que podem ser adaptados automaticamente para satisfazer os requisitos de aplicações específicas (Software/Hardware Integration Lab, a).

1.3.2.2 EPOSMote III

EPOSMote III é um componente de hardware desenvolvido no Laboratório de Integração de Software e Hardware (LISHA) que possui um Sistema-em-um-Chip (SoC) Texas Instruments CC2538, com um microcontrolador baseado no ARM Cortex-M3 de tamanho diminuto, utilizado principalmente em aplicações de IoT (FRÖHLICH; TIENTCHEU; SCHRÖDER-PREIKSCHAT, 2000).

1.3.2.3 TSTP

O TSTP é um protocolo orientado a aplicação projetado para proporcionar suporte a comunicação de dados autenticada, criptografada, cronometrada, georreferenciada e em conformidade com o SI para dispositivos IoT interagindo com um gateway IoT, inicialmente desenvolvido para o EPOS (RESNER; FRöHLICH, 2016).

1.3.2.4 RFID

Identificação por Radiofrequência (RFID) é uma forma de comunicação sem contato que utiliza ondas de rádio para identificar e rastrear objetos. Um sistema RFID é composto por leitores e etiquetas que se comunicam via rádio (WANT, 2006).

1.3.2.5 MFRC522

O MFRC522 é um chip altamente integrado de leitura / escrita para para uma comunicação sem contato a uma frequência de 13,56MHz. O leitor MFRC522 suporta cartões que seguem a norma ISO/IEC 14443 incluindo cartões A/MIFARE e NTAG e fornece as interfaces de comunicação SPI, UART e I2C (NXP SEMICONDUCTORS, 2016).

O transmissor interno do MFRC522 é capaz de controlar uma antena de leitura/escrita feita para se comunicar com cartões ISO/IEC 14443 A/MIFARE sem nenhum circuito ativo adicional. O módulo receptor provê de uma implementação robusta e eficiente para demodulação e decodificação dos sinais de cartões compatíveis com ISO/IEC 14443 A/MIFARE. Seu módulo digital gerencia as funcionalidades de framing e detecção de erros (paridade e CRC) (NXP SEMICONDUCTORS, 2016).

Leitoras que utilizam o chip MFRC522 são facilmente encontradas e possuem um custo baixo, principalmente quando comparado com outros modelos de leitoras de RFID.

1.3.2.6 ESP8266 ESP-01

O módulo ESP-01 é um módulo de Wi-Fi que utiliza o microcontrolador ESP8266, que integra um processador RISC de 32 bits com um baixo consumo de energia (SCHWARTZ, 2016). Ele permite a gravação de uma aplicação customizada pelo usuário, de acordo com as suas necessidades, e permite a comunicação serial utilizando UART.

1.3.2.7 Raspberry Pi

O Raspberry Pi é um computador de placa única, de baixo custo e alto desempenho, utilizando um processador ARMv8 de 64 bits, 1 GB de RAM e chip WiFi integrado (RICHARDSON; WALLACE, 2012).

Um recurso da Raspberry Pi é a coluna de GPIO, que pode ser controlado por software e pode ser usado com uma variedade de funções, permitindo a utilização de diversos dispositivos periféricos (UPTON; HALFACREE, 2014).

1.3.2.8 REST

REST significa Representational State Transfer (Transferência de Estado Representacional), e pode ser definido como um estilo arquitetural utilizado em aplicações web tipicamente voltadas à comunicação com outras aplicações (FIELDING, 2000).

1.3.2.9 Node.js

Node.js é um interpretador JavaScript assíncrono orientado a eventos, projetado para construir aplicações de rede escaláveis. O Node.js permite que desenvolvedores utilizem JavaScript para programação no lado do servidor, para gerar dinamicamente o conteúdo de uma página antes de ela ser enviada ao usuário.(Node.js Foundation, a)

2 FUNDAMENTAÇÃO TEÓRICA

2.1 AUTENTICAÇÃO

Autenticação é uma identificação positiva, com um grau de certeza suficiente para permitir certos direitos ou privilégios à pessoa ou coisa identificada positivamente. Em termos gerais, ela é "O ato de verificar a identidade alegada de um indivíduo, estação ou originador". (BOSWORTH; KABAY, 2002)

Os métodos clássicos para se correlacionar identidades físicas e virtuais são paralelos aos métodos utilizados para se autenticar humanos no mundo físico. Existem quatro maneiras de se autenticar de informações, e elas são definidas com base em:

- Algo que você sabe - como um código ou senha;
- Algo que você tem - um token que pode ser algo como uma credencial, uma chave, um certificado ou um documento de identidade;
- Algo que você é - verificação de atributos biométricos como impressões digitais ou rosto;
- Algo que você faz - e.g. maneira de falar ou assinar.

(BOSWORTH; KABAY, 2002)

2.2 RFID

Nos últimos anos, a identificação por radiofrequência (RFID) tem se tornado cada vez mais utilizada. Ela permite a identificação de etiquetas em até uma certa distância, e suas etiquetas são mais práticas e suportam uma quantidade maior de identificadores únicos do que os códigos de barra (WANT, 2006).

Etiquetas RFID são pequenas e requerem pouca energia para funcionar, não necessitando de baterias e podendo ser ativadas por indução eletromagnética. Isto faz com que elas sejam baratas e torna fácil sua aplicação em qualquer objeto que se queira identificar ou rastrear.

Podemos dividir as RFID em duas classes. A primeira delas é a classe ativa, cuja etiquetas requerem uma fonte de energia para funcionar, necessitando estarem conectadas a uma alimentação externa ou alimentadas por uma bateria interna. Um exemplo de aplicação de um sistema de RFID ativo são os transponders de aeronaves, que identificam a aeronave e sua origem. A segunda classe é a RFID passiva, cuja etiquetas não requerem baterias ou manutenção.

As etiquetas passivas são constituídas de três partes: uma antena, um chip anexado à antena e algum invólucro, que pode ser desde um pequeno frasco de vidro até adesivos ou carteirinhas. O leitor de etiquetas é responsável por alimentar a etiqueta e se comunicar com a etiqueta, enquanto a antena captura a energia e a transfere o seu identificador único da etiqueta, em um processo controlado pelo chip (WANT, 2006).

As carteirinhas atualmente utilizadas pelos estudantes da UFSC já contam com a tecnologia de RFID passiva de curto alcance, aonde o cartão é alimentado por indução eletromagnética ao ser colocado a uma curta distância do leitor. O cartão então envia os dados ao leitor utilizando modulação de carga, puxando corrente da bobina da antena e gerando variações em seu pequeno campo magnético. Essas pequenas diferenças são detectadas pela bobina do leitor como um pequeno aumento na corrente circulando nele, permitindo que o leitor receba os dados do cartão (WANT, 2006).

2.3 CONTROLE DE ACESSO

O controle de acesso eletrônico é uma tarefa muito comum utilizada em ambientes nos quais se queira restringir o acesso a apenas um certo grupo de pessoas. Em sua forma mais simples, um sistema de controle de acesso consiste em uma fechadura eletrônica, um leitor (como um leitor de cartões) e algum tipo de controlador eletrônico (DEUTSCH, 2018).

Os leitores são montados do lado de fora das portas e são a única parte do sistema de controle de acesso que a maioria das pessoas vê. Em um sistema de controle de acesso moderno, os leitores são responsáveis por alguma maneira de autenticação. Se o sistema usar um leitor de código, você insere um número de identificação pessoal (PIN) em um teclado para se identificar no sistema. Com um leitor de credenciais, você apresentaria um cartão ou chaveiro. Um leitor biométrico deve ler uma parte de você (DEUTSCH, 2018).

As credenciais de controle de acesso geralmente vêm na forma de cartões ou etiquetas que podem ser penduradas no chaveiro. As credenciais mais comuns são cartões de identificação por radiofrequência (RFID) que podem ser lidos à distância. Em alguns casos, eles não precisam ser removidos do bolso para serem usados (DEUTSCH, 2018).

Um sistema de controle de acesso eletrônico pode permitir uma auditoria de maneira descomplicada. Um sistema eletrônico pode manter um registro de hora e data de cada acesso ou tentativa de acesso à uma área de acesso restrito, podendo identificar sobre a pessoa que realizou a tentativa de acesso (DEUTSCH, 2018).

Um sistema de acesso eletrônico também pode permitir a restrição de acesso com base em diferentes políticas de acesso, tais como restrição de hora e dia em que o acesso pode ser realizado por uma determinada pessoa, quantidade máxima de vezes que uma determinada pessoa pode acessar em um intervalo de tempo, e outros critérios, como o saldo do usuário disponível no sistema. Um sistema EAC permite que se criem "chaves"personalizadas que só operam em com base nas políticas definidas.

Uma chave perdida ou roubada gera uma grave ameaça de violação de sua segurança física. Reajustar travas mecânicas pode ser muito caro e inconveniente. Por outro lado, uma credencial de controle de acesso eletrônico geralmente pode ser excluída ou desativada em questão de minutos a um baixo custo. Adicionalmente, o sistema de controle de acesso eletrônico pode informar quando e onde alguém tentou desbloquear uma porta com a credencial desativada (DEUTSCH, 2018).

2.4 CRIPTOGRAFIA

A criptografia tem como objetivo embaralhar dados, tais como mensagens e informações, de maneira que somente determinados indivíduos possam desembaralhar e ter acesso ao conteúdo original dos dados. Um algoritmo criptográfico é uma sequência de passos que são executados para cifrar e decifrar dados. Antes do envio de um dado, um indivíduo escolhe uma forma de cifrar a mensagem, incluindo o algorítimo e uma chave. O destinatário final do dado deve conter uma chave específica de acordo com o algoritmo e chave utilizados pelo remetente, caso contrário ele não será capaz de decifrar o dado recebido (HOUSLEY; POLK, 2001).

Existem diversos algoritmos criptográficos que são diferenciados pela segurança oferecida e pelo tipo de chave utilizada (HOUSLEY; POLK, 2001).

2.4.1 Criptografia Simétrica

A criptografia simétrica utiliza única chave para para realizar a cifragem e a decifragem, que é compartilhada entre o emissor e o destinatário de um dado. Essa chave é uma sequência de bits que irá definir a forma como o algoritmo irá cifrar (ou embaralhar) um dado (TRIPATHI; AGRAWAL, 2014).

Uma vantagem na utilização da criptografia simétrica é uma boa performance com a possibilidade de se realizar uma comunicação contínua entre vários indivíduos simultaneamente. Caso a chave seja comprometida, isto é, obtida por alguém que não deveria ter acesso a ela, basta efetuar a troca por uma nova, mantendo o mesmo algoritmo (TRIPATHI; AGRAWAL, 2014).

O nível de segurança de um sistema criptográfico varia de acordo com o tamanho da chave utilizada. Um exemplo é um algoritmo baseado em DES, que possui chave de 56 bits , permitindo a criação de 72 quadrilhões de chaves diferentes. Mesmo sendo um número elevado, esse padrão já é considerado inseguro diante da capacidade de processamento dos dispositivos atuais (TRIPATHI; AGRAWAL, 2014).

Outros sistemas podem possuir chaves de tamanho variável, como o RC2, que utiliza o protocolo S/MIME e possui uma chave com tamanho entre 8 e 1.024 bits, reduzindo consideravelmente as chances de alguém conseguir decifrar um conteúdo criptografado por meio de força bruta (TRIPATHI; AGRAWAL, 2014).

Apesar de possuir um alto desempenho, a criptografia simétrica possui graves falhas de segurança. Ela não possui meios que permitam a verificação de identidade de quem envia ou recebe um dado. Além disso, não há como garantir o armazenamento das chaves em ambientes confiáveis. A gestão de chaves também torna-se mais complexa conforme o número de indivíduos comunicando-se aumenta (TRIPATHI; AGRAWAL, 2014).

2.4.2 Criptografia Assimétrica

A criptografia assimétrica, conhecida também por criptografia de chave pública, é baseada em pares de chave: uma chave privada e outra pública. Elas podem ser utilizadas para cifrar dados e para a verificação de identidade de usuários (TRIPATHI; AGRAWAL, 2014).

A chave privada é utilizada para decifrar dados, enquanto a pública é utilizada para cifrar um dado. Assim, qualquer indivíduo que queira enviar um conteúdo para alguém necessita apenas da chave pública do seu destinatário, que utilizará a chave privada para decifrar a mensagem. Um dado cifrado com uma determinada chave pública pode somente ser decifrado pela chave privada correspondente. (TRIPATHI; AGRAWAL, 2014).

Esse sistema garante a privacidade de usuários e aumenta a confiabilidade de uma troca de dados, pois a quantidade de indivíduos com acesso à chave privada é bastante limitada, reduzindo consideravelmente as chances de se comprometer a segurança de uma comunicação (TRIPATHI; AGRAWAL, 2014).

Um dos principais algoritmos que utiliza a criptografia assimétrica é o RSA, que é baseado na multiplicação de números primos grandes para a geração de um par de chaves. Outros algoritmos, como o ElGamal e o de Curvas Elípticas são baseados em diferentes tipos de operações matemáticas, tais como operações com logaritmos discretos e curvas elípticas sobre corpos infinitos para a criação de chaves (TRIPATHI; AGRAWAL, 2014).

2.4.3 Resumo Criptográfico

Uma função de resumo é um método que, ao ser aplicado sobre um dado de tamanho independente, gera um resultado único e de tamanho fixo chamado de hash. Uma função de hash é unidirecional, ou seja, não é possível reverter o processo para se obter o dado original partindo-se do hash (CERT.br, 2017).

2.4.4 Sal

Um sal, em criptografia, é um dado aleatório adicionado à entrada de uma função de resumo criptográfico. Sua principal função é fazer que uma mesma senha utilizada por diferentes usuários tenham valores de hash diferentes, impedindo ataques que se apropriem desta característica (WOSCHEK, 2015).

2.5 INTERNET DAS COISAS

O termo "Internet" se refere à vasta categoria de aplicações e protocolos construídos sobre uma sofisticada e interconectada rede de computadores. Com a evolução da internet, o foco tem mudado em direção a uma integração de pessoas e dispositivos para convergir o domínio físico com ambientes virtuais construídos pelo ser humano, criando a chamada utopia da Internet das Coisas (BUYYA; DASTJERDI, 2016).

O termo "Internet of Things" (ou "Internet das Coisas") foi introduzido por Kevin Ashton em 1999. Ele acredita que o modo como interagimos e vivemos com as "coisas" do mundo físico necessita de reconsiderações sérias, por conta dos avanços em computação, internet e na taxa de geração de dados por dispositivos inteligentes (BUYYA; DASTJERDI, 2016).

Desde então, várias definições de IoT foram apresentadas, incluindo a definição que foca majoritariamente nos requisitos de conectividade e equipamentos de sensoriamento para entidades envolvidas em ambientes típicos de IoT. Enquanto estas definições discutem os requisitos básicos de IoT, novas definições como "Internet of Everything" (ou "Internet de Tudo") dão mais valor à necessidade de redes autônomas e ubíquas de objetos onde a integração do serviço e a identificação possuem um papel importante e inevitável. Por exemplo, o termo IoE é utilizado pela Cisco para se referir a pessoas, coisas e locais que podem expor seus serviços às outras entidades (BUYYA; DASTJERDI, 2016).

A IoT é criada através de interconexões de diversos objetos, como veículos, celulares, habitações e ocupantes das habitações. Ela utiliza obtenção de informações de baixo custo e dispositivos de disseminação, tais como sensores e etiquetas de RFID, que facilitam interações rápidas entre os próprios objetos, assim como entre pessoas e objetos, em qualquer lugar e a qualquer hora (ZHENG et al., 2011).

Não podemos considerar a IoT como sendo uma mera extensão da Internet, ou como uma interconexão de sistemas de Internet. Ela representa sistemas de ponta-a-ponta inteligentes que permitem soluções inteligentes, abrangendo diversas áreas da tecnologia, tais como sensoriamento, comunicações e processamento de informações (ZHENG et al., 2011).

3 PROPOSTA DE SISTEMA

A proposta consiste no desenvolvimento de um sistema que auxiliará no controle de acesso à áreas restritas na UFSC. A proposta pode ser subdividida em duas etapas, sendo elas o desenvolvimento de um sistema de IoT para auxiliar no controle de acesso, e o desenvolvimento de uma aplicação para que o administrador do sistema possa acompanhar e realizar este controle.

3.1 SISTEMA DE IOT

Considerando que os estudantes, professores e servidores da UFSC já utilizam carteiras de identificação com suporte a RFID, serão utilizadas as carteiras já existentes para a identificação de um aluno ou servidor para liberação do acesso.

Será implementado um sistema com nodos sensores distribuídos nas áreas de acesso restrito da universidade. A proposta é que cada porta de acesso à área restrita tenha um sensor de RFID capaz de ler os cartões na porta, de modo que eles sejam capazes de identificar um indivíduo e liberar ou não o seu acesso, e registrando esta informação para consulta pelo administrador.

Cada nodo sensor fará parte de uma rede de IoT que servirá como controle de acesso eletrônico aos indivíduos credenciados. Para a construção desta rede, serão utilizados dispositivos EPOSMote III, produzidos pelo LISHA (FRÖHLICH; TIENTCHEU; SCHRÖDER-PREIKSCHAT, 2000), que rodarão o sistema operacional EPOS 2 utilizando o protocolo TSTP. Alguns dos dispositivos na rede atuarão como gateway, estando conectados com a internet e transmitindo as informações de tentativa de acesso para o servidor da aplicação de controle.

Serão utilizados sensores com o chip MFRC522, cujo componente controlador para o sistema operacional EPOS será desenvolvido neste trabalho, bem como as aplicações que rodarão nos nodos da rede IoT. Será desenvolvida a aplicação dos nodos sensores, responsável por permitir ou não o acesso, informando para a rede sobre a entrada e saída de indivíduos, assim como a aplicação de gateway para permitir a interação entre a rede e a aplicação de controle.

3.1.1 Catraca

A catraca possui uma trava eletromecânica, composta por um solenoide, controlado por um relé de 12 VCC, e dois sensores de LED infravermelho para o controle de giro e ativação da trava. Neste trabalho será implementado um controlador utilizando EPOSMote III para o controle de acesso da catraca, além de um leitor de cartão RFID modelo MFRC522.

3.1.1.1 Requisitos Funcionais

Para a aplicação que será executada na catraca, foram levantados os seguintes requisitos funcionais:

- A aplicação deve realizar a identificação de cartões RFID na catraca.
- A aplicação deve ser capaz de permitir ou bloquear o giro da catraca, realizando o controle de sua trava.
- Deve ser realizado o controle de giro da catraca, detectando quando um giro está sendo efetuado e quando ele é completo, bem como o sentido do giro.
- Quando um cartão é identificado pela catraca, a informação deve ser transmitida para um sistema que efetue o seu registro.
- Quando um cartão é identificado pela catraca, deve ser determinado se aquele cartão está relacionado a um indivíduo autorizado naquele momento.
- O registro de cartões de indivíduos autorizados e suas políticas de acesso deve poder ser atualizado.
- Quando um cartão relacionado a um individuo autorizado é identificado, deve se permitir a realização de um giro na catraca.
- Quando uma tentativa de giro é iniciada sem autorização, deve ser ativada a trava da catraca até o término da tentativa.

3.1.2 Porta

A porta utilizará uma trava eletromagnética de 12 VCC, além de um botão para a abertura pelo lado interno. Neste trabalho será implementado um controlador utilizando EPOSMote III para o controle de acesso da porta, além de um leitor de cartão RFID modelo MFRC522, instalado do lado externo da porta.

3.1.2.1 Requisitos Funcionais

Para a aplicação que será executada na porta, foram levantados os seguintes requisitos funcionais:

- A aplicação deve permitir a identificação de cartões RFID na porta.
- A aplicação deve ser capaz de permitir ou impedir a abertura da porta, realizando o controle de sua trava.
- Quando um cartão é identificado pela porta, a informação deve ser transmitida para um sistema que efetue o seu registro.
- Quando um cartão é identificado pela porta, deve ser determinado se aquele cartão está relacionado a um indivíduo autorizado naquele momento.
- Quando um cartão relacionado a um individuo autorizado é identificado, a porta deve ser destravada por um período de tempo, permitindo a sua abertura.
- O registro de cartões de indivíduos autorizados e suas políticas de acesso deve poder ser atualizado.

3.2 APLICAÇÃO PARA CONTROLE E AUDITORIA

Para que o controle de acesso seja propriamente realizado, será desenvolvida uma aplicação web para o controle e acompanhamento do acesso de alunos, professores e servidores. Esta aplicação receberá os dados do sistema de IoT sobre o acesso à áreas restritas.

A aplicação possuirá uma interface para controle por parte do administrador, que após autenticado no sistema, poderá verificar os indivíduos que obtiveram acesso à área restrita. O sistema utilizará o horário e a identificação da área com acesso controlado.

3.2.1 Requisitos Funcionais

Para o sistema de controle e auditoria, foram levantados os seguintes requisitos funcionais:

- O sistema deve possuir autenticação por meio de usuário e senha.
- O sistema deve permitir a configuração de portas por um administrador.
- O sistema deve permitir que um administrador adicione novos administradores.
- O sistema deve permitir a configuração de portas conectadas por administradores.
- O sistema deve permitir que um administrador adicione novos gerentes de portas.
- O sistema deve permitir que um administrador controle quais portas um gerente terá acesso.
- O sistema deve permitir que os administradores e gerentes de uma porta possam adicionar e remover usuários com autorização de acesso.
- O sistema deve enviar alterações nas permissões de acesso de usuários às respectivas portas.
- O sistema deve receber informações de tentativas de acesso das portas e armazená-las em um registro.
- O sistema deve permitir que os administradores e gerentes de uma porta possam visualizar o seu registro de tentativas de acesso.

4 IMPLEMENTAÇÃO

4.1 MFRC522 NO EPOS

4.1.1 Comunicação via SPI

Para construir uma aplicação que utilize o módulo RFID no EPOSMote III é necessário implementar uma interface de comunicação entre os dois componentes. O meio de comunicação adotado para ser utilizado entre esses dois componentes foi o SPI.

O SPI é um protocolo de comunicação desenvolvido pela Motorola que permite a comunicação do microcontrolador com outros componentes, especificando uma interface de comunicação síncrona, utilizada amplamente e principalmente em sistemas embarcados. Um barramento SPI com um único escravo possui os pinos SCLK (Serial Clock), MOSI (Master Output Slave Input), MISO (Master Input Slave Output) e $\overline{SS}$ (not Slave Select) (MOTOROLA, INC., 2003).

O sistema operacional EPOS já possui um componente pronto para utilizar a comunicação SPI no EPOSMote III, que utiliza o módulo SSI de seu SoC. Nele contém todos os métodos necessários para fazer as transferências de dados, sendo possível também inicializá-lo com diferentes parâmetros de acordo com as funcionalidades desejadas. A implementação do MFRC522 para o EPOSMote III foi feita utilizando esse meio de comunicação para escrever e ler em seus registradores.

Um problema enfrentado ao utilizar o modulo SPI do EPOS foi que o seu pino de seleção de escravo não é atualizado ao encerrar uma conexão, o que é necessário para que o MFRC522 funcione corretamente. Como solução, foi utilizado um outro pino como $\overline{SS}$ que é atualizado via GPIO no inicio e no final de cada transação.

Na figura (1) é possível analisar o funcionamento do SPI através de um exemplo de escrita em um registrador do leitor MFRC522.

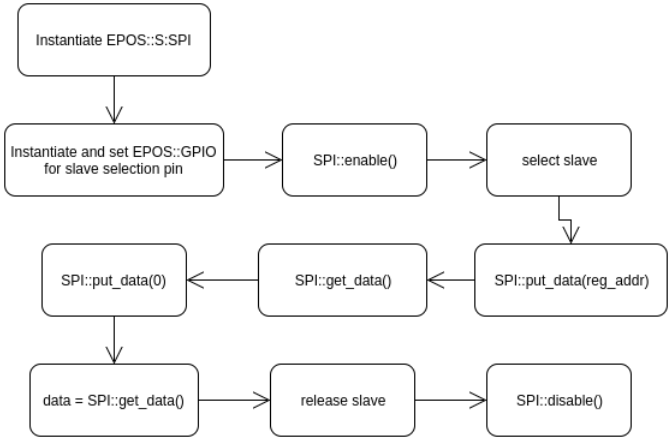

Figura 1 – Exemplo da escrita no MFRC522 utilizando SPI.

O primeiro passo do exemplo é instanciar o componente de comunicação SPI com os parâmetros relacionados ao leitor, como o seu clock e o modo de comunicação do SPI, que podem ser encontrados em sua especificação.

No próximo passo é habilitada a interface SPI para que se possa realizar a transação de dados. O MFRC522 é selecionado como escravo através do pino $\overline{SS}$ e então o mestre EPOSMote III envia via SPI o endereço do registrador no qual ele quer escrever, com o bit mais significativo de valor 1 para indicar uma leitura. Após isso, o chip MFRC522 responde um valor que não importa, e já se pode solicitar o byte com o dado do registrador enviando o valor 0 via SPI, para que então, o chip responda o dado via SPI.

Após a obtenção do dado, a seleção do escravo deve ser desfeita e a interface SPI deve ser desabilitada, para que o MFRC522 saiba que a transação atual terminou, permitindo que a próxima transação possa ser executada sem nenhum conflito com última.

A escrita de registradores funciona de maneira semelhante, mudando apenas o bit mais significativo do endereço para 0, indicando uma escrita, e também o segundo valor enviado via SPI, que passa a ser o dado que se deseja escrever no registrador. Também é possível solicitar a leitura ou a escrita de registradores subsequentes em uma única transação (NXP SEMICONDUCTORS, 2016).

4.1.2 Seleção e Anticolisão de um PICC

Para que um leitor de cartão (PCD) inicie uma comunicação com um cartão sem contato com circuito integrado (PICC), é necessário seguir uma sequência de passos definida pelo padrão ISO/IEC 14443-3, que pode ser melhor entendida analisando o diagrama da Fig. (2) (NXP SEMICONDUCTORS, 2016) (ISO/IEC, 2001).

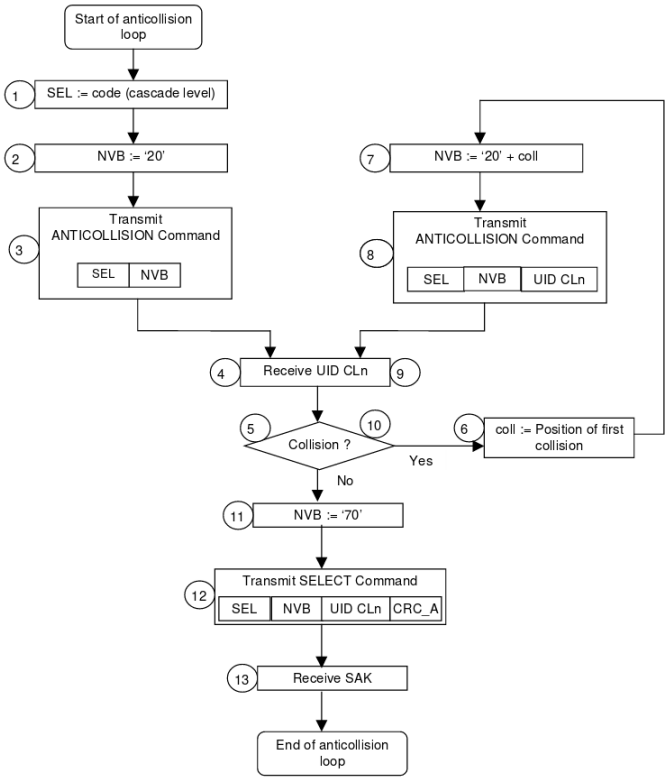

Figura 2 – Processo de anticolisão conforme a ISO/IEC 14443-3 (ISO/IEC, 2001).

Antes de se iniciar uma comunicação, caso seja necessário aguardar pela presença de um cartão (PICC) próximo ao leitor (PCD), deve-se realizar um polling como o definido pela ISO/IEC, solicitando ao PCD o envio dos comandos REQA (Request Type A) ou REQB (Re-

quest Type B) periodicamente, onde o tipo da requisição depende do tipo do cartão. O PCD pode ser consultado para saber se foi recebida alguma resposta ao REQA ou REQB, ou seja, se existe algum PICC presente (NXP SEMICONDUCTORS, 2016).

Com o cartão presente próximo ao leitor, deve-se iniciar o processo de seleção e anticolisão de um cartão. Uma colisão pode ocorrer quando existem dois ou mais PICCs que enviam valores diferentes e pode ser detectada pelo PCD. O mecanismo de anticolisão permite que um único PICC seja selecionado mesmo quando existirem mais de um PICC (NXP SEMICONDUCTORS, 2016).

No processo de anticolisão, o PCD deve enviar o comando de anticolisão para o(s) PICC(s) e o nível de cascata, e então receber o código de identificação único (UID) para o nível solicitado do cartão. Caso tenha sido detectada alguma colisão, o comando de anticolisão é reenviado com o ID recebido até a posição em que ocorreu a colisão, até o momento em que não ocorram mais colisões e o UID esteja completo (NXP SEMICONDUCTORS, 2016).

No momento em que não houverem mais colisões, deve ser calculado um valor para a verificação de redundância cíclica (CRC), para ser enviado no comando de seleção, juntamente com o UID obtido no processo de anticolisão. Após o comando de seleção, se não ocorreu nenhum erro no procedimento, o PICC deve receber uma confirmação de seleção (SAK) e então o cartão estará pronto para a comunicação (NXP SEMICONDUCTORS, 2016).

A implementação da interface de comunicação com o MFRC522 foi realizada seguindo as especificações em sua ficha de dados. Ela é dividida em dois módulos, sendo eles o PICC, que é independente de arquitetura, e o módulo MFRC522 que representa um leitor e possui uma classe genérica, independente de arquitetura, e outra classe específica para a maquina Cortex-M, conforme descrito na Fig. (3).

A classe abstrata do módulo MFRC522 contém enumeradores com os endereços de registradores do leitor e também os valores utilizados para cada comando. Ela também possui declarações de métodos virtuais para funções que devem ser realizadas pelo leitor, que devem ser implementadas pelas classes específicas de cada máquina.

Quando a classe abstrata é compilada, a classe com a sua implementação para a arquitetura na qual o sistema está sendo compilado é compilada juntamente, pois ela é definida no header com a configuração do EPOS.

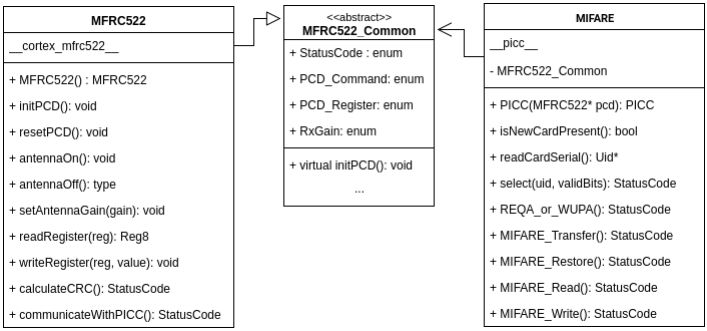

Figura 3 – Diagrama com as novas classes implementadas.

A classe desenvolvida para a máquina Cortex-M possui a implementação dos métodos virtuais da classe abstrata, utilizando para isso os módulos de GPIO e SPI do EPOS próprios para a máquina, utilizando uma pinagem pré-determinada para os componentes de hardware. Uma outra classe com a implementação para outra máquina pode ser desenvolvida da mesma maneira, apenas utilizando os módulos necessários para a comunicação com o MFRC522 nessa outra máquina.

Já o módulo PICC implementa as funções de um cartão (PICC) de acordo com a ISO/IEC 14443-3, com algumas funções específicas para o cartão do tipo MIFARE Classic. Nesse módulo estão contidos os valores dos comandos que podem ser utilizados na comunicação entre o cartão e o leitor. Ele realiza a comunicação com um PICC através de uma instância do módulo MFRC522, que é recebida como parâmetro em seu construtor.

4.2 CATRACA

Durante a execução do trabalho, foram desenvolvidos diferentes protótipos de catraca com três diferentes modelos de catraca, sendo elas a Catraca Lumen Balcão da Henry, a Catraca Slim da Tecnibra e uma catraca antiga do RU da UFSC, sem identificação de marca ou modelo.

Todos os três modelos de catraca possuem um funcionamento interno bastante semelhante, com a trava sendo um solenoide de 12 VCC, e um mesmo sistema mecânico com dois sensores infravermelhos para que se possa controlar o giro.

4.2.1 EPOSMote III

Foram desenvolvidos protótipos de catracas controladas pelo dispositivo EPOSMote III com o modelo de caraca Slim da Tecnibra e com uma antiga catraca do RU da UFSC (sem marca).

Foram montados em cada catraca:

- um EPOSMote III com uma placa hidrológica para EPOSMote III (Software/Hardware Integration Lab, b), na qual um dos relés foi utilizado para controlar o solenoide (trava) da catraca,
- uma leitora RFID MFRC522,
- uma campainha e
- uma tela LCD I2C para comunicação com o usuário.

4.2.1.1 Controle de Giro

Ambas as catracas possuem dois sensores infravermelhos que podem ser utilizados para detectar o estado de uma placa giratória conforme a Fig. (4), podendo ser utilizados para controlar o estado de giro da catraca, permitindo saber se alguém passou ou está tentando passar pela catraca, assim como a direção do giro ou tentativa.

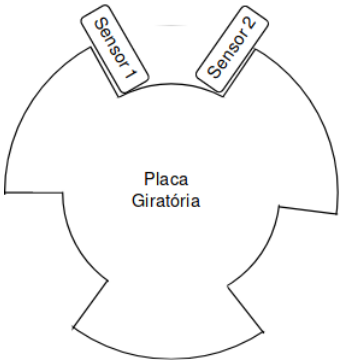

Figura 4 – Diagrama do controlador de giro.

Nestes dois modelos de catraca desenvolvidos com o EPOSMote III, os sensores operam normalmente com 3.3 VCC, que é a mesma voltagem que o EPOSMote III utiliza, fazendo com que a ligação entre eles possa ser feita diretamente.

Para a utilização destes sensores como controlador de giros, foi implementada uma máquina de estados que muda de estado de acordo com a mudança do valor de um dos sensores. Para detectar a mudança dos sensores, foram utilizadas interrupções de GPIO do EPOS que alteram o estado da máquina. A máquina de estados implementada pode ser analisada pelo diagrama da Fig. (5).

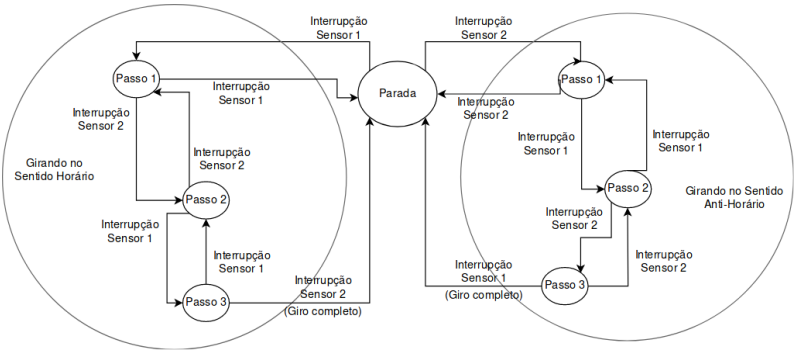

Figura 5 – Diagrama da máquina de estados que controla o giro.

O uso desta máquina de estados permite que seja possível saber quando um giro foi efetuado na catraca, assim como quando uma tentativa de giro está sendo iniciada, para que se possa acionar a trava da catraca quando necessário, bloqueando a passagem de pessoas não autorizadas.

4.2.1.2 Comunicação TSTP

A comunicação entre a catraca e o servidor foi projetada para ocorrer por meio de uma rede TSTP, onde uma catraca envia e recebe dados por meio de uma rede IoT até um nodo gateway, que poderá armazenar em uma cache local com os usuários autorizados, assim como o registro das tentativas de acesso. Este gateway deve possuir uma conexão com um servidor para que se possa modificar autorizações de pessoas e obter-se registros de acessos.

Para o uso do TSTP, foi utilizada a implementação de SmartData já disponível no EPOS, porém foram efetuadas modificações para que ela pudesse ser utilizada na aplicação da catraca. Para isto, foi modificado o transducer do EPOSMote III no EPOS, implementando a classe RFID_Sensor. Esta classe é responsável por se comunicar com a leitora RFID e gerar um novo SmartData quando um cartão é lido, que neste caso contém o UID do cartão lido pela leitora, a autorização de acesso (verdadeiro ou falso) e informações sobre o nodo (catraca) que gerou o dado.

Para testes, foi criada uma aplicação para rodar no gateway da rede TSTP, que recebe um SmartData pela rede TSTP quando um cartão é passado em alguma catraca, imprime o UID na carteirinha, modifica o SmartData para autorizar o acesso e responde pela rede TSTP.

Como o objetivo da aplicação era apenas testar a comunicação TSTP, não foi realizada nenhuma verificação com o UID do cartão, liberando o acesso para qualquer cartão reconhecido pela leitora.

Com três nodos na rede TSTP, sendo dois rodando a aplicação da catraca e um a aplicação do gateway, foi possível testar que a comunicação TSTP funcionou conforme o esperado, transmitindo corretamente o UID do cartão para o gateway, o que permite a identificação do usuário que está realizando a tentativa de acesso, assim como a catraca libera o acesso após o recebimento da resposta.

Ao modificarmos o gateway para não permitir o acesso, a catraca indica na tela que o acesso foi negado e continua travando ao tentar-se girá-la, comprovando que ela interpreta corretamente a resposta do gateway.

4.2.1.3 Testes

Nos testes realizados com TSTP, mesmo com poucos nodos foi identificado um atraso relativamente alto nos tempos de resposta pela rede TSTP, variando entre 1 e 3 segundos do momento em que o cartão é lido até a liberação (ou não) da catraca. Este atraso prejudica muito a experiência do usuário, principalmente em casos onde o cartão não tenha sido lido corretamente e deve ser passado novamente, podendo implicar em uma demora muito maior para que o usuário possa acessar o ambiente, causando-lhe frustrações e aumentando a possibilidade de se criar filas por conta da catraca.

Uma alternativa para se diminuir o atraso seria manter uma cache em cada nodo da rede, porém isso aumentaria a complexidade e quantidade dos dados que devem ser enviados pela rede TSTP, que foi projetada para envios de dados simples, além de elevar bastante a complexidade da implementação.

Uma versão modificada da aplicação da catraca, sem comunicação alguma e liberando para qualquer cartão compatível, foi instalada na catraca Slim da Tecnibra. Para a realização de testes com usuários, esta catraca foi instalada no Restaurante Universitário do Centro de Ciências Agrárias da UFSC.

Os alunos, professores e servidores que fazem uso do restaurante passaram a ter que utilizar a catraca para acessá-lo, necessitando aproximar um cartão para liberar este acesso. Após um curto período de tempo, a catraca caiu e/ou foi arrancada do chão, pois alguns usuários forçaram a catraca ao esquecer de passar o cartão, e o concreto do chão estava se desmanchando. Logo após este ocorrido, a catraca foi reinstalada em um local onde o piso estava mais firme para dar continuidade aos testes.

Durante alguns dias, foi realizado o acompanhamento da utilização da catraca, e pôde-se perceber que alguns usuários acabam sendo violentos, dando tapas ou batendo com o cartão da catraca, o que fez com que a catraca parasse de funcionar algumas vezes por conta de conectores que acabaram se soltando. Após um período maior de tempo, o suporte da tela quebrou e ela afundou para dentro da catraca.

Outro ponto que foi possível observar acompanhando-se a sua utilização, foi que a usabilidade da catraca não é muito boa. O local do leitor não é muito aparente, o que fez com que vários usuários tentassem passar o cartão na tela e não na leitora. A campainha utilizada acabou sendo praticamente inaudível no ambiente barulhento do restaurante, e os usuários acabam não lendo a mensagem da tela, tentando girá-la mesmo quando aconteceu um erro na leitura do cartão.

Uma tentativa de melhoria da usabilidade pode ser realizada destacando-se melhor o local da leitora e utilizando alguma forma de comunicação simples e efetiva, como um LED vermelho e outro verde, para indicar se o acesso está liberado ou não.

4.2.2 Raspberry Pi

A SeTIC desenvolveu um novo sistema de compra de passes e acesso eletrônico aos Restaurantes Universitários da UFSC. Com este sistema, a compra e utilização de passes passa a ocorrer de maneira automatizada (Universidade Federal de Santa Catarina, 2018).

A versão do sistema implantada no acesso de alguns Restaurantes Universitários utiliza uma Raspberry Pi 3 conectada a uma leitora de cartão RFID e a uma tela LCD I2C, que apenas registra o acesso do usuário, descontando o valor do passe do sistema, ou informa o motivo pelo qual o acesso não está autorizado. A operação deste sistema é feita por um segurança ou funcionário do restaurante, ou ainda por um servidor da UFSC, que autoriza ou impede a entrada de pessoas de acordo com o informado pelo sistema.

Para que o sistema possa ser implantando em um ambiente com uma maior quantidade de usuários - como o RU do Campus Trindade da UFSC - uma versão com o controle completamente automatizado do sistema (sem a necessidade de um mediador) foi realizada para que o sistema funcionasse com uma catraca.

A catraca utilizada para este propósito foi o modelo Lumen Balcão da marca Henry. Ela possui um sistema de controle de giro idêntico ao das outras catracas utilizadas, com uma placa giratória e dois sensores infravermelhos.

Para que o sistema realize o controle de giro da catraca, o código da máquina de estados desenvolvida para EPOS (mencionada anteriormente) foi enviado para a equipe da SeTIC responsável pelo sistema, que o adaptou para funcionar na aplicação de controle da Raspberry Pi.

Os sensores do controlador de giro da catraca da Henry são analógicos, sendo necessário desenvolver o circuito adicional da Fig. (6) para que eles possam ser utilizados com a Raspberry Pi.

Para o projeto do circuito da Fig. (6), que recebe o sinal analógico do sensor e retorna um sinal digital binário de acordo com o seu estado, foi utilizada uma calculadora online para se encontrar o valor e definir a resistência necessária a ser utilizada em conjunto com o transistor, de acordo com as voltagens do sinal analógico (VIS, 2018).

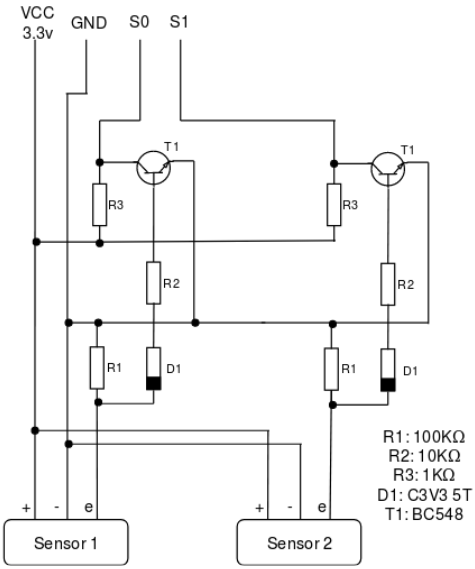

Figura 6 – Esquemático do circuito utilizado nos sensores da catraca.

Para o controle da trava solenoide da catraca, foi utilizado um relé para 12 VCC, que pode ser controlado pela Raspberry Pi, e para a comunicação com o usuário, foram utilizados uma tela LCD I2C e uma campainha que apita quando o acesso é liberado. Todos estes dispositivos periféricos foram conectados ao GPIO da Raspberry Pi 3 de acordo com a pinagem da Tabela (1).

Tabela 1 – Mapa de pinos utilizados na catraca com Raspberry Pi

Pino da Raspberry Pi 3	Tela LCD I2C
6	GND
4	VCC
3	SDA
5	SCL
Pino da Raspberry Pi 3	**Sensores de giro**
29	Sinal Sensor 1
31	Sinal Sensor 2
2	VCC
39	GND
Pino da Raspberry Pi 3	**Relé da trava**
33	Input
2	VCC
39	GND
Pino da Raspberry Pi 3	**Campainha**
12	Input
2	VCC
39	GND

Para a identificação dos cartões RFID, foi utilizada uma leitora Gemalto Prox-SU, conectada à interface USB da Raspberry Pi 3.

A Raspberry Pi utilizada nesta catraca é alimentada por uma bateria RPI SKU:435230 conectada a uma fonte de 5 VCC 3 amperes.

Durante os testes, foi verificado que podem ocorrer problemas se for utilizada uma fonte de 2 amperes com a bateria, pois a fonte é utilizada para alimentar o sistema e ao mesmo tempo recarregar a bateria. Também foi percebido que ao se desligar a fonte, a Raspberry Pi reiniciava devido a uma rápida queda de tensão.

Este último problema foi resolvido adicionando-se um capacitor de 1.5 milifarads entre a bateria e a Raspberry Pi, para manter a tensão durante a troca para a bateria.

Alguns outros problemas da aplicação de controle foram percebidos, reportados à SeTIC e corrigidos, e a catraca está funcionando corretamente sem nenhum problema conhecido. Esta catraca ainda não foi testada em nenhum ambiente com uma grande quantidade de usuários.

4.3 PORTA EPOS

Durante a execução do trabalho, foram montados dois protótipos de portas controladas pelo EPOSMote III. As portas estão situadas no Laboratório de Segurança em Computação (LabSEC) no INE - UFSC. Uma das portas é a de acesso ao laboratório e a outra é a de acesso à sala de reuniões do laboratório. Ambas as portas possuem uma trava magnética de 12 VCC e mola para fechamento automático.

Em ambas as portas foram instalados um EPOSMote III com uma placa hidrológica para EPOSMote III (Software/Hardware Integration Lab, b), na qual um dos relés foi utilizado para controlar a trava magnética da porta. Quando o acesso é liberado, a trava magnética é desligada por 5 segundos, permitindo a abertura da porta.

Em uma das portas foi utilizada uma leitora de RFID MFRC522 pelo lado de fora, com um botão pelo lado interno. Ao aproximar um cartão cadastrado ou apertar o botão, a porta é destravada.

Na outra porta, foram utilizados um sinal de botão de interfone e duas leitoras RFID Genéricas, que utilizam a interface Wiegand. O suporte do EPOS à esta leitora RFID foi implementado pelo Laboratório de Integração Software/Hardware (LISHA) antes do inicio da implementação da aplicação da porta. Como a implementação da leitora utiliza o mesmo módulo (RFID), uma mesma aplicação pode ser utilizada para os dois tipos de leitoras, sendo necessário apenas alterar a configuração da aplicação (traits) no EPOS.

Em ambas as portas, para o acionamento do botão foi utilizada uma interrupção por GPIO com o pino conectado ao botão, e o módulo RFID do EPOS é utilizado para obter o UID do cartão aproximado.

Para o cadastramento de cartões na aplicação de controle da porta, foi utilizado o sistema de arquivos do EPOS, que ainda se encontrava em fase de testes. Em alguns testes realizados durante o desenvolvimento da aplicação, foi verificado que o sistema de arquivos é corrompido ao se adicionar mais do que 16 arquivos ou quando um arquivo ultrapassa um tamanho de cerca de 768 bytes. Os problemas foram reportados ao desenvolvedor do sistema de arquivos do EPOS, porém eles não foram corrigidos, o que limitou bastante o projeto do armazenamento de dados.

Havia sido planejado o desenvolvimento de um sistema que utilizaria um arquivo para cada cartão, que poderia conter políticas de acesso, tais como horário e quantidade máxima de acessos, assim como um arquivo para armazenar o registro dos acessos e tentativas de acesso, porem as limitações causadas pelos problemas no sistema de arquivos

impediram a implementação do mesmo.

Como alternativa, foi utilizado um único arquivo contendo somente os UIDs de cartões autorizados de maneira sequencial, permitindo o cadastramento de uma quantidade limitada de cartões (aproximadamente 96 cartões) sem que ocorram problemas. Quando o EPOS possuir um sistema de arquivos estável, o método de armazenamento descrito anteriormente poderá ser adequadamente implementado.

Conforme observou-se após a implementação da comunicação TSTP nas catracas, ficou constatado que o TSTP não seria uma boa alternativa para as portas, em função dos atrasos e da complexidade dos dados que serão sincronizados pela porta ao utilizar-se políticas de acesso avançadas.

Além disto, normalmente será instalada apenas uma única porta em cada ambiente, fazendo com que seja necessário instalar um gateway para cada porta, por conta do limite de distância da comunicação TSTP. Por conta destes motivos, foi planejado que cada controlador de porta deve se comunicar diretamente com o servidor de gerenciamento via Wi-Fi, utilizando o módulo ESP8266.

Foi realizada uma tentativa de se implementar a comunicação do EPOS via Wi-Fi com o módulo ESP8266, detalhada na próxima seção, porém ela não foi finalizada. Como não é possível comunicar-se com a aplicação, foi definido no seu código um cartão-mestre. Este cartão-mestre é capaz de cadastrar um novo cartão na porta, fazendo com que o novo cartão passe a liberar o acesso na porta.

Com isto, foi possível testar que a aplicação é capaz de fazer o controle de acesso, permitindo apenas cartões autorizados. Porém, para realizarem-se auditorias, é necessário concluir a comunicação para que o EPOS possa atualizar o seu relógio, registrar as tentativas de acesso em um arquivo e enviá-las para o servidor de gerenciamento.

Para armazenar os registros de tentativas de acesso em um arquivo, é importante que o sistema de arquivos do EPOS seja corrigido, uma vez que este arquivo com os registros pode crescer rapidamente, fazendo com que possa acontecer o problema de corromper o sistema de arquivos.

4.4 COMUNICAÇÃO VIA ESP8266

O EPOSMote III não possui uma maneira de se comunicar diretamente com a internet em seu hardware.

Uma opção bastante utilizada para a comunicação do EPOSMote com a internet é conectá-lo via USB ou UART à um computador com acesso à internet. Esta opção acaba possuindo um custo elevado, principalmente se forem instalados vários EPOSMotes distantes sem a utilização de TSTP, fazendo-se necessária a utilização de vários computadores somente para este fim.

Outra maneira utilizada para conexão de um EPOSMote com a internet é a utilização de um módulo Wi-Fi, como o ESP8266, conectado ao EPOSMote via UART para se enviar dados por uma rede Wi-Fi.

Este módulo já foi anteriormente utilizado para o envio de dados inteligentes (SmartData) do TSTP para um servidor na internet, utilizando um firmware para ESP8266 desenvolvido pelo LISHA. Este firmware permite apenas o envio de dados de tamanho fixo para um mesmo servidor configurado, sem a possibilidade de se alterar parâmetros da requisição.

Foram desenvolvidas modificações no firmware para permitir a conexão com dois servidores simultaneamente, um utilizando a implementação já existente para o envio de dados inteligentes, e outro servidor utilizando uma nova implementação, para conectar-se com um WebService comum.

O firmware original recebe comandos, que podem possuir parâmetros ou não, pela serial UART, e os responde pela mesma interface. A versão modificada utiliza este mesmo padrão, porém os comandos foram modificados para identificar o servidor que deve ser utilizado, se é o de dados inteligentes (IoT) ou o WebService comum.

A nova implementação permite a utilização de autenticação Basic do protocolo HTTP, e retorna a resposta da requisição pela serial UART. Isto permite que possam ser recebidas informações do servidor pelo dispositivo conectado ao ESP8266.

Para o envio e recebimento de dados complexos, foi planejada a utilização de dados na notação de objetos de JavaScript (JSON), notação bastante utilizada por vários serviços da Web, inclusive serviços da própria SeTIC.

Durante os testes realizados com ESP8266 conectados diretamente à um terminal UART, sem utilizarmos o EPOSMote III, foi verificado que o módulo apresenta problemas de estabilidade.

Foram testados tanto o novo firmware quanto o firmware original desenvolvido pelo LISHA, e em ambos os casos pode acontecer de o módulo reiniciar sem um motivo aparente, ou até mesmo parar de responder à comandos até que ele seja reiniciado. Foram testados com outros módulos ESP8266 ESP-01, juntamente com diferentes fontes de alimentação, e o problema continuou acontecendo.

Em testes onde não ocorreram problemas com o módulo, foi verificado que o novo firmware pode realizar conexões com um servidor via HTTPS com autenticação Basic, enviar dados recebidos por parâmetro via POST e retornar a resposta da requisição.

A implementação de uma interface entre o EPOS e o ESP8266 foi muito prejudicada por conta destes problemas, além do fato de o tratamento de strings de dados ser muito complicado no EPOS, e de não existir um analisador JSON implementado para o sistema. Por isto, não foi possível implementar uma interface de comunicação durante o prazo de tempo disponível para a execução deste trabalho.

4.5 INTERFACE DE CONTROLE

Para a realização do gerenciamento e auditoria por parte dos gerenciadores do controle de acesso, foi iniciado o desenvolvimento de uma aplicação Web para o controle do sistema.

A aplicação é responsável por fornecer uma interface para controle do sistema de acesso por parte dos gerentes, assim como armazenar e exibir dados para a auditoria de tentativas de acesso. Tanto a interface como o armazenamento de dados necessários foram implementados.

Esta aplicação foi planejada também para ser responsável por informar atualizações de políticas de acesso, cadastramento e descadastramento de usuários aos EPOSMotes que controlam as portas, assim como receber e armazenar os registros para auditoria das tentativas de acesso.

Estas atualizações seriam obtidas como respostas de requisições pelo EPOSMote feitas ao servidor da aplicação por meio de uma interface REST, porém como como a comunicação do EPOSMote não foi finalizada, a interface REST não foi implementada.

A implementação desta aplicação de controle foi feita em JavaScript utilizando o Node.js, em conjunto com o framework Express, que permite o tratamento de requisições HTTP (Node.js Foundation, b). Para a implementação da interface, foram utilizados os frameworks Pug e Bootstrap, que permitem a criação de uma interface gráfica moderna

de maneira ágil (Pugjs,) (Bootstrap,). Os dados do sistemas são armazenados no banco de dados PostgreSQL (The PostgreSQL Global Development Group,).

Foi implementado um sistema de login, com autenticação por usuário e senha. O usuário é armazenado em texto plano, enquanto que para a verificação de senha são armazenados um sal em conjunto com um resumo criptográfico da senha em conjunto com o sal.

Os usuários são divididos em três grupos:

- Gerentes: usuários que irão gerenciar o controle de acesso de uma ou mais portas, podendo adicionar ou remover pessoas e acompanhar os registros de tentativas de acesso;
- Administradores: usuários que podem cadastrar novos usuários, gerenciar as portas às quais os gerentes possuem autorização, alterar o grupo, nome ou senha de um outro usuário, e gerenciar todas as portas;
- Nenhum: usuários desativados que não podem fazer nada.

As portas são identificadas pelo sistema com base no seu endereço físico de rede (MAC). Os administradores podem configurar um nome para facilitar a identificação das portas.

Como a interface REST não foi implementada, foram inseridos manualmente dados falsos no banco de dados para a realização de testes de interface, simulando a existência de portas e tentativas de acesso.

Para o administrador do sistema e gerente de uma porta, foi implementado o cadastramento de cartões no sistema, que pode ser feito de duas maneiras. Uma delas é a inserção manual do nome do usuário e do UID do cartão. A segunda maneira é buscando pelo CPF ou usuário e número de matrícula da UFSC.

As alterações de usuários, como cadastramento e descadastramento, são salvas no banco de dados com um identificador serial, permitindo que as alterações de cadastro possam ser informadas às portas na ordem em que ocorreram, possibilitando a implementação de uma sincronização com a porta, onde apenas as mudanças são enviadas e os usuários cadastrados sejam os mesmos.

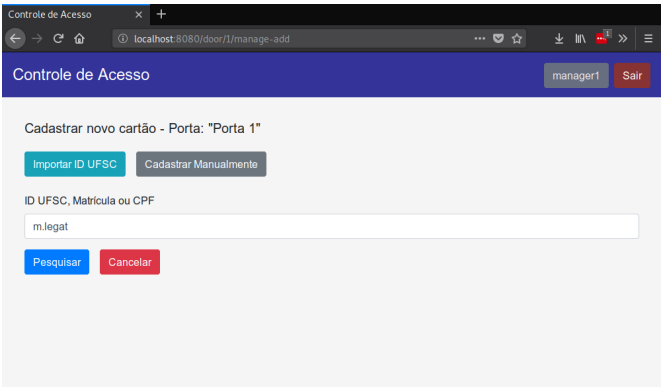

Figura 7 – Exemplo de busca pelo ID UFSC.

A importação de dados da UFSC é feita por meio de um Web-Service da SeTIC para a obtenção de dados de cartões emitidos. Por meio deste serviço, são obtidos e cadastrados o nome completo, UID do cartão e o ID Pessoa UFSC.

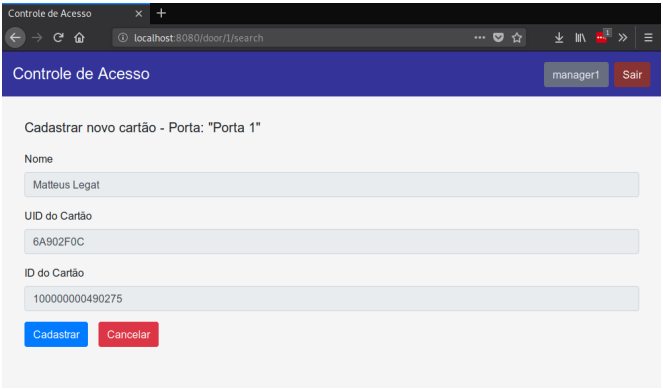

Figura 8 – Resultado da busca pelo ID UFSC.

Ao gerenciar uma porta, é possível também realizar a auditoria, verificando as tentativas de acesso ordenadas a partir da mais recente, com informações de data e hora, UID do cartão, nome do usuário, caso cadastrado, e se o acesso foi autorizado ou negado.

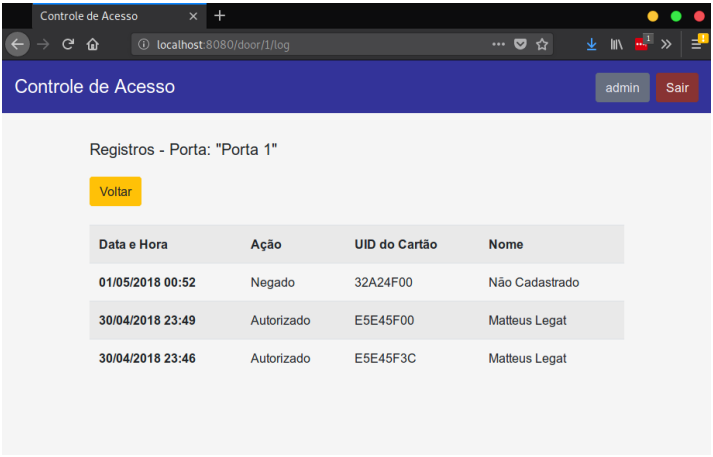

Figura 9 – Tela de auditoria dos registros.

Todas as telas desenvolvidas tiveram a interface testada manualmente, e nenhum erro foi encontrado. Todas as ações executadas obtiveram os resultados esperados, incluindo as atualizações dos bancos de dados.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foram desenvolvidas soluções para controle de acesso e construídos protótipos funcionais que mostram a viabilidade de soluções de baixo custo para a implantação de sistemas de controle de acesso, com a utilização de tecnologias construídas na UFSC e outros equipamentos comerciais de baixo custo.

Foi realizado o desenvolvimento da integração de leitoras MFRC-522 com o EPOSMote III, permitindo a construção de equipamentos capazes de realizar a identificação por RFID com um hardware de custo bastante reduzido.

Foram desenvolvidas também aplicações de controle para portas e catracas, permitindo a utilização destes hardwares de baixo custo para realizar o controle de acesso propriamente dito.

Os sistemas construídos com EPOS são capazes de permitir o acesso somente à um grupo seleto de pessoas, porém, devido à falta de comunicação do EPOS com a rede, não permitem a realização de auditorias e de um gerenciamento mais avançado, onde se poderia aplicar políticas ou adicionar e remover autorizações de acesso mais facilmente.

Um sistema para se realizar o gerenciamento do controle de acesso e auditoria foi preparado e teve sua implementação iniciada, porém ainda é necessário desenvolver uma interface REST para se realizar a sincronização de dados com o sistema de controle no EPOS, assim como a implementação da comunicação no EPOS.

Foi realizada também a adaptação do sistema de acesso ao RU UFSC, desenvolvido pela SeTIC, para funcionamento com catracas, permitindo a automatização do acesso ao RU com a redução da necessidade de intervenção de um segurança ou funcionário para a realização do controle de acesso.

5.1 TRABALHOS FUTUROS

Como trabalho futuro, a principal sugestão seria a da conclusão do desenvolvimento da comunicação entre a aplicação de controle e o EPOSMote. Para isto, são necessários a implementação de uma interface REST na aplicação de controle, assim como o desenvolvimento de uma comunicação estável do EPOSMote com o servidor.

Para isto, sugere-se o estudo da utilização de diferentes firmwares para o módulo ESP8266 ou até mesmo a utilização de outros hardwares para comunicação, assim como a implementaçao de um interpretador JSON no EPOS e mudanças na aplicacão de controle da porta. Para o armazenamento dos registros e autorizações de acesso no EPOS para a sincronização com o servidor, é importante o desenvolvimento de correções para o sistema de arquivos do EPOS.

Outra sugestão importante de trabalho futuro é a utilização de estruturas para o armazenamento de políticas de acesso avançadas, tais como restrições de horário ou a possibilidade de se limitar a quantidade de vezes que um indivíduo pode acessar o local. Para isto, também seria importante a correção do sistema de arquivos do EPOS.

Outro trabalho futuro considerado como interessante seria o desenvolvimento de aplicações controladoras em EPOS para diferentes mecanismos de controle de acesso, tais como portas automáticas, cancelas ou até mesmo catracas para cadeirantes.

Uma sugestão interessante de trabalho futuro seria o estudo de possíveis métodos de fraude, avaliando e até mesmo implementando alguns métodos para a prevenção destas fraudes.

REFERÊNCIAS

Bootstrap. *Bootstrap*. <https://getbootstrap.com/>. Acessado em 25/05/2018.

BOSWORTH, S.; KABAY, M. E. *Computer security handbook*. [S.l.]: John Wiley & Sons, 2002.

BUYYA, R.; DASTJERDI, A. V. *Internet of Things: Principles and Paradigms*. [S.l.]: Elsevier, 2016. 378 p.

CERT.br. *Cartilha de Segurança para Internet*. 2017. <https://cartilha.cert.br/criptografia/>. Acessado em 23/05/2018.

DEUTSCH, W. *Introduction to Electronic Access Control*. 2018. <https://www.thebalancesmb.com/introduction-to-electronic-access-control-394578>. Acessado em 18/05/2018.

EVANS, D. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, v. 1, n. 2011, p. 1–11, 2011.

FIELDING, R. T. Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000.

FRÖHLICH, A. A.; TIENTCHEU, G. P.; SCHRÖDER-PREIKSCHAT, W. Epos and myrinet: Effective communication support for parallel applications running on clusters of commodity workstation. In: SPRINGER. *International Conference on High-Performance Computing and Networking*. [S.l.], 2000. p. 417–426.

HOUSLEY, R.; POLK, T. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. [S.l.]: John Wiley Sons, Inc., 2001. 327 p.

ISO/IEC. *ISO 14443-3. Identification cards – Contactless integrated circuit cards – Proximity cards – Part 3: Initialization and anticollision*. [S.l.], 2001.

MOTOROLA, INC. *SPI Block Guide V03.06*. [S.l.], 2003.

Node.js Foundation. *About | Node.js*. <https://nodejs.org/en/about/>. Acessado em 20/05/2018.

Node.js Foundation. *Express - Node.js web application framework.* <https://expressjs.com/>. Acessado em 25/05/2018.

NXP SEMICONDUCTORS. *MFRC522 - Standard performance MIFARE and NTAG frontend - Product data sheet.* [S.l.], 2016.

Pugjs. *Getting Started - Pug.* <https://pugjs.org/api/getting-started.html>. Acessado em 25/05/2018.

RESNER, D.; FRöHLICH, A. A. Tstp mac: A foundation for the trustful space-time protocol. *Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, 2016.

RICHARDSON, M.; WALLACE, S. *Getting started with raspberry PI.* [S.l.]: "O'Reilly Media, Inc.", 2012.

SCHWARTZ, M. *Internet of Things with ESP8266.* [S.l.]: Packt Publishing Ltd, 2016.

Software/Hardware Integration Lab. *EPOSMote III.* <http://epos.lisha.ufsc.br/HomePage>. Acessado em 12/09/2016.

Software/Hardware Integration Lab. *Hydrologic Station board.* <http://epos.lisha.ufsc.br/Hydrologic+Station+board>. Acessado em 22/05/2018.

The PostgreSQL Global Development Group. *PostgreSQL: The world's most advanced open source database.* <https://www.postgresql.org/>. Acessado em 25/05/2018.

TRIPATHI, R.; AGRAWAL, S. Comparative study of symmetric and asymmetric cryptography techniques. *International Journal of Advance Foundation and Research in Computer (IJAFRC)*, v. 1, n. 6, p. 68–76, 2014.

Universidade Federal de Santa Catarina. *Restaurantes Universitários dos campi iniciam modernização de acesso.* 2018. <http://noticias.ufsc.br/2018/02/restaurantes-universitarios-dos-campi-iniciam-modernizacao-de-acesso/>. Acessado em 22/05/2018.

UPTON, E.; HALFACREE, G. *Raspberry Pi user guide.* [S.l.]: John Wiley & Sons, 2014.

VERMESAN, O.; FRIESS, P. *Internet of Things – From Research and Innovation to Market Deployment*. Aalborg, Denmark: River Publishers, 2014. 355 p.

VIS, P. *Transistor Base Resistor Calculator*. 2018. <https://www.petervis.com/GCSE_Design_and_Technology_Electronic_Products/transistor_base_resistor_calculator/transistor_base_resistor_calculator.html>. Acessado em 17/03/2018.

WANT, R. An introduction to rfid technology. *IEEE Pervasive Computing*, v. 5, n. 1, p. 25–33, 2006.

WOSCHEK, M. *OWASP Cheat Sheets*. [S.l.]: Citeseer, 2015.

ZHENG, J. et al. The internet of things [guest editorial]. *IEEE Communications Magazine*, v. 49, n. 11, p. 30–31, 2011.

APÊNDICE A – Artigo

Sistema de Controle de Acesso em IoT

Matteus Legat[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

matteuslegat@gmail.com

Abstract. *Access control is essential in environments where it is necessary to restrict access to only a certain group of people. In order to make a scientific contribution to society seeking an access control solution for UFSC by seeking the reduction of costs for the implementation of restricted areas in its campus, this Course Conclusion Work proposes the development of an Internet of Things system for the accomplishment of the access control through equipment with RFID sensors present in doors and turnstyles, capable of identifying and controling the access of individuals through compatible identification cards, using technologies of low cost and developed in the UFSC in an efficient way.*

Resumo. *O controle de acesso é essencial em ambientes onde se faz necessário restringir o acesso a apenas um certo grupo de pessoas. Para realizar uma contribuição científica para a sociedade buscando uma solução de controle de acesso para a UFSC, buscando a redução de custos para a implementação de áreas restritas em seu campus, este Trabalho de Conclusão de Curso propõe o desenvolvimento de um sistema de Internet das Coisas para a realização do controle de acesso através de equipamentos com sensores RFID presentes em portas e catracas, capazes de identificar e controlar o acesso de indivíduos através de cartões de identificação compatíveis, utilizando tecnologias de baixo custo e desenvolvidas na UFSC de maneira eficiente.*

1. Introdução

O controle de acesso de alunos, professores e servidores é uma tarefa essencial para que se possa garantir a segurança em áreas específicas da Universidade Federal de Santa Catarina. Até o presente momento, não existe um padrão definido sobre como este controle deve ser realizado, e a solução vigente consiste em um sistema de controle de acesso terceirizado, que possui um custo significativo para cada ponto de acesso.

Os estudantes, professores e servidores da Universidade Federal de Santa Catarina possuem um cartão de identificação que contém a tecnologia radio-frequency identification (RFID), que podem ser utilizados para identificação através de um sistema que seja compatível com a tecnologia. Utilizando esta tecnologia, é possível cadastrar os indivíduos em uma base de dados e identificá-los posteriormente.

Dispositivos de Internet das Coisas (IoT) estão sendo cada vez mais utilizados para se executar diferentes tipos de tarefas, principalmente quando se tratam de tarefas que não exigem muito poder computacional. Um sistema de IoT possui a vantagem de não ficar concentrado em um único ponto, facilitando sua utilização em uma área de grandes dimensões, podendo ser acessado fisicamente em vários pontos, evitando-se assim o deslocamento até um ponto específico.

Considerando esse contexto, é interessante o desenvolvimento de um sistema de IoT automatizado que controle a entrada e saída de alunos, professores e servidores à áreas restritas, utilizando-se de dispositivos presentes nas portas das salas capazes de identificar os indivíduos através da tecnologia RFID, controlando e registrando o acesso.

2. Proposta

A proposta consiste no desenvolvimento de um sistema que auxiliará no controle de acesso à áreas restritas na UFSC. A proposta pode ser subdividida em duas etapas, sendo elas o desenvolvimento de um sistema de IoT para auxiliar no controle de acesso, e o desenvolvimento de uma aplicação para que o administrador do sistema possa acompanhar e realizar este controle.

2.1. Sistema de IoT

Considerando que os estudantes, professores e servidores da UFSC já utilizam carteiras de identificação com suporte a RFID, serão utilizadas as carteiras já existentes para a identificação de um aluno ou servidor para liberação do acesso.

Será implementado um sistema com nodos sensores distribuídos nas áreas de acesso restrito da universidade. A proposta é que cada porta de acesso à área restrita tenha um sensor de RFID capaz de ler os cartões na porta, de modo que eles sejam capazes de identificar um indivíduo e liberar ou não o seu acesso, e registrando esta informação para consulta pelo administrador.

Cada nodo sensor fará parte de uma rede de IoT que servirá como controle de acesso eletrônico aos indivíduos credenciados. Para a construção desta rede, serão utilizados dispositivos EPOSMote III, produzidos pelo LISHA, que rodarão o sistema operacional EPOS 2 utilizando o protocolo TSTP. Alguns dos dispositivos na rede atuarão como gateway, estando conectados com a internet e transmitindo as informações de tentativa de acesso para o servidor da aplicação de controle. Alguns nodos da rede deverão realizar o controle de portas e catracas, permitindo ou não o acesso.

Serão utilizados sensores com o chip MFRC522, cujo componente controlador para o sistema operacional EPOS será desenvolvido neste trabalho, bem como as aplicações que rodarão nos nodos da rede IoT. Será desenvolvida a aplicação dos nodos sensores, responsável por permitir ou não o acesso, informando para a rede sobre a entrada e saída de indivíduos, assim como a aplicação de gateway para permitir a interação entre a rede e a aplicação de controle.

2.2. Aplicação para Controle e Auditoria

Para que o controle de acesso seja propriamente realizado, será desenvolvida uma aplicação web para o controle e acompanhamento do acesso de alunos, professores e servidores. Esta aplicação receberá os dados do sistema de IoT sobre o acesso à áreas restritas.

A aplicação possuirá uma interface para controle por parte do administrador, que após autenticado no sistema, poderá verificar os indivíduos que obtiveram acesso à área restrita. O sistema utilizará o horário e a identificação da área com acesso controlado.

3. Implementação

3.1. MFRC522 no EPOS

Para construir uma aplicação que utilize o módulo RFID no EPOSMote III é necessário implementar uma interface de comunicação entre os dois componentes. O meio de comunicação adotado para ser utilizado entre esses dois componentes foi o SPI.

O sistema operacional EPOS já possui um componente pronto para utilizar a comunicação SPI no EPOSMote III, que utiliza o módulo SSI de seu SoC. Nele contém todos os métodos necessários para fazer as transferências de dados, sendo possível também inicializá-lo com diferentes parâmetros de acordo com as funcionalidades desejadas. A implementação do MFRC522 para o EPOSMote III foi feita utilizando esse meio de comunicação para escrever e ler em seus registradores.

Para que um leitor de cartão (PCD) inicie uma comunicação com um cartão sem contato com circuito integrado (PICC), é necessário seguir uma sequência de passos definida pelo padrão ISO/IEC 14443-3.

Antes de se iniciar uma comunicação, caso seja necessário aguardar pela presença de um cartão (PICC) próximo ao leitor (PCD), deve-se realizar um polling como o definido pela ISO/IEC, solicitando ao PCD o envio dos comandos REQA (Request Type A) ou REQB (Request Type B) periodicamente, onde o tipo da requisição depende do tipo do cartão. O PCD pode ser consultado para saber se foi recebida alguma resposta ao REQA ou REQB, ou seja, se existe algum PICC presente.

Com o cartão presente próximo ao leitor, deve-se iniciar o processo de seleção e anticolisão de um cartão. Uma colisão pode ocorrer quando existem dois ou mais PICCs que enviam valores diferentes e pode ser detectada pelo PCD. O mecanismo de anticolisão permite que um único PICC seja selecionado mesmo quando existirem mais de um PICC.

No processo de anticolisão, o PCD deve enviar o comando de anticolisão para o(s) PICC(s) e o nível de cascata, e então receber o código de identificação único (UID) para o nível solicitado do cartão. Caso tenha sido detectada alguma colisão, o comando de anticolisão é reenviado com o ID recebido até a posição em que ocorreu a colisão, até o momento em que não ocorram mais colisões e o UID esteja completo.

No momento em que não houverem mais colisões, deve ser calculado um valor para a verificação de redundância cíclica (CRC), para ser enviado no comando de seleção, juntamente com o UID obtido no processo de anticolisão. Após o comando de seleção, se não ocorreu nenhum erro no procedimento, o PICC deve receber uma confirmação de seleção (SAK) e então o cartão estará pronto para a comunicação.

A implementação da interface de comunicação com o MFRC522 foi realizada seguindo as especificações em sua ficha de dados. Ela é dividida em dois módulos, sendo eles o PICC, que é independente de arquitetura, e o módulo MFRC522 que representa um leitor e possui uma classe genérica, independente de arquitetura, e outra classe específica para a maquina Cortex-M.

3.2. Catraca

Durante a execução do trabalho, foram desenvolvidos diferentes protótipos de catraca com três diferentes modelos de catraca, sendo elas a Catraca Lumen Balcão da Henry, a Catraca Slim da Tecnibra e uma catraca antiga do RU da UFSC, sem identificação de marca ou modelo.

Todos os três modelos de catraca possuem um funcionamento interno bastante semelhante, com a trava sendo um solenoide de 12 VCC, e um mesmo sistema mecânico com dois sensores infravermelhos para que se possa controlar o giro.

Foram desenvolvidos protótipos de catracas controladas pelo dispositivo EPOSMote III com o modelo de caraca Slim da Tecnibra e com uma antiga catraca do RU da UFSC (sem marca).

Ambas as catracas possuem dois sensores infravermelhos que podem ser utilizados para detectar o estado de uma placa giratória conforme a Fig. (1), podendo ser utilizados para controlar o estado de giro da catraca, permitindo saber se alguém passou ou está tentando passar pela catraca, assim como a direção do giro ou tentativa.

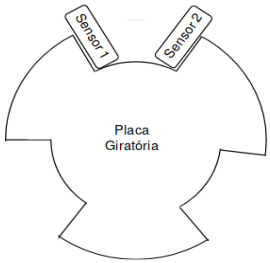

Figura 1. Diagrama do sensor de giro

Nestes dois modelos de catraca desenvolvidos com o EPOSMote III, os sensores operam normalmente com 3.3 VCC, que é a mesma voltagem que o EPOSMote III utiliza, fazendo com que a ligação entre eles possa ser feita diretamente.

Para a utilização destes sensores como controlador de giros, foi implementada uma máquina de estados que muda de estado de acordo com a mudança do valor de um dos sensores. Para detectar a mudança dos sensores, foram utilizadas interrupções de GPIO do EPOS que alteram o estado da máquina. A máquina de estados implementada pode ser analisada pelo diagrama da Fig. (2).

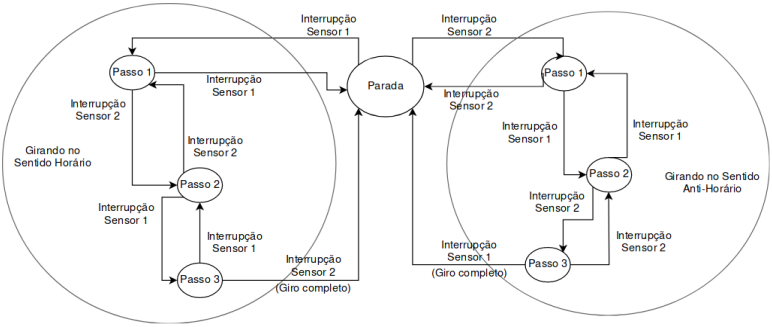

Figura 2. Diagrama da máquina de estados que controla o giro

O uso desta máquina de estados permite que seja possível saber quando um giro foi efetuado na catraca, assim como quando uma tentativa de giro está sendo iniciada, para que se possa acionar a trava da catraca quando necessário, bloqueando a passagem de pessoas não autorizadas.

A SeTIC desenvolveu um novo sistema de compra de passes e acesso eletrônico aos Restaurantes Universitários da UFSC. Com este sistema, a compra e utilização de passes passa a ocorrer de maneira automatizada.

A versão do sistema implantada no acesso de alguns Restaurantes Universitários utiliza uma Raspberry Pi 3 conectada a uma leitora de cartão RFID e a uma tela LCD I2C, que apenas registra o acesso do usuário, descontando o valor do passe do sistema, ou informa o motivo pelo qual o acesso não está autorizado. A operação deste sistema é feita por um segurança ou funcionário do restaurante, ou ainda por um servidor da UFSC, que autoriza ou impede a entrada de pessoas de acordo com o informado pelo sistema.

Para que o sistema possa ser implantando em um ambiente com uma maior quantidade de usuários - como o RU do Campus Trindade da UFSC - uma versão com o controle completamente automatizado do sistema (sem a necessidade de um mediador) foi realizada para que o sistema funcionasse com uma catraca.

A catraca utilizada para este propósito foi o modelo Lumen Balcão da marca Henry. Ela possui um sistema de controle de giro idêntico ao das outras catracas utilizadas, com uma placa giratória e dois sensores infravermelhos.

Para que o sistema realize o controle de giro da catraca, o código da máquina de estados desenvolvida para EPOS (mencionada anteriormente) foi enviado para a equipe da SeTIC responsável pelo sistema, que o adaptou para funcionar na aplicação de controle da Raspberry Pi.

Os sensores do controlador de giro da catraca da Henry são analógicos, sendo necessário desenvolver o circuito adicional da Fig. (3) para que eles possam ser utilizados com a Raspberry Pi.

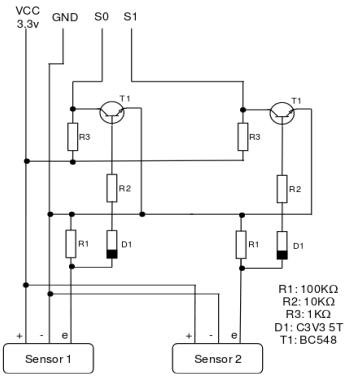

Figura 3. Esquemático do circuito utilizado nos sensores da catraca

Para o controle da trava solenoide da catraca, foi utilizado um relé para 12 VCC, que pode ser controlado pela Raspberry Pi, e para a comunicação com o usuário, foram utilizados uma tela LCD I2C e uma campainha que apita quando o acesso é liberado. Para a identificação dos cartões RFID, foi utilizada uma leitora Gemalto Prox-SU, conectada à interface USB da Raspberry Pi 3.

3.3. Porta

Durante a execução do trabalho, foram montados dois protótipos de portas controladas pelo EPOSMote III. As portas estão situadas no Laboratório de Segurança em Computação (LabSEC) no INE - UFSC. Uma das portas é a de acesso ao laboratório e a outra é a de acesso à sala de reuniões do laboratório. Ambas as portas possuem uma trava magnética de 12 VCC e mola para fechamento automático.

Em ambas as portas foram instalados um EPOSMote III com uma placa hidrológica para EPOSMote III, na qual um dos relés foi utilizado para controlar a trava magnética da porta. Quando o acesso é liberado, a trava magnética é desligada por 5 segundos, permitindo a abertura da porta.

Em uma das portas foi utilizada uma leitora de RFID MFRC522 pelo lado de fora, com um botão pelo lado interno. Ao aproximar um cartão cadastrado ou apertar o botão, a porta é destravada.

Na outra porta, foram utilizados um sinal de botão de interfone e duas leitoras RFID Genéricas, que utilizam a interface Wiegand. O suporte do EPOS à esta leitora RFID foi implementado pelo Laboratório de Integração Software/Hardware (LISHA) antes do inicio da implementação da aplicação da porta. Como a implementação da leitora utiliza o mesmo módulo (RFID), uma mesma aplicação pode ser utilizada para os dois tipos de leitoras, sendo necessário apenas alterar a configuração da aplicação (traits) no EPOS.

Em ambas as portas, para o acionamento do botão foi utilizada uma interrupção por GPIO com o pino conectado ao botão, e o módulo RFID do EPOS é utilizado para obter o UID do cartão aproximado.

Para o cadastramento de cartões na aplicação de controle da porta, foi utilizado o sistema de arquivos do EPOS, que ainda se encontrava em fase de testes. Em alguns testes realizados durante o desenvolvimento da aplicação, foi verificado que o sistema de arquivos é corrompido ao se adicionar mais do que 16 arquivos ou quando um arquivo ultrapassa um tamanho de cerca de 768 bytes. Os problemas foram reportados ao desenvolvedor do sistema de arquivos do EPOS, porém eles não foram corrigidos, o que limitou bastante o projeto do armazenamento de dados.

Havia sido planejado o desenvolvimento de um sistema que utilizaria um arquivo para cada cartão, que poderia conter políticas de acesso, tais como horário e quantidade

máxima de acessos, assim como um arquivo para armazenar o registro dos acessos e tentativas de acesso, porem as limitações causadas pelos problemas no sistema de arquivos impediram a implementação do mesmo.

Como alternativa, foi utilizado um único arquivo contendo somente os UIDs de cartões autorizados de maneira sequencial, permitindo o cadastramento de uma quantidade limitada de cartões (aproximadamente 96 cartões) sem que ocorram problemas. Quando o EPOS possuir um sistema de arquivos estável, o método de armazenamento descrito anteriormente poderá ser adequadamente implementado.

Como normalmente será instalada apenas uma única porta em cada ambiente, fazendo com que seja necessário instalar um gateway para cada porta para se utilizar o TSTP, por conta do limite de distância da comunicação. Por conta disto, foi planejado que cada controlador de porta deve se comunicar diretamente com o servidor de gerenciamento via Wi-Fi, utilizando o módulo ESP8266.

Foi realizada uma tentativa de se implementar a comunicação do EPOS via Wi-Fi com o módulo ESP8266, detalhada na próxima seção, porém ela não foi finalizada. Como não é possível comunicar-se com a aplicação, foi definido no seu código um cartão-mestre. Este cartão-mestre é capaz de cadastrar um novo cartão na porta, fazendo com que o novo cartão passe a liberar o acesso na porta.

Com isto, foi possível testar que a aplicação é capaz de fazer o controle de acesso, permitindo apenas cartões autorizados. Porém, para realizarem-se auditorias, é necessário concluir a comunicação para que o EPOS possa atualizar o seu relógio, registrar as tentativas de acesso em um arquivo e enviá-las para o servidor de gerenciamento.

Para armazenar os registros de tentativas de acesso em um arquivo, é importante que o sistema de arquivos do EPOS seja corrigido, uma vez que este arquivo com os registros pode crescer rapidamente, fazendo com que possa acontecer o problema de corromper o sistema de arquivos.

3.4. Comunicação via ESP8266

O EPOSMote III não possui uma maneira de se comunicar diretamente com a internet em seu hardware.

Uma opção bastante utilizada para a comunicação do EPOSMote com a internet é conectá-lo via USB ou UART à um computador com acesso à internet. Esta opção acaba possuindo um custo elevado, principalmente se forem instalados vários EPOSMotes distantes sem a utilização de TSTP, fazendo-se necessária a utilização de vários computadores somente para este fim.

Outra maneira utilizada para conexão de um EPOSMote com a internet é a utilização de um módulo Wi-Fi, como o ESP8266, conectado ao EPOSMote via UART para se enviar dados por uma rede Wi-Fi.

Este módulo já foi anteriormente utilizado para o envio de dados inteligentes (SmartData) do TSTP para um servidor na internet, utilizando um firmware para ESP8266 desenvolvido pelo LISHA. Este firmware permite apenas o envio de dados de tamanho fixo para um mesmo servidor configurado, sem a possibilidade de se alterar parâmetros da requisição.

Foram desenvolvidas modificações no firmware para permitir a conexão com dois servidores simultaneamente, um utilizando a implementação já existente para o envio de dados inteligentes, e outro servidor utilizando uma nova implementação, para conectar-se com um WebService comum.

O firmware original recebe comandos, que podem possuir parâmetros ou não, pela serial UART, e os responde pela mesma interface. A versão modificada utiliza este mesmo padrão, porém os comandos foram modificados para identificar o servidor que deve ser utilizado, se é o de dados inteligentes (IoT) ou o WebService comum.

A nova implementação permite a utilização de autenticação Basic do protocolo HTTP, e retorna a resposta da requisição pela serial UART. Isto permite que possam ser recebidas informações do servidor pelo dispositivo conectado ao ESP8266.

Para o envio e recebimento de dados complexos, foi planejada a utilização de dados na notação de objetos de JavaScript (JSON), notação bastante utilizada por vários serviços da Web, inclusive serviços da própria SeTIC.

Durante os testes realizados com ESP8266 conectados diretamente à um terminal UART, sem utilizarmos o EPOSMote III, foi verificado que o módulo apresenta problemas de estabilidade.

Foram testados tanto o novo firmware quanto o firmware original desenvolvido pelo LISHA, e em ambos os casos pode acontecer de o módulo reiniciar sem um motivo aparente, ou até mesmo parar de responder à comandos até que ele seja reiniciado. Foram testados com outros módulos ESP8266 ESP-01, juntamente com diferentes fontes de alimentação, e o problema continuou acontecendo.

Em testes onde não ocorreram problemas com o módulo, foi verificado que o novo firmware pode realizar conexões com um servidor via HTTPS com autenticação Basic, enviar dados recebidos por parâmetro via POST e retornar a resposta da requisição.

A implementação de uma interface entre o EPOS e o ESP8266 foi muito prejudicada por conta destes problemas, além do fato de o tratamento de strings de dados ser muito complicado no EPOS, e de não existir um analisador JSON implementado para o sistema. Por isto, não foi possível implementar uma interface de comunicação durante o prazo de tempo disponível para a execução deste trabalho.

3.5. Interface de Controle

Para a realização do gerenciamento e auditoria por parte dos gerenciadores do controle de acesso, foi iniciado o desenvolvimento de uma aplicação Web para o controle do sistema.

A aplicação é responsável por fornecer uma interface para controle do sistema de acesso por parte dos gerentes, assim como armazenar e exibir dados para a auditoria de tentativas de acesso. Tanto a interface como o armazenamento de dados necessários foram implementados.

Esta aplicação foi planejada também para ser responsável por informar atualizações de políticas de acesso, cadastramento e descadastramento de usuários aos EPOSMotes que controlam as portas, assim como receber e armazenar os registros para auditoria das tentativas de acesso.

Estas atualizações seriam obtidas como respostas de requisições pelo EPOSMote feitas ao servidor da aplicação por meio de uma interface REST, porém como como a comunicação do EPOSMote não foi finalizada, a interface REST não foi implementada.

A implementação desta aplicação de controle foi feita em JavaScript utilizando o Node.js, em conjunto com o framework Express, que permite o tratamento de requisições HTTP. Para a implementação da interface, foram utilizados os frameworks Pug e Bootstrap, que permitem a criação de uma interface gráfica moderna de maneira ágil. Os dados do sistemas são armazenados no banco de dados PostgreSQL.

Foi implementado um sistema de login, com autenticação por usuário e senha. O usuário é armazenado em texto plano, enquanto que para a verificação de senha são armazenados um sal em conjunto com um resumo criptográfico da senha em conjunto com o sal.

Os usuários são divididos em três grupos:

- Gerentes: usuários que irão gerenciar o controle de acesso de uma ou mais portas, podendo adicionar ou remover pessoas e acompanhar os registros de tentativas de acesso;

- Administradores: usuários que podem cadastrar novos usuários, gerenciar as portas às quais os gerentes possuem autorização, alterar o grupo, nome ou senha de um outro usuário, e gerenciar todas as portas;
- Nenhum: usuários desativados que não podem fazer nada.

As portas são identificadas pelo sistema com base no seu endereço físico de rede (MAC). Os administradores podem configurar um nome para facilitar a identificação das portas.

Como a interface REST não foi implementada, foram inseridos manualmente dados falsos no banco de dados para a realização de testes de interface, simulando a existência de portas e tentativas de acesso.

Para o administrador do sistema e gerente de uma porta, foi implementado o cadastramento de cartões no sistema, que pode ser feito de duas maneiras. Uma delas é a inserção manual do nome do usuário e do UID do cartão. A segunda maneira é buscando pelo CPF ou usuário e número de matrícula da UFSC.

As alterações de usuários, como cadastramento e descadastramento, são salvas no banco de dados com um identificador serial, permitindo que as alterações de cadastro possam ser informadas às portas na ordem em que ocorreram, possibilitando a implementação de uma sincronização com a porta, onde apenas as mudanças são enviadas e os usuários cadastrados sejam os mesmos.

A importação de dados da UFSC é feita por meio de um WebService da SeTIC para a obtenção de dados de cartões emitidos. Por meio deste serviço, são obtidos e cadastrados o nome completo, UID do cartão e o ID Pessoa UFSC.

Ao gerenciar uma porta, é possível também realizar a auditoria, verificando as tentativas de acesso ordenadas a partir da mais recente, com informações de data e hora, UID do cartão, nome do usuário, caso cadastrado, e se o acesso foi autorizado ou negado.

5. Conclusão e Trabalhos Futuros

Neste trabalho, foram desenvolvidas soluções para controle de acesso e construídos protótipos funcionais que mostram a viabilidade de soluções de baixo custo para a implantação de sistemas de controle de acesso, com a utilização de tecnologias construídas na UFSC e outros equipamentos comerciais de baixo custo.

Foi realizado o desenvolvimento da integração de leitoras MFRC-522 com o EPOSMote III, permitindo a construção de equipamentos capazes de realizar a identificação por RFID com um hardware de custo bastante reduzido.

Foram desenvolvidas também aplicações de controle para portas e catracas, permitindo a utilização destes hardwares de baixo custo para realizar o controle de acesso propriamente dito.

Os sistemas construídos com EPOS são capazes de permitir o acesso somente à um grupo seleto de pessoas, porém, devido à falta de comunicação do EPOS com a rede, não permitem a realização de auditorias e de um gerenciamento mais avançado, onde se poderia aplicar políticas ou adicionar e remover autorizações de acesso mais facilmente.

Um sistema para se realizar o gerenciamento do controle de acesso e auditoria foi preparado e teve sua implementação iniciada, porém ainda é necessário desenvolver uma interface REST para se realizar a sincronização de dados com o sistema de controle no EPOS, assim como a implementação da comunicação no EPOS.

Foi realizada também a adaptação do sistema de acesso ao RU UFSC, desenvolvido pela SeTIC, para funcionamento com catracas, permitindo a automatização do acesso ao RU com a redução da necessidade de intervenção de um segurança ou funcionário para a realização do controle de acesso.

Como trabalho futuro, a principal sugestão seria a da conclusão do desenvolvimento da comunicação entre a aplicação de controle e o EPOSMote. Para isto, são necessários a implementação de uma interface REST na aplicação de controle, assim como o desenvolvimento de uma comunicação estável do EPOSMote com o servidor.

Para isto, sugere-se o estudo da utilização de diferentes firmwares para o módulo ESP8266 ou até mesmo a utilização de outros hardwares para comunicação, assim como a implementaçao de um interpretador JSON no EPOS e mudanças na aplicacão de controle da porta. Para o armazenamento dos registros e autorizações de acesso no EPOS para a sincronização com o servidor, é importante o desenvolvimento de correções para o sistema de arquivos do EPOS.

Outra sugestão importante de trabalho futuro é a utilização de estruturas para o armazenamento de políticas de acesso avançadas, tais como restrições de horário ou a possibilidade de se limitar a quantidade de vezes que um indivíduo pode acessar o local. Para isto, também seria importante a correção do sistema de arquivos do EPOS.

Outro trabalho futuro considerado como interessante seria o desenvolvimento de aplicações controladoras em EPOS para diferentes mecanismos de controle de acesso, tais como portas automáticas, cancelas ou até mesmo catracas para cadeirantes.

Uma sugestão interessante de trabalho futuro seria o estudo de possíveis métodos de fraude, avaliando e até mesmo implementando alguns métodos para a prevenção destas fraudes.

Referências

FRÖHLICH, A. A.; TIENTCHEU, G. P.; SCHRÖDER-PREIKSCHAT, W. Epos and myrinet: Effective communication support for parallel applications running on clusters of commodity workstation. In: SPRINGER. International Conference on High-Performance Computing and Networking. [S.l.], 2000. p. 417–426.

ISO/IEC. ISO 14443-3. Identification cards – Contactless integrated circuit cards – Proximity cards – Part 3: Initialization and anticollision. [S.l.], 2001.

NXP SEMICONDUCTORS. MFRC522 - Standard performance MIFARE and NTAG frontend - Product data sheet. [S.l.], 2016.

RESNER, D.; FRöHLICH, A. A. Tstp mac: A foundation for the trustful space-time protocol. Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES), 2016.

RICHARDSON, M.; WALLACE, S. Getting started with raspberry PI. [S.l.]: "O'Reilly Media, Inc.", 2012.

SCHWARTZ, M. Internet of Things with ESP8266. [S.l.]: Packt Publishing Ltd, 2016.

Software/Hardware Integration Lab. EPOSMote III. <http://epos.lisha.ufsc.br/HomePage>. Acessado em 12/09/2016.

Software/Hardware Integration Lab. Hydrologic Station board. <http://epos.lisha.ufsc.br/Hydrologic+Station+board>. Acessado em 22/05/2018.

WANT, R. An introduction to rfid technology. IEEE Pervasive Computing, v. 5, n. 1, p. 25–33, 2006.

ZHENG, J. et al. The internet of things [guest editorial]. IEEE Communications Magazine, v. 49, n. 11, p. 30–31, 2011.

ANEXO A – Módulo para leitora MFRC522 no EPOS

src/catraca/include/mifare.h

```
// Identification cards - Contactless integrated
    circuit(s) cards - Proximity cards standard
    (ISO/IEC 14443-3)

#ifndef __mifare_h
#define __mifare_h

#include <cpu.h>

__BEGIN_SYS

class MIFARE
{
    typedef CPU::Reg8 Reg8;

public:
    // MIFARE constants
    enum MIFARE_Misc {
        MF_ACK      = 0xA,
        MF_KEY_SIZE = 6,
    };

    // Commands sent to the MIFARE PICC (proximity
        integrated circuit card).
    // The commands used by the PCD (reader) to
        manage communication
    // with several PICCs (cards) (ISO 14443-3, Type
        A, section 6.4)
    enum PICC_Command {
        REQA            = 0x26,   // Request command,
            Type A. Invites PICCs in state IDLE to go
            to READY and prepare for anticollisionor
            selection.
        WUPA            = 0x52,   // Wake-UP command,
            Type A. Invites PICCs in state IDLE and
            HALT to go to READY and prepare
            foranticollision or selection.
        CT              = 0x88,   // Cascade Tag.
            Used during anti collision.
        SEL_CL1         = 0x93,   // Anti
            collision/Select, Cascade Level 1
        SEL_CL2         = 0x95,   // Anti
            collision/Select, Cascade Level 2
        SEL_CL3         = 0x97,   // Anti
            collision/Select, Cascade Level 3
        HLTA            = 0x50,   // Halt command,
            Type A. Instructs an ACTIVE PICC to go to
            state HALT.
```

```
    // MIFARE Classic commands
        (http://www.mouser.com/ds/2/302/MF1S503x-89574
        .pdf, Section 9.1)
    // Use MFAuthent to authenticate access to a
        sector, then use these commands to
        read/write/modify the blocks on the
        sector.
    MF_AUTH_KEY_A    = 0x60,  // Perform
        authentication with Key A
    MF_AUTH_KEY_B    = 0x61,  // Perform
        authentication with Key B
    MF_READ          = 0x30,  // Reads one 16 byte
        block from the authenticated sector of
        the PICC.
    MF_WRITE         = 0xA0,  // Writes one 16
        byte block to the authenticated sector of
        the PICC.
    MF_DECREMENT     = 0xC0,   // Decrements the
        contents of a block and stores the result
        in the internal data register.
    MF_INCREMENT     = 0xC1,   // Increments the
        contents of a block and stores the result
        in the internal data register.
    MF_RESTORE       = 0xC2,   // Reads the
        contents of a block into the internal
        data register.
    MF_TRANSFER      = 0xB0,    // Writes the
        contents of the internal data register to
        a block.
    // MIFARE Ultralight commands
        (http://www.nxp.com/documents/data_sheet/
        MF0ICU1.pdf, Section 7.6)
    // The PICC_CMD_MF_READ and PICC_CMD_MF_WRITE
        can also be used for MIFARE Ultralight.
    PICC_CMD_UL_WRITE = 0xA2 // Writes one 4 byte
        page to the PICC.
};

// PICC types
enum PICC_Type {
    MIFARE_UL   = 0x00, // MIFARE Ultralight or
        Ultralight C
    NOT_COMPLETE = 0x04, // SAK of an incomplete
        UID
    MIFARE_1K   = 0x08, // MIFARE Classic - 1KB
    MIFARE_MINI = 0x09, // MIFARE Classic - 320
        bytes
    MIFARE_PLUS = 0x11, // MIFARE Plus
    MIFARE_4K   = 0x18, // MIFARE Classic - 4KB
```

```
    ISO_14443_4 = 0x20, // PICC compliant with
        ISO/IEC 14443-4
    ISO_18092   = 0x40, // PICC compliant with
        ISO/IEC 18092
};

class UID
{
public:
    static const unsigned int SIZE_MAX = 10;

    UID() : _size(0) {}
    UID(const void * id, unsigned int size =
        SIZE_MAX, Reg8 sak = 0) {
        uid(id, size);
        _uid_sak[_size] = sak;
    }
    UID(unsigned int v) : _size(4) {
        unsigned int i;
        for(i = 0; i < 4; i++)
            _uid_sak[i] =
                reinterpret_cast<Reg8*>(&v)[i];
        for(; i < SIZE_MAX + 1; i++)
            _uid_sak[i] = 0;
    }
    UID(unsigned long long v) : _size(4) { // ID
        size must be 4, 7, or 10
        Debug dbg;
        dbg << "CHEGOU:" << v << endl;
        unsigned int i;
        for(i = 0; i < 8; i++) {
            _uid_sak[i] =
                reinterpret_cast<Reg8*>(&v)[i];
        }
        for(; i < SIZE_MAX + 1; i++)
            _uid_sak[i] = 0;
    }

    const Reg8 * uid() const { return _uid_sak; }
    Reg8 * uid() { return _uid_sak; }

    void sak(Reg8 s) { _uid_sak[_size] = s; }
    PICC_Type type() const { return
        static_cast<PICC_Type>(_uid_sak[_size] &
        0x7F); }

    Reg8 size() const { return _size; }
    void size(unsigned int s) { assert(_size <=
        SIZE_MAX); _size = s; }
```

```
    void uid(const void * u, unsigned int s) {
        size(s); uid(u); }
    void uid(const void * u) {
        for(unsigned int i = 0; i < SIZE_MAX; i++)
            _uid_sak[i] = (i < _size) ?
                reinterpret_cast<const
                Reg8*>(u)[i] : 0;
    }

    operator unsigned int() const {
        unsigned long long ret = 0;
        for(unsigned int i = 0; i < SIZE_MAX; i++)
            ret ^= _uid_sak[i] << ((i %
                sizeof(unsigned int)) * 8);
        return ret;
    }

    // operator unsigned long long() const {
    //    unsigned long long ret = 0;
    //    for(unsigned int i = 0; i < SIZE_MAX;
        i++)
    //        ret ^= _uid_sak[i] << ((i %
        sizeof(unsigned long long)) * 8);
    //    return ret;
    // }

    friend Debug & operator<<(Debug & db, const
        UID & u) {
        db << "{s=" << u._size << ",u=";
        db << hex << u._uid_sak[0];
        for(unsigned int i = 1; i < u._size; i++)
            db << "," << hex << u._uid_sak[i];
        db << ",sa=" << u._uid_sak[u._size] << "}";
        return db;
    }

    friend OStream & operator<<(OStream & os,
        const UID & u) {
        os << "{s=" << u._size << ",u=";
        os << hex << u._uid_sak[0];
        for(unsigned int i = 1; i < u._size; i++)
            os << "," << hex << u._uid_sak[i];
        os << ",sa=" << u._uid_sak[u._size] << "}";
        return os;
    }

private:
    Reg8 _size;
```

```
    Reg8 _uid_sak[SIZE_MAX + 1];
}__attribute__((packed));

class Key
{
public:
    static const unsigned int KEY_SIZE = 6;

    enum Key_Type {
        TYPE_A,
        TYPE_B,
    };

    Key() {}
    Key(const void * key, Key_Type type) :
        _type(type) {
        memcpy(_key, key, KEY_SIZE);
    }

    static unsigned int size() { return KEY_SIZE;
        }
    const Reg8 * key() const { return _key; }
    const Key_Type type() const { return _type; }

    void key(const void * k) { memcpy(_key, k,
        KEY_SIZE); }

    friend Debug & operator<<(Debug & db, const
        Key & k) {
        db << "{t=" << (k._type == TYPE_A ? 'A' :
            'B') << ",k=[";
        db << hex << k._key[0];
        for(unsigned int i = 1; i < KEY_SIZE; i++)
            db << "," << hex << k._key[i];
        db << "]}";
        return db;
    }

    friend OStream & operator<<(OStream & os,
        const Key & k) {
        os << "{t=" << (k._type == TYPE_A ? 'A' :
            'B') << ",k=[";
        os << hex << k._key[0];
        for(unsigned int i = 1; i < KEY_SIZE; i++)
            os << "," << hex << k._key[i];
        os << "]}";
        return os;
    }
```

```
    private:
        Key_Type _type;
        Reg8 _key[KEY_SIZE];
    }__attribute__((packed));
};

__END_SYS

#endif
```

src/catraca/include/machine/cortex/rfid_reader.h

```
// EPOS ARM Cortex RFID Reader Mediator Declarations

#include <system/config.h>

#ifndef __cortex_rfid_reader_h
#define __cortex_rfid_reader_h

#include <rfid_reader.h>
#include <machine.h>
#include <mifare.h>
#include <spi.h>
#include <gpio.h>

__BEGIN_SYS

// MFRC522 MIFARE Reader/Writer chip from NXP
class MFRC522: private Machine_Model, public MIFARE
{
    static const unsigned int BLOCK_SIZE = 16;

public:
    enum PCD_Status_Code {
        OK            = 0, // Success
        ERROR         = 1, // Error in communication
        COLLISION     = 2, // Collission detected
        NO_ROOM       = 3, // A buffer is not big
            enough.
        INVALID       = 4, // Invalid argument.
        CRC_WRONG     = 5, // The CRC_A does not match
        MIFARE_NACK   = 6, // A MIFARE PICC responded
            with NAK.
        TIMEOUT       = 7, // Request time exceeded
        INTERNAL_ERROR = 8, // Error in the module
            code
    };
```

```
// MFRC522 reader commands
enum PCD_Command {
    IDLE              = 0x00, // no action,
        cancels current command execution
    MEM               = 0x01, // stores 25 bytes
        into the internal buffer
    GENERATE_RANDOM_ID = 0x02, // generates a
        10-byte random ID number
    CALC_CRC          = 0x03, // activates the CRC
        coprocessor or performs a self-test
    TRANSMIT          = 0x04, // transmits data
        from the FIFO buffer
    NO_CMD_CHANGE     = 0x07, // no command
        change, can be used to modify the
        CommandRegbits without affecting the
        command
    RECEIVE           = 0x08, // activates the
        receiver circuits
    TRANSCEIVE        = 0x0C, // transmits data
        from FIFO buffer to antenna and activates
        the receiver after transmission
    MF_AUTHENT        = 0x0E, // performs the
        MIFARE standard authentication as a reader
    SOFT_RESET        = 0x0F, // resets the MFRC522
};

// MFRC522 reader registers addresses
enum PCD_Register {
    // Command and status
    COMMAND           = 0x01 << 1, // starts and
        stops command execution
    COM_I_EN          = 0x02 << 1, // enable and
        disable interrupt request control bits
    DIV_I_EN          = 0x03 << 1, // enable and
        disable interrupt request control bits
    COM_IRQ           = 0x04 << 1, // interrupt
        request bits
    DIV_IRQ           = 0x05 << 1, // interrupt
        request bits
    ERROR_REG         = 0x06 << 1, // error bits
        showing the error status of the last
        command executed
    STATUS_1          = 0x07 << 1, // communication
        status bits
    STATUS_2          = 0x08 << 1, // receiver and
        transmitter status bits
    FIFO_DATA         = 0x09 << 1, // input and
        output of 64 byte FIFO buffer
    FIFO_LEVEL        = 0x0A << 1, // number of
```

```
    bytes stored in the FIFO buffer
WATER_LEVEL      = 0x0B << 1, // level for
    FIFO underflow and overflow warning
CONTROL          = 0x0C << 1, // miscellaneous
    control registers
BIT_FRAMING      = 0x0D << 1, // adjustments
    for bit-oriented frames
COLL             = 0x0E << 1, // bit position
    of the first bit-collision detected on
    the RF interface
// Command
MODE             = 0x11 << 1, // defines
    general modes for transmitting and
    receiving
TX_MODE          = 0x12 << 1, // defines
    transmission data rate and framing
RX_MODE          = 0x13 << 1, // defines
    reception data rate and framing
TX_CONTROL       = 0x14 << 1, // controls the
    logical behavior of the antenna driver
    pins TX1 and TX2
TX_ASK           = 0x15 << 1, // controls the
    setting of the transmission modulation
TX_SEL           = 0x16 << 1, // selects the
    internal sources for the antenna driver
RX_SEL           = 0x17 << 1, // selects
    internal receiver settings
RX_THRESHOLD     = 0x18 << 1, // selects
    thresholds for the bit decoder
DEMOD            = 0x19 << 1, // defines
    demodulator settings
MF_TX            = 0x1C << 1, // controls some
    MIFARE communication transmit parameters
MF_RX            = 0x1D << 1, // controls some
    MIFARE communication receive parameters
SERIAL_SPEED     = 0x1F << 1, // selects the
    speed of the serial UART interface
// Configuration
CRC_RESULT_H     = 0x21 << 1, // shows the MSB
    and LSB values of the CRC calculation
CRC_RESULT_L     = 0x22 << 1, // shows the MSB
    and LSB values of the CRC calculation
MOD_WIDTH        = 0x24 << 1, // controls the
    ModWidth setting
RF_CFG           = 0x26 << 1, // configures
    the receiver gain
GS_N             = 0x27 << 1, // selects the
    conductance of the antenna driver pins
    TX1 and TX2 for modulation
```

```
        CW_GS_P          = 0x28 << 1, // defines the
            conductance of the p-driver output during
            periods of no modulation
        MOD_GS_P         = 0x29 << 1, // defines the
            conductance of the p-driver output during
            periods of modulation
        T_MODE           = 0x2A << 1, // defines
            settings for the internal timer
        T_PRESCALER      = 0x2B << 1, // defines
            settings for the internal timer
        T_RELOAD_H       = 0x2C << 1, // defines the
            16-bit timer reload value
        T_RELOAD_L       = 0x2D << 1, // defines the
            16-bit timer reload value
        T_COUNTER_VALUE_H = 0x2E << 1, // shows the
            16-bit timer value
        T_COUNTER_VALUE_L = 0x2F << 1, // shows the
            16-bit timer value
        // Test registers
        TEST_SEL_1       = 0x31 << 1, // general test
            signal configuration
        TEST_SEL_2       = 0x32 << 1, // general test
            signal configuration
        TEST_PIN_EN      = 0x33 << 1, // enables pin
            output driver on pins D1 to D7
        TEST_PIN_VALUE   = 0x34 << 1, // defines the
            values for D1 to D7 when it is used as an
            I/O bus
        TEST_BUS         = 0x35 << 1, // shows the
            status of the internal test bus
        AUTO_TEST        = 0x36 << 1, // controls the
            digital self-test
        VERSION          = 0x37 << 1, // shows the
            software version
        ANALOG_TEST      = 0x38 << 1, // controls the
            pins AUX1 and AUX2
        TEST_DAC_1       = 0x39 << 1, // defines the
            test value for TestDAC1
        TEST_DAC_2       = 0x3A << 1, // defines the
            test value for TestDAC2
        TEST_ADC         = 0x3B << 1, // shows the
            value of ADC I and Q channels
    };

protected:
    typedef MIFARE::UID UID;
    typedef MIFARE::Key Key;

    typedef unsigned char Block[BLOCK_SIZE];
```

```
    MFRC522(SPI * spi, GPIO * select, GPIO * reset);
    ~MFRC522();

    void initialize();
    void reset();
    bool card_present();
    bool ready_to_get() { return card_present(); }
    bool ready_to_put() { return card_present(); }
    bool read_card(UID * uid, unsigned int valid_bits
        = 0);
    bool select(UID & uid) { return read_card(&uid,
        uid.size() * 8); }
    void halt_card();
    unsigned int put(unsigned int block, const Block
        data);
    unsigned int read(unsigned int block, Block data);
    bool authenticate(unsigned int sector, const Key
        & key, const UID & UID);

    void int_enable(); // Unused
    void int_disable(); // Unused
    static bool input0_handler(const IC::Interrupt_Id
        & id); // Unused
    static bool input1_handler(const IC::Interrupt_Id
        & id); // Unused

    void deauthenticate() {
        // clear MFCrypto1On bit
        write_reg(PCD_Register::STATUS_2,
            read_reg(PCD_Register::STATUS_2) &
            (~0x08));
    }

public:
    static unsigned int sector(unsigned int block) {
        if(block < 128)
            return block / 4;
        return 32 + (block - 128) / 16;
    }
    static unsigned int key_block(unsigned int
        sector) {
        if(sector < 32)
            return sector * 4 + 3;
        return 128 + (sector - 32) * 16 + 15;
    }

private:
    PCD_Status_Code crc(const void * data, unsigned
```

```
        int length, CPU::Reg8 * result);

    PCD_Status_Code communicate_with_picc(
        CPU::Reg8 command, CPU::Reg8 wait_irq,
            CPU::Reg8 * send_data, CPU::Reg8
            send_len, CPU::Reg8 * back_data = 0,
        CPU::Reg8 * back_len = 0, CPU::Reg8 *
            valid_bits = 0, CPU::Reg8 rx_align = 0,
            bool check_crc = false
    );

    PCD_Status_Code mifare_transceive(const
        CPU::Reg8* data, CPU::Reg8 length, bool
        accept_timeout = false);

    CPU::Reg8 read_reg(PCD_Register reg);
    void read_reg(PCD_Register reg, CPU::Reg8 count,
        CPU::Reg8 * values, CPU::Reg8 rx_align);
    void write_reg(PCD_Register reg, CPU::Reg8 value);
    void write_reg(PCD_Register reg, unsigned int
        count, const void * values);

    SPI * _spi;
    GPIO * _select, * _reset;
};


typedef MFRC522 RFID_Reader_Engine;

class RFID_Reader: public RFID_Reader_Common,
    private RFID_Reader_Engine
{
    typedef RFID_Reader_Engine Engine;

public:
    typedef Engine::UID UID;
    typedef Engine::Key Key;
    typedef Engine::Block Block;

    typedef _UTIL::Observer Observer;
    typedef _UTIL::Observed Observed;

    using Engine::int_enable;
    using Engine::int_disable;

    static void attach(Observer * obs) {
        _observed.attach(obs); }
    static void detach(Observer * obs) {
        _observed.detach(obs); }
```

```
    static void input0_handler(const IC::Interrupt_Id
        & id) {
        if(Engine::input0_handler(id))
            notify();
    }

    static void input1_handler(const IC::Interrupt_Id
        & id) {
        if(Engine::input1_handler(id))
            notify();
    }

public:
    RFID_Reader(SPI * spi, GPIO * gpio0, GPIO *
        gpio1) : Engine(spi, gpio0, gpio1),
        _enabled(true) {
        Engine::initialize();
    }

    void enabled(bool b) {
        _enabled = b;
    }

    UID get() {
        UID u;
        while (true) {
            while(!ready_to_get());
            if (Engine::read_card(&u))
                return u;
        }
    }

    bool halt(UID & u) {
        for (unsigned int i = 0; i < 2; i++) {
            if(ready_to_put() && Engine::select(u)) {
                Engine::halt_card();
                return true;
            }
        }
        return false;
    }

    unsigned int put(UID & u, unsigned int block,
        const Block data) {
        unsigned int ret = 0;
        if(select(u))
            ret = Engine::put(block, data);
        return ret;
```

```
    }

    unsigned int put(UID & u, unsigned int block,
        const Block data, const Key & key) {
        unsigned int ret = 0;
        if(select(u)) {
            unsigned int s = sector(block);
            if(Engine::authenticate(s, key, u))
                ret = Engine::put(block, data);
            Engine::deauthenticate();
        }
        return ret;
    }

    unsigned int read(UID & u, unsigned int block,
        Block data) {
        unsigned int ret = 0;
        if(select(u))
            ret = Engine::read(block, data);
        return ret;
    }

    unsigned int read(UID & u, unsigned int block,
        Block data, const Key & key) {
        unsigned int ret = 0;
        if(select(u)) {
            unsigned int s = sector(block);
            if(Engine::authenticate(s, key, u))
                ret = Engine::read(block, data);
            Engine::deauthenticate();
        }
        return ret;
    }

    void reset() { Engine::reset(); }

    bool ready_to_get() { return _enabled &&
        Engine::ready_to_get(); }
    bool ready_to_put() { return _enabled &&
        Engine::ready_to_put(); }

protected:
    static bool notify() { return _observed.notify();
        }

private:
    bool select(UID & u) {
        while(!ready_to_get());
        return Engine::select(u);
```

```
    }

    bool _enabled;
    static Observed _observed;
};

__END_SYS

#endif
```

src/catraca/src/machine/cortex/rfid_reader.cc

```
// EPOS ARM Cortex RFID Reader Mediator
    Implementation

#include <system/config.h>

#ifdef __RFID_READER_H

#include <machine/cortex/rfid_reader.h>

__BEGIN_SYS

// RFID Reader API

// Class attributes
RFID_Reader::Observed RFID_Reader::_observed;

// Methods


// MFRC522 MIFARE Reader chip from NXP

// Class attributes

// Methods
MFRC522::MFRC522(SPI * spi, GPIO * select, GPIO *
    reset) : _spi(spi), _select(select), _reset(reset)
{
    _spi->disable();
    _select->direction(GPIO::OUT);
    _select->set();
    _reset->direction(GPIO::INOUT);
    _reset->clear();
}

MFRC522::~MFRC522()
{
```

```
    _spi->disable();
}

void MFRC522::initialize()
{
    if(!_reset->get()) {
        _reset->set();
        Machine::delay(100000); // According to
            datasheet section 8.8.2, the MFRC522
            start-up delay time is the start-up time
            of the crystal oscilator circuit +
            37.74us.
    }
    else
        reset();

    // Set timeout
    write_reg(PCD_Register::T_MODE, 0x80); // Start
        timer automatically

    write_reg(PCD_Register::T_PRESCALER, 0xA9); //
        freq timer = 40kHz (25us)
    write_reg(PCD_Register::T_RELOAD_H, 0x03); //
        Reload timer after 1000 (0x03E8) ticks (25ms)
    write_reg(PCD_Register::T_RELOAD_L, 0xE8);

    write_reg(PCD_Register::TX_ASK, 0x40); // Force
        100% ASK modulation
    write_reg(PCD_Register::MODE, 0x3D); // Set CRC
        coprocessor preset value

    // Turn antenna on
    CPU::Reg8 value =
        read_reg(PCD_Register::TX_CONTROL);
    if((value & 0x03) != 0x03)
        write_reg(PCD_Register::TX_CONTROL, value |
            0x03);
}

void MFRC522::reset()
{
    write_reg(PCD_Register::COMMAND,
        PCD_Command::SOFT_RESET);
    Machine::delay(100000); // According to datasheet
        section 8.8.2, the MFRC522 start-up delay
        time is the start-up time of the crystal
        oscilator circuit + 37.74us.
    while(read_reg(PCD_Register::COMMAND) & (1 <<
        4)); // PCD still restarting
```

```
}

bool MFRC522::card_present()
{
    CPU::Reg8 buffer_answer[2];
    CPU::Reg8 buffer_size = sizeof(buffer_answer);

    write_reg(PCD_Register::COLL,
        read_reg(PCD_Register::COLL) & (~0x80)); //
        Clear bits received after collision.
    CPU::Reg8 valid_bits = 7;
    CPU::Reg8 command = MIFARE::PICC_Command::REQA;
    CPU::Reg8 wait_irq = 0x30;
    PCD_Status_Code status =
        communicate_with_picc(PCD_Command::TRANSCEIVE,
        wait_irq, &command, 1, buffer_answer,
        &buffer_size, &valid_bits);
    if(status == PCD_Status_Code::COLLISION)
        return true;

    if((buffer_size != 2) || (valid_bits != 0)) //
        ATQA must have 16 bits
        return false;

    return status == PCD_Status_Code::OK;
}

bool MFRC522::read_card(UID * uid, unsigned int
    valid_bits)
{
    write_reg(PCD_Register::COLL,
        read_reg(PCD_Register::COLL) & (~0x80));

    bool use_cascade_tag;
    CPU::Reg8 cascade_level = 1;
    CPU::Reg8 buffer[9];
    CPU::Reg8 uid_index, count, index;

    for(;;) {
        switch(cascade_level) {
            case 1:
                buffer[0] =
                    MIFARE::PICC_Command::SEL_CL1;
                uid_index = 0;
                use_cascade_tag = (valid_bits > 0) &&
                    (uid->size() > 4); // Use only
                    when it's known that the UID has
                    more than 4 bytes.
            break;
```

```
        case 2:
            buffer[0] =
                MIFARE::PICC_Command::SEL_CL2;
            uid_index = 3;
            use_cascade_tag = (valid_bits > 0) &&
                (uid->size() > 7); // Use only
                when it's known that the UID has
                more than 7 bytes.
        break;
        case 3:
            buffer[0] =
                MIFARE::PICC_Command::SEL_CL3;
            uid_index = 6;
            use_cascade_tag = false; // Never used
                in level 3.
        break;
        default:
            return false;
        break;
    }

    // How many UID bits are known in the current
        level?
    int current_level_known_bits = valid_bits -
        (8 * uid_index);
    if(current_level_known_bits < 0)
        current_level_known_bits = 0;

    // Copy known bits from uid to buffer
    index = 2; // destination index in buffer[]
    if(use_cascade_tag)
        buffer[index++] = MIFARE::PICC_Command::CT;

    unsigned int bytes_to_copy =
        current_level_known_bits / 8 +
        ((current_level_known_bits % 8) ? 1 : 0);
        // The number of bytes to be copied.
    if(bytes_to_copy) {
        unsigned int max_bytes = use_cascade_tag ?
            3 : 4; // Max of 4 bytes per level.
            Cascade Tag takes 1 byte.
        if(bytes_to_copy > max_bytes)
            bytes_to_copy = max_bytes;
        for(count = 0; count < bytes_to_copy;
            count++)
            buffer[index++] = uid->uid()[uid_index
                + count];
    }
```

```
// Include the 8 bits in Cascade Tag in
    current_level_known_bits
if(use_cascade_tag)
    current_level_known_bits += 8;

CPU::Reg8 response_length, tx_last_bits,
    buffer_used;
CPU::Reg8* response_buffer;
// Repeat anti collision loop until it is
    possible to transmit all UID bits + BCC
    and receive a SAK
for(;;) {
    if(current_level_known_bits >= 32) { //
        All UID bits in this Cascade Level are
        known. This is a SELECT.
        buffer[1] = 0x70; // Number of Valid
            Bits = Seven bytes
        buffer[6] = buffer[2] ^ buffer[3] ^
            buffer[4] ^ buffer[5]; //
            Calculate BCC.
        unsigned int result = crc(buffer, 7,
            &buffer[7]); // Calculate CRC_A.
        if(result != PCD_Status_Code::OK)
            return false;

        tx_last_bits = 0; // 0 -> 8 valid bits
        buffer_used = 9;
        // Store response in the last 3 bytes
            of buffer (BCC and CRC_A - not
            needed after tx)
        response_buffer = &buffer[6];
        response_length = 3;
    } else { // anti collision
        tx_last_bits  =
            current_level_known_bits % 8;
        count         =
            current_level_known_bits / 8; //
            Number bytes in the UID part.
        index         = 2 + count;
                            // Number bytes:
            SEL + NVB + UIDs
        buffer[1]     = (index << 4) +
            tx_last_bits; // Number of Valid
            Bits
        buffer_used   = index + (tx_last_bits ?
            1 : 0);
        // Store response in the unused part
            of buffer
        response_buffer = &buffer[index];
```

```
    response_length = sizeof(buffer) -
        index;
}

CPU::Reg8 rx_align = tx_last_bits;
write_reg(PCD_Register::BIT_FRAMING,
    (rx_align << 4) + tx_last_bits);

CPU::Reg8 wait_irq = 0x30;
PCD_Status_Code result =
    communicate_with_picc(
    PCD_Command::TRANSCEIVE, wait_irq,
        buffer, buffer_used,
        response_buffer, &response_length,
        &tx_last_bits, rx_align
);

if(result == PCD_Status_Code::COLLISION) {
    // More than one PICC found
    CPU::Reg8 coll_reg_val =
        read_reg(PCD_Register::COLL);
    if(coll_reg_val & 0x20) //
        CollPosNotValid
        return false; // No valid
            collision position.

    CPU::Reg8 collision_pos = coll_reg_val
        & 0x1F;
    if(collision_pos == 0)
        collision_pos = 32;

    if(collision_pos <=
        current_level_known_bits) // No
        progress
        return false;

    current_level_known_bits =
        collision_pos;
    count = (current_level_known_bits - 1)
        % 8; // The bit to be modified
    index = 1 + (current_level_known_bits
        / 8) + (count ? 1 : 0);
    buffer[index] |= (1 << count);
}
else if(result != PCD_Status_Code::OK)
    return false;
else {
    if(current_level_known_bits >= 32) //
        SELECT done
```

```
                    break;
                else
                    current_level_known_bits = 32;
            }
        }

        // Copy UID bytes from buffer to uid
        index = (buffer[2] ==
            MIFARE::PICC_Command::CT) ? 3 : 2;
        bytes_to_copy = (buffer[2] ==
            MIFARE::PICC_Command::CT) ? 3 : 4;
        memcpy(&uid->uid()[uid_index],
            &buffer[index], bytes_to_copy);

        // Check response SAK
        if((response_length != 3) || (tx_last_bits !=
            0)) // SAK must have 24 bits (1 byte +
            CRC_A).
            return false;

        // Verify CRC_A
        if(crc(response_buffer, 1, &buffer[2]) !=
            PCD_Status_Code::OK)
            return false;

        if((buffer[2] != response_buffer[1]) ||
            (buffer[3] != response_buffer[2]))
            return false;

        if(response_buffer[0] & 0x04) // Cascade bit
            set - UID not ready
            cascade_level++;
        else {
            uid->size(3 * cascade_level + 1);
            uid->sak(response_buffer[0]);
            break;
        }
    }

    return true;
}

void MFRC522::halt_card()
{
    CPU::Reg8 buffer[4];

    buffer[0] = MIFARE::PICC_Command::HLTA;
    buffer[1] = 0;
```

```
    if(crc(buffer, 2, &buffer[2]) !=
        PCD_Status_Code::OK)
        return;

    CPU::Reg8 wait_irq = 0x30; // RX_IRQ and IDLE_IRQ
    communicate_with_picc(PCD_Command::TRANSCEIVE,
        wait_irq, buffer, sizeof(buffer));
}

unsigned int MFRC522::put(unsigned int block, const
    Block data)
{
    CPU::Reg8 command[2] =
        {MIFARE::PICC_Command::MF_WRITE,
        static_cast<CPU::Reg8>(block)};
    if(mifare_transceive(command, sizeof(command)) !=
        PCD_Status_Code::OK)
        return 0;
    return mifare_transceive(data, 16) ==
        PCD_Status_Code::OK;
}

unsigned int MFRC522::read(unsigned int block, Block
    data)
{
    CPU::Reg8 buffer[18];
    buffer[0] = MIFARE::PICC_Command::MF_READ;
    buffer[1] = block;
    if(crc(buffer, 2, &buffer[2]) !=
        PCD_Status_Code::OK)
        return 0;
    else {
        CPU::Reg8 wait_irq = 0x30;
        CPU::Reg8 buffer_size = sizeof(buffer);
        PCD_Status_Code status =
            communicate_with_picc(PCD_Command::TRANSCEIVE,
            wait_irq, buffer, 4, buffer,
            &buffer_size, 0, 0, true);
        if(status == PCD_Status_Code::OK)
            memcpy(data, buffer, 16);
        return status == PCD_Status_Code::OK;
    }
}

bool MFRC522::authenticate(unsigned int sector,
    const Key & key, const UID & uid)
{
    CPU::Reg8 block;
    block = MFRC522::key_block(sector);
```

```
    CPU::Reg8 cmd = key.type() == Key::TYPE_A ?
        MIFARE::MF_AUTH_KEY_A : MIFARE::MF_AUTH_KEY_B;

    // buffer[0..1] = {cmd, block}
    CPU::Reg8 buffer[12] = {cmd, block};

    // buffer[2..7] = key bytes
    for(CPU::Reg8 i = 0; i < key.size(); i++)
        buffer[2+i] = key.key()[i];

    for(unsigned int i = 0; i < 4; i++)
        buffer[8+i] = uid.uid()[(uid.size()-4)+i];

    CPU::Reg8 wait_irq = 0x10; //IdleIRq

    return communicate_with_picc(
        PCD_Command::MF_AUTHENT, wait_irq, buffer,
            sizeof(buffer)
    ) == PCD_Status_Code::OK;
}

MFRC522::PCD_Status_Code MFRC522::crc(const void *
    data, unsigned int length, unsigned char * result)
{
    write_reg(PCD_Register::COMMAND,
        PCD_Command::IDLE); // Stop active commands
    write_reg(PCD_Register::DIV_IRQ, 0x04);
                    // Clear CRCIRq interrupt request
        bit
    write_reg(PCD_Register::FIFO_LEVEL,
        read_reg(PCD_Register::FIFO_LEVEL) | 0x80);
        // FIFO initialization
    write_reg(PCD_Register::FIFO_DATA, length, data);
         // Push data to the FIFO
    write_reg(PCD_Register::COMMAND,
        PCD_Command::CALC_CRC); // Start calculation

    // Each loop iteration takes 7.3 us. Timeout if
        it takes more than 90 ms.
    for(unsigned int i = 13000;;) {
        CPU::Reg8 n = read_reg(PCD_Register::DIV_IRQ);
        if(n & 0x04) // CRC_IRQ bit set - calculation
            done
            break;

        if(--i == 0)
            return PCD_Status_Code::TIMEOUT;
    }
```

```
    result[0] = read_reg(PCD_Register::CRC_RESULT_L);
    result[1] = read_reg(PCD_Register::CRC_RESULT_H);

    return PCD_Status_Code::OK;
}


MFRC522::PCD_Status_Code
    MFRC522::communicate_with_picc(
    CPU::Reg8 command, CPU::Reg8 wait_irq, CPU::Reg8*
        send_data, CPU::Reg8 send_len, CPU::Reg8*
        back_data,
    CPU::Reg8* back_len, CPU::Reg8* valid_bits,
        CPU::Reg8 rx_align, bool check_crc
)
{
    // Prepare values forBitFramingReg
    CPU::Reg8 tx_last_bits = valid_bits ? *valid_bits
        : 0;
    CPU::Reg8 bit_framing = (rx_align << 4) +
        tx_last_bits; // rx_align =
        bit_framing_reg[6..4]. tx_last_bits =
        bit_framing_reg[2..0]

    write_reg(PCD_Register::COMMAND,
        PCD_Command::IDLE); // Stop active commands
    write_reg(PCD_Register::COM_IRQ, 0x7f);
                    // Clear all interrupt request bits
    write_reg(PCD_Register::FIFO_LEVEL,
        read_reg(PCD_Register::FIFO_LEVEL) | 0x80);
        // FIFO initialization
    write_reg(PCD_Register::FIFO_DATA, send_len,
        send_data); // Push data to the FIFO
    write_reg(PCD_Register::BIT_FRAMING,
        bit_framing); // Bit adjustments
    write_reg(PCD_Register::COMMAND, command);
                // Start command execution

    if(command == PCD_Command::TRANSCEIVE)
        write_reg(PCD_Register::BIT_FRAMING,
            read_reg(PCD_Register::BIT_FRAMING) |
            0x80); // StartSend=1, transmission of
            data starts

    // Each loop iteration takes 7.3 us. Timeout if
        it takes more than 90 ms.
    for(unsigned int i = 13000;;) {
        CPU::Reg8 n = read_reg(PCD_Register::COM_IRQ);
```

```
        if(n & wait_irq) // Success interruption bit
            set
            break;
        else if(n & 0x01)
            return PCD_Status_Code::TIMEOUT;
        else if(--i == 0)
            return PCD_Status_Code::TIMEOUT;
    }

    CPU::Reg8 error =
        read_reg(PCD_Register::ERROR_REG);
    if(error & 0x13) // BufferOvfl ParityErr
        ProtocolErr
        return PCD_Status_Code::ERROR;

    CPU::Reg8 aux_valid_bits = 0;

    // Get data back from MFRC522 when requested.
    if(back_data && back_len) {
        CPU::Reg8 n =
            read_reg(PCD_Register::FIFO_LEVEL);
        if(n > *back_len)
            return PCD_Status_Code::NO_ROOM;

        *back_len = n;
        read_reg(PCD_Register::FIFO_DATA, n,
            back_data, rx_align); // Get data from
            FIFO
        aux_valid_bits =
            read_reg(PCD_Register::CONTROL) & 0x07;
            // Number of valid bits in the last
            received byte. 0b000 -> byte valid.
        if(valid_bits)
            *valid_bits = aux_valid_bits;
    }

    if(error & 0x08) // CollErr
        return PCD_Status_Code::COLLISION;

    // Perform CRC_A validation when requested.
    if(back_data && back_len && check_crc) {
        if((*back_len == 1) && (aux_valid_bits == 4))
            return PCD_Status_Code::MIFARE_NACK;

        if((*back_len < 2) || (aux_valid_bits != 0))
            return PCD_Status_Code::CRC_WRONG;

        CPU::Reg8 control_buffer[2];
        PCD_Status_Code status = crc(&back_data[0],
```

```
            *back_len - 2, &control_buffer[0]);
        if(status != PCD_Status_Code::OK)
            return status;

        if((back_data[*back_len - 2] !=
            control_buffer[0]) ||
            (back_data[*back_len - 1] !=
            control_buffer[1]))
            return PCD_Status_Code::CRC_WRONG;
    }

    return PCD_Status_Code::OK;
}

MFRC522::PCD_Status_Code
    MFRC522::mifare_transceive(const CPU::Reg8* data,
    CPU::Reg8 length, bool accept_timeout)
{
    if((data == 0) || (length > 16))
        return PCD_Status_Code::INVALID;

    CPU::Reg8 buffer[18]; // 16 bytes of data and 2
        bytes of CRC_A
    // Copy data[] to buffer[]
    memcpy(buffer, data, length);

    // Add CRC_A to buffer[]
    PCD_Status_Code result = crc(buffer, length,
        &buffer[length]);
    if(result != PCD_Status_Code::OK)
        return result;
    length += 2;

    // transceive data and store reply in buffer[]
    CPU::Reg8 wait_irq = 0x30; // RxIRq and IdleIRq
    CPU::Reg8 bufferSize = sizeof(buffer);
    CPU::Reg8 validBits = 0;
    result = communicate_with_picc(
        PCD_Command::TRANSCEIVE, wait_irq, buffer,
        length, buffer, &bufferSize, &validBits
    );

    if(accept_timeout && (result ==
        PCD_Status_Code::TIMEOUT))
        return PCD_Status_Code::OK;
    else if(result != PCD_Status_Code::OK)
        return result;

    // PICC must reply an ACK with 4 bits
```

```
    if((bufferSize != 1) || (validBits != 4))
        return PCD_Status_Code::ERROR;
    else if(buffer[0] != MIFARE::MIFARE_Misc::MF_ACK)
        return PCD_Status_Code::MIFARE_NACK;

    return PCD_Status_Code::OK;
}

CPU::Reg8 MFRC522::read_reg(PCD_Register reg)
{
    _spi->enable();
    _select->clear();
    _spi->put_data(0x80 | (reg & 0xfe)); // MSB == 1
        -> read // LSB is always 0
    _spi->get_data();
    _spi->put_data(0);
    CPU::Reg8 data = _spi->get_data();
    _select->set();
    _spi->disable();
    return data;
}

void MFRC522::read_reg(PCD_Register reg, CPU::Reg8
    count, CPU::Reg8* values, CPU::Reg8 rx_align)
{
    if(count == 0)
        return;

    CPU::Reg8 address = 0x80 | (reg & 0xfe); // MSB
        == 1 -> read // LSB is always 0
    unsigned int index = 0;
    count--;
    _spi->enable();
    _select->clear();
    _spi->put_data(address);
    _spi->get_data();
    while (index < count) {
        if(index == 0 && rx_align) { // Only update
            bit positions rx_align..7 in values[0]
            CPU::Reg8 mask = 0;
            for(CPU::Reg8 i = rx_align; i <= 7; i++)
                mask |= (1 << i);

            _spi->put_data(address);
            CPU::Reg8 value = _spi->get_data();
            // Apply mask to current value of
                values[0] and the new data.
            values[0] = (values[index] & ~mask) |
                (value & mask);
```

```
        } else {
            _spi->put_data(address);
            values[index] = _spi->get_data();
        }
        index++;
    }
    _spi->put_data(0);
    values[index] = _spi->get_data();
    _select->set();
    _spi->disable();
}

void MFRC522::write_reg(PCD_Register reg, CPU::Reg8
    value)
{
    _spi->enable();
    _select->clear();
    _spi->put_data((reg & 0x7e)); // MSB == 0 ->
        write // LSB is always 0
    _spi->get_data();
    _spi->put_data(value);
    _spi->get_data();
    _select->set();
    _spi->disable();
}

void MFRC522::write_reg(PCD_Register reg, unsigned
    int count, const void * values)
{
    _spi->enable();
    _select->clear();
    _spi->put_data(reg & 0x7E); // MSB == 0 -> write
        // LSB is always 0
    _spi->get_data();
    for(unsigned int i = 0; i < count; i++) {
        _spi->put_data(reinterpret_cast<const
            char*>(values)[i]);
        _spi->get_data();
    }
    _select->set();
    _spi->disable();
}



__END_SYS

#endif
```

ANEXO B – Aplicação da catraca EPOS

src/catraca/app/emote3_racket.cc

```
#include <utility/ostream.h>
#include <utility/string.h>
#include <rfid_reader.h>
#include <alarm.h>
#include <spi.h>
#include <gpio.h>
#include <timer.h>
#include <pwm.h>
#include <mutex.h>
#include <machine/cortex/i2c.h>
#include <machine.h>
#include <thread.h>
#include <smart_data.h>

using namespace EPOS;

OStream cout;

class LCD_Display
{
private:
    // LCD Commands
    enum {
        CLEAR_DISPLAY = 0x01,
        ENTRY_MODE_SET = 0x04,
        FUNCTION_SET = 0x20,
        DISPLAY_CONTROL = 0x08,
        SET_DRAM_ADDR = 0x80,
    };

    // Function set flags
    enum {
        FOUR_BIT = 0x00,
        ONE_LINE = 0x00,
        TWO_LINES = 0x08,
        DOTS_5X8 = 0x00,
    };

    // Display control flags
    enum {
        CURSOR_OFF = 0x00,
        BLINK_OFF = 0x00,
        BLINK_ON = 0x01,
        CURSOR_ON = 0x02,
        DISPLAY_ON = 0x04,
    };

    // Entry mode flgas
```

```
    enum {
        ENTRY_SHIFT_DECREMENT = 0x00,
        ENTRY_LEFT = 0x02,
    };

    // Backlight control
    enum {
        LCD_BACKLIGHT = 0xFF,
        LCD_NOBACKLIGHT = 0x00,
    };

    // Define COMMAND and DATA LCD Rs (used by send
        method).
    enum {
        COMMAND = 0,
        LCD_DATA = 1,
        FOUR_BITS = 2,
    };

    // This constant defines the numer of
        microseconds it takes for the home
    // and clear commands in the LCD
    enum {
        HOME_CLEAR_EXEC = 2000,
    };

    enum {
        POSITIVE,
        NEGATIVE
    };

    enum {
        RS = 0,
        RW = 1,
        EN = 2,
        BACKLIGHT = 3,
        D4 = 4,
        D5 = 5,
        D6 = 6,
        D7 = 7,
    };

public:
    LCD_Display(I2C * i2c, int address, int lines,
        int cols): _i2c(i2c),
            _address(address), _lines(lines),
                _cols(cols) {
        _backlightStsMask = LCD_NOBACKLIGHT;
        _polarity = POSITIVE;
```

```
// Initialise pin mapping
_Rs = 1 << RS;
_Rw = 1 << RW;
_En = 1 << EN;
_backlight = 1 << BACKLIGHT;
_data_pins[0] = 1 << D4;
_data_pins[1] = 1 << D5;
_data_pins[2] = 1 << D6;
_data_pins[3] = 1 << D7;

_displayfunction = FOUR_BIT | ONE_LINE |
    DOTS_5X8;

_i2c->put(_address, 0);

if(lines > 1)
    _displayfunction |= TWO_LINES;

// See page 45/46 for initialization
    specification! We need at least
// 40ms after power rises above 2.7V before
    sending commands.
Machine::delay(100000);

// Configuring 4-bit mode according to the
    hitachi HD44780 datasheet figure 24, page
// 46. The controller start in 8-bit mode.

// Special case of "Function Set"
send(0x03, FOUR_BITS);
Machine::delay(4500); // wait min 4.1ms

// Second try
send(0x03, FOUR_BITS);
Machine::delay(150); // wait min 100us

// Third go!
send(0x03, FOUR_BITS);
Machine::delay(150); // wait min of 100us

// Finally, set to 4-bit interface
send(0x02, FOUR_BITS);
Machine::delay(150); // wait min of 100us

// Finally, set line number, font size, etc.
command(FUNCTION_SET | _displayfunction);
Machine::delay(60); // Wait more
```

```
    // Turn the display on with no cursor or
        blinking default
    _displaycontrol = DISPLAY_ON | CURSOR_OFF |
        BLINK_OFF;
    display();

    // Clear the LCD
    clear();

    // Initialize to default text direction (for
        romance languages)
    _displaymode = ENTRY_LEFT |
        ENTRY_SHIFT_DECREMENT;

    // Set the entry mode
    command(ENTRY_MODE_SET | _displaymode);

    backlight(true);
}

void print(const char * text) {
//void print(char * text, int n) {
    //while(n--)
        //write(*text++);
    while(*text != '\0')
        write(*text++);

    // XXX: Not sure why but this is necessary
        after avery print
    backlight(true);
}

void cursor(int col, int row) {
    const int row_offsetsDef[] = { 0x00, 0x40,
        0x14, 0x54 };

    if(row >= _lines)
        row = _lines - 1; // Rows start at 0

    command(SET_DRAM_ADDR | (col +
        row_offsetsDef[row]));
}

void backlight(bool on) {
    if(on)
        _i2c->put(_address, _backlight);
    else
        _i2c->put(_address, 0);
}
```

```
    void clear() {
        command(CLEAR_DISPLAY);
        Machine::delay(HOME_CLEAR_EXEC);
    }

private:
    void write(int value) { send(value, LCD_DATA); }

    void display() {
        _displaycontrol |= DISPLAY_ON;
        command(DISPLAY_CONTROL | _displaycontrol);
    }


    void command(int value) { send(value, COMMAND); }

    void send(int value, int mode) {
        // No need to use the delay routines since
            the time taken to write takes
        // longer that what is needed both for
            toggling and enable pin an to
        // execute the command.

        if(mode == FOUR_BITS) {
            write4bits((value & 0x0F), COMMAND);
        } else {
            write4bits((value >> 4), mode);
            write4bits((value & 0x0F), mode);
        }
    }

    void write4bits(int value, int mode) {
        int pinMapValue = 0;

        // Map the value to LCD pin mapping
        for(int i = 0; i < 4; i++) {
            if((value & 0x1) == 1)
                pinMapValue |= _data_pins[i];

            value = (value >> 1);
        }

        // Is it a command or data
        if(mode == LCD_DATA)
            mode = _Rs;

        pinMapValue |= mode | _backlightStsMask;
        pulseEnable(pinMapValue);
```

```
    }

    void pulseEnable(int data) {
        _i2c->put(_address, data | _En);
        Machine::delay(100);
        _i2c->put(_address, data & (~_En));
        Machine::delay(100);
    }

private:
    I2C * _i2c;
    int _address;
    int _backlight;
    int _backlightStsMask;

    int _displaymode;
    int _lines;
    int _cols;
    int _polarity;

    int _displayfunction;
    int _displaycontrol;

    int _En; // LCD expander word for enable pin
    int _Rw; // LCD expander word for R/W pin
    int _Rs; // LCD expander word for Register Select
        pin
    int _data_pins[4]; // LCD data lines
};

char *h2toa(int h, char *str) {
    char *s = str;
    char map[17] = "0123456789ABCDEF";
    unsigned char high = h / 16;
    unsigned char low = h % 16;
    *s++ = map[high];
    *s++ = map[low];
    *s++ = 0;
    return str;
}


enum State {
  STOPPED = 9,
  SPINNING_LEFT = 8,
  SPINNING_RIGHT = 7,
  UNKNOW = 6
};
```

```
enum Permission {
    LOCKED,
    ALLOW_LEFT,
    ALLOW_RIGHT,
    ALLOW_BOTH,
};

#define SENSOR_DEFAULT false
#define LOCKED_VALUE true
#define BUZZER_PERIOD 1500

// Thread * main_thread = 0;

State state = UNKNOW;
int stage = -1;
Permission permission = LOCKED;

GPIO sensor_left('C', 4, GPIO::IN);
GPIO sensor_right('C', 5, GPIO::IN);
GPIO lock('B', 0, GPIO::OUT);
GPIO buzzer('C', 1, GPIO::OUT);

User_Timer timer(0, BUZZER_PERIOD, 0);
PWM buzzer_pwm(&timer, &buzzer, 50);

GPIO in0('B', 5, GPIO::IN);
GPIO in1('C', 6, GPIO::INOUT);
SPI spi(0, 4000000, SPI::FORMAT_MOTO_0, SPI::MASTER,
    500000, 8);
RFID_Sensor rfid_sensor(1, &spi, &in0, &in1);

void updateLock() {
    if (permission == LOCKED) {
        if (state == STOPPED) {
            lock.set(!LOCKED_VALUE);
        }
        else {
            lock.set(LOCKED_VALUE);
        }
    }
    else if (permission == ALLOW_LEFT) {
        if (state == STOPPED || state ==
            SPINNING_LEFT) {
            lock.set(!LOCKED_VALUE);
        }
        else {
            lock.set(LOCKED_VALUE);
        }
    }
```

```
    else if (permission == ALLOW_RIGHT) {
        if (state == STOPPED || state ==
            SPINNING_RIGHT) {
            lock.set(!LOCKED_VALUE);
        }
        else {
            lock.set(LOCKED_VALUE);
        }
    }
    else { // if ALLOW_BOTH
        lock.set(!LOCKED_VALUE);
    }
    if (lock.get() == LOCKED_VALUE){
        buzzer_pwm.enable();
    } else {
        buzzer_pwm.disable();
    }
}

void onFullSpin() {
    if (permission != LOCKED) {
        permission = LOCKED;
        rfid_sensor.enabled(true);
        // if (main_thread) {
            // while (main_thread->state() !=
                Thread::SUSPENDED) {
            //     main_thread->pass();
            // }
            // main_thread->resume();
        // }
    }
}

void updateState(const unsigned int &id) {
    const bool left = sensor_left.get() !=
        SENSOR_DEFAULT;
    const bool right = sensor_right.get() !=
        SENSOR_DEFAULT;

    if (state == STOPPED) {
        if (!left && right) {
            state = SPINNING_RIGHT;
            stage = 1;
        }
        else if (left && !right) {
            state = SPINNING_LEFT;
            stage = 1;
        }
    }
```

```
    else if (state == UNKNOW) {
        if (!left && !right) {
            state = STOPPED;
            stage = 0;
        }
    }
    else if (stage == 1) {
        if (left && right) {
            stage = 2;
        }
        else if (!left && !right) {
            state = STOPPED;
            stage = 0;
        }
    }
    else if (stage == 2) {
        if (left && !right) {
            stage = (state == SPINNING_LEFT) ? 1 : 3;
        }
        else if (!left && right) {
            stage = (state == SPINNING_LEFT) ? 3 : 1;
        }
    }
    else if (stage == 3) {
        if (left && right) {
            stage = 2;
        }
        else if (!left && !right) {
            state = STOPPED;
            stage = 0;
            onFullSpin();
        }
    }
    updateLock();
}

RFID_Sensor::Observed RFID_Sensor::_observed;
RFID_Sensor *
    RFID_Sensor::_dev[RFID_Sensor::MAX_DEVICES];
Switch_Sensor::Observed Switch_Sensor::_observed;
Switch_Sensor *
    Switch_Sensor::_dev[Switch_Sensor::MAX_DEVICES];

I2C i2c(I2C_Common::MASTER, 'B', 7, 'B', 6);
LCD_Display lcd(&i2c, 0x3F, 20, 4);

class Racket_Transform
{
```

```
    static const unsigned int DOOR_OPEN_TIME = 15 *
        1000;
    typedef RFID_Sensor::Data Data;

public:
    Racket_Transform(Permission * p) :
        _door_control_condition(),
        _door_control_thread(&door_control, p,
        &_door_control_condition) { }

    void apply(Switch * o, Switch * button, RFID *
        rfid) {
        cout << "apply(RFID) : " << *rfid << endl;
        unsigned int d = *rfid;
        RFID_Sensor::Data data = d;
        cout << "apply() - data.uid(): " <<
            data.uid() << endl;
        if (*rfid != 0){
            permission = ALLOW_LEFT;
            rfid_sensor.enabled(false);
        }
    }

private:
    static int door_control(Permission * p, Condition
        * condition) {
        cout << "Door_Control" << endl;
        *p = LOCKED;
        while(true) {
            condition->wait();
            *p = ALLOW_LEFT;
        }

        return 0;
    }
    Condition _door_control_condition;
    Thread _door_control_thread;
};

int main()
{
    sensor_left.handler(&updateState, GPIO::BOTH);
    sensor_right.handler(&updateState, GPIO::BOTH);
    state = UNKNOW;
    stage = -1;
    permission = LOCKED;

    Alarm::delay(1000000);
```

```
updateState(0);

cout << "EPOSMote III racket" << endl;

unsigned int count = 0;
unsigned char * uid_sak;
char countstr[17], uidstr[17];

// SPI spi(0, 4000000, SPI::FORMAT_MOTO_0,
    SPI::MASTER, 500000, 8);
// GPIO select('B', 5, GPIO::OUT);
// GPIO reset('C', 6, GPIO::INOUT);
// RFID_Reader reader(&spi, &select, &reset);
// // RFID_Reader::Key
    key("\xff\xff\xff\xff\xff\xff",
    RFID_Reader::Key::TYPE_A);

Switch_Sensor open_door_sensor(0, 'D', 3,
    GPIO::IN, GPIO::UP, GPIO::RISING);
Switch open_door(0, 100000, Switch::PRIVATE);


// Acceleration a(1,0, Acceleration::ADVERTISED);

Switch_Sensor door_control_actuator(2, 'D', 1,
    GPIO::OUT);
Switch door_control(2, 0, Switch::COMMANDED);

Racket_Transform racket_transform(&permission);




RFID rfid(1, 0, RFID::COMMANDED);
Actuator<Switch, Racket_Transform, Switch, RFID>
    door_actuator(&door_control,
    &racket_transform, &open_door, &rfid);


while(true) {
    lcd.clear();
    lcd.cursor(0, 0);

    lcd.print("Contador: ");
    lcd.print(itoa(count, countstr));
    lcd.cursor(0, 1);
    lcd.print("Aproximar cartao");
```

```
        cout << "Place a card next to the reader" <<
            endl;

        while (permission != ALLOW_LEFT) {
            Alarm::delay(100000);
        }
        count++;

        lcd.clear();
        lcd.cursor(0, 0);
        lcd.print("Contador: ");
        lcd.print(itoa(count, countstr));
        lcd.cursor(0, 1);
        lcd.print("Bem-vindo");
        lcd.cursor(0, 3);
        lcd.print("UID:");

        // uid_sak = uid.uid();

        // for (int i = 0; i < uid.size(); i++)
        //     lcd.print(h2toa(uid_sak[i], uidstr));

        // cout << "UID: " << uid << endl;
        cout << "count: " << count << endl;

        permission = ALLOW_LEFT;
        updateLock();
        buzzer_pwm.enable();
        Alarm::delay(100000);
        buzzer_pwm.disable();
        // main_thread->suspend();
        while (permission == ALLOW_LEFT) {
            Alarm::delay(100000);
        }
    }
}
```

ANEXO C – Aplicação do gateway EPOS

src/catraca/app/gateway_testbed.cc

```
// TSTP Gateway to be used with
    tools/eposiotgw/eposiotgw

#include <transducer.h>
#include <smart_data.h>
#include <tstp.h>
#include <utility/ostream.h>
#include <gpio.h>
#include <semaphore.h>

using namespace EPOS;

// TODO
RFID_Sensor::Observed RFID_Sensor::_observed;
RFID_Sensor *
    RFID_Sensor::_dev[RFID_Sensor::MAX_DEVICES];
Switch_Sensor::Observed Switch_Sensor::_observed;
Switch_Sensor *
    Switch_Sensor::_dev[Switch_Sensor::MAX_DEVICES];

IF<Traits<USB>::enabled, USB, UART>::Result io;

typedef Smart_Data_Common::DB_Series DB_Series;
typedef Smart_Data_Common::DB_Record DB_Record;
typedef TSTP::Coordinates Coordinates;
typedef TSTP::Region Region;

const unsigned int INTEREST_PERIOD = 5 * 60 *
    1000000;
const unsigned int INTEREST_EXPIRY = 2 *
    INTEREST_PERIOD;
const unsigned int PRESENCE_EXPIRY = 15 * 60 *
    1000000;

void print(const DB_Record & d)
{
    CPU::int_disable();
    for(unsigned int i = 0; i <
        sizeof(Smart_Data_Common::DB_Record); i++)
        io.put(reinterpret_cast<const char *>(&d)[i]);
    CPU::int_enable();
}

template<typename T>
class Printer: public Smart_Data_Common::Observer
{
public:
    Printer(T * t) : _data(t) {
```

```
        _data->attach(this);
    }
    ~Printer() { _data->detach(this); }

    void update(Smart_Data_Common::Observed * obs) {
        print(_data->db_record());
    }

private:
    T * _data;
};

class Presence_Transform
{
public:
    void apply(Current * destination, Presence *
        source) {
        Presence::Value p = *source;
        if(p)
            *destination = 1;
    }
};

class Door_Transform
{
public:
    void apply(RFID * destination, RFID * source) {
      unsigned int d = *source;
      DB_Record r = source->db_record();
      RFID_Sensor::Data data = d;

      bool authorized = false;
      if(data.authorized()) {
          if(!authorize(data.uid())) {
              data.authorized(false);
              data.open(false);
              *destination = data;
          } else {
              authorized = true;
          }
      } else {
          if(authorize(data.uid())) {
              authorized = true;
              data.authorized(true);
              data.open(true);
              *destination = d;
          }
      }
      if (authorized) {
```

```
        r.value = data.uid();
        print(r);
      }
    }

    bool authorize(const RFID_Reader::UID & u) {
        return u != 0; // Accept everyone
    }
};

int main()
{
    // Get epoch time from serial
    TSTP::Time epoch = 0;
    char c = io.get();
    if(c != 'X') {
        epoch += c - '0';
        c = io.get();
        while(c != 'X') {
            epoch *= 10;
            epoch += c - '0';
            c = io.get();
        }
        TSTP::epoch(epoch);
    }

    Alarm::delay(5000000);


    // Interest center points
    Coordinates center_dummy0(10,5,0);
    Coordinates center_dummy1(10,10,0);
    Coordinates center_dummy2(5,15,0);
    //Coordinates center_dummy3(0,15,0);
    //Coordinates center_dummy4(-5,10,0);
    Coordinates center_dummy5(-5,5,0);
    Coordinates center_outlet0(460-730, -250-80, -15);
    Coordinates center_outlet1(-5-730, -30-80, -15);
    Coordinates center_lights1(305-730, -170-80, 220);
    Coordinates center_lux0(-720,-90, 0);
    Coordinates center_door0(-200,150,200);
    Coordinates center_presence0(-720,-100,0);


    // Regions of interest
    Region region_dummy0(center_dummy0, 0, 0, -1);
    Region region_dummy1(center_dummy1, 0, 0, -1);
    Region region_dummy2(center_dummy2, 0, 0, -1);
```

```
//Region region_dummy3(center_dummy3, 0, 0, -1);
//Region region_dummy4(center_dummy4, 0, 0, -1);
Region region_dummy5(center_dummy5, 0, 0, -1);
Region region_outlet0(center_outlet0, 0, 0, -1);
Region region_outlet1(center_outlet1, 0, 0, -1);
Region region_lights1(center_lights1, 0, 0, -1);
Region region_lux0(center_lux0, 0, 0, -1);
Region region_door0(center_door0, 0, 0, -1);
Region region_presence0(center_presence0, 0, 0,
    -1);


// Data of interest
Current data_dummy0(region_dummy0,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Current data_dummy1(region_dummy1,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Current data_dummy2(region_dummy2,
    INTEREST_EXPIRY, INTEREST_PERIOD);
//Current data_dummy3(region_dummy3,
    INTEREST_EXPIRY, INTEREST_PERIOD);
//Current data_dummy4(region_dummy4,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Current data_dummy5(region_dummy5,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Current data_outlet0(region_outlet0,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Current data_outlet1(region_outlet1,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Current data_lights1(region_lights1,
    INTEREST_EXPIRY, INTEREST_PERIOD);
Luminous_Intensity data_lux0(region_lux0,
    INTEREST_EXPIRY, INTEREST_PERIOD);
RFID data_rfid0(region_door0, INTEREST_EXPIRY, 0);
Switch data_door0(region_door0, INTEREST_EXPIRY,
    INTEREST_PERIOD);
Presence data_presence0(region_presence0,
    PRESENCE_EXPIRY, 0);


// Data transforms and aggregators
Percent_Transform<Luminous_Intensity>
    lux0_transform(0, 1833);
Presence_Transform presence0_transform;
Door_Transform rfid0_transform;


// Event-driven actuators
```

```
    Actuator<RFID, Door_Transform, RFID>
        door_actuator(&data_rfid0, &rfid0_transform,
        &data_rfid0);
    Actuator<Current, Presence_Transform, Presence>
        presence0_actuator(&data_lights1,
        &presence0_transform, &data_presence0);


    // Printers to output received messages to serial
    Printer<Current> p0(&data_dummy0);
    Printer<Current> p1(&data_dummy1);
    Printer<Current> p2(&data_dummy2);
    Printer<Current> p5(&data_dummy5);
    Printer<Current> p6(&data_outlet0);
    Printer<Current> p7(&data_outlet1);
    Printer<Current> p8(&data_lights1);
    Printer<Luminous_Intensity> p9(&data_lux0);
    // Door_Transform already prints RFID messages
    Printer<Switch> p10(&data_door0);
    Printer<Presence> p11(&data_presence0);

    Alarm::delay(INTEREST_EXPIRY);

    // Time-triggered actuators
    while(true) {
        Alarm::delay(PRESENCE_EXPIRY);
        if(data_presence0.expired())
            data_lights1 = 0;
        else
            lux0_transform.apply(&data_lights1,
                &data_lux0);

        data_door0 = 0;
    }

    return 0;
}
```

ANEXO D – Adições no transducer do EPOS para transmitir dados de RFID via TSTP

src/catraca/include/machine/cortex/transducer.h.add

```
class RFID_Sensor
{
    static const unsigned int MAX_DEVICES = 8;

public:
    static const unsigned int UNIT =
        CUSTOM_UNITS::UNIT_RFID;
    static const unsigned int NUM = TSTP::Unit::I32;
    static const int ERROR = 0; // Unknown

    static const bool INTERRUPT = true;
    static const bool POLLING = false;

    typedef RFID_Reader::Observer Observer;
    typedef RFID_Reader::Observed Observed;

public:
    class Data : private RFID_Reader::UID
    {
    public:
        enum Code {
            DENIED    = 0,
            OPEN_NOW  = 1 << 0,
            AUTHORIZED = 1 << 1,
        };

        Data() : _code(0) {}
        Data(unsigned int v) : RFID_Reader::UID(v >>
            8), _code(v) {}
        Data(unsigned long long v) :
            RFID_Reader::UID(v >> 8), _code(v) {}
        Data(const RFID_Reader::UID & u) :
            RFID_Reader::UID(u) {}
        Data(const RFID_Reader::UID & u, unsigned
            char code) : RFID_Reader::UID(u),
            _code(code) {}

        operator unsigned int() const {
            unsigned int i =
                RFID_Reader::UID::operator unsigned
                int();
            return (i << 8) | _code;
        }

        // operator unsigned long long() const {
        //    unsigned long long i =
            RFID_Reader::UID::operator unsigned long
            long();
```

```
    //     return (i << 8) | _code;
    // }

    const RFID_Reader::UID & uid() const { return
        *this; }

    unsigned char code() const { return _code; }
    void code(unsigned char c) { _code = c; }

    bool authorized() const { return _code &
        AUTHORIZED; }
    void authorized(bool a) { if(a) _code |=
        AUTHORIZED; else _code &= ~AUTHORIZED; }
    bool open() const { return _code & OPEN_NOW; }
    void open(bool a) { if(a) _code |= OPEN_NOW;
        else _code &= ~OPEN_NOW; }

private:
    unsigned char _code;
}__attribute__((packed));

RFID_Sensor(unsigned int dev, SPI * reader_spi,
    GPIO * reader_gpio0, GPIO * reader_gpio1)
: _device(reader_spi, reader_gpio0, reader_gpio1)
{
    assert(dev < MAX_DEVICES);
    _dev[dev] = this;
}

~RFID_Sensor() {
    for(unsigned int i = 0; i < MAX_DEVICES; i++)
        if(_dev[i] == this)
            _dev[i] = 0;
}

void enabled(bool b) {
    _device.enabled(b);
}

static void sense(unsigned int dev,
    Smart_Data<RFID_Sensor> * data) {
    assert(dev < MAX_DEVICES);
    if(_dev[dev])
        _dev[dev]->sense(data);
}

static void actuate(unsigned int dev,
    Smart_Data<RFID_Sensor> * data, const
    Smart_Data<RFID_Sensor>::Value & command) {
```

```
        data->_value = command;
    }

    static void attach(Observer * obs) {
        RFID_Reader::attach(obs); }
    static void detach(Observer * obs) {
        RFID_Reader::detach(obs); }

private:
    void sense(Smart_Data<RFID_Sensor> * data) {
        if(_device.ready_to_get()) {
            EPOS::S::MIFARE::UID uid = _device.get();
            Data id_with_code = uid;
            _device.halt(uid);
            data->_value = id_with_code;
        } else
            data->_value = 0;
    }

private:
    RFID_Reader _device;
    static Observed _observed;
    static RFID_Sensor * _dev[MAX_DEVICES];
};

typedef Smart_Data<RFID_Sensor> RFID;
```

ANEXO E – Aplicação da porta EPOS

src/porta/app/door_labsec.cc

```
#include <utility/ostream.h>
#include <utility/string.h>
#include <alarm.h>
#include <gpio.h>
#include <riffs.h>
#include <clock.h>
#include <chronometer.h>
#include <condition.h>

#include <rfid_reader.h>
#include <spi.h>


using namespace EPOS;


OStream cout;
GPIO lock('B', 0, GPIO::OUT);

class FileRecord {
public:
    FileRecord(){
        for(int i = 0; i < 64; i++)
            name[i] = 'A';
    }
    unsigned int uid;
    bool status;
    char name[64];
};


char *h2toa(int h, char *str) {
    char *s = str;
    char map[17] = "0123456789ABCDEF";
    unsigned char high = h / 16;
    unsigned char low = h % 16;
    *s++ = map[high];
    *s++ = map[low];
    *s++ = 0;
    return str;
}

#define OPEN_DOOR_TIME 5000000
#define BUZZER_TIME 100000
#define POLLING_REBOOT 10*60*5 /* 5 minutes */

GPIO buzzer('C', 3, GPIO::OUT);
GPIO in0('B', 5, GPIO::IN);
```

```
GPIO in1('C', 6, GPIO::INOUT);
SPI spi(0, 4000000, SPI::FORMAT_MOTO_0, SPI::MASTER,
    500000, 8);
RFID_Reader reader(&spi, &in0, &in1);

Condition unlockCondition;

int lockControl() {
    lock.set(true);
    buzzer.set(true);
    while(true) {
        unlockCondition.wait();
        cout << endl << endl << "ABRE PORTA" << endl
            << endl << endl;
        lock.set(false);
        buzzer.set(false);
        Delay(BUZZER_TIME);
        buzzer.set(true);
        Delay(OPEN_DOOR_TIME);
        cout << endl << endl << "FECHA PORTA" << endl
            << endl << endl;
        lock.set(true);
        Delay(10000);
        reader.reboot();

    }
    return 0;
}


void unlockDoor(const unsigned int &id) {
    cout << "Interrupt" << endl;
    unlockCondition.signal();
}

FileSystem fs;
File *file;

void registerUid(char * uid) {
    // char append_str[8] = "";
    // strcat(append_str, uid);
    // file = fs.open("/authorized_uid_list");
    file->seek(file->size());
    file->append(uid, 8);
}

bool isUidRegistered(char * cardUid) {
    char c = '/';
    uint32_t fileIndex = 0;
```

```
    while (true) {
        char lastUid[9];
        for (int i = 0; i < 8; i++) {
            if (fileIndex > file->size()) {
                return false;
            }
            file->seek(fileIndex++);
            file->read(&c, 1);
            lastUid[i] = c;
        }

        lastUid[9] = '\0';
        char append_str[9] = "";
        strcat(append_str, cardUid);
        cardUid[9] = '\0';
        cout << "ONFILE: " << lastUid << endl;
        cout << "CARD:  " << append_str << endl;
        if (strncmp(lastUid, append_str, 8) == 0) {
            return true;
        }
    }
}

int main(){
    buzzer.set(true);
    Thread unlockDoorThread(lockControl);
    lock.set(false);
    // Delay(1000000);

    int polling_count = 0;

    //fs.format();
    fs.mount();
    //  File *dir = fs.create_dir("test_directory");

    file = fs.open("/authorized_uid_list");
    if(file == 0){
        file = fs.create("authorized_uid_list", 1);
        file = fs.open("/authorized_uid_list");
        if (file == 0) {
            cout << "COULD NOT OPEN OR CREATE UID
                LIST" << endl;
            return 0;
        }
    }

    GPIO in2('A', 0, GPIO::IN);
    in2.handler(&unlockDoor, GPIO::FALLING);
```

```
/*
    Utilizar fs.format quando utilizar o mote
        pela primeira vez com o riffs
    ou para limapr a memoria. Comentar o resto.
*/
//fs.format();

Delay(100000);
cout << "Teste Porta" << endl;


unsigned char * str2;
char uidstr[8];


File *file;
while(1){
    reader.reboot();
    cout << "Passe um cartao" << endl;

    RFID_Reader::UID uid;

    while (uid.size() == 0) {
        while (!reader.ready_to_get()) {
            polling_count++;
            if (polling_count >= POLLING_REBOOT) {
                cout << "rebooting reader" << endl;
                reader.reboot();
            }
            Delay(1000);
            cout << "Sem Cartao" << endl;
        }

        uid = reader.get();
        cout << "Com Cartao: " << uid << endl;
    }
    reader.halt(uid);

    char struid[8] = "";

    str2 = uid.uid();
    for (int i = 0; i < uid.size(); i++)
        strcat(struid, h2toa(str2[i], uidstr));

    if (strcmp(struid, "0240D535") == 0){
        cout << "Registre um cartao" << endl;
        buzzer.set(false);
```

```
        Delay(BUZZER_TIME);
        buzzer.set(true);
        Delay(BUZZER_TIME);
        buzzer.set(false);
        Delay(BUZZER_TIME);
        buzzer.set(true);
        uid = RFID_Reader::UID();
        while (uid.size() == 0) {
            while (!reader.ready_to_get()) {
                polling_count++;
                if (polling_count >=
                    POLLING_REBOOT) {
                    cout << "rebooting reader" <<
                        endl;
                    reader.reboot();
                }
                Delay(1000);
            }

            uid = reader.get();
        }
        reader.halt(uid);

        str2 = uid.uid();
        char struid2[8] = "";
        for (int i = 0; i < uid.size(); i++)
            strcat(struid2, h2toa(str2[i],
                uidstr));

        cout << struid2 << endl;

        if (isUidRegistered(struid2)) {
            cout << "Cartao ja registrado" << endl;
        } else {
            registerUid(struid2);
            cout << "Cartao registrado" << endl;
            unlockDoor(1);
            Delay(1000000);
            return 0;
        }

    } else {
        if(isUidRegistered(struid)){
            unlockDoor(1);
        } else {
            cout << "Acesso Negado" << endl;
        }
    }
```

```
        Delay(100000);
    }
    return 0;
}
```

ANEXO F – Aplicação Web para gerenciamento do controle de acesso

src/server/server.js

```
const express = require('express')
const session = require('express-session')
const flash = require('express-flash')
const crypto = require('crypto')
const bodyParser = require('body-parser')
const config = require('./config.json')
const User = require('./controllers/user.js')
const Door = require('./controllers/door.js')(User)

const app = express()
const jsonParser = bodyParser.json()
const urlencodedParser = bodyParser.urlencoded({
    extended: false })

app.use(session({
  secret: crypto.randomBytes(64).toString('hex'),
  resave: true,
  saveUninitialized: true
}))
app.use(flash())
app.use(express.static('public'));
app.set('views', './views')
app.set('view engine', 'pug');

app.get('/', (req, res) => {
  if (req.session.role == undefined) {
    res.redirect('/login')
  } else {
    User.getAuthorizedDoors(req.session.login,
        (doors) => {
      res.render('manage-door-list', {session:
          req.session, doors: doors})
    })
  }
})

app.get('/logout', (req, res) => {
  User.logout(req, res)
})

app.get('/accounts', (req, res) => {
  User.renderList(req, res)
})

app.get('/accounts/:operation/:id?/:doorAction?/:doorId?',
    (req, res) => {
  User.renderEditForm(req, res)
```

```
})

app.post('/accounts/:operation/:id?/:doorAction?/:doorId?',
    urlencodedParser, (req, res) => {
  User.accountPost(req, res)
})

app.get('/door/:id/:op', urlencodedParser, (req,
    res) => {
  Door.renderManaggement(req, res)
})

app.post('/door/:id/:op', urlencodedParser, (req,
    res) => {
  Door.managementPost(req, res)
})

app.get('/login',urlencodedParser, (req, res) => {
  User.renderLoginForm(req, res)
})

app.post('/login', urlencodedParser, (req, res) => {
  User.login(req, res)
})

app.all('/favicon.ico', (req, res) => {
  res.send('')
})

// 404 - must be last
app.all('*', function(req, res){
  req.flash('danger', '404 - ' + req.method + ' @ ' +
      req.url)
  res.status(404).redirect('/');
});

app.listen(config.port, () => {
  console.log('app listening on port ' + config.port
      + '!')
})
```

src/server/config.json

```
{
  "port": 8080,
  "db": {
    "user": "postgres",
    "password": "postgres",
```

```
    "database": "door_server",
    "server": "localhost",
    "port": 5432
  },
  "ws" : {
    "url": "https://ws.ufsc.br/CadastroCartaoService
        /dadosPessoasCartaoByIdentificacao/",
    "user": "labsecws",
    "password": "awEq5EMAJ7yeFXkE"
  }
}
```

src/server/package.json

```
{
  "name": "door_server",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
        exit 1"
  },
  "author": "Matteus Legat",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.2",
    "crypto": "^1.0.1",
    "express": "^4.16.3",
    "express-flash": "0.0.2",
    "express-session": "^1.15.6",
    "pg": "^6.1.0",
    "pug": "^2.0.3",
    "request": "^2.85.0"
  }
}
```

src/server/controllers/db.js

```
const dbConfig = require('../config.json').db
const pg = require ('pg')

const connection =
    `postgres://${dbConfig.user}:${dbConfig.password}@`
    +
    `${dbConfig.server}:${dbConfig.port}/${dbConfig.database}`

const client = new pg.Client(connection)
```

```
client.connect()

module.exports = client
```

src/server/controllers/door.js

```
const DB = require('./db.js')
const WS = require('./ws.js')

class DoorController {

  constructor(UserController) {
    this.user = UserController
  }

  managementPost(req, res) {
    if (req.session.role == undefined) {
      res.redirect('/login')
    } else {
      this.user.getAuthorizedDoors(req.session.login,
          (doors) => {
        const door = doors.filter(door => door.id ==
            req.params.id)[0]
        if(!door) {
          req.flash('danger', 'A porta não existe ou
              você não possui permissões sobre ela.')
          res.redirect('/')
        } else {
          if (req.params.op == 'manage') {
            if (req.body.delete) {
              this.removeCard(req.body.delete,
                  req.params.id, (err) => {
                if (err) {
                  console.log(err)
                  req.flash('danger', 'Erro ao remover
                      cartão.')
                  res.redirect(req.url)
                } else {
                  req.flash('success', 'Cartão
                      removido.')
                  res.redirect(req.url)
                }
              })
            } else {
              req.flash('danger', 'Ação Incompleta')
              res.redirect('/')
            }
          } else if (req.params.op == 'manage-add') {
```

```
    const oldUid = req.body.uid
    const newUid = oldUid
    const name = req.body.name
    const doorId = req.params.id
    const ufscId = req.body.ufsc_id
    const constraints = null
    this.updateCard(oldUid, newUid, ufscId,
        name, doorId, constraints, (err) => {
      if (err) {
        console.log(err)
        req.flash('danger', 'Erro ao cadastrar
            cartão.')
        res.redirect(req.url)
      } else {
        req.flash('success', 'Cartão
            cadastrado/atualizado.')
        res.redirect('/door/'+doorId+'/manage')
      }
    })
  } else if (req.params.op == 'search') {
    WS.search(req, res, (error, response) => {
      if (error) {
        req.flash('danger', 'Erro no WS: ' +
            error)
        res.redirect('/door/' + req.params.id
            + '/manage-add')
      } else {
        const card = response[0]
        if (card) {
          res.render('manage-add-card',
              {session: req.session, card:
              card, door: door})
        } else {
          req.flash('danger', 'Cadastrado não
              encontrado.')
          res.redirect('/door/' +
              req.params.id + '/manage-add')
        }
      }
    })
  } else if (req.params.op == 'config') {
    if (req.session.role != 0) {
      req.flash('danger', 'Não Autorizado.')
      res.redirect('/')
    } else {
      this._updateDoor(req, res)
    }
  } else {
    req.flash('danger', 'Operação Inválida')
```

```
            res.redirect('/')
          }
        }
      })
    }
  }

  renderManaggement(req, res) {
    if (req.session.role == undefined) {
      res.redirect('/login')
    } else {
      this.user.getAuthorizedDoors(req.session.login,
          (doors) => {
        const door = doors.filter(door => door.id ==
            req.params.id)[0]
        if(!door) {
          req.flash('danger', 'A porta não existe ou
              você não possui permissões sobre ela.')
          res.redirect('/')
        } else {
          if (req.params.op == 'manage') {
            this.getAuthorizedCards(door.id, (cards)
                => {
              res.render('door-card-list', {session:
                  req.session, cards: cards, door:
                  door})
            })
          } else if (req.params.op == 'manage-add') {
            res.render('manage-add-card', {session:
                req.session, card: null, door: door})
          } else if (req.params.op == 'log') {
            this.getLog(req.params.id, (log) => {
              res.render('door-log', {session:
                  req.session, log: log, door: door})
            })
          } else if (req.params.op == 'config') {
            if (req.session.role != 0) {
              req.flash('danger', 'Não Autorizado.')
              res.redirect('/')
            } else {
              res.render('door-config', {session:
                  req.session, door: door})
            }
          } else {
            req.flash('danger', 'Operação Inválida.')
            res.redirect('/')
          }
        }
      })
```

```
  }
}

renderEditDoors(req, res) {
  this.user.getUserByLogin(req.params.id, (user) =>
      {
    if (!user || user.role != 1) {
      req.flash('danger', 'Operacão Inválida.')
      res.redirect('/accounts')
    } else if (req.params.doorAction) {
      if (req.params.doorAction == 'delete') {
        if (!req.params.doorId) {
          req.flash('danger', 'Porta não
              especificada.')
          res.redirect('/accounts/doors/'+req.params.id)
        } else {
          return this._removeManager(user,
              req.params.doorId, req, res)
        }
      } else if (req.params.doorAction == 'add') {
        if (req.params.doorId) {
          return this._addManager(user,
              req.params.doorId, req, res)
        } else {
          this.user.getAuthorizedDoors(user.login,
              (authorized) => {
            const authorizedIds =
                authorized.map(door => door.door)
            this.getAll((all) => {
              const notAuthorized = all.filter(door
                  =>
                  !authorizedIds.includes(door.id))
              res.render('user-door-list', {session:
                  req.session, doors: notAuthorized,
                  user: user, add: true})
            })
          })
        }
      }
    } else {
      this.user.getAuthorizedDoors(user.login,
          (list) => {
        res.render('user-door-list', {session:
            req.session, doors: list, user: user})
      })
    }
  })
}
```

```
getAuthorizedCards(doorId, callback) {
  const results = [];
  const query = DB.query(
    `SELECT card_uid, id_pessoa, name FROM
        "DoorUpdates"
     WHERE upper(action) = 'ADD' AND door = $1
     ORDER BY name`,
    [doorId]
  );
  query.on('row', (row) => {
    results.push(row);
  });
  query.on('end', function() {
    return callback(results);
  });
}

removeCard(card, doorId, callback) {
  const query = DB.query('BEGIN')
    .then(() => {
      return DB.query('DELETE FROM "DoorUpdates"
          WHERE upper(card_uid) = upper($1) AND
          door = $2', [card, doorId])
    })
    .then(() => {
      return DB.query(`INSERT INTO "DoorUpdates"
          (door, card_uid, action) VALUES ($2, $1,
          'REMOVE')`, [card, doorId])
    })
    .then(() => {
      return DB.query('COMMIT')
    })
    .then(() => {
      callback()
    })
    .catch((err) => {
      callback(err)
    })
}

updateCard(oldUid, newUid, ufscId, name, doorId,
    constraints, callback) {
  const query = DB.query('BEGIN')
    .then(() => {
      return DB.query('DELETE FROM "DoorUpdates"
          WHERE upper(card_uid) = upper($1) AND
          door = $2', [oldUid, doorId])
    })
    .then(() => {
```

```
      return DB.query(`INSERT INTO "DoorUpdates"
          (door, card_uid, action) VALUES ($1, $2,
          'REMOVE')`, [doorId, oldUid])
    })
    .then(() => {
      return DB.query(
        `INSERT INTO "DoorUpdates" (door, card_uid,
            id_pessoa, name, action, constraints)
         VALUES ($1, $2, $3, $4, 'ADD', $5)`,
        [doorId, newUid, ufscId, name, constraints]
      )
    })
    .then(() => {
      return DB.query('COMMIT')
    })
    .then(() => {
      callback()
    })
    .catch((err) => {
      callback(err)
    })
}

getAll(callback) {
  const results = [];
  const query = DB.query('SELECT id, mac, name,
      salt FROM "Doors"', []);
  query.on('row', (row) => {
    results.push(row);
  });
  query.on('end', function() {
    return callback(results);
  });
}

getLog(doorId, callback) {
  const results = [];
  const query = DB.query(
    `SELECT log.card_uid, log.timestamp,
        log.action, updates.name FROM "DoorLogs" AS
        log
     LEFT JOIN "DoorUpdates" AS updates
      ON log.card_uid = updates.card_uid
      AND log.door = updates.door
      AND upper(updates.action) = 'ADD'
     WHERE log.door = $1
     ORDER BY log.timestamp DESC`,
    [doorId]
  );
```

```
    query.on('row', (row) => {
      results.push(row);
    });
    query.on('end', function() {
      return callback(results);
    });
  }

  getAuthorized(login, callback) {
    const results = [];
    const query = DB.query(
      `SELECT door AS id, mac, name, salt
       FROM "Doors" AS d
       JOIN "DoorManagers" ON (d.id = door)
       WHERE manager = $1
      `,
      [login]
    );
    query.on('row', (row) => {
      results.push(row);
    });
    query.on('end', function() {
      return callback(results);
    });
  }

  _addManager(user, door, req, res) {
    const query = DB.query(
      'INSERT INTO "DoorManagers"(door, manager)
          VALUES($1, $2)',
      [door, user.id],
      (err, dbRes) => {
        if (err) {
          console.log(err.stack)
          req.flash('danger', 'Erro ao cadastrar
              gerência.')
        } else {
          req.flash('success', 'Gerência de porta
              adicionada ao usuário ' + user.login +
              '.')
        }
        res.redirect('/accounts/doors/' + user.login)
      }
    )
  }

  _removeManager(user, door, req, res) {
    const query = DB.query(
      `DELETE FROM "DoorManagers"
```

```
      WHERE door = $1 AND manager = $2‘,
    [door, user.id],
    (err, dbRes) => {
      if (err) {
        console.log(err.stack)
        req.flash(’danger’, ’Erro ao remover
            gerência.’)
      } else {
        req.flash(’success’, ’Gerência de porta
            removida do usuário ’ + user.login + ’.’)
      }
      res.redirect(’/accounts/doors/’ + user.login)
    }
  )
}

_addDoor(mac, callback) {
  const query = DB.query(
    ‘INSERT INTO "Doors" (mac)
     VALUES (upper($1))‘,
    [newMac, newName, id],
    (err, dbRes) => {
      callback(err)
    }
  )
}

_updateDoor(req, res) {
  const id = req.params.id
  const newName = req.body.name
  const newMac = req.body.mac
  const query = DB.query(
    ‘UPDATE "Doors"
     SET mac = upper($1), name = $2
     WHERE id = $3;‘,
    [newMac, newName, id],
    (err, dbRes) => {
      if (err) {
        console.log(err.stack)
        req.flash(’danger’, ’Erro ao alterar
            porta.’)
      } else {
        req.flash(’success’, ’Porta alterada com
            sucesso.’)
      }
      res.redirect(’/’)
    }
  )
}
```

```
}

module.exports = (UserController) => {
  const instance = new DoorController(UserController)
  Object.freeze(instance)
  return instance
}
```

src/server/controllers/user.js

```
const DB = require('./db.js')
const crypto = require('crypto')

// USER ROLES:
// 0 - MASTER - Manage any door and users
// 1 - MANAGER - Manage only doors authorized by
    master

class UserController {

  constructor() {
    this.door = require('./door.js')(this)
  }

  renderList(req, res) {
    if (req.session.role != 0) {
      res.redirect('/accounts/edit/' +
          req.session.login)
    } else {
      this._getLoginList((list) => {
        res.render('account-list', {session:
            req.session, accounts: list})
      })
    }
  }

  renderEditForm(req, res) {
    this._checkAccountAuthorization(req, res, () => {
      if (req.params.operation == 'doors') {
        this.door.renderEditDoors(req, res)
      } else if (req.params.operation == 'new') {
        res.render('account', {session: req.session,
            account: {login: '', new: true}})
      } else {
        this._getData(req.params.id, req, res, (data)
            => {
          res.render('account', {session:
              req.session, account: data})
```

```
      })
    }
  })
}

renderLoginForm(req, res) {
  if (req.session.login) {
    res.redirect('/')
  } else {
    res.render('login', {session: req.session})
  }
}

accountPost(req, res) {
  this._checkAccountAuthorization(req, res, () => {
    const operation = req.params.operation
    const passLength = req.body.password.length
    if (req.body.username.length < 4) {
      req.flash('danger', 'O nome de usuário deve
          possuir ao menos 4 caracteres.')
      return res.redirect(req.url)
    } else if (!/^\w+$/.test(req.body.username)) {
      req.flash('danger', 'O nome de usuário deve
          possuir somente letras, números e
          underlines.')
      return res.redirect(req.url)
    } else if (passLength < 8 && !(passLength == 0
        && operation == 'edit')) {
      req.flash('danger', 'A senha deve possuir ao
          menos 8 caracteres.')
      return res.redirect(req.url)
    } else {
      if (operation == 'new') {
        return this._addUser(req, res)
      } else if (operation == 'edit') {
        this._updateUser(req, res)
      } else {
        req.flash('danger', 'POST inválido.')
        res.redirect('/')
      }
    }
  })
}

login(req, res) {
  const login = req.body.username || ""
  const password = req.body.password || ""
  this.getUserByLogin(login, (user) => {
    if (user) {
```

```
      if (this._saltedHash(user.salt, password) ==
          user.password) {
        if (user.role < 0) {
          req.flash('danger', 'Conta Desativada!')
          return res.redirect('/login')
        }
        req.session.login = user.login
        req.session.role = user.role
        res.redirect('/')
        return
      }
    }
    req.flash('danger', 'Login Inválido!')
    res.redirect('/')
  })
}

logout(req, res) {
  req.session.destroy(() => {
    res.redirect('/login')
  })
}

getAuthorizedDoors(login, callback) {
  this.getUserByLogin(login, (user) => {
    if (!user || user.role == -1) {
      callback([])
    } else if (user.role == 0) {
      this.door.getAll(callback)
    } else {
      this.door.getAuthorized(user.id, callback)
    }
  })
}

_getData(login, req, res, callback) {
  this.getUserByLogin(login, (user) => {
    if (user) {
      callback(user)
    } else {
      req.flash('danger', 'Usuário inexistente.')
      res.redirect('/accounts')
    }
  })
}

_updateUser(req, res) {
  const login = req.params.id
  if (req.body.role != undefined &&
```

```
    (req.session.role != 0 || req.session.login
    == login)) {
  delete req.body.role
}
this.getUserByLogin(login, (user) => {
  if (user) {
    const newLogin = req.body.username
    this.getUserByLogin(newLogin, (newUser) => {
      if (newUser && newUser.login != login) {
        req.flash('danger', 'Nome de usuário já
            cadastrado.')
        res.redirect(req.url)
      } else {
        let newRole = req.body.role
        let newPassword = req.body.password
        let newSalt = ''
        if (newPassword.length > 0) {
          newSalt =
              crypto.randomBytes(64).toString('hex')
          newPassword = this._saltedHash(newSalt,
              newPassword)
        } else {
          newSalt = user.salt
          newPassword = user.password
        }
        if (newRole == undefined) {
          newRole = user.role
        }
        const query = DB.query(
          `UPDATE "Users"
           SET login = lower($1), password = $2,
               salt = $3, role = $4::integer
           WHERE login = $5`,
          [newLogin, newPassword, newSalt,
              newRole, login],
          (err, dbRes) => {
            if (err) {
              console.log(err.stack)
              req.flash('danger', 'Erro ao alterar
                  usuário.')
            } else {
              req.flash('success', 'Usuário ' +
                  login + ' alterado com sucesso.')
              if (login == req.session.login) {
                req.session.login = newLogin
              }
            }
            res.redirect('/accounts')
          }
```

```
          )
        }
      })
    } else {
      req.flash('danger', 'Usuário inexistente.')
      res.redirect('/accounts')
    }
  })
}

_addUser(req, res) {
  if (!req.session.login || req.session.role != 0) {
    return res.redirect('/')
  }

  const role = req.body.role
  const login = req.body.username
  this.getUserByLogin(login, (user) => {
    if (user) {
      req.flash('danger', 'Nome de usuário já
          cadastrado.')
      return res.redirect(req.url)
    } else {
      const salt =
          crypto.randomBytes(64).toString('hex')
      const saltedHash = this._saltedHash(salt,
          req.body.password)
      const query = DB.query(
        'INSERT INTO "Users"(login, password, salt,
            role) VALUES(lower($1), $2, $3, $4)',
        [login, saltedHash, salt, role],
        (err, dbRes) => {
          if (err) {
            console.log(err.stack)
            req.flash('danger', 'Erro ao cadastrar
                usuário.')
          } else {
            req.flash('success', 'Usuário ' + login
                + ' cadastrado com sucesso.')
          }
          res.redirect('/accounts')
        }
      )
    }
  })
}

_saltedHash(salt, secret) {
  return crypto.createHash('sha512').update(salt +
```

```
        secret).digest('base64');
  }

  getUserByLogin(login, callback) {
    const results = [];
    const query = DB.query('SELECT * FROM "Users"
        WHERE upper(login) = upper($1)', [login]);
    query.on('row', (row) => {
      results.push(row);
    });
    query.on('end', () => {
      return callback(results[0]);
    });
  }

  _getLoginList(callback) {
    const results = [];
    const query = DB.query('SELECT login, role FROM
        "Users" ORDER BY login', []);
    query.on('row', (row) => {
      results.push(row);
    });
    query.on('end', () => {
      return callback(results);
    });
  }

  _checkAccountAuthorization(req, res, callback) {
    if (!['new', 'edit',
        'doors'].includes(req.params.operation)) {
      req.flash('danger', 'Operação Inválida!')
      res.redirect('/')
    } else if (req.session.role == undefined) {
      res.redirect('/login')
    } else if (
      req.session.role != 0 &&
      (req.params.operation != 'edit' ||
          req.params.id != req.session.login)
    ) {
      res.redirect('/accounts/edit/' +
          req.session.login)
    } else {
      callback()
    }
  }
}

const instance = new UserController()
Object.freeze(instance)
```

```
module.exports = instance
```

src/server/controllers/ws.js

```
const wsConfig = require('../config.json').ws
const request = require ('request')

class WebService {
  search(req, res, callback) {
    request.get({
      url: wsConfig.url + req.body.search,
      json: true,
      headers: {
        "Authorization": "Basic " + new
            Buffer(wsConfig.user + ":" +
            wsConfig.password).toString("base64")
      }
    }, (error, response, body) => {
      if (!error && typeof body != 'object') {
        error = 'Not a JSON'
      }
      if (!error && body.error_description) {
        error = body.error_description
      } else if (!error && body.error) {
        error = body.error_description
      }
      callback(error, body)
    })
  }
}

const instance = new WebService()
Object.freeze(instance)
module.exports = instance
```

src/server/db/sql/create.sql

```
CREATE TABLE "Users" (
  "id" serial primary key not null,
  "login" varchar(255) unique not null,
  "password" varchar(255) not null,
  "salt" varchar(255) not null,
  "role" int not null
);

CREATE TABLE "Doors" (
  "id" serial primary key not null,
  "mac" varchar(255) unique not null,
```

```
  "name" varchar(255) null,
  "password" varchar(255) null,
  "salt" varchar(255) null
);

CREATE TABLE "DoorUpdates" (
  "id" serial primary key not null,
  "door" integer not null,
  "action" varchar(255) not null,
  "card_uid" varchar(255) not null,
  "id_pessoa" varchar(255) null,
  "name" varchar(255) null,
  "constraints" varchar(255) null
);

CREATE TABLE "DoorLogs" (
  "id" serial primary key not null,
  "door" integer not null,
  "card_uid" varchar(255) not null,
  "timestamp" timestamp not null,
  "action" varchar(255) not null
);

CREATE TABLE "DoorManagers" (
  "door" integer not null,
  "manager" integer not null,
  PRIMARY KEY(door, manager)
);

ALTER TABLE "DoorUpdates"
  ADD CONSTRAINT fk_door
  FOREIGN KEY (door)
  REFERENCES "Doors"(id);

ALTER TABLE "DoorLogs"
  ADD CONSTRAINT fk_door
  FOREIGN KEY (door)
  REFERENCES "Doors"(id);

ALTER TABLE "DoorManagers"
  ADD CONSTRAINT fk_door
  FOREIGN KEY (door)
  REFERENCES "Doors"(id);

ALTER TABLE "DoorManagers"
  ADD CONSTRAINT fk_manager
  FOREIGN KEY (manager)
  REFERENCES "Users"(id);
```

```
/* USER admin PASS admin */
INSERT INTO "Users"(login, password, salt, role)
    VALUES ('admin',
    'flcluikTfN2QrVSQ8q2T7OTuGfclzbL07uauROVLx9eX+0g1d
    ByF4Ofqhh7B8IsgEhlGXLc6c3A9wYqBD9PTOA==',
    'e7f45c4e68a77552c4fcb2beab7b961610a577c65bb2ede96
    cae3537c9ae45fd325732ffd772b8c61afc6051c79cebedabc
    943a24dc248ec696e2a21e0a91759', 0);
```

src/server/public/css/style.css

```
body {
  background-color: #f5f5f5;
  font-family: arial, sans-serif;
}

.topbar {
  padding: 1em;
  background-color: #339;
  margin-bottom: 2em;
}

div.title {
  font-size: 15pt;
  margin-bottom: 1em;
}

.topbar .title a {
  font-size: 18pt;
  color: #fff;
  text-decoration: none;
}

.topbar .buttons {
  position: absolute;
  right: 0;
  margin-right: 1em;
}

.topbar .buttons .btn {
  margin-left: 0.5em;
}

.flash {
  margin: 0 5em 1em 5em;
}

td.td-btn {
```

```
  width: 1px;
  white-space: nowrap;
  text-align: center;
}

.topbar .btn-secondary {
  opacity: 0.9;
}

.topbar .btn-danger {
  background-color: #833;
  border-color: #833;
}

.topbar .btn-danger:hover {
  background-color: #a33;
  border-color: #a33;
}

.container {
  position: relative;
}

.cr {
  position: absolute;
  right: 0;
}
```

src/server/views/account.pug

```
extends template.pug

block content
  script(type='text/javascript')
    include form-validate.js

  if account.new
    .title Cadastrar novo usuário
  else
    .title Atualizar dados do usuário:
        #{account.login}

  form(method='post', onsubmit='return validate()')
    .form-group
      label(for='username') Usuário
      input.form-control#username(type='text',
          name='username', value=account.login).
    .form-group
```

```
      label(for='password') Senha
      input.form-control#password(type='password',
          name='password', placeholder=account.new ?
          '' : 'Inalterado').
    .form-group
      label(for='password-confirm') Confirmação de
          Senha
      input.form-control#password-confirm(type='password',
          name='password-confirm',
          placeholder=account.new ? '' :
          'Inalterado').
    if session.role == 0 && account.login !=
        session.login
      .form-group
        label(for='role') Função
        select.form-control#role(name='role')
          option(value='1', selected=account.role ==
              1 ? 'selected' : false)
            | Gerente - gerencia apenas portas
                autorizadas
          option(value='0', selected=account.role ==
              0 ? 'selected' : false)
            | Administrador - gerencia todas as portas
                e usuários
          option(value='-1', selected=account.role ==
              -1 ? 'selected' : false)
            | Nenhum - conta desativada
    button.btn.btn-primary(type='submit')
        #{account.new ? 'Cadastrar' : 'Atualizar'}
    if session.role == 0
      a.btn.btn-danger.ml-3(class='button'
          href='/accounts') Cancelar
    else
      a.btn.btn-danger.ml-3(class='button' href='/')
          Cancelar
```

src/server/views/account–list.pug

```
extends template.pug

block content


  .title Gerenciamento de Usuários
    a.btn.btn-outline-primary.cr.mr-3.mb-4(class='button'
        href='/') Gerenciar Portas

  a.btn.btn-success.mb-3(class='button'
```

```
      href='/accounts/new') Cadastrar novo usuário
  table.table.table-striped
    tr
      th(scope='col') Usuário
      th(scope='col') Função
      th.text-center(scope='col' colspan='2') Editar
    each user in accounts
      tr
        th(scope='row') #{user.login}

        case user.role
          when 0
            td Administrador
          when 1
            td Gerente
          when -1
            td Nenhum

        td.td-btn
          if user.role == 1
            a.btn.btn-primary(href='/accounts/doors/'
                + user.login) Portas
        td.td-btn
          a.btn.btn-secondary(href='/accounts/edit/'
              + user.login) Conta
```

src/server/views/card–form.js

```
function cardForm() {
  document.getElementById('card-form').classList
    .remove('d-none')
  document.getElementById('card-btn').classList
    .remove('btn-secondary')
  document.getElementById('card-btn').classList
    .add('btn-info')
  document.getElementById('ufsc-form').classList
    .add('d-none')
  document.getElementById('ufsc-btn').classList
    .add('btn-secondary')
  document.getElementById('ufsc-btn').classList
    .remove('btn-info')
  document.getElementById('initial-cancel').classList
    .add('d-none')
}
function ufscForm() {
  document.getElementById('ufsc-form').classList
    .remove('d-none')
  document.getElementById('ufsc-btn').classList
```

```
    .remove('btn-secondary')
  document.getElementById('ufsc-btn').classList
    .add('btn-info')
  document.getElementById('card-form').classList
    .add('d-none')
  document.getElementById('card-btn').classList
    .add('btn-secondary')
  document.getElementById('card-btn').classList
    .remove('btn-info')
  document.getElementById('initial-cancel').classList
    .add('d-none')
}
function validate() {
  return true
}
```

src/server/views/door–card–list.pug

```
extends template.pug

block content
  .title Gerenciar Cartões - Porta: "#{door.name ||
      '#'+door.id}" - MAC: #{door.mac}

  a.btn.btn-warning.mb-3(class='button' href='/')
      Voltar
  a.btn.btn-success.ml-3.mb-3(class='button'
      href='/door/'+door.id+'/manage-add') Adicionar
  form(method='post')
    table.table.table-striped
      tr
        th(scope='col') Nome
        th(scope='col') UID do Cartão
        th(scope='col') ID Pessoa UFSC
        th.text-center(scope='col') Editar
      each card in cards
        tr
          th #{card.name || 'Sem Nome'}
          td(scope='row') #{card.card_uid}
          td(scope='row') #{card.id_pessoa || 'Não
              Cadastrado'}
          td.td-btn
            button.btn.btn-danger(type='submit',
                name='delete', value=card.card_uid)
                Remover
```

src/server/views/door–config.pug

```
extends template.pug

block content
  script(type='text/javascript')
    include form-validate.js

  .title Configurar Porta - Porta: "#{door.name ||
      '#'+door.id}" - MAC: #{door.mac}

  form(method='post', onsubmit='return validate()')
    .form-group
      label(for='name') Nome
      input.form-control#username(type='text',
          name='name', value=door.name).
    .form-group
      label(for='name') MAC
      input.form-control#mac(type='text', name='mac',
          value=door.mac).

    button.btn.btn-primary(type='submit') Atualizar
    if session.role == 0
      a.btn.btn-danger.ml-3(class='button'
          href='/accounts') Cancelar
    else
      a.btn.btn-danger.ml-3(class='button' href='/')
          Cancelar
```

src/server/views/door–log.pug

```
extends template.pug

block content
  .title Registros - Porta: "#{door.name ||
      '#'+door.id}"
  a.btn.btn-warning.mb-4(class='button' href='/')
      Voltar


  table.table.table-striped
    tr
      th(scope='col') Data e Hora
      th(scope='col') Ação
      th(scope='col') UID do Cartão
      th(scope='col') Nome
    each entry in log
      tr
        - var date =
            entry.timestamp.toLocaleTimeString('pt-BR',
```

```
            {day:'numeric', month:'numeric',
            year:'numeric', hour: '2-digit',
            minute:'2-digit'})
        th(scope='row') #{date}
        case entry.action
          when 'AUTHORIZED'
            td Autorizado
          when 'DENIED'
            td Negado
          default
            td #{entry.action}
        #{entry.action}
        td #{entry.card_uid}
        td #{entry.name || 'Não Cadastrado'}
```

src/server/views/form–validate.js

```
function validate() {
  var pass1 =
      document.getElementById('password').value
  var pass2 =
      document.getElementById('password-confirm').value
  if (pass1 != pass2) {
    alert('ERRO\nA senha e a confirmação são
        diferentes!')
    return false
  }
  return true
}
```

src/server/views/login.pug

```
extends template.pug

block content
  form(method='post')
    .form-group
      label(for='username') Usuário
      input.form-control#username(type='text',
          name='username').
    .form-group
      label(for='password') Senha
      input.form-control#password(type='password',
          name='password').
    button.btn.btn-primary(type='submit') Entrar
```

src/server/views/manage–add–card.pug

```
extends template.pug

block content

  .title Cadastrar novo cartão - Porta: "#{door.name
      || '#'+door.id}"

  if (!card)
    script(type='text/javascript')
      include card-form.js


    button.btn.btn-secondary#ufsc-btn(type='button',
        onclick="ufscForm()")
      | Importar ID UFSC
    button.btn.btn-secondary.ml-3#card-btn(type='button',
        onclick="cardForm()")
      | Cadastrar Manualmente

    .mb-4


    form.d-none#ufsc-form(method='post'
        action='/door/'+door.id+'/search')
      .form-group
        label(for='search') ID UFSC, Matrícula ou CPF
        input.form-control#search(type='text',
            name='search').
      button.btn.btn-primary(type='submit') Pesquisar
      a.btn.btn-danger.ml-3(href='/door/' + door.id +
          '/manage') Cancelar


    a.btn.btn-danger#initial-cancel(href='/door/' +
        door.id + '/manage') Cancelar

  form#card-form(method='post',
      action='/door/'+door.id+'/manage-add',
      class=(card ? '' : 'd-none'))
    -var readonly = card? true : null
    .form-group
      label(for='name') Nome
      input.form-control#name(type='text',
          name='name', value=card ? card.nome : '',
          readonly=readonly).
    .form-group
      label(for='uid') UID do Cartão
      input.form-control#uid(type='text', name='uid',
```

```
          value=card ? card.uidCartao : '',
          readonly=readonly).
    if (card)
      .form-group
        label(for='uid') ID do Cartão
        input.form-control#ufsc_id(type='text',
            name='ufsc_id', value=card.idPessoa,
            readonly).
    button.btn.btn-primary(type='submit') Cadastrar
    a.btn.btn-danger.ml-3(href='/door/' + door.id +
        '/manage') Cancelar
```

src/server/views/manage–door–list.pug

```
extends template.pug

block content
  .title Gerenciamento de Portas
    if session.role == 0
      a.btn.btn-outline-primary.cr.mr-3.mb-4(class='button'
          href='/accounts') Gerenciar Usuários


  table.table.table-striped
    tr
      th(scope='col') Nome
      th(scope='col') MAC
      th.text-center(scope='col' colspan='3') Editar
    each door in doors
      tr
        th #{door.name || '#'+door.id}
        td(scope='row') #{door.mac}
        td.td-btn
          if session.role == 0
            a.btn.btn-secondary(href='/door/'+door.id+'/config')
                Configurar
        td.td-btn
          a.btn.btn-secondary(href='/door/'+door.id+'/log')
              Log
        td.td-btn
          a.btn.btn-secondary(href='/door/'+door.id+'/manage')
              Gerenciar
```

src/server/views/template.pug

```
doctype html
html
  head
```

```
    title Controle de Acesso
    link(rel='stylesheet',
        href='/css/bootstrap.min.css')
    link(rel='stylesheet', href='/css/style.css')
  body
    .topbar
      span.title
        a(href='/') Controle de Acesso
      if session.role != undefined
        span.buttons
          a.btn.btn-secondary(class='button'
              href='/accounts/edit/' + session.login)
              #{session.login}
          a.btn.btn-danger(class='button'
              href='/logout') Sair

    // Flash messages
    each key in Object.keys(messages)
      .flash.alert(class='alert-'+key)
          #{messages[key]}

    .container
      block content
```

src/server/views/user–door–list.pug

```
extends template.pug

block content
  .title #{add ? 'Adicionar Porta' : 'Portas
      Autorizadas'} - Usuário #{user.login}

  if add
    a.btn.btn-danger.mb-3(class='button'
        href='/accounts/doors/'+user.login) Cancelar
  else
    a.btn.btn-warning.mb-3(class='button'
        href='/accounts/') Voltar
    a.btn.btn-success.ml-3.mb-3(class='button'
        href='/accounts/doors/' + user.login +
        '/add') Adicionar Porta
  table.table.table-striped
    tr
      th(scope='col') Nome
      th(scope='col') MAC
      th.text-center(scope='col') Editar
    each door in doors
      tr
```

```
        th #{door.name}
        td(scope='row') #{door.mac}
        td.td-btn
          if add
            a.btn.btn-success(href='/accounts/doors/'
                + user.login + '/add/'+door.id)
                Adicionar
          else
            a.btn.btn-danger(href='/accounts/doors/' +
                user.login + '/delete/'+door.door)
                Remover
```

ANEXO G – Firmware modificado para o ESP8266

src/esp8266/espv4.ino

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <time.h>
#include <rBase64.h> //
    https://github.com/boseji/rBASE64
#include "FS.h"

class WiFiConnection {
  static const int SSID_MAX_SIZE = 32;
  static const int PASS_MAX_SIZE = 32;

private:
  char ssid[SSID_MAX_SIZE + 1] = {0};
  int ssid_size;

  char pass[PASS_MAX_SIZE + 1] = {0};
  int pass_size;

private:
  bool can_try_connection(){
    if(ssid_size == 0)
      return false;

    return true;
  }

public:
  WiFiConnection() {
    ssid_size = 0;
    pass_size = 0;
  }

  bool connect_to_wifi(){
    if(!can_try_connection())
      return false;

    WiFi.disconnect();

    WiFi.begin(ssid, pass);

    return (WiFi.waitForConnectResult() ==
        WL_CONNECTED);
  }

  bool wifi_connected() {
    return !(WiFi.status() != WL_CONNECTED);
  }
```

```
bool disconnect_from_wifi(){
  WiFi.disconnect();

  return true;
}

void get_ip(){
  Serial.println(WiFi.localIP());
}

String get_mac(){
  return WiFi.macAddress();
}

bool set_ssid(char *newssid, int ssidsize) {
  if(ssidsize > SSID_MAX_SIZE || newssid == 0)
    return false;

  ssid_size = ssidsize;
  memcpy(ssid, newssid, ssidsize);
  ssid[ssidsize] = '\0';

  return true;
}

bool set_pass(char *newpass, int passsize) {
  if(passsize > PASS_MAX_SIZE || newpass == 0)
    return false;

  pass_size = passsize;
  memcpy(pass, newpass, passsize);
  pass[passsize] = '\0';

  return true;
}

 char *get_ssid() {
  return ssid;
}

int get_ssid_size(){
  return ssid_size;
}

char *get_pass() {
  return pass;
}
```

```
  int get_pass_size(){
    return pass_size;
  }

};

class ServerConnection {
protected:
  static const int HOST_MAX_SIZE = 128;
  static const int ROUTE_MAX_SIZE = 128;
  static const int MAX_MESSAGE_SIZE = 255;

  char host[HOST_MAX_SIZE + 1] = {0};
  int host_size;

  char route[ROUTE_MAX_SIZE + 1] = {0};
  int route_size;

  int port;

  unsigned long int last_time_recorded;
  unsigned long int last_response_time;

  WiFiConnection * wifi;
  WiFiClientSecure client;

protected:

  bool is_all_setup_correctly(){
    if(host_size == 0 || route_size == 0 || port == 0)
      return false;

    return true;
  }

  bool say_hello(){
    unsigned long int timeNow = millis();

    if (wifi->wifi_connected()) {
      return false;
    }

    while(client.connect(host, port) && (unsigned
        long int) millis() - timeNow < 5000) {
      delay(0);
    }

    return client.connected();
```

```
  }

public:
  ServerConnection(WiFiConnection * wifi){
    this->wifi = wifi;
    host_size = 0;
    route_size = 0;
    port = 0;
  }

  bool set_host(char *newhost, int hostsize) {
    if(hostsize > HOST_MAX_SIZE || newhost == 0)
      return false;

    host_size = hostsize;
    memcpy(host, newhost, hostsize);
    host[hostsize] = '\0';

    return true;
  }

  bool set_route(char *newroute, int routesize) {
    if(routesize > ROUTE_MAX_SIZE || newroute == 0)
      return false;

    route_size = routesize;
    memcpy(route, newroute, routesize);
    route[routesize] = '\0';

    return true;
  }

  bool set_port(int newport) {
    port = newport;

    return true;
  }

  char *get_host() {
    return host;
  }

  int get_host_size(){
    return host_size;
  }

  char *get_route() {
    return route;
  }
```

```
  int get_route_size(){
    return route_size;
  }

  int get_port(){
    return port;
  }

  virtual bool send_data(String method_t, unsigned
      char * message, int message_size) = 0;
  virtual bool wait_for_response(int timeout) = 0;

};

class WSConnection : public ServerConnection {

  static const int FINGERPRINT_MAX_SIZE = 128;
  static const int USER_MAX_SIZE = 32;
  static const int PASS_MAX_SIZE = 32;

private:
  char cert_fingerprint[FINGERPRINT_MAX_SIZE + 1] =
      {0};
  int fingerprint_size;

  char basic_user[USER_MAX_SIZE + 1] = {0};
  int user_size;

  char basic_password[PASS_MAX_SIZE + 1] = {0};
  int pass_size;

  String basic_auth;

public:
  WSConnection(WiFiConnection * wifi) :
      ServerConnection(wifi) {
    fingerprint_size = 0;
    user_size = 0;
    pass_size = 0;
  }

  bool set_cert_fingerprint(char * fingerprint, int
      fsize) {
    if(fsize > FINGERPRINT_MAX_SIZE || fingerprint ==
        0)
      return false;

    fingerprint_size = fsize;
```

```
  memcpy(cert_fingerprint, fingerprint, fsize);
  cert_fingerprint[fsize] = '\0';

  return true;
}

bool set_basic_user(char * newuser, int usersize) {
  if(usersize > USER_MAX_SIZE || newuser == 0)
    return false;

  user_size = usersize;
  memcpy(basic_user, newuser, usersize);
  basic_user[usersize] = '\0';

  return update_auth(usersize > 0);
}

bool set_basic_pass(char * newpass, int passsize) {
  if(passsize > PASS_MAX_SIZE || newpass == 0)
    return false;

  pass_size = passsize;
  memcpy(basic_password, newpass, passsize);
  basic_password[passsize] = '\0';

  return update_auth(passsize > 0);
}

bool update_auth(bool use_auth) {
  if (use_auth) {
    basic_auth = rbase64.encode(String(basic_user)
        + ":" + String(basic_password));
  } else {
    basic_auth = "";
  }
  return true;
}

bool send_data(String method_t, unsigned char *
    message, int message_size) {

  if(!is_all_setup_correctly() || message_size >=
      MAX_MESSAGE_SIZE) {
    Serial.print("{\"error\":\"setup_error\",\"error");
    Serial.print("_description\":\"ESP8266 setup is
        incomplete or incorrect\"}");
    return false;
  }
  if (!wifi->wifi_connected()) {
```

```
    Serial.print("{\"error\":\"wifi_error\",\"error");
    Serial.print("_description\":\"Device not
        connected to a wifi network\"}");
    return false;
  }
  if (!client.connect(host, port)) {
    Serial.print("{\"error\":\"connection_fail\",\"error");
    Serial.print("_description\":\"Connection to
        server failed\"}");
    return false;
  }
  if (cert_fingerprint[0] != '\0' &&
      !client.verify(cert_fingerprint, host)) {
    Serial.print("{\"error\":\"cert_fail\",");
    Serial.print(\"error_description\":\"Server
        certificate does not match\"}");
    return false;
  }

  String s_request = method_t + " " + route + "
      HTTP/1.0\r\n" +
      "Host: " + host + "\r\n" +
      (basic_auth != "" ? ("Authorization: Basic "
          + basic_auth + "\r\n") : "") +
      "User-Agent: ESP8266\r\n" +
      "Content-Type: text/plain\r\n" +
      "UWS-Client-NetworkInterfaces: {\"" +
          wifi->get_mac() + "\":\"wlan0\"}\r\n" +
      "Connection: close\r\n" +
      "Content-Length: " + String(message_size) +
          "\r\n" +
      "\r\n";

  unsigned char c_request[s_request.length() +
      message_size];

  for(int i = 0; i < s_request.length(); i++)
    c_request[i] = s_request.charAt(i);

  memcpy(c_request + s_request.length(), message,
      message_size);

  last_time_recorded = millis();

  client.write(c_request, s_request.length() +
      message_size);

  return true;
}
```

```
  bool wait_for_response (int timeout){
    unsigned long int timeNow = millis();
    bool first_line = false;

    while(client.available() == 0 && (unsigned long
        int) millis() - timeNow < timeout) {
      delay(0);
    }

    if((unsigned long int) millis() - timeNow >
        timeout)
      return false;

    while(client.connected()) {
      String line = client.readStringUntil('\n');
        if (line == "\r") {
        break;
      }
    }
    String response = client.readStringUntil('\n');
    Serial.print(response);
    delay(5);

    return true;
  }

};

class IoTConnection : public ServerConnection {

public:
  IoTConnection(WiFiConnection * wifi) :
      ServerConnection(wifi) {
    File certificate = SPIFFS.open("/cert.der", "r");
    File key = SPIFFS.open("/key.der", "r");
    client.loadCertificate(certificate);
    client.loadPrivateKey(key);
  }


  bool send_data(String method_t, unsigned char *
      message, int message_size) {
    if(!is_all_setup_correctly() || message_size >=
        MAX_MESSAGE_SIZE)
      return false;

    if (!wifi->wifi_connected()) {
      Serial.print("WIFI_DISCONNECTED\r\n");
```

```
    return false;
  }

  if(!client.connected())
    say_hello();

  String s_route;

  for(int i = 0; i < route_size; i++)
    s_route += route[i];

  String s_request = method_t + " " + s_route + "
      HTTP/1.0\r\nConnection:
      keep-alive\r\nContent-type:
      application/octet-stream\r\nContent-Length: "
      + String(message_size) + "\r\n\r\n";

  unsigned char c_request[s_request.length() +
      message_size];

  for(int i = 0; i < s_request.length(); i++)
    c_request[i] = s_request.charAt(i);

  memcpy(c_request + s_request.length(), message,
      message_size);

  last_time_recorded = millis();

  client.write(c_request, s_request.length() +
      message_size);

  return true;
}

bool wait_for_response(int timeout){
  unsigned long int timeNow = millis();
  bool first_line = false;

  while(client.available() == 0 && (unsigned long
      int) millis() - timeNow < timeout) {
    delay(0);
  }

  if((unsigned long int) millis() - timeNow >
      timeout)
    return false;

  while(client.available() != 0) {
    String line = client.readStringUntil('\r');
```

```
      if(!first_line) {
        last_response_time = (unsigned long int)
            millis() - last_time_recorded;
        first_line = true;
      }

      Serial.print(line);
      delay(5);
    }

    return true;
  }

};

void act();

WiFiConnection wifi;
IoTConnection iot(&wifi);
WSConnection ws(&wifi);

void setup() {
  Serial.begin(115200);
}

char command[32];
int command_pos = 0;

char data[4096];
int data_pos = 0;

bool is_command = true; //alternates between command
    or data

void loop() {

  if(Serial.available() != 0){ //if received a
      command or data

    while(Serial.available() != 0){ //read until it
        finds an end

      if(is_command) {
        command[command_pos] = Serial.read();

        if(command[command_pos] == '=') {
          is_command = false;
```

```
        continue;
      }

      if(command[command_pos] == '\n' &&
         command[command_pos - 1] == '\r') {
        act();

        command_pos = 0;
        break;
      }

     command_pos++;


    } else {
      data[data_pos] = Serial.read();

      if(data[data_pos] == '+' && data[data_pos-1]
          == '+' && data[data_pos-2] == '+') {
        act();

        is_command = true;
        data_pos = 0;
        command_pos = 0;

        break;
      }

      data_pos++;
    }

  }

}

void act(){

  if(command[0] != 'A' || command[1] != 'T' ||
      command[2] != '+') {
    return;
  }

  ServerConnection * connection;
  int action_begin;
  int action_size;
  int data_size;
  char action[action_size + 1];
```

```
if(command[3] == 'I' && command[4] == 'O' &&
    command[5] == 'T' && command[6] == '+') {
  connection = &iot;
  action_begin = 7;
}

else if(command[3] == 'W' && command[4] == 'S' &&
    command[5] == '+') {
  connection = &ws;
  action_begin = 6;
}

else {
  connection = NULL;
  action_begin = 3;
}

data_size = data_pos - 2;
action_size = command_pos - action_begin - 1;

if(action_size < 1) {
  return;
}

for(int i = action_begin; i < command_pos; i++){
  action[i - action_begin] = command[i];
}
action[action_size] = '\0';

if (connection == NULL) {
  if(strncmp(action,"SETSSID",action_size) == 0){
    if(data_size == 0) {
      Serial.print("ERR");
    } else {
      bool result = wifi.set_ssid(data, data_size);

      if(result)
        Serial.print("OK");
      else
        Serial.print("ERR");
    }

  } else
      if(strncmp(action,"SETPASSWORD",action_size)
      == 0){
    bool result = wifi.set_pass(data, data_size);
```

```
  if(result)
    Serial.print("OK");
  else
    Serial.print("ERR");

} else if(strncmp(action,"GETSSID",action_size)
    == 0){

  Serial.print(wifi.get_ssid());

} else if(strncmp(action,"GETPASS",action_size)
    == 0){

  Serial.print(wifi.get_pass());

} else
    if(strncmp(action,"CONNECTWIFI",action_size)
    == 0){
  bool result = wifi.connect_to_wifi();

  if(result)
    Serial.print("OK");
  else
    Serial.print("ERR");

} else
    if(strncmp(action,"DISCONNECTWIFI",action_size)
    == 0){

  bool result = wifi.disconnect_from_wifi();

  if(result)
    Serial.print("OK");
  else
    Serial.print("ERR");

} else if(strncmp(action,"GETIP",action_size) ==
    0){

  wifi.get_ip();

} else if(strncmp(action,"GETMAC",action_size) ==
    0){

  Serial.println(wifi.get_mac());

} else
    if(strncmp(action,"CONNECTIONSTATUS",action_size)
    == 0){
```

```
      Serial.print(wifi.wifi_connected());

    } else
        if(strncmp(action,"GETTIMESTAMP",action_size)
        == 0){

      if(!wifi.wifi_connected()) {
        Serial.print("ERR");
      } else {
        configTime(8 * 3600, 0, "pool.ntp.org",
            "time.nist.gov");
        time_t now = time(nullptr);
        while (now < 1000) {
          delay(2);
          now = time(nullptr);
        }
        Serial.print(now);
      }

    } else
        if(strncmp(action,"GETHEAPSIZE",action_size)
        == 0){

      Serial.print(ESP.getFreeHeap());

    } else {
      Serial.print("INVALIDCOMMAND");
    }


  } else {
    if(strncmp(action,"SETHOST",action_size) == 0){
      if(data_size == 0) {
        Serial.print("ERR");
      } else {

        bool result = connection->set_host(data,
            data_size);

        if(result)
          Serial.print("OK");
        else
          Serial.print("ERR");

      }
    } else if(strncmp(action,"SETROUTE",action_size)
        == 0){
      if(data_size == 0) {
```

```
    Serial.print("ERR");
  } else {

    bool result = connection->set_route(data,
        data_size);

    if(result)
      Serial.print("OK");
    else
      Serial.print("ERR");

  }
} else if(strncmp(action,"SETPORT",action_size)
    == 0){
  if(data_size == 0) {
    Serial.print("ERR");
  } else {

    String temp;

    for(int i = 0; i < data_size; i++)
      temp += data[i];

    bool result =
        connection->set_port(temp.toInt());

    if(result)
      Serial.print("OK");
    else
      Serial.print("ERR");

  }
} else if(strncmp(action,"GETHOST",action_size)
    == 0){

  Serial.print(connection->get_host());

} else if(strncmp(action,"GETROUTE",action_size)
    == 0){

  Serial.print(connection->get_route());

} else if(strncmp(action,"GETPORT",action_size)
    == 0){

  Serial.print(connection->get_port());

} else
    if(strncmp(action,"RESPONSETIME",action_size)
```

```
    == 0){

  Serial.println("OK");

} else if(strncmp(action,"SENDPOST",action_size)
    == 0){
  if(data_size == 0) {
    Serial.print("ERR");
  }

  if (connection->send_data(String("POST"),
      (unsigned char *) data, data_size)) {
    if (connection->wait_for_response(5000)) {
      Serial.print("OK");
    } else {
      Serial.print("ERR");
    }
  } else {
    Serial.print("ERR");
  }

} else if(strncmp(action,"SENDGET",action_size)
    == 0){
  connection->send_data(String("GET"), (unsigned
      char *) data, data_size);

  bool res = connection->wait_for_response(5000);

  if(res)
    Serial.print("OK");
  else
    Serial.print("ERR");

} else {
  if (connection == &ws) {
    if(strncmp(action,"SETFINGERPRINT",action_size)
        == 0){

      bool result = ws.set_cert_fingerprint(data,
          data_size);

      if(result)
        Serial.print("OK");
      else
        Serial.print("ERR");

    } else
        if(strncmp(action,"SETUSER",action_size)
        == 0){
```

```
          bool result = ws.set_basic_user(data,
              data_size);

          if(result)
            Serial.print("OK");
          else
            Serial.print("ERR");

        } else
            if(strncmp(action,"SETPASS",action_size)
            == 0){

          bool result = ws.set_basic_pass(data,
              data_size);

          if(result)
            Serial.print("OK");
          else
            Serial.print("ERR");

        } else {
          Serial.print("INVALIDCOMMAND");
        }
      } else {
        Serial.print("INVALIDCOMMAND");
      }
    }
  }

  Serial.print('\r');
  Serial.print('\n');

}
```