CENTRO UNIVERSITÁRIO DE BRASÍLIA – UNICEUB

FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS

CURSO DE ENGENHARIA DA COMPUTAÇÃO

LUCAS GENNARI SILVA ABREU

SISTEMA DE CONTROLE DE PONTO AUXILIADO POR APLICATIVO ANDROID

Orientador: Prof.º MsC. Francisco Javier De Obaldía Díaz

Brasília, DF – Brasil

Julho de 2016

LUCAS GENNARI SILVA ABREU

SISTEMA DE CONTROLE DE PONTO AUXILIADO POR APLICATIVO ANDROID

Trabalho de Conclusão de Curso apresentado à Banca examinadora do curso de Engenharia da Computação da FATECS – Faculdade de Tecnologia e Ciências Sociais Aplicadas – Centro Universitário de Brasília como requisito para obtenção do título de Engenheiro da Computação

Orientador: Prof.º MsC. Francisco Javier De Obaldía Díaz

BRASÍLIA, DF – BRASIL

JULHO DE 2016

LUCAS GENNARI SILVA ABREU

SISTEMA DE CONTROLE DE PONTO AUXILIADO POR APLICATIVO ANDROID

Trabalho de Conclusão de Curso apresentado à Banca examinadora do curso de Engenharia da Computação da FATECS – Faculdade de Tecnologia e Ciências Sociais Aplicadas – Centro Universitário de Brasília como requisito para obtenção do título de Engenheiro da Computação

Orientador: Prof.º MsC. Francisco Javier De Obaldía Díaz

BANCA EXAMINADORA

_____

Prof.º MsC. Francisco Javier De Obaldía Díaz

Orientador

_____

Prof.º MsC. Nilo Sérgio Soares Ribeiro

UniCEUB

_____

Prof.º Dr. Sidney Cerqueira Bispo Dos Santos

UniCEUB

BRASÍLIA, DF – BRASIL

JULHO DE 2016

AGRADECIMENTOS

À instituição UniCEUB que proporcionou a realização do sonho de me formar Engenheiro com ênfase em Computação.

Aos colegas Guilherme Pimenta, Luís Matheus Versiane, Matheus Paz, Ana Débora e Rebeca Hannah que foram promovidos à amigos devido ao contínuo suporte técnico e moral.

Ao grande amigo de trabalho Diego Kubota que durante todo o ano apresentou-se sempre solícito às minhas necessidades e esteve presente em todos os momentos.

Ao grande amigo da família Gustavo Aires com seu incansável incentivo e apoio.

À minha namorada, Mariana, que me incentivou durante todo o semestre e não me deixou desmotivar.

À minha mãe que sempre quis o melhor para mim e nunca deixou de me apoiar, mesmo com todas as dificuldades que ocorreram.

A todos que de uma forma ou outra estiveram empenhados neste trabalho.

À Deus por não me deixar desistir nunca.

CITAÇÃO

"I've missed more than 9000 shots in my career. I have lost almost 300 games. 26 times, I have been trusted to take the game winning shot and missed. I have failed over and over and over again in my life. And that is why I succeed."

Michael Jordan

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

Art.	Artigo
CLT	Consolidação das Leis Trabalhistas
EEPROM	Memória Programável Eletricamente Apagável Somente de Leitura
Gb/s	Giga Bits por Segundo
HTML	Linguagem de Marcação de Hipertexto
HTTP	Protocolo de Transferência de Hipertexto
ID	Identidade
IDE	Ambiente de Desenvolvimento Integrado
IEEE	Instituto para Engenheiros Elétricos e Eletrônicos
IP	Protocolo de Internet
ISO	Organização Internacional para Padronização
KB	Kilo Bytes
LCD	Display de Cristal Líquido
MAC	Controle de Acesso à Mídia
Mb/s	Mega Bits por Segundo
MER	Modelo Entidade Relacional
MHz	Mega Hertz
OSI	Interconexão de Sistemas Abertos
PHP	Pré-processador de Hipertexto
R$	Reais
RFID	Identificação por Radiofrequência
SPI	Interface Serial Periférica
SQL	Linguagem de Consulta Estruturada

SRAM	Memória Estática de Acesso Aleatório
URL	Localizador Padrão de Recursos
USB	Porta Serial Universal
V	Volts

RESUMO

Este trabalho propõe um sistema de controle de ponto auxiliado por um aplicativo Android, ou seja, é um sistema que permite que o usuário realize o registro de frequência em um dispositivo desenvolvido com Arduino, utilizando-se da tecnologia RFID, e armazene todo o seu histórico em um banco de dados. A informação pode ser visualizada via aplicativo Android, mostrando as informações de horas cumpridas e faltantes diárias, semanais e mensais. Além disso, permite o controle de horas trabalhadas por parte do empregado e da empresa, dando uma maior transparência para esta atividade. Após o desenvolvimento do projeto, foi constatado sua viabilidade não só em termos de funcionalidade, mas também por apresentar um baixo custo ao se comparar com produtos que apresentam as mesmas funções de relógio de ponto e pelo aplicativo Android que agrega ao possibilitar visualizar as informações guardadas no banco de dados, além de ter obtidos resultados satisfatórios durante os testes realizados com nenhuma falha.

Palavras chave: Leis trabalhistas, ponto eletrônico, Arduino, Android, empregado, empresa.

ABSTRACT

This paper proposes a work clock system aided by an Android app, that is, a system that allows the user to perform the work clock hit on a device developed with Arduino, using RFID technology, and store all your history in a database. The information is viewed via Android application, showing information of daily, weekly and monthly worked and missing hours. Moreover, allowing control of worked hours by the employee and the company, giving greater transparency to this activity. After the development of the project, it was found its viability not only in terms of functionality, but also by presenting a low cost when compared with products that have the same functions, and by the Android application aggregating when enables the view of information stored in the database, and having obtained satisfactory results during the tests performed.

Keywords: Work laws, electronic work clock, Arduino, Android, employee, company.

CAPÍTULO 1 – INTRODUÇÃO

1.1. Apresentação do Problema

De acordo com a CLT (Consolidação das Leis Trabalhistas), mais precisamente no Art. 74 § 2º, é obrigatório o registro manual, mecânico ou eletrônico da hora de saída e de entrada para estabelecimentos e empresas que possuem mais de dez empregados.

A falta de um controle de ponto não permite que a empresa saiba se os seus funcionários estão cumprindo toda a sua carga horária diária e mensal. Um controle de ponto manual possibilita que sejam colocados valores alterados dos horários de entrada e saída. Dessa forma, o uso do controle de ponto automatizado é necessário para que as empresas possam contabilizar as horas de trabalho de seus funcionários, minimizando erros.

Além disso, a maioria dos sistemas eletrônicos de ponto não possuem uma forma de notificar os funcionários sobre o cumprimento das suas horas de trabalho de forma prática e fácil.

Assim, a implementação de um produto que ofereça os benefícios de gerenciamento do ponto dos funcionários, tanto para as empresas quanto para os próprios empregados, é o objetivo deste trabalho.

1.2. Objetivos do Trabalho

Desenvolver um projeto de controle e gerenciamento de ponto, de baixo custo, para ser implementado nas empresas, a fim de cumprir com as leis trabalhistas e permitir a visualização das informações por parte dos empregados.

1.2.1. Objetivos específicos

- Desenvolver protótipo de leitura de cartões RFID;
- Desenvolver servidor de banco de dados com as informações de ponto;
- Desenvolver um aplicativo Android que acesse o banco de dados e apresente as informações lidas;
- Facilitar o controle e gerenciamento dos horários de ponto.

1.3. Justificativa e Importância do Trabalho

A dificuldade que os funcionários de uma empresa possuem para visualizar o cumprimento de sua carga horária diária, semanal e mensal em seus locais de trabalho foi o principal motivo que levou a buscar a implementação de uma solução para esse problema.

Já que dessa forma, os funcionários podem ter o controle das horas que estão cumprindo durante os dias do mês e saibam quando estão devendo horas ou quantas horas extras possuem. E também, possibilita que as empresas possuam o controle de todas as horas feitas por seus empregados.

1.4. Escopo do Trabalho

Será montado um dispositivo por meio da plataforma Arduino com um leitor de cartões utilizando a tecnologia de RFID (Identificação por Radiofrequência) em conjunto com um módulo Ethernet para se conectar com um computador que funciona como servidor local e, dessa forma, transmitir os dados de acesso. O servidor receberá esses dados e irá armazená-los em um banco de dados para que seja possível ter todos os horários de entrada e saída dos funcionários salvos.

Esses dados serão acessíveis aos funcionários por meio de um aplicativo Android que também fará parte do desenvolvimento do projeto, fazendo uso de uma ferramenta *open-source* chamada Android Studio para a escrita e compilação dos

códigos de programação e permitirá a visualização das horas cumpridas e quantas horas faltam para completar a carga horária diária, semanal e mensal.

O produto final será uma máquina de ponto posicionada próxima à entrada de funcionários, que irão aproximar o cartão RFID para informar sua hora de entrada ou saída do estabelecimento. E um aplicativo móvel que apresenta essas informações tanto para o empregado quanto para o empregador.

1.5. Resultados Esperados

O dispositivo de ponto automático será capaz de fazer a leitura dos dados do cartão RFID, verificar se o funcionário está cadastrado no sistema e irá registrar o horário que foi passado o cartão, tanto na saída como na entrada da empresa.

Como resultado desse registro, as informações serão imediatamente transmitidas para o servidor de banco de dados local. Assim, fazendo uso da linguagem de programação Java, o aplicativo *mobile* Android será capaz de recuperar essas informações e disponibilizá-las na tela do celular.

Espera-se que com a implementação deste projeto, os dados possam ser acessados tanto por empregado como empregador, via *smartphone*, completando assim a funcionalidade do sistema.

1.6. Estrutura do Trabalho

O conteúdo deste trabalho foi estruturado conforme a descrição a seguir:

No **capítulo 1** é apresentado uma breve introdução sobre o tema a ser tratado neste projeto, o **capítulo 2** faz um aprofundamento no tema, apresentando o contexto do problema a ser solucionado, abordando a direção em que o trabalho será realizado. O **capítulo 3** descreve a revisão bibliográfica, a metodologia e referencial teórico, que fundamentam a pesquisa necessária ao desenvolvimento do projeto. O **capítulo 4** apresenta a proposta em concordância com o estudo realizado, ilustrando o modelo desenvolvido para a resolução do problema. O **capítulo 5** traz a aplicação prática do

modelo proposto, os custos do projeto e os resultados obtidos com ele, assim como a avaliação e análise do sistema em funcionamento. E por fim, o **capítulo 6** apresenta as conclusões finais e as possíveis sugestões para trabalhos futuros

CAPÍTULO 2 – APRESENTAÇÃO DO PROBLEMA

No ambiente de trabalho, sempre foi necessária a medição da produtividade de cada funcionário para saber se ele está trabalhando bem e sendo um bom investimento para a empresa ou se está sendo displicente com suas responsabilidades no seu trabalho.

Além disso, existe uma necessidade de se computar as horas trabalhadas de cada empregado como uma forma de validar a sua remuneração mensal. Com isto, levando em consideração a existência de diferentes categorias salarias por formação, competências, responsabilidades que demandam formas mais sofisticadas de se avaliar o desempenho e a remuneração.

Ao longo dos tempos, as empresas têm implementado diversas formas de avaliação dos seus funcionários, algumas com modelos bem definidos que vão desde o estabelecimento dos horários de entrada e saída, bem como os horários de intervalos, a existência dos bancos de horas, descontos de horários, horas extras, produtividade, competências, entre outros.

Como é de se esperar, isto implica numa relação de patrão e empregado envolvendo resultados por remuneração e abre espaço para possíveis discordâncias. Devido a isso, foi necessário o surgimento de formas regulamentares que colocassem normas para um melhor relacionamento patrão-funcionário buscando um acordo que fosse justo para ambas as partes.

Em 10 de novembro de 1943 foi aprovada a Consolidação das Leis do Trabalho, a partir de um decreto-lei, introduzindo alterações na legislação vigente que regulam as relações individuais e coletivas de trabalho (Decreto-Lei Nº 5.452, 1943). Dessa forma, alguns artigos e parágrafos serão utilizados como base para o projeto a ser desenvolvido.

Primeiramente, o Art. 58 diz que “A duração normal do trabalho, para os empregados em qualquer atividade privada, não excederá de 8 (oito) horas diárias, desde que não seja fixado expressamente outro limite”. Já o Art. 59 explicita a

existência de horas complementares à jornada de trabalho, que não podem exceder a duas horas e necessitam de um acordo entre empregador e empregado.

Também serão seguidos os Artigos 66 e 71, que falam sobre os períodos de descanso nas jornadas de trabalho. O primeiro diz que entre duas jornadas, é obrigatório um descanso mínimo de 11 horas consecutivas, enquanto o segundo diz que em trabalhos que tenham mais de 6 horas contínuas de duração, é obrigatório um intervalo de no mínimo 1 hora e no máximo 2 horas. E trabalhos que não excedam de 6 horas, após 4 horas de trabalho um intervalo de 15 minutos é obrigatório.

Em conjunto com esses Artigos, será trabalhado o Art. 74 que trata do quadro de horário, que deverá seguir um modelo criado pelo Ministro do Trabalho, Indústria e Comércio. Porém, o mais importante está no seu parágrafo segundo, que fala que estabelecimentos com mais de dez trabalhadores são obrigados a registrar os horários de entrada e saída dos funcionários de forma manual, mecânica ou eletrônica obedecendo os horários de repouso e intervalo já citados nos artigos anteriores.

Esse registro é uma das questões que gera bastante polêmica no ambiente de trabalho, devido à falta de transparência que o controle de entradas e saídas possui tanto para empregador quanto para o funcionário.

Uma das questões que gera bastante polêmica é a questão do controle das horas trabalhadas em função dos controles de entradas e saídas que as empresas estabelecem. Ocorre que em muitos casos não é tratado com a devida transparência para ambas as partes e assim, embora os funcionários registrem suas entradas e saídas, o controle e o histórico desses eventos, em geral, são de ampla consulta do patrão.

Por outro lado, existem limitações quanto ao número de horas-extras mensais permitidas por lei, no Art. 59 é dito que o máximo de horas suplementares diárias não pode passar de 2 horas, exceto em alguns casos de força maior como explicado no Art. 61. Ainda de acordo com o Art. 59, essas horas suplementares devem possuir uma remuneração de no mínimo 20% superior à hora normal ou podem ser utilizadas como banco de horas para serem diminuídas das horas feitas em outros dias.

Será que é possível desenvolver um sistema que permita o registro automático das entradas e saídas dos empregados, permitindo a consulta das informações online, tanto pelos empregados como pelos patrões?

O foco deste trabalho é desenvolver um sistema com transparência, que permita mostrar os registros de entradas e saída ao ambiente de trabalho.

CAPÍTULO 3 – BASES METODOLÓGICAS PARA RESOLUÇÃO DO PROBLEMA

3.1. Consolidação das Leis Trabalhistas

A Consolidação das Leis Trabalhistas trouxe diversas regras para as condições de trabalho e permitiu que tanto empregados como empregadores fossem protegidos de situações adversas. Os artigos que foram usados de base para a construção deste projeto estão listados abaixo.

- Art. 58 - A duração normal do trabalho, para os empregados em qualquer atividade privada, não excederá de 8 (oito) horas diárias, desde que não seja fixado expressamente outro limite.
- Art. 58 § 1º - Não serão descontadas nem computadas como jornada extraordinária as variações de horário no registro de ponto não excedentes de cinco minutos, observado o limite máximo de dez minutos diários.
- Art. 59 - A duração normal do trabalho poderá ser acrescida de horas suplementares, em número não excedente de 2 (duas), mediante acordo escrito entre empregador e empregado, ou mediante contrato coletivo de trabalho.
- Art. 59 § 1º - Do acordo ou do contrato coletivo de trabalho deverá constar, obrigatoriamente, a importância da remuneração da hora suplementar, que será, pelo menos, 20% (vinte por cento) superior à da hora normal.
- Art. 59 § 2º - Poderá ser dispensado o acréscimo de salário se, por força de acordo ou convenção coletiva de trabalho, o excesso de horas em um dia for compensado pela correspondente diminuição em outro dia, de maneira que não exceda, no período máximo de um ano, à soma das jornadas semanais de trabalho previstas, nem seja ultrapassado o limite máximo de dez horas diárias.
- Art. 61 - Ocorrendo necessidade imperiosa, poderá a duração do trabalho exceder do limite legal ou convencionado, seja para fazer face a motivo de força maior, seja para atender à realização ou conclusão de serviços inadiáveis ou cuja inexecução possa acarretar prejuízo manifesto.
- Art. 66 - Entre 2 (duas) jornadas de trabalho haverá um período mínimo de 11 (onze) horas consecutivas para descanso.

- Art. 71 - Em qualquer trabalho contínuo, cuja duração exceda de 6 (seis) horas, é obrigatória a concessão de um intervalo para repouso ou alimentação, o qual será, no mínimo, de 1 (uma) hora e, salvo acordo escrito ou contrato coletivo em contrário, não poderá exceder de 2 (duas) horas.
- Art. 71 § 1º - Não excedendo de 6 (seis) horas o trabalho, será, entretanto, obrigatório um intervalo de 15 (quinze) minutos quando a duração ultrapassar 4 (quatro) horas.
- Art. 74 - O horário do trabalho constará de quadro, organizado conforme modelo expedido pelo Ministro do Trabalho, Industria e Comercio, e afixado em lugar bem visível. Esse quadro será discriminativo no caso de não ser o horário único para todos os empregados de uma mesma seção ou turma.
- Art. 74 § 2º - Para os estabelecimentos de mais de dez trabalhadores será obrigatória a anotação da hora de entrada e de saída, em registro manual, mecânico ou eletrônico, conforme instruções a serem expedidas pelo Ministério do Trabalho, devendo haver pré-assinalação do período de repouso.

3.2. Controle de Acesso

Com a finalidade de gerenciar o fluxo de pessoas, carros e motos para facilitar a localização e o controle destes, é importante o uso de práticas para monitorar quaisquer ambientes como estacionamento, recepção e entrada em empresas. As Figuras 3.1 e 3.2 apresentam algumas das maneiras possíveis de se obter este controle de acesso, seja pelo uso de catracas ou o uso de relógios de ponto.

Figura 3.1 - Catraca para controle de acesso da empresa 3AVTech. Fonte: 3AVTech, 2014. Disponível em <http://www.3avtec.com.br/blog/catraca-empresas/>

Figura 3.2 - Relógio de ponto utilizado pela empresa TopData. Fonte: TopData, s.d. Disponível em <https://www.topdata.com.br/controle-de-ponto/>

Note que na Figura 3.1 é utilizado um sistema com um cartão RFID, em que a catraca reconhece o cartão e libera a passagem para o usuário. Já no caso da Figura 3.2 um sistema que faz o uso de um cartão RFID em conjunto com uma senha é utilizado para registrar o horário de entrada ou saída do funcionário de uma empresa.

Para controlar, gerenciar e consultar os horários de entrada e saída dos funcionários de uma empresa, o uso de um relógio de ponto é necessário. A Figura 3.3 apresenta um modelo semelhante ao que será desenvolvido neste projeto, no qual o usuário utiliza um cartão RFID para registrar o ponto.

Figura 3.3 - Funcionária utilizando um cartão RFID para registrar a hora de sua entrada na Prefeitura do Lago da Pedra. Fonte: Prefeitura Lago da Pedra, 2012. Disponível em <http://www.lagodapedra.ma.gov.br/noticias2012.php>

3.3. RFID – Identificação por Radiofrequência

O uso de formas automáticas de identificação se tornou bastante popular nos últimos anos em vários setores provendo informações sobre pessoas, animais, bens de consumo e produtos em trânsito. A Figura 3.4 traz um pequeno esquemático de alguns tipos diferentes de sistemas de identificação.

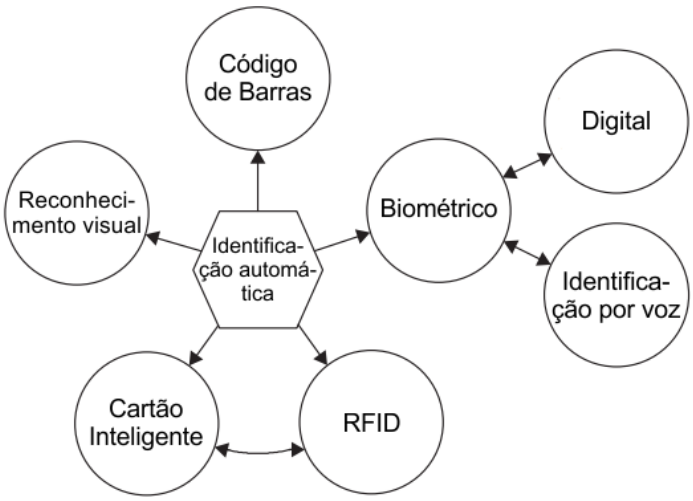

Figura 3.4 – Diferentes tipos de sistemas de identificação. Fonte: Adaptado de FINKENZELLER, 2010

Pode-se notar na Figura 3.4 que existem vários tipos de sistemas de identificação, como sistema de código de barras comumente utilizado para identificação e precificação de produtos em supermercados e afins, os tipos de sistemas biométricos, como identificação por voz ou por digital. O foco desde trabalho é a utilização da tecnologia de RFID, que diferentemente do código de barras não precisa de contato visual com a leitora.

Assim como o cartão inteligente, a informação é armazenada em um dispositivo eletrônico chamado transponder, um dispositivo eletrônico que recebe e retransmite uma mensagem pré-determinada. Porém, diferentemente do cartão eletrônico, a tecnologia faz o uso de campos magnéticos e eletromagnéticos sem necessitar de uma fonte de energia conectada ao dispositivo (FINKENZELLER, 2010).

A Figura 3.5 ilustra o funcionamento da tecnologia de RFID através da *tag*, que é o cartão que possui a identificação e da leitora, que por ondas eletromagnéticas consegue energizar a *tag* e receber o sinal transmitido por ela.

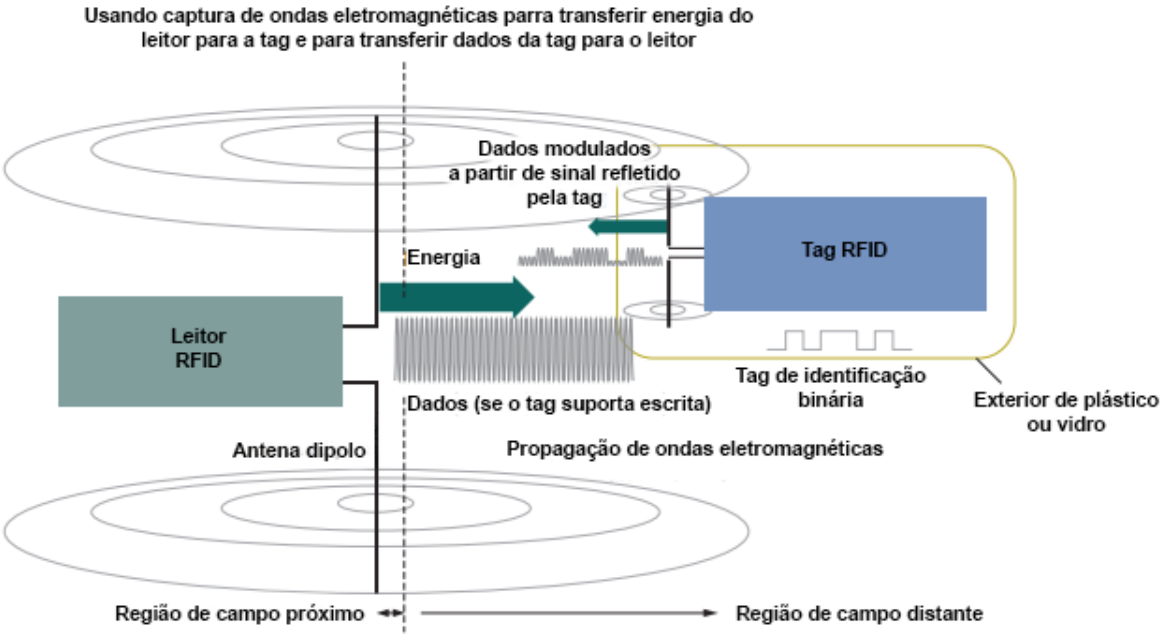

Figura 3.5 - Funcionamento de uma *tag* e um leitor RFID por ondas eletromagnéticas. Fonte: Adaptado de PERVARSIVE *Computing*, 2006

3.4. Banco de Dados

Banco de dados é um repositório de dados, feito para suportar armazenamento, recuperação e manutenção de dados de forma eficiente. Existem diversos tipos de banco de dados para atender às diversas exigências da indústria, podendo ser especializados em armazenar arquivos binários, documentos, imagens, vídeos e outros (SHARMA et al, 2010).

A linguagem base para os bancos de dados se chama SQL, que significa Linguagem de Consulta Estruturada, e foi desenvolvida na IBM por Chamberlin e Codd e foi padronizada pela ISO em 1987 (DELIGHT, 2013). Esta linguagem permite que os bancos sejam criados e manipulados com algumas linhas de códigos, fazendo uso de comando como criar, selecionar, deletar e atualizar.

Neste projeto será utilizado um banco de dados relacional incorporado *Open Source* chamado SQLite. Ele foi desenvolvido em 2000 para prover um uma forma conveniente de gerenciar dados sem a sobrecarga que muitas vezes vem com sistemas dedicados de banco de dados, além de possuir a reputação de ser altamente portável, de fácil utilização, compacto, eficiente e confiável (Owens, 2006).

A Figura 3.6 ilustra como será a utilização do banco de dados SQLite em conjunto com um processo Apache e um servidor feito na linguagem PHP. O código PHP que roda dentro do Apache implementa a API (Interface de Programação de Aplicações) do SQLite e com o auxílio do sistema operacional consegue utilizar o banco de dados.

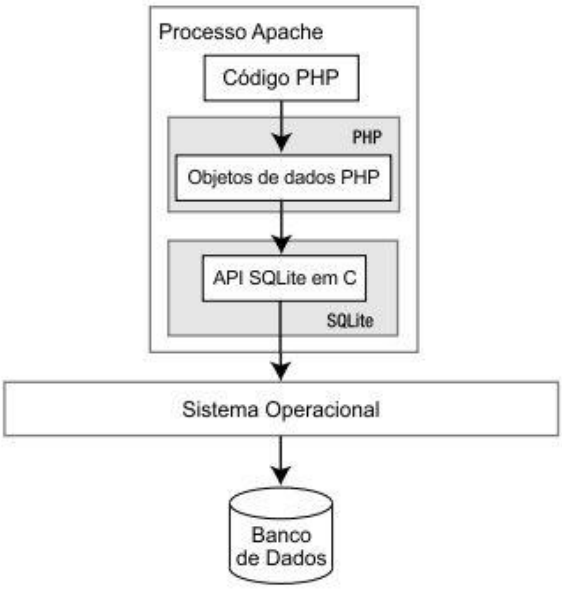

Figura 3.6 - Formato de servidor utilizado no projeto. Fonte: Adaptado de OWENS, 2006

3.4.1. Modelo Entidade Relacional (MER)

Modelo de dados é um plano de ação para a criação de um banco de dados. Ele dá uma forma concreta de se pensar no banco que será desenvolvido, além de deixar todas as pessoas do projeto cientes do que precisa ser feito. O MER faz o uso de diagramas para mapear as classes ou entidades nas devidas tabelas e é muito utilizado para se começar a desenhar um banco de dados (STEPHENS, 2009).

Para se entender esses diagramas, é precisa saber três conceitos que formam esse modelo e definem o projeto de um banco de dados, são eles as entidades, os atributos e a cardinalidade.

A entidade representa uma instância específica sobre algo que se quer controlar no modelo, podendo ser objetos físicos (empregados, pedido de trabalho) ou uma abstração lógica (reuniões, desculpas). Cada entidade possui atributos, que descrevem o objeto ao qual eles representam. Já a cardinalidade, representa as relações que as tabelas têm entre si, dando o máximo e o mínimo de entidades que podem se associar nas relações (STEPHENS, 2009).

3.5. Ethernet

A tecnologia Ethernet foi criada em 1973 e é utilizada até hoje em dia para criar redes de computadores das mais pequenas às maiores e das mais simples às mais complexas. Apesar de ser uma tecnologia relativamente antiga, ela tem sido constantemente reinventada com novas capacidades para se adequar à rápida transformação que o mundo da tecnologia sofre, tornando-se a tecnologia de rede mais usada no mundo inteiro (SPURGEON; ZIMMERMAN, 2014).

É através desta tecnologia que é possível que o uso da Internet em escala mundial, pois é ela que conecta os computadores, os *laptops*, os *smartphones* e *tablets* (pelo uso de pontos *wireless*) aos servidores que proveem a Internet.

Após a criação da Ethernet, ela se tornou um padrão definido pelo IEEE – Instituto para Engenheiros Elétricos e Eletrônicos, que tornou padrão o modelo 10Mb/s criado em 1980, o padrão do cabo de par trançado em 1990, o padrão *full-duplex*, o padrão 100Gb/s largamente utilizado nos dias de hoje, dentre outras várias padronizações que podem ser vistas na Tabela 3.1.

Para o melhor funcionamento da rede, é utilizado um modelo de referência chamado OSI – *Open Systems Interconnection*, que divide as funcionalidades da rede em sete camadas. A Figura 3.7 mostra quais são essas camadas, que vão desde a parte física até a parte de aplicação da rede, sendo que as camadas de interesse para a tecnologia Ethernet são as duas camadas mais baixas.

Tabela 3.1 - Alguns dos suplementos que formam o padrão 802.3 do IEEE.

Suplemento	Descrição
802.3a-1998	10BASE2 *thin Ethernet*
802.3c-1985	10 Mb/s especificações de repetidor
802.3i-1990	10BASE-T *twisted-pair*
802.3j-1993	10BASE-F fibra óptica
802.3x-1997	Padrão *full-duplex*
802.3ae-2002	802.3ae-2002
802.3ak-2004	10GBASE-CX4 10 Gigabit *Ethernet* sobre cabo coaxial de curto alcance
802.3an-2006	10GBASE-T 10 Gigabit *Ethernet* sobre par trançado
802.3aq-2007	10GBASE-LRM 10 Gigabit sobre fibra óptica de longo alcance
802.3ba-2010	40 Gb/s e 100 Gb/s *Ethernet*

Fonte: Adaptado de Spurgeon e Zimmerman, 2014

Figura 3.7 - Camadas do modelo OSI. Fonte: Spurgeon e Zimmerman, 2014

A primeira camada, que é a camada física, padroniza o controle elétrico, mecânico e funcional dos circuitos de dados que se conectam à mídia física, ou seja, ela é responsável pela transmissão e recepção do fluxo bruto de bits. Já a segunda camada, que é a camada de enlace, estabelece a comunicação entre as estações conectadas ao mesmo sistema de rede, sendo encarregada de transmitir e receber *frames* e conhecer endereços de rede, padronizando o tamanho e formato do *frame* e o protocolo de Controle de Acesso à Mídia, conhecido como MAC.

3.6. Plataforma Arduino

A plataforma Arduino tem sido utilizada em diversos projetos ao longo dos anos, desde projetos complexos à projetos mais simples que são utilizados no dia a dia das pessoas. Ela é uma plataforma *open-source* com *softwares* e *hardwares* de fácil utilização, custo relativamente baixo e um grande apoio da comunidade por meio de fóruns.

Existem várias placas Arduino no mercado, porém, neste projeto será utilizada a placa Arduino Nano v3.0, que pode ser visualizada na Figura 3.8. Ela é equipada com o micro controlador ATmega328, possui 14 canais digitais de entrada/saída e 6 analógicos, tem 32 KB de memória *flash*, 2 KB de SRAM, 1 KB de EEPROM e opera em um *clock* de 16 MHz. A placa opera em uma tensão de 5V mas possui um regulador de tensão que aceita entrada de 7 a 12V e utiliza um cabo USB para se comunicar com o computador e realizar a transferência do código fonte (Arduino, s.d.).

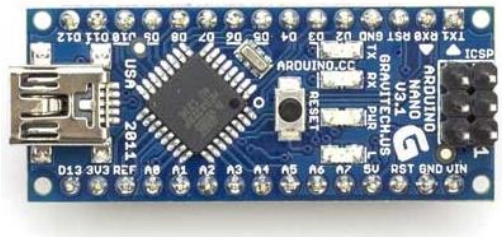

Figura 3.8 - Modelo NANO da plataforma Arduino. Fonte: Disponível em <www.arduino.cc>

Este código fonte, também chamado de *sketch*, é desenvolvido na IDE (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado) do Arduino, que é compatível com todos os sistemas operacionais. Faz uso das linguagens de programação C e C++, sendo derivada da plataforma de prototipagem e framework para micro controladores *Wiring*. A Figura 3.9 apresenta a IDE do Arduino com um trecho de código feito para teste do projeto.

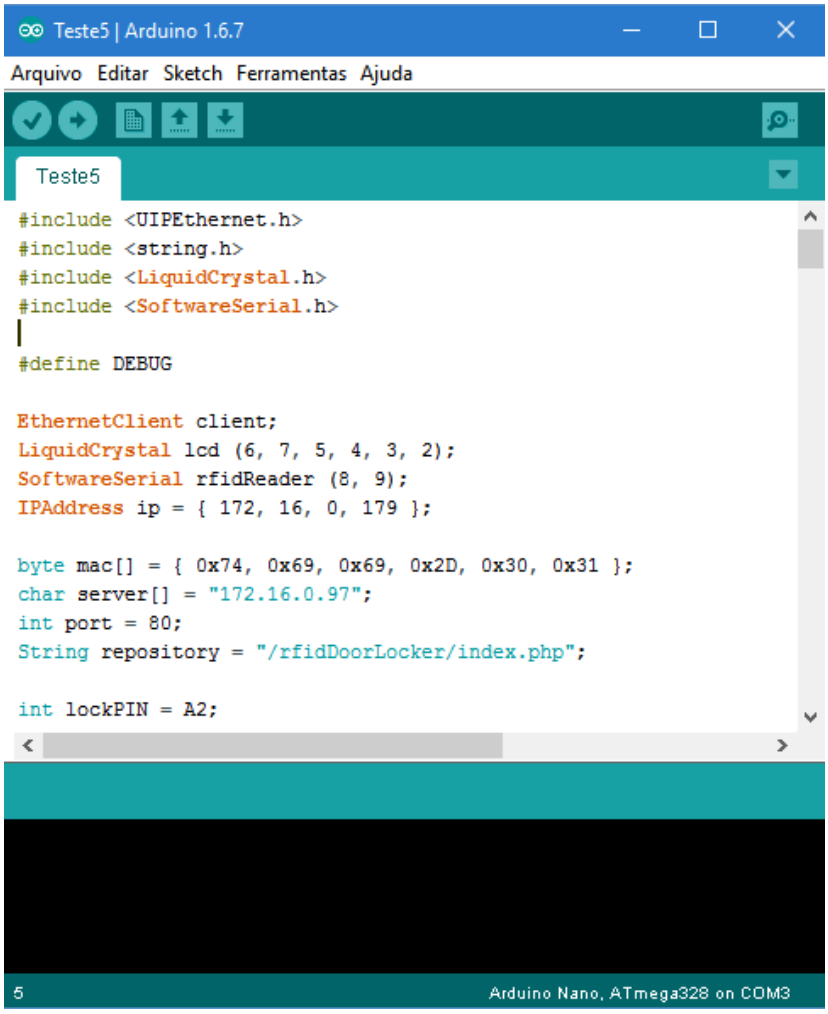

Figura 3.9 - IDE do Arduino com um trecho de código utilizado no projeto. Fonte: elaborado pelo autor

O fato da plataforma Arduino ser *Open Source*, ter um baixo custo e ser compatível com os módulos Ethernet e leitor de RFID que serão utilizados contribuíram para a sua integração no projeto.

3.7. Sistema Operacional Android

O sistema operacional Android, é um sistema baseado em Java que utiliza o kernel 2.6 do Linux. Ele foi liberado em novembro de 2007 pela Google, juntamente com a OHA (*Open Handset Alliance*), um grupo formado por desenvolvedores de hardware e software. Por ser um sistema de software livre, esse sistema é utilizado pela maioria das empresas que vendem *tablets* e *smartphones*, dentre elas a Motorola, Samsung e Sony (DIMARZIO, 2008).

A Figura 3.10 mostra o *smartphone* Moto X Play da empresa Motorola rodando o sistema operacional Android que será utilizado neste projeto. O sistema Android foi escolhido por já estar difundido no mundo inteiro, com mais de um milhão de ativações de dispositivos por dia segundo a Google e por possuir diversas APIs já desenvolvidas e prontas para serem utilizadas como bibliotecas do projeto em questão.

Figura 3.10 - Moto X Play rodando o Sistema Operacional Android. Fonte: Disponível em <http://www.motorola.com.br/products/moto-x-play>

O desenvolvimento do aplicativo utilizará da IDE Android Studio, que foi desenvolvida pela própria Google e é otimizada para o desenvolvimento de aplicativos Android.

3.8. Linguagens de Programação

Linguagem de programação não é nada mais que instruções que são interpretadas pelo hardware dos computadores, nesse caso, linguagem de máquina. A linguagem de máquina é a “linguagem natural” que os computadores conseguem compreender e são reduzidas à números binários, 0s e 1s. Por serem difíceis de serem compreendidas, os programadores começaram a utilizar abreviações em inglês para substituir a fim de substituir a escrita em binário e assim foi criada a linguagem *assembler*, que necessitam de programas tradutores para converter as abreviações na linguagem binária que os computadores entendem (H. DEITEL e P. DEITEL, 2003).

Com o passar dos anos e a popularização do uso dos computadores, foram criadas as linguagens de alto nível, como C e Java, para diminuir a quantidade de instruções necessárias para realizar tarefas mais simples e acelerar o processo de programação. Programas chamados de compiladores foram criados para conseguirem traduzir as linguagens de alto nível para a linguagem de máquina e posteriormente criaram programas interpretadores para diminuir o tempo necessário para executar um programa sem a necessidade de os compilar para linguagem de máquina (H. DEITEL e P. DEITEL, 2003).

Neste projeto, serão utilizadas linguagens de alto nível e programação orientada à objetos, que segundo H. Deitel e P. Deitel (2003) *“são essencialmente componentes de software reutilizáveis que modelam itens do mundo real”*. Comparada à programação estruturada, ela é mais produtiva, fácil de entender e corrigir (H. DEITEL e P. DEITEL, 2003). As linguagens abordadas aqui serão as linguagens C, C++, Java e PHP.

3.8.1. Linguagem C++

A linguagem C++ foi desenvolvida a partir da linguagem C, desenvolvida por Dennis Ritchie e foi utilizada pela primeira vez em 1972. A linguagem C possui uma programação orientada a estrutura e foi popularizada com o desenvolvimento do sistema operacional UNIX e apesar disso, só foi padronizada em 1989 após a criação de diversas variações incompatíveis (H. DEITEL e P. DEITEL, 2003).

Bjarne Stroustrup, no início da década de 1980, criou a extensão da linguagem C conhecida como C++. Além de aprimorar a versão anterior, também possibilitou a programação com orientação a objetos, a programação estruturada, ou ambas (H. DEITEL e P. DEITEL, 2003).

Mesmo sendo uma linguagem orientada a objetos, também possui características de linguagens estruturadas e de baixo nível, possuindo uma portabilidade que não a torna dependente do *hardware* em que está sendo utilizada. Por isso, é uma linguagem extremamente rápida em termos de compilação e execução de instruções, além de permitir sua programação em qualquer computador e possuir uma simples curva de aprendizado.

O Arduino faz uso das linguagens de programação C e C++ na sua IDE para o desenvolvimento e compilação dos seus códigos fonte.

3.8.2. Linguagem Java

Em 1991, a empresa Sun Microsystems criou um projeto visando o uso dos microprocessadores em dispositivos eletrônicos inteligentes que recebeu o codinome Green. A partir disso, James Gosling criou uma linguagem baseada nas linguagens C e C++ que posteriormente foi chamada de Java (H. DEITEL e P. DEITEL, 2003).

Mas foi somente em 1995, após ter passado por diversas dificuldades, que a linguagem Java fora anunciada pela Sun, graças ao *boom* da *Wold Wide Web* que trouxe grande interesse à essa linguagem. Segundo H. Deitel e P. Deitel (2003), *“Java é agora utilizada para criar páginas da Web com conteúdo interativo e dinâmico [...], fornecer aplicativos para dispositivos destinados ao consumidor final [...] e para muitas outras finalidades. ”*

Atualmente, por ser uma linguagem muito robusta, de fácil aprendizado e possuir compatibilidade com qualquer tipo de plataforma, é uma das principais linguagens para o desenvolvimento de aplicações e soluções *Web* e está traçando o seu caminho para se tornar a principal linguagem de desenvolvimento móvel, com o crescente aumento dos dispositivos que utilizam o sistema Android, que segundo a

IDC – Corporação Internacional de Dados (2015), chegou à uma participação de mercado de 82,8% no mundo inteiro.

Logo, por também ser uma linguagem orientada a objetos como a C++ e possuir diversos códigos genéricos, que são disponibilizados até pela Google, o desenvolvimento de aplicativos Android pode ser realizado de forma simplificada. Portanto, o Java foi escolhido para o desenvolvimento do aplicativo Android por ser a linguagem de programação nativa deste sistema operacional.

3.8.3. Linguagem PHP

O PHP, que é um acrônimo recursivo de PHP: *Hypertext Preprocessor* (Pré-processador de hipertexto), é uma linguagem *open source* muito utilizada para o desenvolvimento *web*. Esta linguagem foi criada em 1994 por Ramus Lerdof, e após diversas atualizações chegou à sua última versão em 2004, o PHP 5.0 (PHP.net, s.d.).

Ele possui uma sintaxe que se assemelha bastante à sintaxe das linguagens C, Java e Perl, sendo bastante simples de se aprender e programar sem deixar de oferecer recursos avançados para programadores experientes. Um destaque do PHP, é que o código é executado no lado do servidor, desta forma o usuário não sabe qual é o código fonte já que ele só irá ler o código HTML gerado pelo servidor (PHP.net, s.d.).

Um diferencial desta última versão, é que ela possui suporte nativo para o banco de dados SQLite, facilitando o seu uso neste projeto, além da sua facilidade para uso como já foi dito.

Todos os conceitos técnicos e bases teóricas apresentados neste capítulo serão de grande importância para que o desenvolvimento do projeto seja completado com sucesso, como pode ser visto no próximo capítulo.

CAPÍTULO 4 – DESENVOLVIMENTO DA SOLUÇÃO PROPOSTA

Este capítulo irá apresentar o desenvolvimento do sistema de controle de ponto com o auxílio de um aplicativo Android. Nele, serão apresentados os passos necessários para a montagem e desenvolvimento do controle de ponto, do desenvolvimento do aplicativo Android e do desenvolvimento do servidor de banco de dados levando em consideração todos os conceitos apresentados no capítulo anterior.

4.1. Apresentação Geral da Solução

Este projeto propõe a implementação de um sistema de controle de ponto fazendo uso da plataforma Arduino através de um módulo de RFID. Portanto, o sistema realiza a captura das informações do cartão RFID do usuário e as insere em um banco de dados localizado em um servidor local, permitindo que este usuário possa acessar essas informações por meio de um aplicativo Android. Esta proposta será utilizada para facilitar e proporcionar transparência no controle de ponto tanto para a empresa quanto para o funcionário.

Primeiramente, o funcionário da empresa terá um cartão RFID vinculado ao seu usuário. Ao entrar na empresa, ele terá que posicionar este cartão em frente ao dispositivo de controle de ponto que através do leitor de RFID irá identificar o cartão e armazenar a hora que foi batido o ponto de acordo com o usuário que está relacionado àquele cartão RFID.

Em um segundo momento, um aplicativo Android será desenvolvido para acessar o banco de dados aonde foram armazenados os horários das batidas do ponto de acordo com cada usuário. Dessa forma o usuário poderá acompanhar como está a sua folha de frequência no mês e quantas horas faltam para completar a carga horária do dia.

O diagrama de blocos mostrado na Figura 4.1 exibe de forma simplificada o funcionamento do projeto proposto.

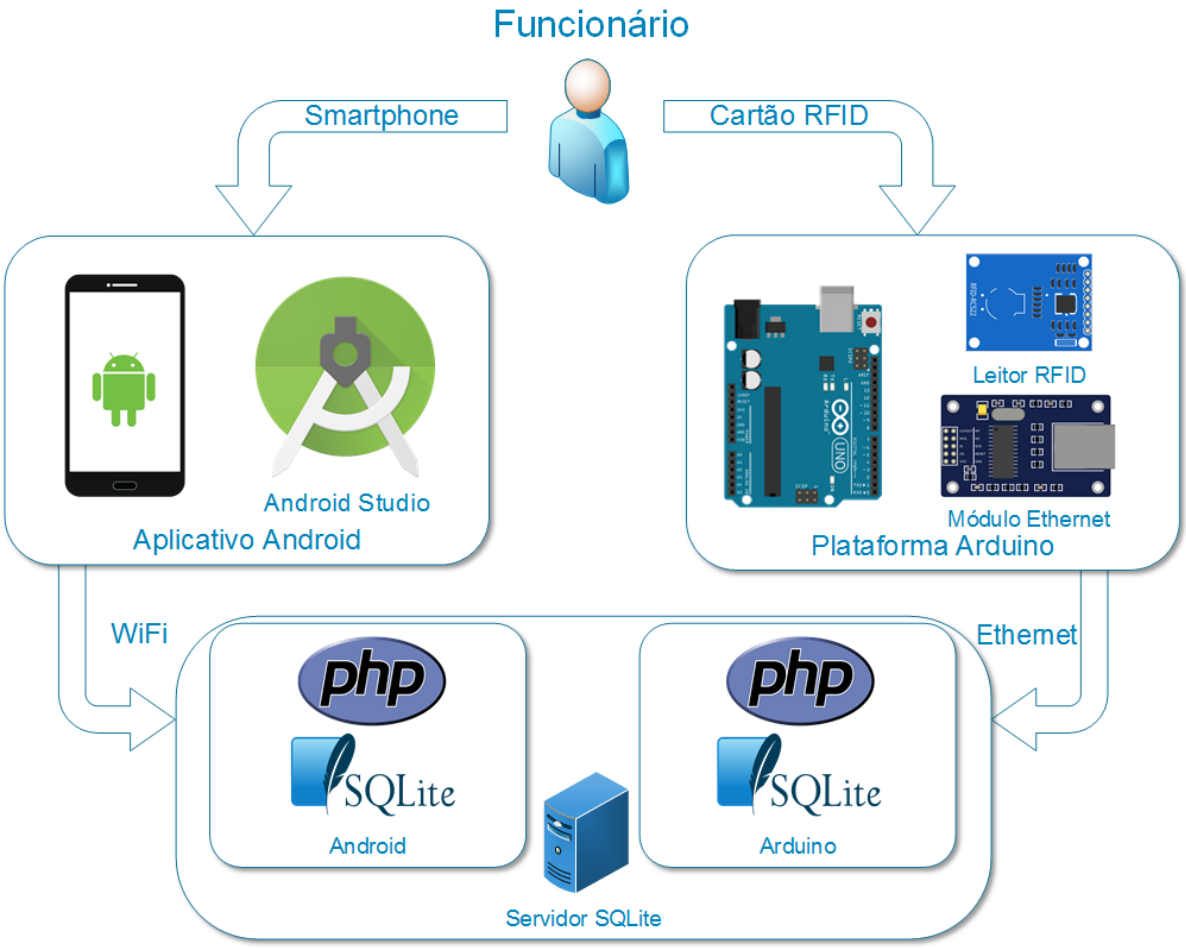

Figura 4.1 - Diagrama de blocos do projeto. Fonte: Elaborado pelo autor.

Para o armazenamento das informações do ponto no banco de dados, será estabelecida a comunicação, com tecnologia Ethernet, entre a plataforma Arduino e um servidor de banco de dados SQLite. Já para a consulta pelo aplicativo Android, este precisará estar na mesma rede local que o servidor, acessando-o via WiFi.

Assim, o desenvolvimento do projeto foi dividido em cinco etapas:

- Desenvolvimento na Plataforma Arduino;
- Desenvolvimento do Aplicativo Android;
- Desenvolvimento do Banco de Dados;
- Desenvolvimento do Servidor Arduino;
- Desenvolvimento do Servidor Android.

4.2. Descrição das Etapas

Neste tópico serão apresentados todos os passos necessários para o desenvolvimento de cada uma das etapas mencionadas no tópico acima.

4.2.1. Desenvolvimento na Plataforma Arduino

O primeiro passo para o desenvolvimento com uso da plataforma Arduino, como dispositivo de ponto, foi a aquisição dos componentes necessários para o correto funcionamento, tendo essa plataforma como base. Para complementar a placa Arduino Nano que está sendo utilizada, os componentes que foram utilizados são: o *display* LCD – *Liquid Crystal Display* (Display de Cristal Líquido), um módulo Ethernet e um módulo RFID.

O modelo de display LCD utilizado possui tamanho de 16x02 com *backlight* (luz traseira) na cor azul e letra na cor branca. Ele é baseado no *chipset* HD44780 da Hitachi, e funciona tanto em modo 4-bit quanto em modo 8bit, mas como ele será utilizado somente para escrever na tela o modo 4-bit é suficiente, fazendo uso de sete portas I/O do Arduino. A Figura 4.2 apresenta o modelo de *display* LCD que será utilizado neste projeto (SPARKFUN, s.d.).

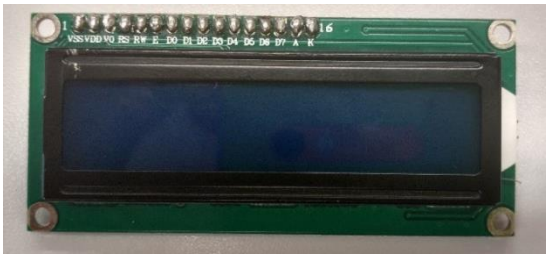

Figura 4.2 - Display LCD baseado no *chipset* HD44780. Fonte: Elaborado pelo autor.

Outro periférico que será utilizado é o módulo Ethernet baseado no *chipset* ENC28J60 da Microship. O módulo possui uma entrada RJ45, um cristal de 25MHz e um conector de apenas 10 pinos, facilitando o uso desse módulo por micro controladores. Sua comunicação com o Arduino é via SPI (*Serial Peripheral Interface*

– Interface Periférica Serial), um protocolo síncrono de comunicação serial, tendo suporte para pacotes *Unicast*, *Multicast* e *Broadcast*. Além disso, opera em uma tensão de 3,3V. O periférico utilizado é este que está sendo mostrado na Figura 4.3 (MICROSHIP, s.d.).

Figura 4.3 - Módulo Ethernet ENC28J60. Fonte: Elaborado pelo autor.

Por fim, o último componente utilizado é o leitor RFID que utiliza o *chip* MFRC522 feito pela empresa NXP, sendo capaz de se comunicar a uma frequência de 13,56MHz. Trabalhando em uma tensão de 3,3V, ele possui um cristal de 27MHz, taxa de transferência de 10 Mb/s e também faz uso do protocolo SPI para se comunicar com o Arduino. A Figura 4.4 - Módulo RFID RC522 com o cartão e a *tag* que o acompanham. Fonte: Elaborado pelo autor. mostra o módulo juntamente com uma *tag* e um cartão que conseguem se comunicar com ele na frequência de 13,56MHz (NXP, s.d.).

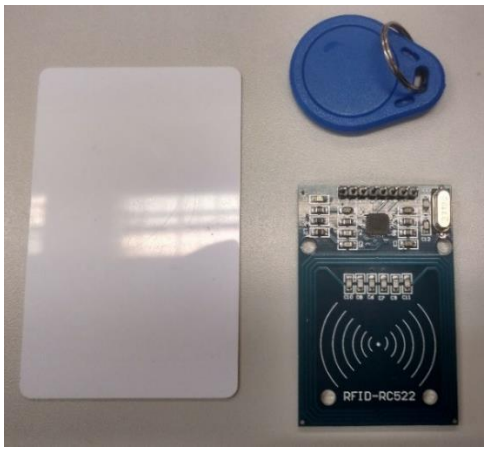

Figura 4.4 - Módulo RFID RC522 com o cartão e a *tag* que o acompanham. Fonte: Elaborado pelo autor.

Terminada a aquisição dos componentes, o próximo passo foi o desenvolvimento de um protótipo do dispositivo de controle de ponto. Para isso, cada um dos periféricos foi testado separadamente para validar seu funcionamento e código fonte e depois foi feita a conciliação para gerar apenas um código fonte.

O primeiro periférico que teve seu código desenvolvido foi o display de LCD, por ser mais simples de testar. Foi montado um pequeno circuito, utilizando a ferramenta gratuita Fritzing (Disponível em: <http://fritzing.org/>), como o mostrado na Figura 4.5 - Circuito para testes do display LCD. Fonte: para a realização dos testes. Na figura podemos ver o display LCD conectado ao Arduino, além de um potenciômetro para controlar a intensidade da luz de fundo do display.

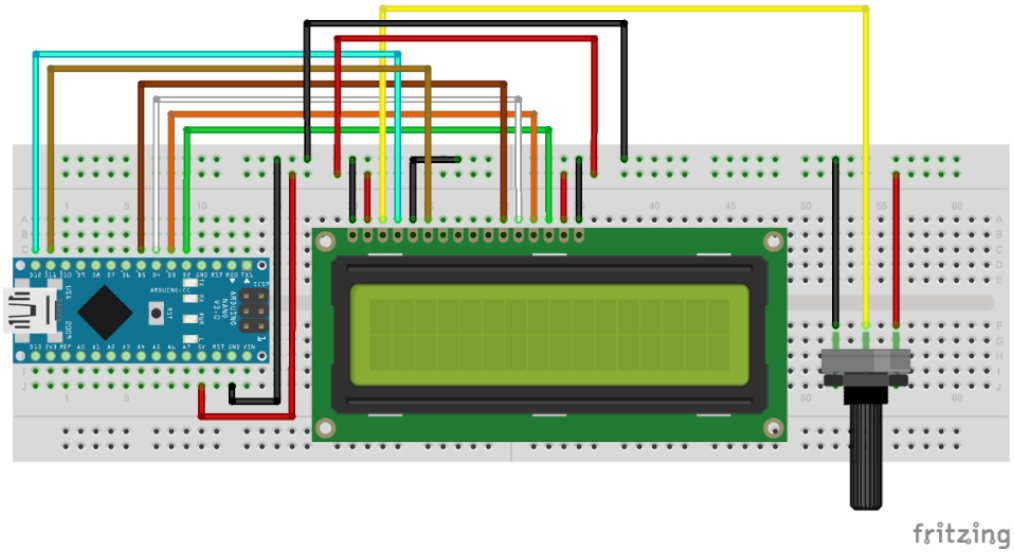

Figura 4.5 - Circuito para testes do display LCD. Fonte: Elaborado pelo autor.

Para este teste foi utilizado a biblioteca *LiquidCrystal*, que por padrão já está inclusa na IDE do Arduino, sendo que a definição dos pinos do display que serão conectados ao Arduino foi feita da seguinte forma:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Para o teste, foi utilizada somente a função void setup() do Arduino, que é uma função que vai executar somente uma vez quando este for ligado. Por isso, foi feita a inicialização do display com o número de colunas e linhas que são utilizadas, foi posicionado o cursor na coluna e linha 0 para que fosse escrito uma frase de teste no display, como mostra o código abaixo:

```
//Define o número de colunas e linhas do LCD
lcd.begin(16, 2);
//Posiciona o cursor na coluna 0, linha 0x10
lcd.setCursor(0, 0);
//Mostra o texto entre aspas no display
lcd.print("Teste de LCD");
```

Constatado o funcionamento do display LCD, o próximo periférico a ser testado foi o módulo de Ethernet. O circuito utilizado pode ser visto na Figura 4.6 - Circuito para testes do módulo Ethernet. Fonte: Elaborado pelo autor. e foi necessário utilizar uma biblioteca *open source* (código aberto) disponibilizada no repositório do GitHub, chamada *UIPEthernet* para que o módulo funcionasse com a plataforma Arduino.

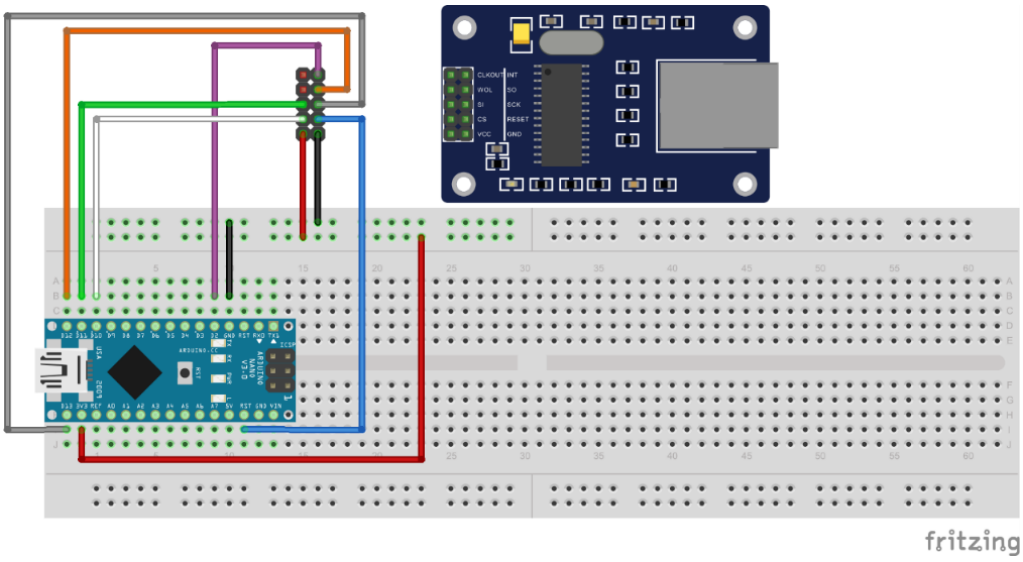

Figura 4.6 - Circuito para testes do módulo Ethernet. Fonte: Elaborado pelo autor.

Após a montagem do circuito, algumas linhas de código foram inseridas para configurar o módulo Ethernet com o objetivo de definir qual será o seu endereço IP e MAC *Address* e qual porta será criado o servidor de teste. O trecho de código abaixo mostra quais comando foram utilizados com esta finalidade:

```
// Cria uma instância do cliente
EthernetClient client;
// Cria uma instância do servidor na porta 1000
EthernetServer server = EthernetServer(1000);
```

```
// Define qual ip será utilizado pelo módulo
IPAddress ip = {192, 168, 1, 128 };
// Define qual mac será utilizado pelo módulo
byte mac[] = { 0x74, 0x69, 0x69, 0x2D, 0x30, 0x31 };
```

Na função *void.setup()*, assim como no exemplo anterior, foi feita a inicialização do módulo passando os parâmetros do MAC Address e do IP que ele vai utilizar e logo em seguida do servidor.

```
Ethernet.begin(mac, ip);
server.begin();
```

Feita a inicialização do módulo e do servidor, para verificar se o Arduino conseguiu criar uma instância do servidor utilizou-se o comando *telnet* feito em um terminal de um computador Windows que estivesse na mesma rede para validar o funcionamento correto do módulo.

Para finalizar os testes dos periféricos, o leitor de RFID mostrou-se bem simples de configurar e utilizar com a plataforma Arduino. A montagem de seu circuito elétrico com o Arduino pode ser vista na Figura 4.7 - Circuito para testes do leitor RFID. Fonte: Elaborado pelo autor., e assim como o módulo Ethernet é alimentado pela porta de 3,3V do Arduino. Foi necessário fazer o download de uma biblioteca *open source*, que assim como o módulo Ethernet, está localizada no repositório do GitHub chamada MFRC522, mesmo nome do leitor RFID.

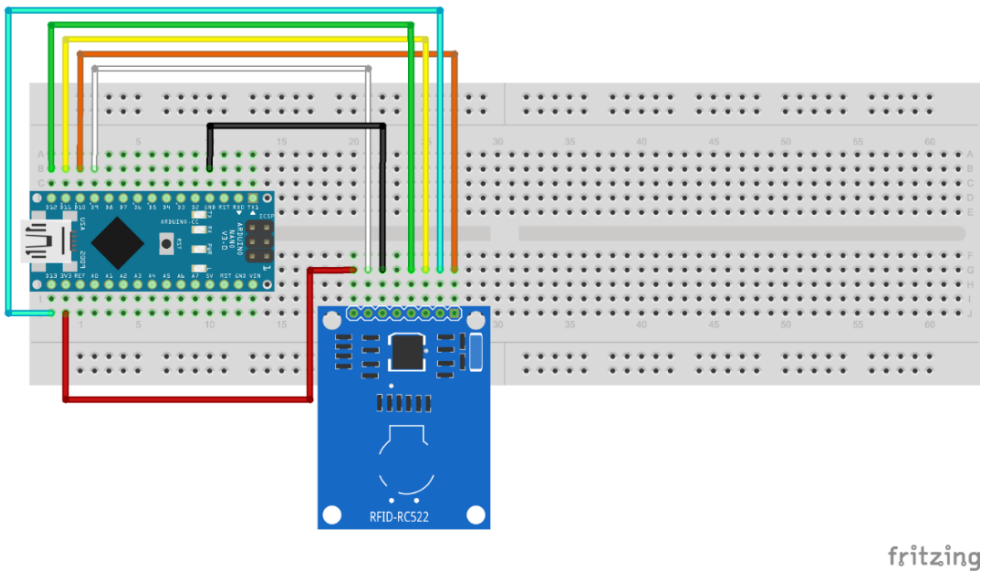

Figura 4.7 - Circuito para testes do leitor RFID. Fonte: Elaborado pelo autor.

Para utilizar o leitor com sucesso na plataforma Arduino, as seguintes linhas de código foram adotadas para importar as bibliotecas necessárias e criar a instância do leitor nos pinos 10 e 9 do Arduino.

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9
// Cria a instância do leitor MFRC522 nos pinos 10 e 9
MFRC522 rfid(SS_PIN, RST_PIN);
```

Em seguida, dentro da função *void setup()* é feita a inicialização do protocolo SPI e do leitor RFID enquanto na função *void loop()*, o Arduino fica à espera da leitura de um cartão ou *tag* para então mostrar na serial qual o ID do cartão/*tag* lidos, como pode ser visto no trecho de código abaixo.

```
void setup() {
  // Inicia a serial
  Serial.begin(9600);
  // Inicia SPI bus
  SPI.begin();
  // Inicia MFRC522
  rfid.PCD_Init();
}
void loop() {
  // Verifica a aproximação de um cartão/tag
  if ( ! rfid.PICC_IsNewCardPresent()) {
    return;
  }
  // Verifica se foi aproximado um cartão ou uma tag
  if ( ! rfid.PICC_ReadCardSerial()) {
    return;
  }
  //Mostra o ID na serial do Arduino
  Serial.print("UID da tag :");
  for (byte i = 0; i < mfrc522.uid.size; i++) {
```

```
    Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(rfid.uid.uidByte[i], HEX);
  }
}
```

Ao executar este pequeno trecho de código, o monitor serial da plataforma Arduino apresentava o ID de todos os cartões e *tags* lidos pelo leitor RFID. A Figura 4.8 - Teste de leitura de RFID do cartão e *tag*. Fonte: Elaborado pelo autor. mostra a tela do monitor serial e dois IDs que foram lidos durante este teste.

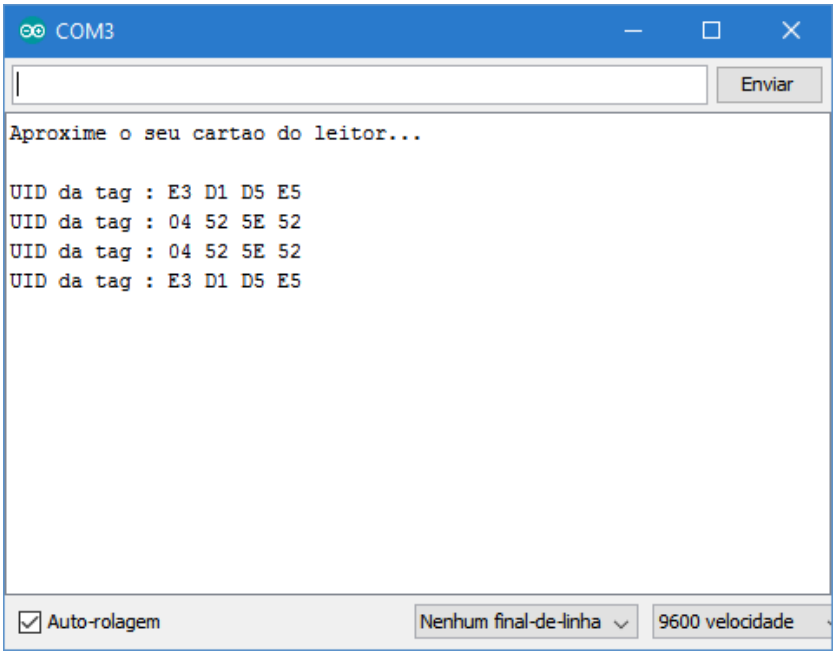

Figura 4.8 - Teste de leitura de RFID do cartão e *tag*. Fonte: Elaborado pelo autor.

Feita a verificação de todos os três periféricos, o próximo passo foi realizar a combinação dos códigos fontes de todos eles em somente um, para o desenvolvimento do protótipo que será utilizado para o controle de ponto. Na hora de montar o circuito, o maior problema encontrado foi a união de dois periféricos que utilizam do protocolo SPI para se comunicar com a plataforma Arduino e desta forma deveriam utilizar as mesmas portas digitais do Arduino Nano.

No protocolo SPI, a plataforma Arduino age como dispositivo mestre e seus periféricos, no caso o leitor RFID e o módulo Ethernet, agem como dispositivos escravos. Portanto, estes dois dispositivos escravos utilizam as mesmas portas

digitais para receber dados do dispositivo mestre, enviar dados para a plataforma Arduino e para sincronizar o sinal de todos os dispositivos escravos no mesmo *clock*.

A única mudança é na linha de sinal que controla com qual escravo o mestre vai se comunicar. Logo, utilizando a porta digital 10 (dez) do Arduino para o módulo Ethernet e a porta digital 8 (oito) para o leitor RFID foi possível fazer com que os dois periféricos funcionassem, contanto que somente um periférico fosse utilizado por vez.

Portanto, Tabela 4.1 lista a pinagem utilizada pelos periféricos ao se conectarem com o Arduino Nano, a Figura 4.9 apresenta o circuito do protótipo final do dispositivo de ponto fazendo uso de todos os três periféricos apresentados neste tópico e a Figura 4.10 apresenta o circuito real montado em uma *protoboard* com a utilização de *jumpers* para realizar a conexão dos periféricos.

Tabela 4.1 - Lista de pinagem dos periféricos utilizados com o Arduino Nano.

Arduino	LCD	Leitor RFID	Módulo Ethernet
D02			INT
D03	D6		
D04	D5		
D05	D4		
D06	RS		
D07	E		
D08		SDA	
D10			SDA
D11		MOSI	MOSI
D12		MISO	MISO
D13		SCK	SCK
A2	D7		

Fonte: Elaborado pelo autor.

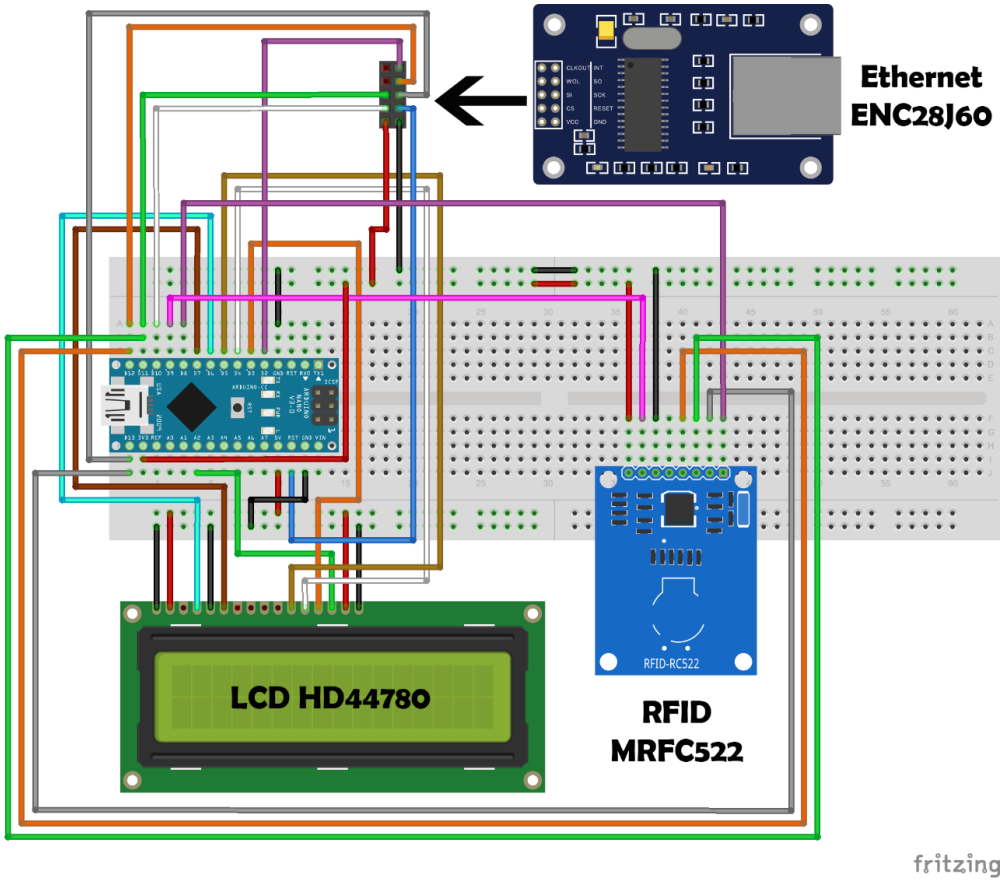

Figura 4.9 - Circuito do protótipo feito no Fritzing. Fonte: Elaborado pelo autor.

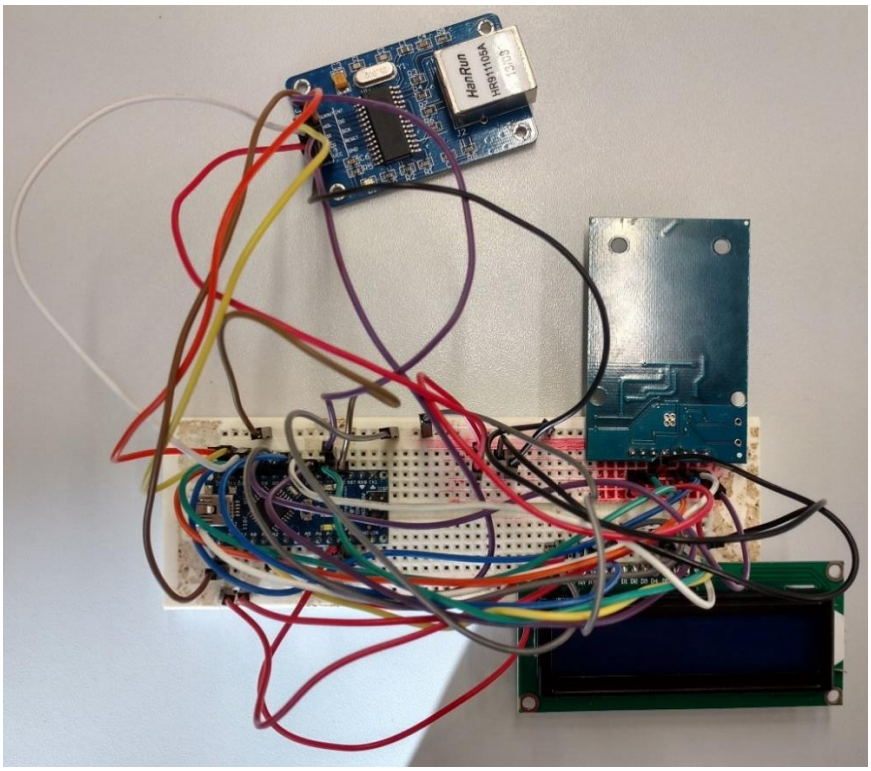

Figura 4.10 - Protótipo do dispositivo de ponto montado. Fonte: Elaborado pelo autor.

Desta forma, o combinado de todos os códigos de testes dos periféricos em um código fonte final, faz uso de todas as ferramentas e funções previamente utilizadas. O trecho de código abaixo apresenta a primeira parte de código, com a importação de todas as bibliotecas que serão utilizadas, a definição dos pinos de comunicação do leitor RFID, nas portas 8 e 9 do Arduino, a definição dos pinos que serão utilizados pelo leitor LCD e por fim, toda a configuração do módulo Ethernet, com o IP que ele utilizará, o IP e o caminho do servidor que ele vai se conectar e qual MAC *Address* ele vai utilizar.

```
#include <MFRC522.h>
#include <UIPEthernet.h>
#include <string.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal.h>
#include <SPI.h>
//#define DEBUG
#define SS_RFID_PIN 8
#define RST_RFID_PIN 9
LiquidCrystal lcd ( 6, 7, 5, 4, 3, A2 );
EthernetClient client;
IPAddress ip = { 192, 168, 1, 128 };
byte mac[] = { 0x74, 0x69, 0x69, 0x2D, 0x30, 0x31 };
char server[] = "192.168.1.4";
String repository = "/WebServer/consultas/arduino.php";
int port = 80;
```

Continuando pelo código fonte final, é possível notar no trecho de código abaixo que foi utilizado um *delay* de 100 milissegundos entre o uso do leitor RFID e do módulo de Ethernet para que não houvesse conflito na hora do Arduino fazer uso de cada um desses periféricos.

```
#ifdef DEBUG
  Serial.print ( "UID da tag :" );
#endif
for (byte i = 0; i < rfid.uid.size; i++) {
  [...]
```

```
  rfidTag.concat(String(rfid.uid.uidByte[i] < 0x10 ? "0" : ""));
  rfidTag.concat(String(rfid.uid.uidByte[i], HEX));
}
delay ( 100 ); // Pausa entre a comunicação do leitor RFID e do módulo Ethernet
if ( !client.connect ( server, port ) ) {
  [...]
  Enc28J60.init ( mac );
} else {
  if ( sendRequest ( rfidTag )) {
    String ret = getResponse();
    #ifdef DEBUG
      Serial.println ( "Resposta[ " + ret + " ]" );
    #endif
  [...]
}
```

Ao final do desenvolvimento do código fonte do protótipo, o *display* LCD apresenta uma mensagem de que está aguardando a aproximação de um cartão ou *tag* LCD. Ao aproximar um cartão válido, que esteja cadastrado no banco de dados e vinculado a um funcionário, o display LCD vai apresentar o nome do funcionário e a mensagem de que o ponto foi batido com sucesso.

Caso contrário, se o cartão não estiver cadastrado, o cartão não estiver vinculado a um funcionário, não for feita a conexão com o servidor ou houver alguma falha na hora de armazenar ou buscar um dado no banco de dados, o display LCD irá mostrar uma mensagem de erro para o usuário.

4.2.2. Desenvolvimento do Aplicativo Android

O aplicativo foi desenvolvido unicamente com o uso do programa Android Studio e da linguagem de programação Java em conjunto com o padrão XML (*eXtensible Markup Language*) para a criação do *layout* do aplicativo.

O aplicativo é dividido em duas telas, a de *login* e a de visualização do ponto. Por padrão, todo aplicativo Android é iniciado com a chamada da função *onCreate*

que define a instância do aplicativo na memória do dispositivo e o *layout* principal a ser utilizado.

A classe principal do aplicativo chama-se PontoActivity.java, e é responsável pela tela de *login* que o usuário irá utilizar. Como pode ser visto na Figura 4.11, a tela de *login* possui um campo de texto para que o usuário digite o seu email cadastrado no banco de dados, um campo de texto protegido para que o usuário digite a senha correspondente, a opção do aplicativo lembrar o *login* na próxima vez que for iniciado e o botão de logar no aplicativo.

Figura 4.11 - Tela de *login* do aplicativo. Fonte: Elaborado pelo autor.

O botão de logar é responsável pelas ações mais importantes que estão presentes na classe PontoActivity, como o envio do *login* e senha digitados pelo usuário para o servidor e o tratamento da resposta que o servidor retorna para o aplicativo. O trecho de código abaixo é responsável por identificar quando o botão de logar é pressionando, a função *fetchLogin* recebe os parâmetros de *login* e senha do usuário. Em seguida, a classe HttpRetriever.java é acionada para realizar a requisição de *login*, também recebendo os mesmos parâmetros que a função *fetchLogin* recebeu.

```
loginButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        [...]
        fetchLogin(emailLogin.getText().toString(), paswdLogin.getText().toString());
    }
});
```

O texto retornado pelo servidor de banco de dados não é completamente legível e por isso recebe um tratamento para que as informações necessárias sejam armazenadas em variáveis criadas para permitir que o usuário acesse a tela que possui o calendário e as informações de ponto. A Tabela 4.2 mostra o texto retornado pelo servidor e como ele fica após este tratamento citado acima.

Tabela 4.2 - Texto retornado pelo servidor e resultado do tratamento que ele sofre.

Texto retornado pelo servidor: {"cpf":"04136321139","nome":"Lucas Gennari","perfil":"PERFIL USER"}
CPF: 04136321139
Nome: Lucas Gennari
Perfil: PERFIL USER

Fonte: Elaborado pelo autor.

Se o servidor retornar uma mensagem positiva, como a apresentada na Tabela 4.1, a classe PontoActivity faz uma chamada para a classe UserCalendar.java, que é a classe que o usuário poderá visualizar as informações de ponto. Essa classe recebe o CPF, para conseguir realizar as buscas de ponto no servidor e o nome do usuário, somente para apresentar na tela. O *layout* dessa classe é a apresentação do calendário, já com o dia atual selecionado, e dois botões, um para realizar o cálculo do ponto semanal e outro para o cálculo mensal. No calendário, ao selecionar qualquer dia, é identificado a seleção do dia e um *popup* será apresentado com os horários de entrada e saída que foram recebidos do servidor e o total de horas feito naquele dia, como pode ser visto na Figura 4.12.

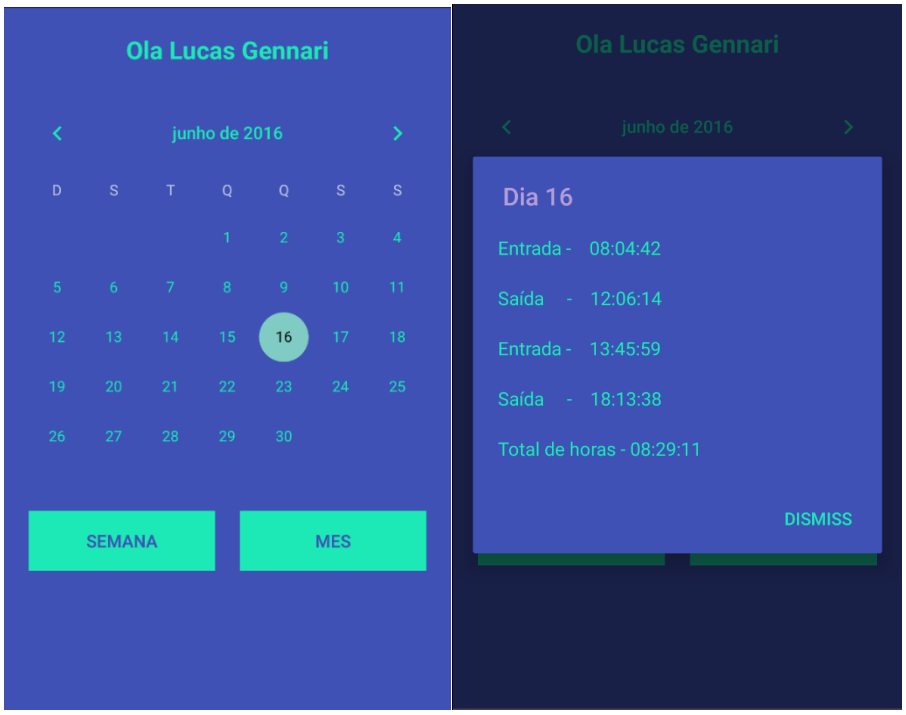

Figura 4.12 - Telas do calendário e das informações de ponto do dia 16 de junho recebidas pelo servidor. Fonte: Elaborado pelo autor.

A classe UserCalendar também possui um método chamado *onCreate*, que é automaticamente executado após o usuário fazer *login* no aplicativo. Nela, é utilizado uma função própria da API do calendário feita pela própria Google que identifica quando o usuário seleciona uma nova data no *layout*. O trecho de código abaixo retrata essa função, que recebe os dados do ano, mês e dia que foi selecionado e logo em seguida os armazena em uma *String* (texto) no padrão utilizado pelo banco de dados.

```
calendarView.setOnDateChangeListener(new
CalendarView.OnDateChangeListener() {
    @Override
    public void onSelectedDayChange(CalendarView view, int year, int month, int
dayOfMonth) {
        month = month + 1;
        String dia = (year + "-" + month + "-" + dayOfMonth);
```

Em seguida, a classe HttpRetriever é chamada para buscar no servidor de banco de dados os horários de entrada e saída do funcionário que retorna um texto com os horários que o ponto foi batido, a identificação se foi uma batida de entrada

ou saída do estabelecimento ou empresa e o total de horas cumpridos naquele dia. Este texto também é tratado, armazenando em uma lista de String todas essas informações, que posteriormente são populadas em uma caixa de diálogo que apresenta todas as informações de forma legível, como apresentado acima na Figura 4.12.

Os botões semana e mês também chamam a classe HttpRetriever que faz a requisição para o servidor, sendo que quando o usuário aperta o botão semana é passado o primeiro e o último dia da semana que tem o dia selecionado e apresenta na tela o total de horas realizado, a carga horária semanal que deve ser cumprida e a diferença entre as duas. Enquanto o botão mês passa o primeiro e o último dia do mês para o servidor e apresenta na tela o total de horas já realizado no mês, a carga horária mensal a ser cumprida e a diferença entre as duas. A Figura 4.13 mostra como estas informações são mostradas para o usuário ao se clicar em qualquer um dos dois botões.

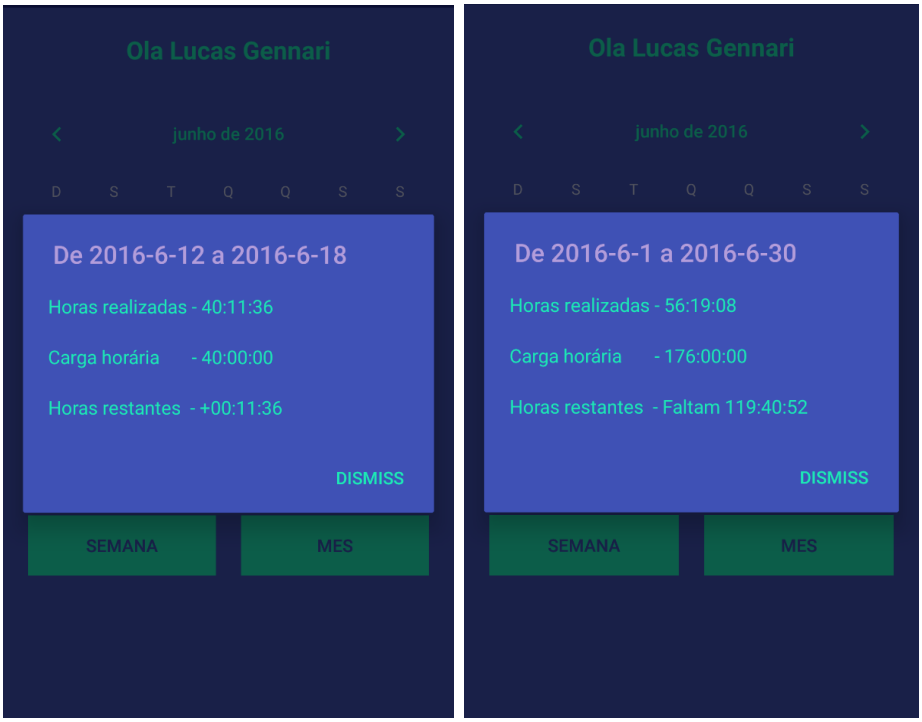

Figura 4.13 – Informações de ponto da semana de 12 a 18 de junho e do mês de junho. Fonte: Elaborado pelo autor.

Enquanto isso, a classe HttpRetriever age em *background* fazendo as requisições ao servidor de banco de dados tanto de login quanto em relação às ações que o usuário faz na tela do calendário. O trecho de código abaixo mostra como é feita a criação da URL (*Uniform Resource Locator* – Localizador Padrão de Recursos) para a requisição de login.

```
URI uri = new URI(properties.getProperty("protocol"),
properties.getProperty("server") + ":" + properties.getProperty("port"), "/" +
properties.getProperty("context") + "/" + properties.getProperty("pathLogin") +
"?login=" + login + "&passwd=" + password, null, null);
String urlStr = URLDecoder.decode(uri.toString(), "UTF-8");
```

Para esta requisição, são passados os parâmetros do protocolo a ser utilizado, que é HTTP, o servidor e a porta para conexão, e o endereço do servidor com as informações de login e senha. Desta forma, é feita a conexão com o servidor e a Tabela 4.3 abaixo mostra a URL de requisição formada e o texto que o servidor retorna para que o aplicativo faça o tratamento.

Tabela 4.3 - URL enviada ao servidor e resposta recebia pelo aplicativo Android.

http://192.168.1.101:80/WebServer/consultas/android_login.php?login=lucas.gennari@gmail.com&passwd=123456
{"cpf":"04136321139","nome":"Lucas Gennari","perfil":"PERFIL USER"}

Fonte: Elaborado pelo autor.

A explicação sobre o funcionamento do servidor com relação ao aplicativo Android está descrita no tópico 4.2.5, onde é detalhado como o servidor faz as buscas no banco de dados e o envio das informações para o aplicativo.

4.2.3. Desenvolvimento do Banco de Dados

Para o desenvolvimento do banco de dados, o primeiro passo foi estudar os requisitos mínimos necessários para o funcionamento do projeto proposto. Desta

forma, foram listados todos os atributos que deveriam ter no banco, bem como todos os atributos que poderiam complementar o próprio banco de dados.

Assim que foram listados todos estes atributos, eles foram separados em entidades de acordo com o seu papel. As entidades foram divididas em:

- RFID – armazena todos os IDs dos cartões e tag utilizados;
- Pessoa – armazena dados do usuário para o controle de ponto;
- Ponto – armazena a data em que o ponto foi batido;
- Batidas – armazena a hora em que os pontos foram batidos;
- Feriado – armazena informações das datas dos feriados nacionais.

Uma vez que foram definidas todas as entidades do banco, o próximo passo foi fazer o relacionamento entre estas entidades. A entidade RFID só precisa estar relacionada à entidade Pessoa, e como cada pessoa só terá um cartão ou *tag* em sua posse, ou seja, foi criado um relacionamento um para um entre elas.

Seguindo esta lógica, todo usuário vai bater o ponto durante todos os dias úteis do mês, logo, o relacionamento entre a entidade Pessoa e a entidade Ponto é de um para muitos, lembrando que uma mesma Pessoa não poderá ter mais de um registro de Ponto com a mesma data. A relação entre a entidade Ponto e a entidade Batidas parte do mesmo princípio, uma vez que em uma mesma data o usuário vai ter diversas batidas de ponto com horas diferentes referentes às entradas e saídas do local de trabalho, fazendo com que também seja uma relação um para muitos.

Para finalizar, a entidade Feriado não necessita de relação com nenhuma entidade, pois ela só vai armazenar as informações de todos os feriados nacionais, não deixando o aplicativo Android penalizar o usuário por não bater o ponto no dia do feriado.

A Figura 4.14 - Modelo Entidade Relacional do banco de dados utilizado no projeto. Fonte: Elaborado pelo autor. apresenta o MER feito durante o desenvolvimento do banco de dados de acordo com as informações que foram escritas acima. Foi utilizado a ferramenta para a criação deste diagrama chamada MySQL Workbench disponibilizada gratuitamente no site da linguagem MySQL.

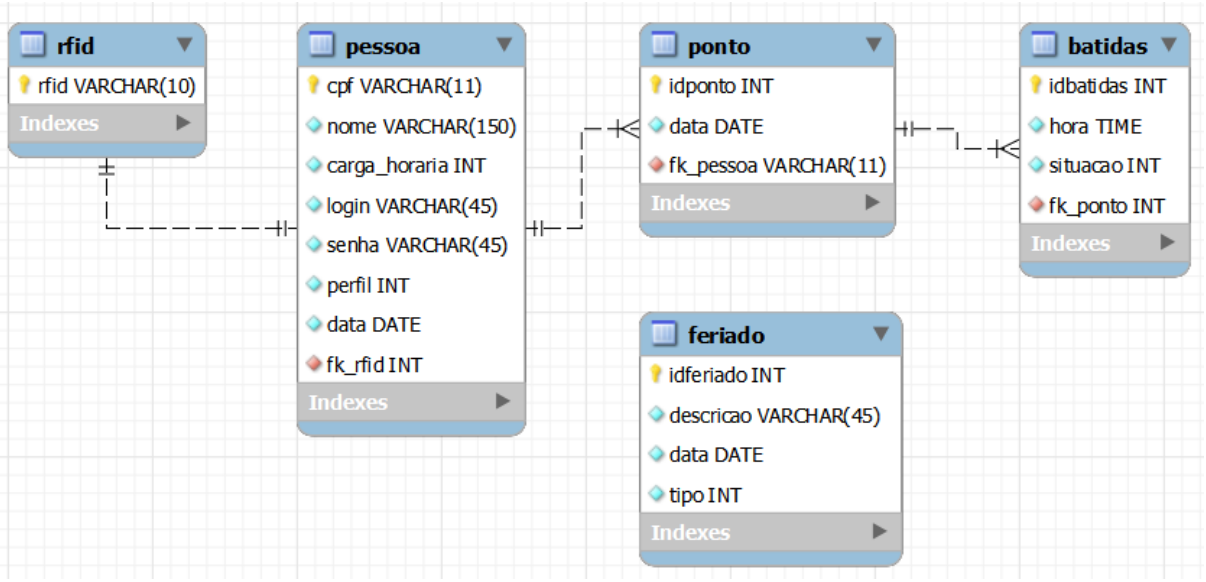

Figura 4.14 - Modelo Entidade Relacional do banco de dados utilizado no projeto. Fonte: Elaborado pelo autor.

4.2.4. Desenvolvimento do Servidor Arduino

Como dito no Capítulo 3, o servidor que se comunica com a plataforma Arduino foi desenvolvido com a linguagem de programação PHP, porém, foi necessário a instalação do programa XAMPP para a criação e configuração de um servidor *web* Apache com a conveniência de já conter todos os recursos ativados que permitam que o servidor receba requisições *web* e retorne uma resposta para o dispositivo que pediu as informações.

Após a instalação do programa, é necessário criar uma pasta dentro do diretório “C:\xampp\htdocs\” que vai abrigar todo o código referente ao servidor Arduino, Android e também vai abrigar o banco de dados. A pasta criada se chama “WebServer” e o arquivo com os códigos do servidor Arduino se encontra no caminho “consultas\arduino.php”. Portanto, sempre que for feita uma requisição *web* com o IP da máquina onde está o servidor seguida de “\WebServer\consultas\arduino.php” o código do servidor será executado.

A primeira parte do código define o *header* da página e mensagens de alerta, informação e erro que são utilizadas para ajudar a depurar o código. Logo em seguida, é feita a identificação de uma requisição GET com o parâmetro RFID e a verificação

se ele está vazio. Caso não esteja vazio, inicia-se a parte da comunicação com o banco de dados para validar o ID que foi identificado.

Primeiramente é salvo em uma variável, chamada $rfid, o ID que foi recebido pela requisição, também é salvo uma instância do banco de dados SQLite em uma variável e então, uma chamada uma função que irá fazer uma *query* no banco de dados para saber se existe uma pessoa relacionada àquele ID, como pode ser visto no trecho de código a seguir.

```
<?php
    header ("Content-Type:application/json");
    define ('E', 'ERRO');
    define ('I', 'INFO');
    define ('W', 'WARN');
    if (!empty($_GET['rfid'])) {
        $rfid = $_GET['rfid'];
        $database = new SQLite3('../Ponto.db');
        insere_log(I ,"--------------------------------------------------");
        $pessoa = consultaRfid($database, $rfid);
```

Esta *query*, que pode ser vista no trecho de código a seguir, vai fazer uma busca no banco de dados de acordo com o ID recebido pela requisição GET e retornar o CPF e o nome da pessoa, além do ID cadastrado por questões de validação, e vai coloca-los em um objeto chamado Pessoa.

```
$fetch = "SELECT cpf, nome, fk_rfid FROM pessoa WHERE fk_rfid=\"$rfid\"";
```

Em seguida, se o objeto não for retornado com sucesso, é porque o ID recebido não está vinculado a nenhum funcionário no banco de dados. Caso exista um funcionário vinculado a este ID, serão armazenados em duas variáveis diferentes: o dia e a hora em que o cartão ou *tag* foi passado no dispositivo. Com a informação do dia, será feito uma outra *query* que vai identificar se já existe algum registro de batida de ponto naquele dia, como pode ser visto no trecho PHP e SQL abaixo.

```
$result = $database->querySingle("SELECT COUNT (*) as count FROM ponto
WHERE
fk_pessoa=\"$pessoa->cpf_pessoa\" and data=\"$dt_atual\"");
```

Já existindo um registro, serão buscadas no banco as informações armazenadas na tabela Ponto, referente à chave primária da tabela, a data cadastrada

e a chave estrangeira da tabela Pessoa e depois elas serão armazenadas em um objeto chamado Ponto. Abaixo pode ser visto a *query* que foi feita para buscar estas informações no banco de dados.

```
$check_fetch = "SELECT * FROM ponto where fk_pessoa=\"$pessoa->cpf_pessoa\" and data=\"$dt_atual\"";
```

Com estas informações armazenadas em um objeto, a função “insereBatida” é chamada para inserir na tabela Batidas os valores referentes à hora em que foi identificado o ID do cartão. Como pode ser visto no código abaixo, os valores inseridos são a hora, a situação que vai ser o número um ou dois, sendo que um se refere à entrada e dois à saída, e a chave estrangeira do ponto, para que a hora armazenada possa ser vinculada ao dia da batida de ponto.

```
$sql_batida = "INSERT INTO batidas VALUES (null, :hora, :situacao, :fkponto)";
```

Caso não seja encontrado o registro de batida de ponto no dia, a função “inserePonto” é chamada para armazenar na tabela Ponto a data que foi obtida e a chave estrangeira contendo o CPF do funcionário, vinculando as duas tabelas, a partir do trecho de código abaixo.

```
$sql_ponto = "INSERT INTO ponto VALUES (null, :date, :cpfpessoa)";
```

Ao final da função “inserePonto”, a função “insereBatida” é chamada para inserir a hora em que foi registrado o ponto, da mesma forma como foi explicado acima. Após estas alterações no banco, é chamada uma função que vai retornar para a plataforma Arduino o nome do funcionário que acabou de bater o ponto, a data e a hora referentes à batida de ponto e uma confirmação que as alterações no banco de dados foram realizadas com sucesso. Desta forma, a plataforma Arduino pode apresentar no display LCD estas informações para que o usuário saiba que tudo ocorreu bem.

Se em algum momento do código acontecer alguma falha, seja na hora de verificar o ID ou na hora de registrar o ponto, será enviado uma mensagem de falha para a plataforma Arduino informar ao usuário que ocorreu um problema durante o processo.

4.2.5. Desenvolvimento do Servidor Android

O servidor Android também foi desenvolvido como o servidor Android, utilizando a linguagem PHP e o programa XAMPP para a criação do serviço web, que está localizado no caminho “WebServer\android_login.pho” para a parte referente ao login do aplicativo Android e no caminho “\WebServer\android_ponto.php” na parte que se refere aos pedidos do usuário pelas informações de ponto diária, semanal e mensal.

Então, assim como o tópico anterior, o primeiro trecho de código do servidor que pode ser visualizado abaixo recebe o número do CPF do usuário e um número que significa qual tipo de busca que ele deverá fazer no banco de dados, sendo que o número um representa que o usuário deseja saber os horários de ponto de um dia fixo, o número dois representa que ele deseja o total de horas realizados na semana e quanto falta para atingir a carga horária semanal ou quantas horas já passaram dessa carga e por fim, o número três ira retornar par ao usuário o total de horas realizados no mês e quantas horas faltam para completar a carga horária mensal ou quantas horas extras já foram realizadas no mês.

```
if (!empty($_GET['cpf']) && !empty($_GET['tipo'])) {
    $cpf = $_GET['cpf'];
    $tipo = $_GET['tipo'];
    $database = new SQLITE3('../Ponto.db');
```

No primeiro caso, em que o usuário deseja saber somente os horários de ponto do dia, primeiramente é feita uma *query* na tabela ponto do banco de dados utilizando os parâmetros de CPF do usuário e data escolhida, o trecho de código responsável por isso pode ser visto abaixo.

```
$fetchPD = "SELECT idponto FROM ponto WHERE fk_pessoa=\"$fk_cpf\" and data=\"$ponto_date\"";
```

Após recuperar o idponto referente à data e ao CPF do usuário é feita uma consulta na tabela batidas de acordo utilizando esta informação para obter a hora e a situação das batidas de ponto que foram realizadas, referente ao trecho de código descrito abaixo.

```
$fetchBD = "SELECT hora, situacao FROM batidas WHERE fk_ponto=\"$fk_ponto\"";
```

Com o resultado dessa *query* é armazenado em um *array* o histórico de batidas de ponto no dia do usuário, permitindo saber quantas horas foram cumpridas entre os horários de entrada e saída. A partir das horas de permanência em cada momento que o usuário ficou na empresa, é feito um cálculo que retorna o total de horas trabalhadas no dia. Sendo que para a realização deste cálculo foi preciso transformar as horas e minutos para segundos e só então foi feita a soma de todos os segundos que foram trabalhados pelo usuário naquele dia.

Então, o resultado da soma é transformado novamente no padrão de horas, minutos e segundos, para que possa ser retornado para o usuário de uma forma legível. O trecho de código abaixo é responsável por realizar todas estas ações.

```
foreach ($diff as $tempo) {
        list ($h, $m, $s) = explode(':', $tempo);
        $segundos += $h * 3600;
        $segundos += $m * 60;
        $segundos += $s;
}
$horas = str_pad(floor( $segundos / 3600 ), 2, "0", STR_PAD_LEFT) ;
$segundos %= 3600;
$minutos = str_pad(floor( $segundos / 60 ), 2, "0", STR_PAD_LEFT);
$segundos %= 60;
$total = $horas . ":" . $minutos . ":" . str_pad($segundos, 2, "0", STR_PAD_LEFT);
```

No segundo e terceiro caso, onde o usuário busca as informações de horário de ponto semanal em comparação com a carga horária semanal e de ponto mensal em comparação com a carga mensal, a *query* feita na tabela ponto é parecida com a feita no primeiro caso, em que a parte da data é feita de forma que se busque o idponto entre duas datas, que são o primeiro dia da semana e o último ou o primeiro e último dia do mês.

```
$fetchPS = "SELECT idponto FROM ponto WHERE fk_pessoa=\"$fk_cpf\" AND data BETWEEN \"$firstDay\" AND \"$lastDay\"";
```

Isto retorna um *array* com todos os idpontos referentes às datas que foram passadas e assim como no primeiro caso, é feita outra *query* para se buscar as horas e a situação de entrada e saída de todos os dias da semana ou do mês, que pode ser

vista abaixo, sendo que a variável $fk_pontos armazena todos os idpontos que foram recebidos na *query* anterior.

```
$fetchBS = "SELECT hora, situacao FROM batidas WHERE fk_ponto IN ($fk_pontos)";
```

Desta forma, são retornadas todo o histórico de batida de ponto semanal ou mensal, que passa pelo mesmo processo de cálculo feito no primeiro caso. São calculados os segundos de permanência em cada dia e por fim é feito a soma desses segundos, que posteriormente são transformados no padrão de horas, minutos e segundos.

Logo, é possível realizar a conexão com o aplicativo Android e responder a todos os pedidos de histórico de ponto, seja ele diário, semanal ou mensal.

4.3. Considerações Finais

Ao final de todas essas etapas de desenvolvimento, o protótipo de dispositivo de ponto e do aplicativo Android foram finalizados. O código fonte escrito para cada uma dessas etapas pode ser encontrado no DVD anexado à monografia.

Os testes realizados com todo o projeto em funcionamento serão apresentados no Capítulo 5, assim como os resultados encontrados ao final dos testes.

CAPÍTULO 5 – APLICAÇÃO PRÁTICA DA SOLUÇÃO

5.1. Apresentação da Área de Aplicação da Solução

O foco do projeto proposto é atuar em qualquer empresa que deseja utilizar um controle de ponto eletrônico nas suas dependências e permitir que seus funcionários consigam acompanhar de forma transparente o andamento das suas respectivas folhas de ponto.

Para o modelo de negócio, a proposta é fazer um preço único para compra do dispositivo com as configurações de servidor, o aplicativo Android sairia gratuito para os funcionários da empresa e uma taxa mensal para manutenção do servidor e do dispositivo de ponto além de atualizações trimestrais do aplicativo Android.

5.2. Descrição da Aplicação da Solução

Para a aplicação do projeto, será necessário que a empresa adquira o produto final, instale o dispositivo de ponto na entrada do seu estabelecimento e distribua cartões com a tecnologia RFID para todos os seus funcionários. Assim, o funcionário deve aproximar o cartão do dispositivo de ponto e esperar a mensagem de que o registro foi feito com sucesso, realizando essa ação sempre que entrar ou sair do local de trabalho.

Após realizar essa ação, o funcionário poderá acompanhar via aplicativo Android o andamento do seu histórico de registro de frequência, visualizando as horas de entrada e saída e também, visualizando o total de horas trabalhadas no dia. Além disso, o aplicativo permite que o usuário acompanhe as horas trabalhadas durante toda a semana e compare com a carga horária semanal e as horas trabalhadas durante todo o mês, comparando também com a carga horária mensal, possibilitando o acompanhamento de seu histórico de horas realizadas, horas faltantes e horas extras.

Durante o desenvolvimento do projeto foram encontradas apenas duas dificuldades. Uma delas foi a realização da conexão do módulo Ethernet utilizado pela plataforma Arduino para a conexão com o servidor de banco de dados, por causa da dificuldade para se encontrar uma biblioteca que trabalhasse com o módulo escolhido.

Neste caso, a melhor biblioteca encontrada é chamada UIPEthernet e pode ser encontrada de forma gratuita no seu repositório do GitHub, possuindo diversos tutoriais e uma base de usuários ampla na internet.

A outra adversidade encontrada foi a criação do modelo de banco de dados a ser utilizado, necessitando da ajuda de um especialista em banco de dados SQLite para a definição do MER.

5.3. Resultados da Aplicação da Solução

Para assegurar que o protótipo está de acordo com o que foi escrito no desenvolvimento do projeto, foi feita a verificação de funcionamento do dispositivo de ponto que utiliza a plataforma Arduino e os seus três periféricos, a utilização do aplicativo Android desenvolvido e por consequência, da funcionalidade do servidor de banco de dados com todas as suas funções, além de consultas e alterações no próprio banco de dados.

A primeira bateria de testes ocorreu com a plataforma Arduino, conectando um cabo de rede no módulo Ethernet, energizando o Arduino Nano pelo cabo USB e verificando se a troca de informações entre o Arduino e o servidor ocorria de forma correta pelo monitor serial que a IDE do próprio Arduino disponibiliza.

Neste primeiro teste, foi observado se o Arduino conseguia identificar a aproximação do cartão RFID, se enviava a requisição de inserção da batida de ponto para o servidor e recebia a resposta do resultado. Para isso, foram inseridos no banco de forma manual os dois IDs que foram identificados na parte do desenvolvimento e dois usuários relacionados com esses IDs. A Figura 5.1 mostra os dois usuários já inseridos no banco e relacionados com os devidos IDs do cartão e da *tag*.

	cpf	nome	carga_horaria	login	senha	perfil	data	fk_rfid
1	04136321139	Lucas Gennari	8	lucas.gennari@gmail.com	123456	1	2016-1-01	04525E52
2	01234567890	Joao Paulo	8	joao@teste.com.br	123456	1	2016-1-1	E3D1D5E5

Figura 5.1 - Tabela Pessoa, como visualizado no gerenciador de banco SQLite. Fonte: Elaborado pelo autor.

Com os usuários devidamente inseridos no banco de dados, foram feitos cinquenta registros de frequência com o cartão RFID e mais cinquenta com a *tag* RFID para avaliar o comportamento do servidor e da plataforma Arduino com o recebimento e envio de várias informações em sequência.

O gráfico presente na Figura 5.2 - Tempo de resposta do servidor para requisição na plataforma Arduino. Fonte: Elaborado pelo autor. foi disposto da seguinte maneira, a variável do eixo Y representa o tempo em segundos que o Arduino levou para enviar o registro de frequência para o servidor e receber a resposta. Enquanto o eixo X representa a quantidade de amostras que foram feitas. Os testes realizados com o cartão RFID que está ligado ao usuário João está representado na cor azul e na cor laranja estão os resultados dos testes realizados com a *tag* RFID do usuário Lucas.

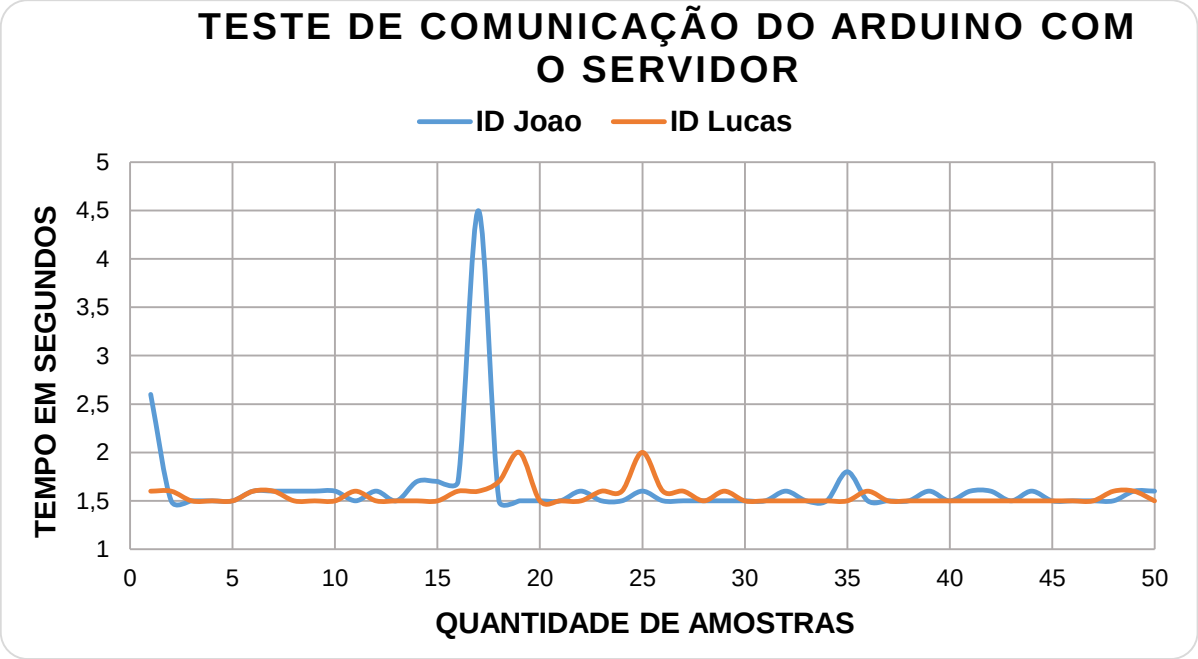

Figura 5.2 - Tempo de resposta do servidor para requisição na plataforma Arduino. Fonte: Elaborado pelo autor.

Do total de cem amostras recolhidas, nenhuma apresentou falha no registro de frequência tanto por parte do servidor como por parte da plataforma Arduino. Somente uma anormalidade foi notada nos testes, que como pode ser visto na Figura 5.2 - Tempo de resposta do servidor para requisição na plataforma Arduino. Fonte: Elaborado pelo autor., foi uma requisição que demorou 4,5 segundos devido à lentidão na rede que estava sendo utilizada enquanto o resto dos testes ficou variando na maior parte do tempo entre 1,5 e 1,6 segundos para que o Arduino recebesse a reposta do servidor.

Já a Figura 5.3 representa o arquivo de *log* que o servidor criou com todos os pedidos da plataforma Arduino e o estado da resposta, que como foi dito anteriormente, obteve sucesso em todas as cem amostras colhidas, tendo conseguido inserir no banco de dados todas as batidas de ponto feitas. E a Figura 5.4 mostra as mensagens que o display LCD apresenta para o usuário ao ser realizada a batida de ponto com sucesso, de acordo com o nome do usuário que as fez.

```
 2  [INFO]17/06/2016 15:26:35  - ------------------------------------------------------
 3  [INFO]17/06/2016 15:26:35  - CPF[01234567890] Nome[Joao Paulo] RFID[E3D1D5E5]
 4  [INFO]17/06/2016 15:26:35  - Nome[Joao Paulo] Data [2016-6-17] Hora [15:26:35]
 5  [INFO]17/06/2016 15:26:35  - ------------------------------------Ponto Batido-----
 6  [INFO]17/06/2016 15:26:35  - ------------------------------------------------------
 7  [INFO]17/06/2016 15:27:10  - ------------------------------------------------------
 8  [INFO]17/06/2016 15:27:10  - CPF[01234567890] Nome[Joao Paulo] RFID[E3D1D5E5]
 9  [INFO]17/06/2016 15:27:10  - Nome[Joao Paulo] Data [2016-6-17] Hora [15:27:10]
10  [INFO]17/06/2016 15:27:10  - ------------------------------------Ponto Batido-----
11  [INFO]17/06/2016 15:27:10  - ------------------------------------------------------
12  [INFO]17/06/2016 15:27:15  - ------------------------------------------------------
13  [INFO]17/06/2016 15:27:15  - CPF[01234567890] Nome[Joao Paulo] RFID[E3D1D5E5]
14  [INFO]17/06/2016 15:27:15  - Nome[Joao Paulo] Data [2016-6-17] Hora [15:27:15]
15  [INFO]17/06/2016 15:27:16  - ------------------------------------Ponto Batido-----
16  [INFO]17/06/2016 15:27:16  - ------------------------------------------------------
17  [INFO]17/06/2016 15:27:49  - ------------------------------------------------------
18  [INFO]17/06/2016 15:27:49  - CPF[01234567890] Nome[Joao Paulo] RFID[E3D1D5E5]
19  [INFO]17/06/2016 15:27:49  - Nome[Joao Paulo] Data [2016-6-17] Hora [15:27:49]
20  [INFO]17/06/2016 15:27:49  - ------------------------------------Ponto Batido-----
```

Figura 5.3 - Tela de log do servidor com informações de batida de ponto. Fonte: Elaborado pelo autor.

Nos testes realizados com o aplicativo Android, as funcionalidades de *login*, apresentação do registro de frequência diário, do resultado dos registros semanal e mensal foram avaliados da mesma forma que os testes feitos com a plataforma Arduino. Foram obtidas um total de 100 (cem) amostras, sendo que destas, 25 (vinte

e cinco) foram testes de login, 25 (vinte e cinco) testes de visualização das batidas diárias e 50 (cinquenta) testes de visualização do total de horas semanal e mensal.

Neste teste, também, não foram encontrados problemas durante a sua execução. Desta forma, o aplicativo conseguiu se conectar com sucesso em todas as tentativas com o servidor de banco de dados para apresentar ao usuário o seu histórico de batidas de ponto.

5.4. Custos

Em relação aos custos do projeto, foi feito um levantamento de preço de todos os componentes utilizados para a confecção do dispositivo de ponto utilizando a plataforma Arduino que pode ser observado no gráfico presente na Figura 5.4 - Preço dos componentes utilizados no dispositivo de ponto. Fonte: Elaborado pelo autor., que relaciona o preço em reais com cada componente utilizado, do Arduino Nano aos *jumpers* que realizam as conexões dos periféricos. Sendo que, a soma dos preços destes componentes totaliza R$ 197,06.

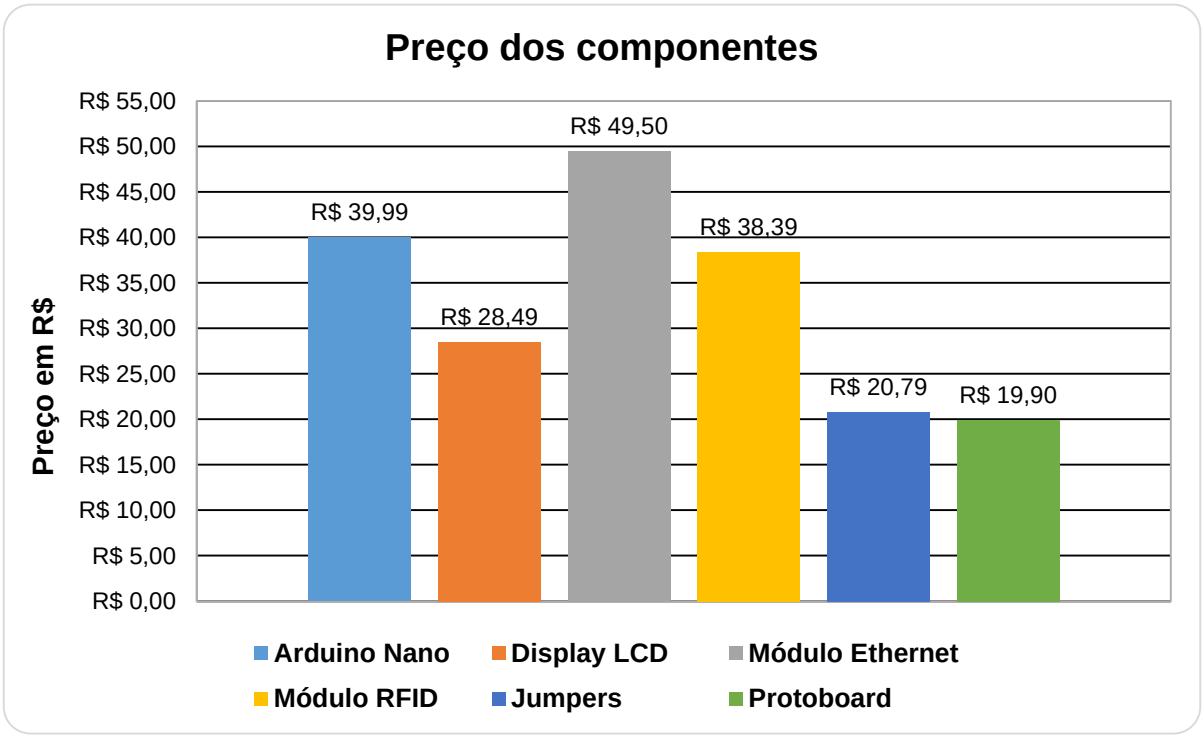

Figura 5.4 - Preço dos componentes utilizados no dispositivo de ponto. Fonte: Elaborado pelo autor.

Ao fazer a simples soma de cada componente o valor obtido é relativamente barato ao se comparar com produtos já existentes no mercado. Esta comparação pode ser observada na Figura 5.5 - Comparação de preço de produtos existentes com o projeto desenvolvido. Fonte: Elaborado pelo autor., que apresenta quatro produtos existentes no mercado e possuem pelo menos um sensor RFID para a batida de ponto.

Desses produtos, apenas o Prisma Super Fácil R01 possui um leitor biométrico, por isso seu preço acaba sendo mais alto que os outros. Esse levantamento também só leva em conta o dispositivo de relógio de ponto, ainda são acrescidos desses valores a bobina de papel para imprimir o comprovante da batida de ponto e o *software* de controle de ponto. Em um orçamento feito em uma revendedora autorizada o valor a ser acrescido do relógio de ponto é de aproximadamente R$350 reais. Sendo que neste valor total já está inclusa a instalação e configuração do produto, além de sua manutenção mensal.

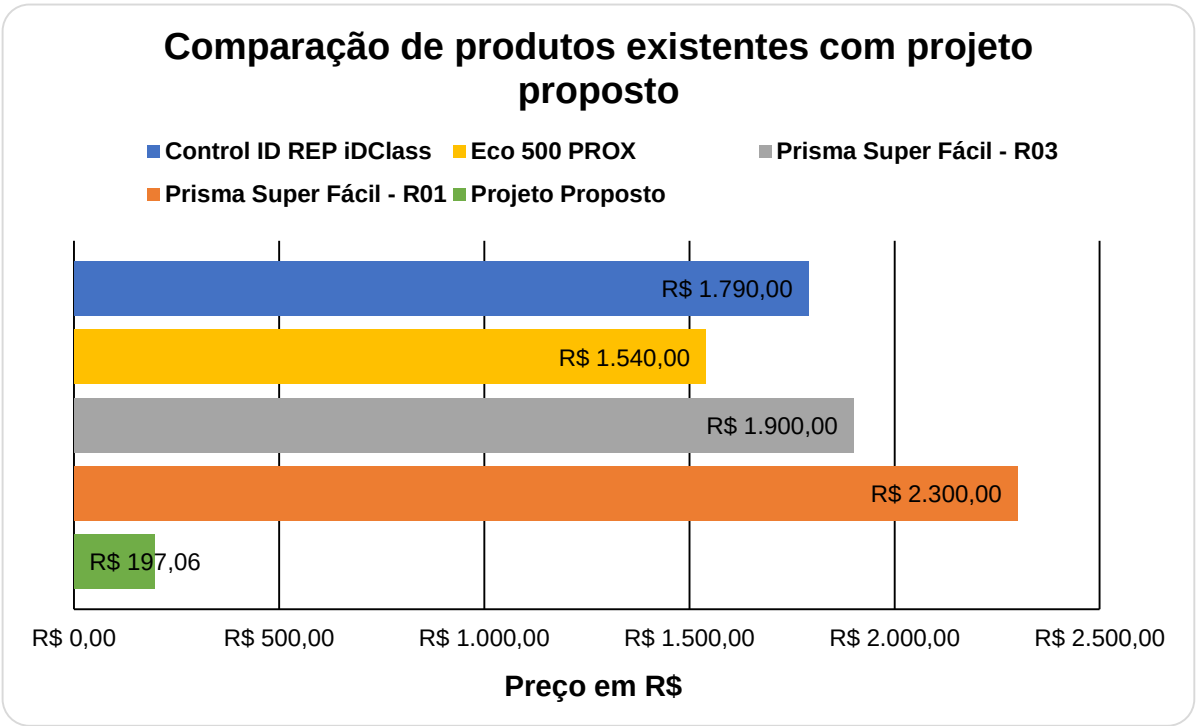

Figura 5.5 - Comparação de preço de produtos existentes com o projeto desenvolvido. Fonte: Elaborado pelo autor.

Para comercializar o projeto aqui desenvolvido, como foi dito anteriormente, será colocado um valor somente no dispositivo de ponto e nos servidores, sem ser feita a cobrança pelo aplicativo Android e a configuração e instalação do produto. Além disso, também será cobrado um valor mensal para realizar a manutenção do produto como um todo e a atualização do aplicativo Android com novas funcionalidades ou correções de *bugs*.

Portanto, um valor justo para se cobrar pelo projeto levando em consideração todos os periféricos utilizados e a mão de obra utilizada para produzir o produto é de R$450,00 e R$50,00 mensais pela manutenção. Uma sugestão para reduzir o custo do projeto é a utilização de uma placa de circuito impresso no lugar da protoboard e dos jumpers, que dão um melhor aspecto visual para o produto final levando a uma redução de aproximadamente R$40,00.

5.5. Avaliação Global

Para finalizar, o protótipo cumpre bem o objetivo de realizar os registros de frequência e armazená-los em um banco de dados e o aplicativo Android também foi capaz de realizar sua função de visualização do histórico de batidas de ponto diária, semanal e mensal para cada usuário, sendo o maior atrativo do produto final desenvolvido, uma vez que nenhum produto similar pesquisado possui um aplicativo *mobile* vinculado ao seu sistema de relógio de ponto.

CAPÍTULO 6 – CONCLUSÃO

6.1. Conclusões

Como dito ao final do capítulo anterior, conclui-se que o projeto conseguiu atingir todos os objetivos específicos citados com um protótipo de relógio de ponto equipado com um leitor RFID capaz de armazenar no banco de dados todas as informações de batidas de ponto que os usuários possam fazer e com o aplicativo Android realizando com sucesso as consultas no servidor do banco de dados para mostrar as informações lá inseridas ao usuário.

Durante o desenvolvimento do projeto, apesar das dificuldades que foram mencionadas no capítulo anterior, notou-se que as pesquisas feitas para a escrita do referencial teórico contribuíram profundamente ao desenvolvimento de todos os códigos e montagem do dispositivo de ponto, de forma que fosse possível terminar o projeto com sucesso.

Isto pode ser confirmado com os resultados obtidos após os testes feitos ao final do desenvolvimento do projeto, que apresentam uma taxa de sucesso de 100%, considerando a repetitividade de até 50 (cinquenta) eventos realizados para cada situação, sendo que foram realizados testes com dois dispositivos RFID no Arduino e testes de estresse do aplicativo Android. Assim, pode ser observado que o projeto atinge a todos os objetivos que foram especificados durante a criação do projeto. Dessa forma, é possível ver que o produto final é confiável e também viável para comercialização.

A pesquisa de valores de produtos similares comprova que o projeto proposto cumpriu com o objetivo de proporcionar um produto de baixo custo, ao ser comparado a estes outros produtos, permitindo que empresas menores não necessitem desembolsar muito dinheiro para a aquisição de um relógio de ponto.

Dessa forma, é possível que o usuário acompanhe todo o seu histórico de ponto e tenha o controle das horas extras ou horas faltantes que possui no dia, na semana e no mês.

6.2. Sugestões para Trabalhos Futuros

Com o intuito de complementar o trabalho aqui desenvolvido, algumas adições e alterações são sugeridas, entre elas:

- Implementar funcionalidade de usuário administrador, que consegue visualizar o histórico de todos os funcionários;
- Desenvolvimento de um sistema *web* para melhor gerenciamento do ponto de todos os funcionários;
- Utilização de um dispositivo WiFi para se conectar à rede local do estabelecimento no lugar no módulo Ethernet que necessita da utilização de cabos;
- Adição de um leitor biométrico, a fim de tornar o produto final mais completo em relação às alternativas encontradas no mercado;
- E aprofundar o projeto no cumprimento de todas as leis e normas propostas na CLT.

REFERÊNCIAS

ARDUINO. Disponivel em: <www.arduino.cc>. Acesso em: 30 Março 2016.

BRASIL. **Decreto-Lei n.º 5.452, de 1º de maio de 1943**. Rio de Janeiro: Aprova a Consolidação das Leis do Trabalho, Consolidação das Leis do Trabalho, 1943.

DEITEL, H. M.; DEITEL, P. J. **C++ How to program:** Third edition. 3ª. ed. Porto: Deitel & Associates, mc. , 2003.

DELIGHT, B. **Optimizing the execution time of SQLite on an ABB robot controller**. Mälardalen University. Mälardalen. 2013.

DIMARZIO, J. F. **Android:** A Programmer's Guide. 1ª. ed. New York: The McGraw-Hill Companies, 2008.

FINKENZELLER, K. **RFID Handbook**. 1ª. ed. Munich: John Wiley & Sons, Ltd, 2010.

FRITZING. Disponivel em: <http://fritzing.org>. Acesso em: 7 Maio 2016.

MICROCHIP. Disponivel em: <http://www.microchip.com/wwwproducts/en/en022889>. Acesso em: 10 Maio 2016.

NXP. Disponivel em: <http://www.nxp.com/documents/data_sheet/MFRC522.pdf>. Acesso em: 10 Maio 2016.

OWENS, M. **The Definitive Guide to SQLite**. New York: Springer-Verlag New York, Inc., 2006.

PHP. **PHP:** Hypertext Preprocessor. Disponivel em: <www.php.net>. Acesso em: 3 Abril 2016.

SHARMA, N. et al. **Database Fundamentals**. 1ª. ed. Markham, ON: IBM Corporation 2010, 2010.

SPARKFUN. Disponivel em: <https://www.sparkfun.com/datasheets/LCD/HD44780.pdf>. Acesso em: 10 Maio 2016.

SPURGEON, C. E.; ZIMMERMAN, J. **Ethernet:** The Definitive Guide. 2ª. ed. Sebastopol, CA: O'Reilly Media, Inc., 2014.

STEPHENS, R. **Beginning Database Design Solutions**. Indianapolis: Wiley Publishing, Inc., 2009.

APÊNDICE A

Código feito na Plataforma Arduino para a comunicação do módulo Ethernet, do leitor RFID e do display LCD com o Arduino Nano e, do Arduino Nano com o servidor Arduino.

```
#include <MFRC522.h>
#include <UIPEthernet.h>
#include <string.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal.h>
#include <SPI.h>
//#define DEBUG
#define SS_RFID_PIN 8
#define RST_RFID_PIN 9
LiquidCrystal lcd (6, 7, 5, 4, 3, A2);
EthernetClient client;
IPAddress ip = {192, 168, 1, 249 };
byte mac[] = { 0x74, 0x69, 0x69, 0x2D, 0x30, 0x31 };
char server[] = "192.168.1.108";
String repository = "/WebServer/consultas/arduino.php";
int port = 80;
String rfidTag = "";
MFRC522 rfid(SS_RFID_PIN, RST_RFID_PIN);
struct resposta {
String nome;
String msg;
};
unsigned long start, finish, sub;
void setup() {
Serial.begin(9600);
SPI.begin();
lcd.begin( 16, 2);
lcd.clear();
rfid.PCD_Init();
Ethernet.begin ( mac, ip );
mensagemInicial();
delay ( 1000 );
}
void loop() {
if ( ! rfid.PICC_IsNewCardPresent() ) {
return;
}
if ( ! rfid.PICC_ReadCardSerial() ) {
return;
}
start = millis();
rfidTag = "";
#ifdef DEBUG
Serial.print ( "UID da tag : " );
#endif
for (byte i = 0; i < rfid.uid.size; i++) {
```

```
#ifdef DEBUG
Serial.print(rfid.uid.uidByte[i] < 0x10 ? "0" : "");
Serial.print(rfid.uid.uidByte[i], HEX);
#endif
rfidTag.concat(String(rfid.uid.uidByte[i] < 0x10 ? "0" : ""));
rfidTag.concat(String(rfid.uid.uidByte[i], HEX));
}
delay ( 250 );
rfidTag.toUpperCase();
if ( !client.connect ( server, port ) ) {
#ifdef DEBUG
Serial.println ( "\n--> Connection failure detected: Resetting ENC!" );
#endif
Enc28J60.init ( mac );
} else {
if ( sendRequest ( rfidTag )) {
String ret = getResponse();
finish = millis();
sub = ( finish - start ) / 100;
Serial.println ( "Resposta[ " + ret + " ]" );
Serial.print ( "Tempo em segundos: " );
Serial.print ( sub );
Serial.println ( "s" );
if ( ret != "" ) {
struct resposta resp = split ( ret, "," );
lcd.clear();
Serial.println ( "Nome: " + resp.nome );
lcd.setCursor ( 0, 0 );
lcd.print ( "Ola " + resp.nome );
#ifdef DEBUG
#endif
Serial.println ( "Mensagem: " + resp.msg );
lcd.setCursor ( 0, 1 );
lcd.print ( resp.msg );
} else {
#ifdef DEBUG
Serial.println ( "Resposta nula" );
#endif
lcd.clear();
lcd.setCursor ( 0, 0 );
lcd.print ( "Sem resposta" );
lcd.setCursor ( 0, 1 );
lcd.print ( "do servidor" );
}}}
client.stop();
delay ( 2500 );
mensagemInicial();
}
void mensagemInicial() {
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(" Aproxime o seu");
lcd.setCursor(0, 1);
lcd.print("cartao do leitor");
#ifdef DEBUG
Serial.println("Aproxime o seu cartao do leitor...");
Serial.println();
#endif
}
bool sendRequest ( String tag ) {
```

```
delay ( 150 );
if (client.connect(server, port)) {
#ifdef DEBUG
Serial.println("\n-> Connected");  // only use serial when debugging
#endif
client.println("GET " + repository + "?rfid=" + tag);
client.print("Host: ");
client.println(server);
client.println("Content-Type: application/x-www-form-urlencoded");
} else {
#ifdef DEBUG
Serial.println("--> connection failed !!");
#endif
return false;
}
return true;
}
String getResponse() {
String response = "";
bool begin = false;
while (client.connected()) {
while (client.available()) {
char in = client.read();
if (in == 123) {
begin = true;
}
if (begin && in != 123 && in != 125 && in != 34) {
response += (in);
}
if (in == 125) {
break;
}
delay(1);
}}
return response;
}
struct resposta split(String texto, String t) {
int start = 0;
int count = 0;
struct resposta r;
texto = texto + ",";
for (int i = 0; i < texto.length(); i++) {
if (texto.substring(i, i + 1) == t) {
switch (count) {
case 0:
r.nome = texto.substring(start, i);
break;
case 1:
r.msg = texto.substring(start, i);
break;
}
start = i + 1;
count++;
}}
r.nome = r.nome.substring(5, r.nome.length());
r.msg = r.msg.substring(4, r.msg.length());
return r;
}
```

APÊNDICE B

Código feito no aplicativo Android, que permite o usuário visualizar seu histórico de batidas de ponto. Está dividido em várias classes, utilizando-se do conceito de programação orientada a objetos.

Classe PontoActivity.java

```
package br.tcc.ponto;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.Handler;
import android.support.v7.widget.SwitchCompat;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import br.tcc.ponto.comm.HttpRetriever;
import br.tcc.ponto.util.PropUtils;
public class PontoActivity extends Activity {
Button loginButton;
EditText emailLogin;
EditText paswdLogin;
SwitchCompat rememberMe;
String cpfUser;
String nomeUser;
String perfilUser;
private static final String SPF_NAME = "pontoLogin";
private static final String USERNAME = "username";
private static final String PASSWORD = "password";
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.login);
PropUtils.loadProps(this);
loginButton = (Button) findViewById(R.id.btnLogin);
emailLogin = (EditText) findViewById(R.id.email);
paswdLogin = (EditText) findViewById(R.id.password);
rememberMe = (SwitchCompat) findViewById(R.id.switch_remember);
SharedPreferences loginPreferences = getSharedPreferences(SPF_NAME,
Context.MODE_PRIVATE);
if (!loginPreferences.getString(USERNAME, "").contentEquals("")) {
rememberMe.setChecked(true);
}
emailLogin.setText(loginPreferences.getString(USERNAME, ""));
paswdLogin.setText(loginPreferences.getString(PASSWORD, ""));
loginButton.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
```

```
Log.d(getCallingPackage(), "Email [" + emailLogin.getText().toString() + "] - Senha [" +
paswdLogin.getText().toString() + "]");
fetchLogin(emailLogin.getText().toString(), paswdLogin.getText().toString());
}});
}
public void fetchLogin(String login, String password) {
try {
String returnLogin = HttpRetriever.getInstance().getLogin(login, password);
System.out.println(returnLogin);
String[] firstSplit = returnLogin.replace("\"", "").replace("}", "").split(",");
String[] cpfSplit = firstSplit[0].split(":");
String[] nomeSplit = firstSplit[1].split(":");
String[] perfilSplit = firstSplit[2].split(":");
cpfUser = cpfSplit[1];
nomeUser = nomeSplit[1];
perfilUser = perfilSplit[1];
if (!cpfUser.equals("")) {
if (rememberMe.isChecked()) {
SharedPreferences loginPreferences = getSharedPreferences(SPF_NAME,
Context.MODE_PRIVATE);
loginPreferences.edit().putString(USERNAME, login).putString(PASSWORD, password).apply();
}
System.out.println("CPF: " + cpfUser);
System.out.println("Nome: " + nomeUser);
System.out.println("Perfil: " + perfilUser);
if (perfilUser.equals("PERFIL USER")) {
new Handler().post(new Runnable() {
@Override
public void run() {
Intent mainIntent = new Intent(PontoActivity.this, UserCalendar.class);
Bundle extra = new Bundle();
extra.putString("cpf", cpfUser);
extra.putString("nome", nomeUser);
mainIntent.putExtras(extra);
PontoActivity.this.startActivity(mainIntent);
}});
} else if (perfilUser.equals("PERFIL ADM")) {
}} else {
}} catch (Exception e) {
e.printStackTrace();
}}}
```

Classe UserCalendar.java

```
package br.tcc.ponto;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CalendarView;
import android.widget.ListView;
import android.widget.TextView;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import br.tcc.ponto.comm.HttpRetriever;
```

```
@SuppressWarnings("deprecation")
public class UserCalendar extends AppCompatActivity {
CalendarView calendarView;
Button totalHorasSemana;
Button totalHorasMes;
ListView ponto_dia;
TextView titulo;
String cpf;
String nome;
Date date;
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.user_calendar);
Bundle extra = getIntent().getExtras();
if (extra != null) {
cpf = extra.getString("cpf");
nome = extra.getString("nome");
System.out.println("Extra - Cpf [" + cpf + "]");
System.out.println("Extra - Nome [" + nome + "]");
}
titulo = (TextView) findViewById(R.id.welcome_user);
calendarView = (CalendarView) findViewById(R.id.userCalendarView);
ponto_dia = (ListView) findViewById(R.id.ponto_dia);
totalHorasSemana = (Button) findViewById(R.id.btnHorasSemana);
totalHorasMes = (Button) findViewById(R.id.btnHorasMes);
titulo.setText(getString(R.string.welcome, nome));
date = new Date();
calendarView.setOnDateChangeListener(new CalendarView.OnDateChangeListener() {
@Override
public void onSelectedDayChange(CalendarView view, int year, int month, int dayOfMonth) {
month = month + 1;
String dia = (year + "-" + month + "-" + dayOfMonth);
date = new Date();
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
try {
date = dateFormat.parse(dia);
} catch (ParseException e) {
e.printStackTrace();
}
String pontoDia = HttpRetriever.getInstance().getPontoDia(cpf, dia, 1);
String[] firstSplit = pontoDia.replace("\"", "").replace("{", "").replace("}", "").split(",");
String[] message = firstSplit[0].split(":");
System.out.println("Mensagem: " + message[1]);
String[] batidas = firstSplit[1].replace("message:", "").split(";");
String totalHorasDia = firstSplit[2].replace("total_horas:", "");
List<String> arrayList = new ArrayList<>();
int i = 1;
for (String batida : batidas) {
if (i % 2 != 0) {
arrayList.add(batida.replace("1-", "Entrada -    "));
} else {
arrayList.add(batida.replace("2-", "Saída    -    "));
}
i++;
}
arrayList.add("Total de horas - " + totalHorasDia);
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(UserCalendar.this, R.layout.list_item,
arrayList);
new AlertDialog.Builder(UserCalendar.this).setAdapter(arrayAdapter, new
DialogInterface.OnClickListener() {
```

```
@Override
public void onClick(DialogInterface dialog, int which) {
}}).setNegativeButton("DISMISS", null).setTitle("Dia " + dayOfMonth).show();
}});
totalHorasSemana.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Calendar week = Calendar.getInstance();
week.setTime(date);
week.set(Calendar.DAY_OF_WEEK, Calendar.SUNDAY);
String firstDayOfWeek = week.get(Calendar.YEAR) + "-" + (week.get(Calendar.MONTH) + 1) + "-" +
week.get(Calendar.DATE);
String lastDayOfWeek = "";
for (int i = 6; i >= 0; i--) {
if (week.getActualMaximum(Calendar.DAY_OF_MONTH) == (week.get(Calendar.DATE) + i)) {
lastDayOfWeek = week.get(Calendar.YEAR) + "-" + (week.get(Calendar.MONTH) + 1) + "-" +
(week.get(Calendar.DATE) + i);
break;
} else {
lastDayOfWeek = week.get(Calendar.YEAR) + "-" + (week.get(Calendar.MONTH) + 1) + "-" +
(week.get(Calendar.DATE) + 6);
}}
String pontoSemana = HttpRetriever.getInstance().getPontoSemana(cpf, 2, firstDayOfWeek,
lastDayOfWeek);
String[] firstSplit = pontoSemana.replace("\"", "").replace("{", "").replace("}", "").split(",");
String[] message = firstSplit[0].split(":");
System.out.println("Mensagem: " + message[1]);
String[] batidas = firstSplit[1].replace("message:", "").split(";");
String totalHorasSemana = firstSplit[2].replace("total_horas:", "");
List<String> arrayList = new ArrayList<>();
arrayList.add("Horas realizadas - " + batidas[0]);
arrayList.add("Carga horária      - " + batidas[1]);
arrayList.add("Horas restantes  - " + totalHorasSemana.replace("-", "Faltam "));
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(UserCalendar.this, R.layout.list_item,
arrayList);
new AlertDialog.Builder(UserCalendar.this).setAdapter(arrayAdapter, new
DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
}}).setNegativeButton("DISMISS", null).setTitle("De " + firstDayOfWeek + " a " +
lastDayOfWeek).show();
}});
totalHorasMes.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Calendar month = Calendar.getInstance();
month.setTime(date);
String firstDayOfMonth = month.get(Calendar.YEAR) + "-" + (month.get(Calendar.MONTH) + 1) + "-" +
month.getActualMinimum(Calendar.DAY_OF_MONTH);
String lastDayOfMonth = month.get(Calendar.YEAR) + "-" + (month.get(Calendar.MONTH) + 1) + "-" +
month.getActualMaximum(Calendar.DAY_OF_MONTH);
System.out.println("Primeiro dia mes: " + firstDayOfMonth);
System.out.println("Ultimo dia mes:   " + lastDayOfMonth);
int diasUteis = 0;
Date first = new Date();
Date last = new Date();
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
try {
first = dateFormat.parse(firstDayOfMonth);
last = dateFormat.parse(lastDayOfMonth);
```

```
} catch (ParseException e) {
e.printStackTrace();
}
Calendar start = Calendar.getInstance();
Calendar end = Calendar.getInstance();
start.setTime(first);
end.setTime(last);
do {
if (start.get(Calendar.DAY_OF_WEEK) != Calendar.SATURDAY &&
start.get(Calendar.DAY_OF_WEEK) != Calendar.SUNDAY) {
diasUteis++;
}
start.add(Calendar.DAY_OF_MONTH, 1);
} while (start.get(Calendar.DAY_OF_YEAR) <= end.get(Calendar.DAY_OF_YEAR));
String pontoMes = HttpRetriever.getInstance().getPontoMes(cpf, 3, firstDayOfMonth, lastDayOfMonth,
diasUteis);
String[] firstSplit = pontoMes.replace("\"", "").replace("{", "").replace("}", "").split(",");
String[] message = firstSplit[0].split(":");
System.out.println("Mensagem: " + message[1]);
String[] batidas = firstSplit[1].replace("message:", "").split(";");
String totalHorasSemana = firstSplit[2].replace("total_horas:", "");
List<String> arrayList = new ArrayList<>();
arrayList.add("Horas realizadas - " + batidas[0]);
arrayList.add("Carga horária - " + batidas[1]);
arrayList.add("Horas restantes - " + totalHorasSemana.replace("-", "Faltam "));
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(UserCalendar.this, R.layout.list_item,
arrayList);
new AlertDialog.Builder(UserCalendar.this).setAdapter(arrayAdapter, new
DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
}}).setNegativeButton("DISMISS", null).setTitle("De " + firstDayOfMonth + " a " +
lastDayOfMonth).show(); }
});
}}
```

Classe HttpRetriever.java

```
package br.tcc.ponto.comm;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URI;
import java.net.URL;
import java.net.URLDecoder;
import java.util.Properties;
import android.os.StrictMode;
import br.tcc.ponto.util.PropUtils;
public class HttpRetriever {
private static HttpRetriever instance = new HttpRetriever();
private Properties properties = PropUtils.getInstance();
public static HttpRetriever getInstance() {
if (instance != null) {
return instance;
} else {
instance = new HttpRetriever();
return instance;
}}
public String getLogin(String login, String password) {
try {
```

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
StrictMode.setThreadPolicy(policy);
URI uri = new URI(properties.getProperty("protocol"), properties.getProperty("server") + ":" +
properties.getProperty("port"), "/" + properties.getProperty("context") + "/" +
properties.getProperty("pathLogin") + "?login=" + login + "&passwd=" + password, null, null);
String urlStr = URLDecoder.decode(uri.toString(), "UTF-8");
System.out.println("Requisicao a ser enviada: [" + urlStr + "]");
URL url = new URL(urlStr);
HttpURLConnection con = (HttpURLConnection) url.openConnection();
String txt = readStream(con.getInputStream());
System.out.println("Text retornado pelo servidor: " + txt);
return txt;
} catch (Exception e) {
System.out.println("ERRO ao tentar logar - " + e.getMessage());
e.printStackTrace();
return null;
}}
public String getPontoDia(String cpf, String date, int tipo) {
try {
URI uri = new URI(properties.getProperty("protocol"), properties.getProperty("server") + ":" +
properties.getProperty("port"), "/" + properties.getProperty("context") + "/" +
properties.getProperty("pathPonto") + "?cpf=" + cpf + "&tipo=" + tipo + "&date=" + date, null, null);
String urlStr = URLDecoder.decode(uri.toString(), "UTF-8");
System.out.println("Requisicao a ser enviada: [" + urlStr + "]");
URL url = new URL(urlStr);
HttpURLConnection con = (HttpURLConnection) url.openConnection();
String txt = readStream(con.getInputStream());
System.out.println("Texto retornado pelo servidor: " + txt);
return txt;
} catch (Exception e) {
return null;
}}
public String getPontoSemana(String cpf, int tipo, String firstDay, String lastDay) {
try {
URI uri = new URI(properties.getProperty("protocol"),
properties.getProperty("server") + ":" + properties.getProperty("port"), "/" +
properties.getProperty("context") + "/" + properties.getProperty("pathPonto") + "?cpf=" + cpf + "&tipo="
+ tipo + "&datestart=" + firstDay + "&dateend=" + lastDay, null, null);
String urlStr = URLDecoder.decode(uri.toString(), "UTF-8");
System.out.println("Requisicao a ser enviada: [" + urlStr + "]");
URL url = new URL(urlStr);
HttpURLConnection con = (HttpURLConnection) url.openConnection();
String txt = readStream(con.getInputStream());
System.out.println("Texto retornado pelo servidor: " + txt);
return txt;
} catch (Exception e) {
return null;
}}
public String getPontoMes(String cpf, int tipo, String firstDay, String lastDay, int diasUteis) {
try {
URI uri = new URI(properties.getProperty("protocol"),
properties.getProperty("server") + ":" + properties.getProperty("port"), "/" +
properties.getProperty("context") + "/" + properties.getProperty("pathPonto") + "?cpf=" + cpf + "&tipo="
+ tipo + "&datestart=" + firstDay + "&dateend=" + lastDay + "&diasuteis=" + diasUteis, null, null);
String urlStr = URLDecoder.decode(uri.toString(), "UTF-8");
System.out.println("Requisicao a ser enviada: [" + urlStr + "]");
URL url = new URL(urlStr);
HttpURLConnection con = (HttpURLConnection) url.openConnection();
String txt = readStream(con.getInputStream());
System.out.println("Texto retornado pelo servidor: " + txt);
```

```
return txt;
} catch (Exception e) {
return null;
}}
private String readStream(InputStream in) {
BufferedReader reader = null;
String out = "";
try {
reader = new BufferedReader(new InputStreamReader(in, "UTF-8"));
String line;
while ((line = reader.readLine()) != null) {
System.out.println(line);
out += line;
}
return out;
} catch (IOException e) {
e.printStackTrace();
} finally {
if (reader != null) {
try {
reader.close();
} catch (IOException e) {
e.printStackTrace();
}}}
return null;
}}
```

APÊNDICE C

Código utilizado para o desenvolvimento do servidor Arduino, que insere no banco de dados os horários das batidas de ponto.

```
<?php
header ("Content-Type:application/json");
define ('E', 'ERRO');
define ('I', 'INFO');
define ('W', 'WARN');
if (!empty($_GET['rfid'])) {
$rfid = $_GET['rfid'];
$database = new SQLite3('../Ponto.db');
insere_log(I ,"----------------------------------------------------------------------------------------------");
$pessoa = consultaRfid($database, $rfid);
if (empty($pessoa->cpf_pessoa)) {
insere_log(E ,"RFID nao encontrado[" . $rfid . "]");
deliver_response(200, "NAO CADASTRADO", NULL);
} else {
insere_log(I ,"CPF[" . $pessoa->cpf_pessoa . "] Nome[" . $pessoa->nome_pessoa . "] RFID[" . $pessoa-
>fk_rfid . "]");
if (!empty($pessoa->fk_rfid)) {
$dt_atual = date("Y-n-d");
$hr_atual = date("H:i:s");
insere_log(I, "Nome[" . $pessoa->nome_pessoa . "] Data [" . $dt_atual . "] Hora [" . $hr_atual . "]");
$result = $database->querySingle("SELECT COUNT (*) as count FROM ponto WHERE
fk_pessoa=\"$pessoa->cpf_pessoa\" and data=\"$dt_atual\"");
if ($result > 0 ) {
$check_fetch = "SELECT * FROM ponto where fk_pessoa=\"$pessoa->cpf_pessoa\" and
data=\"$dt_atual\"";
$resp = $database->query($check_fetch);
$ponto_row = $resp->fetchArray(SQLITE3_ASSOC);
$ponto_obj = new Ponto();
$ponto_obj->id_ponto = $ponto_row['idponto'];
$ponto_obj->data_ponto = $ponto_row['data'];
$ponto_obj->fk_pessoa = $ponto_row['fk_pessoa'];
if (insereBatida($database, $ponto_obj->id_ponto, $hr_atual)) {
insere_log(I ,"-------------------------------------Ponto Batido-------------------------------------------");
deliver_response(200, "PONTO BATIDO", $pessoa->nome_pessoa);
} else {
insere_log(E ,"-------------------------------------Falha Batida-------------------------------------------");
deliver_response(200, "FALHA BATIDA", $pessoa->nome_pessoa);
}} else {
if (inserePonto($database, $dt_atual, $pessoa->cpf_pessoa, $hr_atual)) {
insere_log(I ,"-------------------------------------Ponto Batido-------------------------------------------");
deliver_response(200, "PONTO BATIDO", $pessoa->nome_pessoa);
} else {
insere_log(E ,"-------------------------------------Falha Ponto-------------------------------------------");
deliver_response(200, "FALHA PONTO", $pessoa->nome_pessoa);
}}} else {
insere_log(E ,"RFID nao cadastrado");
deliver_response(200, "NAO CADASTRADO", NULL);
}}
```

```
insere_log(I ,"----------------------------------------------------------------------------------------------");
} else {
insere_log(E ,"-----------------------------------Invalid Request-------------------------------------");
deliver_response(400, "INVALID REQUEST", NULL);
}
function consultaRfid ($db, $rfid) {
if (!$db) {
insere_log(E ,$db->lastErrorMsg());
} else {
$fetch = "SELECT cpf, nome, fk_rfid FROM pessoa WHERE fk_rfid=\"$rfid\"";
$ret = $db->query($fetch);
while($row = $ret->fetchArray(SQLITE3_ASSOC)) {
$obj = new Pessoa();
$obj->cpf_pessoa = $row['cpf'];
$obj->nome_pessoa = $row['nome'];
$obj->fk_rfid = $row['fk_rfid'];
return $obj;
}}}
function inserePonto($db, $ponto_data, $ponto_cpf, $ponto_hora) {
$sql_ponto = "INSERT INTO ponto VALUES (null, :date, :cpfpessoa)";
$stmt_ponto = $db->prepare($sql_ponto);
$stmt_ponto->bindParam(':date', $ponto_data);
$stmt_ponto->bindParam(':cpfpessoa', $ponto_cpf);
if ($stmt_ponto->execute()) {
$last_id = $db->lastInsertRowID();
if (insereBatida($db, $last_id, $ponto_hora )){
return true;
} else {
return false;
}} else {
return false;
}}
function insereBatida($db, $batida_ponto, $batida_hora) {
$sql_batida = "INSERT INTO batidas VALUES (null, :hora, :situacao, :fkponto)";
$stmt_batida = $db->prepare($sql_batida);
$stmt_batida->bindParam(':hora', $batida_hora);
$count = $db->querySingle("SELECT COUNT (*) as count FROM batidas WHERE
fk_ponto=\"$batida_ponto\"");
if ($count % 2 == 0) {
$situacao1 = 1;
$stmt_batida->bindParam(':situacao', $situacao1);
} else {
$situacao2 = 2;
$stmt_batida->bindParam(':situacao', $situacao2);
}
$stmt_batida->bindParam(':fkponto', $batida_ponto);
if ($stmt_batida->execute()) {
return true;
} else {
return false;
}}
function deliver_response($status, $status_message, $nome){
header("HTTP/1.1 $status $status_message");
$response['nome'] = $nome;
$response['msg'] = $status_message;
$json_response = json_encode($response);
echo $json_response;
}
function insere_log ($tipo ,$texto) {
$texto = "\r\n[" . $tipo . "]" . date("d/m/Y H:i:s ") . " - " . $texto;
```

```
$nome_arquivo = '../logs/logArduino.txt';
$file = fopen($nome_arquivo, 'a');
fwrite($file, $texto);
fclose($file);
}
class Pessoa {
public $cpf_pessoa;
public $nome_pessoa;
public $fk_rfid;
}
class Ponto {
public $id_ponto;
public $data_ponto;
public $fk_pessoa;
}
?>
```

APÊNDICE D

Código utilizado para o desenvolvimento do servidor Android, que retorta para o aplicativo as informações dos horários das batidas de ponto.

```
<?php
header ("Content-Type:application/json");
define ('E', 'ERRO');
define ('I', 'INFO');
define ('W', 'WARN');
if (!empty($_GET['login']) && !empty($_GET['passwd'])) {
$login = $_GET['login'];
$paswd = $_GET['passwd'];
$database = new SQLite3('../Ponto.db');
insere_log(I ,"-----------------------------------------------------------------------------------------------");
$pessoa = consultaLogin ($database, $login, $paswd);
if (empty($pessoa->cpf_pessoa)) {
insere_log(E ,"USUÁRIO não encontrado - Login[" . $login . "] Senha[" . $paswd . "]");
deliver_response(200, "NAO CADASTRADO", NULL, NULL);
} else {
insere_log(I ,"CPF[" . $pessoa->cpf_pessoa . "] Nome [" . $pessoa->nome_pessoa . "] Login[" . $pessoa-
>login_pessoa . "] Senha[" . $pessoa->senha_pessoa . "] Perfil[" . $pessoa->perfil_pessoa . "]");
switch ($pessoa->perfil_pessoa) {
case 1:
deliver_response(200, $pessoa->cpf_pessoa, "PERFIL USER", $pessoa->nome_pessoa);
break;
case 2:
deliver_response(200, $pessoa->cpf_pessoa, "PERFIL ADM", $pessoa->nome_pessoa);
break;
default:
deliver_response(200, $pessoa->cpf_pessoa, "ERRO PERFIL", $pessoa->nome_pessoa);
break;
}}} else {
insere_log(E ,"-----------------------------------Invalid Request----------------------------------------");
deliver_response(400, "INVALID REQUEST", NULL, NULL);
}
function consultaLogin ($db, $get_login, $get_paswd) {
if (!$db) {
insere_log(E ,$db->lastErrorMsg());
} else {
$fetch = "SELECT cpf, nome, login, senha, perfil FROM pessoa WHERE login=\"$get_login\" and
senha=\"$get_paswd\"";
$ret = $db->query($fetch);
while($row = $ret->fetchArray(SQLITE3_ASSOC)) {
$obj = new Pessoa();
$obj->cpf_pessoa = $row['cpf'];
$obj->nome_pessoa = $row['nome'];
$obj->login_pessoa = $row['login'];
$obj->senha_pessoa = $row['senha'];
$obj->perfil_pessoa = $row['perfil'];
return $obj;
}}}
function deliver_response($status, $cpf, $perfil, $nome){
```

```
header("HTTP/1.1 $status");
$response['cpf'] = $cpf;
$response['nome'] = $nome;
$response['perfil'] = $perfil;
$json_response = json_encode($response);
echo $json_response;
}
function insere_log ($tipo ,$texto) {
$texto = "\r\n[" . $tipo . "]" . date("d/m/Y H:i:s") . " - " . $texto;
$nome_arquivo = '../logs/logAndroidLogin.txt';
$file = fopen($nome_arquivo, 'a');
fwrite($file, $texto);
fclose($file);
}
class Pessoa {
public $cpf_pessoa;
public $nome_pessoa;
public $login_pessoa;
public $senha_pessoa;
public $perfil_pessoa;
}
?>

<?php
header ("Content-Type:application/json");
define ('E', 'ERRO');
define ('I', 'INFO');
define ('W', 'WARN');
if (!empty($_GET['cpf']) && !empty($_GET['tipo'])) {
$cpf = $_GET['cpf'];
$tipo = $_GET['tipo'];
$database = new SQLite3('../Ponto.db');
if (!$database) {
} else {
switch ($tipo) {
case 1:
if ($date = $_GET['date']) {
$idponto = consultaPontoDia($database, $cpf, $date);
$batidas = consultaBatidasDia($database, $idponto);
break;
} else {
break;
}
case 2:
$date_start = $_GET['datestart'];
$date_end = $_GET['dateend'];
$idpontos = consultaPontoSemana($database, $cpf, $date_start, $date_end);
$batidasSemana = consultaBatidasSemana($database, $idpontos, $cpf);
break;
case 3:
$date_start = $_GET['datestart'];
$date_end = $_GET['dateend'];
$dias_uteis = $_GET['diasuteis'];
$idpontos = consultaPontoMes($database, $cpf, $date_start, $date_end);
$batidasMes = consultaBatidasMes($database, $idpontos, $cpf, $dias_uteis);
break;
default:
break;
}}}
function consultaPontoDia($db, $fk_cpf, $ponto_date) {
```

```
if (!$db) {
return null;
} else {
$fetchPD = "SELECT idponto FROM ponto WHERE fk_pessoa=\"$fk_cpf\" AND data=\"$ponto_date\"";
$resp = $db->querySingle($fetchPD);
return $resp;
}}
function consultaPontoSemana($db, $fk_cpf, $firstDay, $lastDay) {
if (!$db) {
return null;
} else {
$fetchPS = "SELECT idponto FROM ponto WHERE fk_pessoa=\"$fk_cpf\" AND data BETWEEN
\"$firstDay\" AND \"$lastDay\"";
$resp = $db->query($fetchPS);
$ids = array();
$i = 0;
while ($row = $resp->fetchArray(SQLITE3_ASSOC)) {
$ids[] = $row['idponto'];
$i++;
}
return $ids;
}}
function consultaPontoMes($db, $fk_cpf, $firstDay, $lastDay) {
if (!$db) {
return null;
} else {
$fetchPS = "SELECT idponto FROM ponto WHERE fk_pessoa=\"$fk_cpf\" AND data BETWEEN
\"$firstDay\" AND \"$lastDay\"";
$resp = $db->query($fetchPS);
$ids = array();
$i = 0;
while ($row = $resp->fetchArray(SQLITE3_ASSOC)) {
$ids[] = $row['idponto'];
$i++;
}
return $ids;
}}
function consultaBatidasDia($db, $fk_ponto) {
if (!$db) {
} else {
$fetchBD = "SELECT hora, situacao FROM batidas WHERE fk_ponto=\"$fk_ponto\"";
$resp = $db->query($fetchBD);
$final = "";
$entrada;
$diff = array();
while ($row = $resp->fetchArray(SQLITE3_ASSOC)) {
$final = $final . $row['situacao'] .  "-" . $row['hora'] . ";";
if ($row['situacao'] == 2) {
$saida = new DateTime($row['hora']);
$diff[] = $saida->diff($entrada)->format('%H:%I:%S');
}
$entrada = new DateTime($row['hora']);
}
$segundos = 0;
foreach ($diff as $tempo) {
list ($h, $m, $s) = explode(':', $tempo);
$segundos += $h * 3600;
$segundos += $m * 60;
$segundos += $s;
}
```

```
$horas = str_pad(floor( $segundos / 3600 ), 2, "0", STR_PAD_LEFT) ;
$segundos %= 3600;
$minutos = str_pad(floor( $segundos / 60 ), 2, "0", STR_PAD_LEFT);
$segundos %= 60;
$total = $horas . ":" . $minutos . ":" . str_pad($segundos, 2, "0", STR_PAD_LEFT);
deliver_response(200, "Consulta com sucesso", $final, $total);
return $final;
}}
function consultaBatidasSemana ($db, $fk, $pessoa) {
if (!$db) {
return null;
} else {
$fk_pontos = "";
foreach ($fk as $x) {
$fk_pontos = $fk_pontos . $x . ",";
}
$fk_pontos = rtrim($fk_pontos, ",");
$fetchBS = "SELECT hora, situacao FROM batidas WHERE fk_ponto IN ($fk_pontos)";
$fetchCarga = "SELECT carga_horaria FROM pessoa WHERE cpf=\"$pessoa\"";
$resp = $db->query($fetchBS);
$final = "";
$entrada;
$diff = array();
while ($row = $resp->fetchArray(SQLITE3_ASSOC)) {
$final = $final . $row['situacao'] .  "-" . $row['hora'] . ";";
if ($row['situacao'] == 2) {
$saida = new DateTime($row['hora']);
$diff[] = $saida->diff($entrada)->format('%H:%I:%S');
}
$entrada = new DateTime($row['hora']);
}
$segundos = 0;
foreach ($diff as $tempo) {
list ($h, $m, $s) = explode(':', $tempo);
$segundos += $h * 3600;
$segundos += $m * 60;
$segundos += $s;
}
$horas = str_pad(floor( $segundos / 3600 ), 2, "0", STR_PAD_LEFT) ;
$segundos %= 3600;
$minutos = str_pad(floor( $segundos / 60 ), 2, "0", STR_PAD_LEFT);
$segundos %= 60;
$segundos = str_pad($segundos, 2, "0", STR_PAD_LEFT);
$total = $horas . ":" . $minutos . ":" . $segundos;
$carga = $db->querySingle($fetchCarga);
$carga = $carga * 5 . ":00:00";
$secT = $secC = 0;
list ($hT, $mT, $sT) = explode (':', $total);
list ($hC, $mC, $sC) = explode (':', $carga);
$secT += $hT * 3600;
$secC += $hC * 3600;
$secT += $mT * 60;
$secC += $mC * 60;
$secT += $sT;
$secC += $sC;
if ($secT > $secC)
$segundos = $secT - $secC;
if ($secC > $secT)
$segundos = $secC - $secT;
$horas = str_pad(floor( $segundos / 3600 ), 2, "0", STR_PAD_LEFT) ;
```

```
$segundos %= 3600;
$minutos = str_pad(floor( $segundos / 60 ), 2, "0", STR_PAD_LEFT);
$segundos %= 60;
$segundos = str_pad($segundos, 2, "0", STR_PAD_LEFT);
if ($secT > $secC)
$restante = "+" . $horas . ":" . $minutos . ":" . $segundos;
if ($secC > $secT)
$restante = "-" . $horas . ":" . $minutos . ":" . $segundos;
$all = $total . ";" . $carga . ";" . $restante;
deliver_response(200, "Consulta com sucesso", $total . ";" . $carga, $restante);
return $all;
}}
function consultaBatidasMes ($db, $fk, $pessoa, $dias) {
if (!$db) {
return null;
} else {
$fk_pontos = "";
foreach ($fk as $x) {
$fk_pontos = $fk_pontos . $x . ",";
}
$fk_pontos = rtrim($fk_pontos, ",");
$fetchBS = "SELECT hora, situacao FROM batidas WHERE fk_ponto IN ($fk_pontos)";
$fetchCarga = "SELECT carga_horaria FROM pessoa WHERE cpf=\"$pessoa\"";
$resp = $db->query($fetchBS);
$final = "";
$entrada;
$diff = array();
while ($row = $resp->fetchArray(SQLITE3_ASSOC)) {
$final = $final . $row['situacao'] .  "-" . $row['hora'] . ";";
if ($row['situacao'] == 2) {
$saida = new DateTime($row['hora']);
$diff[] = $saida->diff($entrada)->format('%H:%I:%S');
}
$entrada = new DateTime($row['hora']);
}
$segundos = 0;
foreach ($diff as $tempo) {
list ($h, $m, $s) = explode(':', $tempo);
$segundos += $h * 3600;
$segundos += $m * 60;
$segundos += $s;
}
$horas = str_pad(floor( $segundos / 3600 ), 2, "0", STR_PAD_LEFT) ;
$segundos %= 3600;
$minutos = str_pad(floor( $segundos / 60 ), 2, "0", STR_PAD_LEFT);
$segundos %= 60;
$segundos = str_pad($segundos, 2, "0", STR_PAD_LEFT);
$total = $horas . ":" . $minutos . ":" . $segundos;
$carga = $db->querySingle($fetchCarga);
$carga = $carga * $dias . ":00:00";
$secT = $secC = 0;
list ($hT, $mT, $sT) = explode (':', $total);
list ($hC, $mC, $sC) = explode (':', $carga);
$secT += $hT * 3600;
$secC += $hC * 3600;
$secT += $mT * 60;
$secC += $mC * 60;
$secT += $sT;
$secC += $sC;
if ($secT > $secC)
```

```
$segundos = $secT - $secC;
if ($secC > $secT)
$segundos = $secC - $secT;
$horas = str_pad(floor( $segundos / 3600 ), 2, "0", STR_PAD_LEFT) ;
$segundos %= 3600;
$minutos = str_pad(floor( $segundos / 60 ), 2, "0", STR_PAD_LEFT);
$segundos %= 60;
$segundos = str_pad($segundos, 2, "0", STR_PAD_LEFT);
if ($secT > $secC)
$restante = "+" . $horas . ":" . $minutos . ":" . $segundos;
if ($secC > $secT)
$restante = "-" . $horas . ":" . $minutos . ":" . $segundos;
$all = $total . ";" . $carga . ";" . $restante;
deliver_response(200, "Consulta com sucesso", $total . ";" . $carga, $restante);
return $all;
}}
function deliver_response($status, $status_message, $message, $total_horas){
header("HTTP/1.1 $status");
$response['status_message'] = $status_message;
$response['message'] = $message;
$response['total_horas'] = $total_horas;
$json_response = json_encode($response);
echo $json_response;
}
?>
```