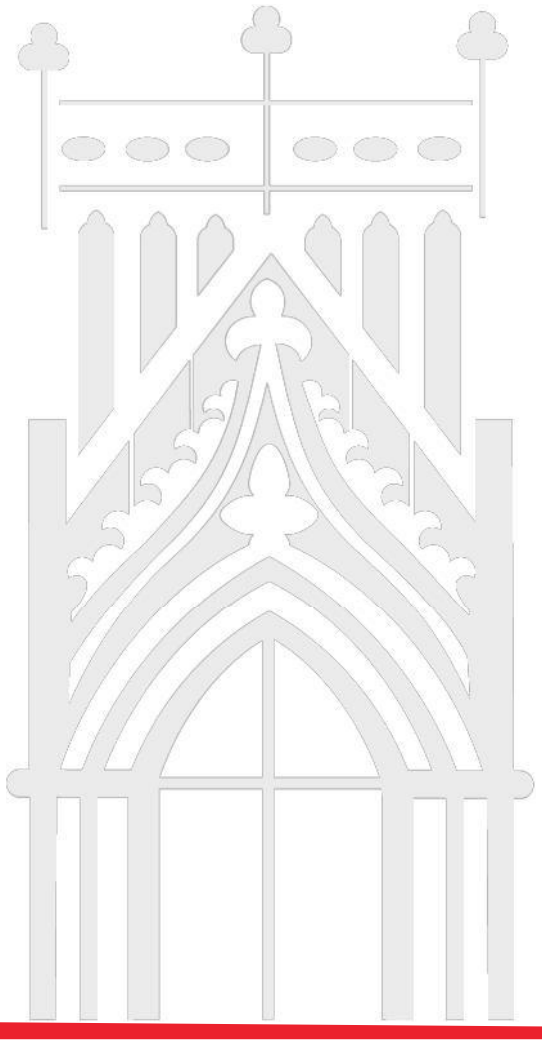
IPG Politécnico da Guarda
Polytechnic of Guarda

Mestrado em Computação Móvel

MFootBall - More Football

Noémio de Jesus da Encarnação Dória

março | 2016



Escola Superior
de Tecnologia
e Gestão

MFootBall - More Football

Noémio de Jesus da Encarnação Dória

RELATÓRIO DE PROJETO APLICADO SUBMETIDO COMO REQUISITO PARCIAL
PARA OBTENÇÃO DO GRAU DE MESTRE EM COMPUTAÇÃO MÓVEL

Março 2016

MFootBall - More Football

Noémio de Jesus da Encarnação Dória

RELATÓRIO DE PROJETO APLICADO SUBMETIDO COMO REQUISITO PARCIAL
PARA OBTENÇÃO DO GRAU DE MESTRE EM COMPUTAÇÃO MÓVEL

Orientador: Professor Doutor Paulo Alexandre Andrade Vieira
Co-orientador: Professor Doutor Adérito Neto Alcaso

Março 2016

Ao meu filho Santiago.
Que o seu olhar e a sua inteligência,
lhe permitam atingir todo o sucesso.

Agradecimentos

Gostaria de agradecer ao meu orientador Professor Doutor Paulo Alexandre Andrade Vieira e ao meu coorientador Professor Doutor Adérito Neto Alcaso pela oportunidade de desenvolvimento do presente projeto.

Deixo uma nota de agradecimento aos colegas do Projeto MagicKey, cujas orientações e ilações me fizeram enveredar pelo caminho correto, indagando sobre a melhor solução.

Um grande agradecimento à minha família, que de uma maneira ou de outra, me apoiou nesta jornada.

À Rosa pela paciência, pelo apoio e compreensão, que sempre me deram espaço de manobra para poder avançar.

Ao meu filho que, com o seu sorriso e a sua alegria, me incentivou a seguir em frente nos momentos mais difíceis.

Resumo

Este projeto centra-se no estudo e implementação de um protótipo de um sistema de gestão de entradas eletrónicas, baseado em redes sem fios, de nome MFootball.

As entradas eletrónicas são, hoje em dia, algo comum. Existem no nosso quotidiano diversas situações em que os acessos a determinadas áreas e serviços são geridos eletronicamente. Na maior parte dos casos, em entradas para eventos como espetáculos, conferências, eventos desportivos e culturais, isso já acontece.

A implementação destes sistemas é, muitas vezes, dispendiosa, dado que os seus componentes se encontram tradicionalmente conectados entre si através de cablagem. Isto implica custos na reconstrução de estruturas, na instalação de cablagens associadas e em equipamentos. A tudo isso acresce o tempo despendido no processo.

O projeto proposto visa a criação de uma solução que resolva estas questões, sendo de implementação não invasiva, sem reconstruções ou necessidade de passar grandes quantidades de cabo. Os equipamentos selecionados utilizam tecnologias de baixo custo, baseadas na plataforma Arduino. O protótipo em estudo pode caracterizar-se fisicamente por três módulos, designadamente, um módulo associado à bilheteira, outro que se encontra a nível dos utilizadores e um módulo de comunicação.

O desenvolvimento do protótipo trouxe alguns desafios, nomeadamente:

- A conectividade entre XBee;
- A conectividade via módulo de rede Wi-Fi/GPRS;
- O estudo da receção de dados do módulo de rede Wi-Fi/GPRS.

Em termos funcionais o Mfootball poderá adaptar-se a diversos cenários, com os mais diversos fins, podendo ser implementado em ambientes *indoor* e *outdoor*.

Palavras-chave: Arduino, RFID, XBee, GPRS.

Abstract

This project is focused on the implementation of a system prototype for access management based on wireless networks. Its name is MFootball.

Nowadays electronic entry systems are quite common. They exist in many situations that we face when we want to access areas or services that are managed electronically. In most cases, like conferences, sport and cultural events this is already common.

The implementation of these systems is often expensive, since their components are traditionally connected to each other through wiring. This also implies costs in rebuilding structures, installation of cabling and costs in equipment. There is also the time spent in the process.

The proposed project aims to create a solution that addresses these issues with an implementation of a non-invasive system, without reconstructions or the need to spend large amounts of cable. The selected equipment use low cost technologies, based on Arduino platform. The prototype under study can be characterized physically by three modules, namely, a module associated with the ticket selling, a module that is on users level and a communication module.

The development of this prototype brought some challenges:

- The connectivity between Xbee modules;
- The conectivity through Wi-Fi/GPRS modules;
- The reception of the data on Wi-Fi/GPRS modules comunication.

In functionally terms, the Mfootball can adapt to different scenarios with different purposes on indoor and outdoor environments.

Keywords: Arduino, RFID, XBee, GPRS.

Glossário

ACID	Atomicidade Consistência Isolamento Durabilidade
AES	*Advanced Encryption Standard*
AP	*Access Points*
API	*Application Programming Interface*
AT	*Transparent Mode*
ASP	*Active Server Pages*
bps	*bits* por segundo
CCK	*Complementary Code Keying*
CPA	*Collaboration Protocol Agreement*
CPP	*Collaboration Protocol Profile*
CORBA	*Common Object Request Broker Architecture*
CSS	*Cascading Style Sheets*
COM	*Component Object Model*
DCOM	*Distributed Component Object Model*
DSSS	*Direct Sequence Spread Spectrum*
ER	*Entity Relationship*
FCC	*Federal Communications Commission*
FHSS	*Frequency Hopping Spread Spectrum*
FTDI	*Future Technology Devices International*
GLP	Gás Liquefeito de Petróleo
GND	*Ground*
GSM	*Global System for Mobile Communications*
GPL	*General Public License*
GPRS	*General Packet Radio Service*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
ICSP	*In-circuit Serial Programming*

IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
IIS	*Internet Information Services*
ISO	*International Organization of Standardization*
JDO	*Java Data Objects*
JPA	*Java Persistence API*
JSP	*Java Server Pages*
LED	*Light Emitting Diode*
MCU	*Microcontroller Unit*
MIMO	*Multiple-Input Multiple-Output*
MU-MIMO	*Multi-User MIMO*
NFC	*Near Field Communication*
OFDM	*Orthogonal Frequency Division Multiplexing*
PC	*Personal Computer*
PWM	*Pulse-Width Modulation*
RAID	*Redundant Array of Independent Drives*
RFID	*Radio-Frequency IDentification*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SAML	*Security Assertion Markup Language*
SMTP	*Simple Mail Transport Protocol*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SSL	*Secure Socket Layer*
SGBD	Sistemas de Gestão de Bases de Dados
TCP	*Transmission Control Protocol*
TKIP	*Temporal Key Integrity Protocol*
TTL	*Transistor-Transistor Logic*
UART	*Universal Asynchronous Receiver/Transmitter*

UDDI	*Universal Description Discovery and Integration*
UHF	*Ultra-High Frequency*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VB	*Visual Basic*
VCC	Tensão de alimentação
VOIP	*Voice over Internet Protocol*
W3C	*World Wide Web Consortium*
WEP	*Wired Equivalent Privacy*
WI-FI	*Wireless Fidelity*
WPA	*Wired Protected Access*
WSDL	*Web Services Description Language*
WS-Security	*Web Services Security*
XBee	Nome comercial de produtos ZigBee da Digi International
XML	*Extensible Markup Language*
X-CTU	*Software* para configuração de rádios XBee
ZED	*ZigBee End Device*
ZigBee	Protocolo de comunicação sem fios
ZR	*ZigBee Router*

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

Nos últimos anos tem-se assistido à massificação de aplicações direcionadas para as tecnologias de redes sem fios. Consequentemente, o número de dispositivos, aplicações e soluções móveis tem crescido a grande velocidade.

De uma forma genérica, as redes sem fios encontram-se disseminadas por toda a parte e detêm diversos papéis, desde a conectividade à *Internet*, às chamadas de voz, à monitorização e controlo de diversos elementos, incluindo a sua identificação e contabilização.

O projeto apresentado visa a criação de uma solução para entradas eletrónicas, de fácil implementação, e de forma não invasiva, podendo instalar-se sem necessidade de obras ou operações de grande envergadura, que se adapte às estruturas já existentes e que utilize componentes de baixo custo e baixo consumo. Este projeto integra diversas tecnologias, nomeadamente:

- Wi-Fi (*Wireless Fidelity*): encontram-se tão enraizadas, que podem ser encontradas nos mais diversos sítios e, em muito casos, ser de acesso gratuito. A este facto está associada a dependência atual da *Internet* por parte da sociedade;
- Zigbee: é um conjunto de especificações de comunicação sem fios entre dispositivos eletrónicos de baixo custo e operando através de baixos consumos. Podem encontrar-se estas especificações em redes de sensores, de monitorização e controlo;
- RFID (*Radio-Frequency IDentification*): apesar de existir há algum tempo, tem visto os seus horizontes alargarem-se de diversas formas, podendo ser encontrada em muitas das aplicações e serviços utilizados hoje em dia, sem que para muitos de nós seja percetível.

O desenvolvimento do presente projeto visa a criação de uma solução definida por:

- Componentes de baixo custo;
- Comunicação sem fios, quer nos elementos identificativos, quer nos componentes;
- Fácil implementação, ou seja, sem a necessidade de realizar obras.

Desta forma, podem contemplar-se diferentes cenários, como pequenos estádios, pavilhões desportivos, e/ou espaços no exterior.

O desenvolvimento do relatório encontra-se dividido em capítulos:

- O capítulo 2 apresenta o estado da arte relativo às tecnologias associadas ao projecto desenvolvido;
- O capítulo 3 apresenta os procedimentos efetuados no estudo das tecnologias apresentadas;
- O capítulo 4 apresenta os testes efetuados e os resultados obtidos durante o desenvolvimento;
- No capítulo 5, podem ver-se as conclusões finais e o trabalho futuro pensado para o projeto.

2. O Estado da Arte

Tendo em conta que neste projeto se utilizaram algumas tecnologias já conhecidas, será pertinente fazer a sua apresentação e analise.

2.1. Arduino

O Arduino é uma plataforma de *hardware* livre, de baixo custo e destinada ao público em geral. Devido à sua simplicidade e acessibilidade permite a sua utilização em vários projetos que se destacam em diversos campos, tendo sido adotado também neste projeto.

2.1.1. Conceito

Uma das características do Arduino é a sua simplicidade. Este dispõe de um controlador, algumas portas de entrada e de saída de tipo analógico, outra digital e uma interface USB (*Universal Serial Bus*), a qual que permite a sua programação em tempo real. Dispõe de componentes complementares para facilitar a programação e incorporação para outros circuitos. Um aspeto importante é a maneira padrão como os conectores são expostos, permitindo a conexão a outros módulos expansivos, conhecidos como *shields*, que permitem que o Arduino ganhe mais versatilidade. (BANZI, 2008)

Esta placa, que está a difundir-se cada vez mais a nível mundial, surgiu em Itália, em 2005, com o intuito de ser integrado em projetos escolares de forma a obter uma plataforma de baixo custo e uso fácil. Em 2006, o seu sucesso foi sinalizado com a atribuição de uma menção honrosa na categoria Comunidades Digitais, pela Prix Ars Electronica.

O utilizador pode montar a placa ou adquiri-las previamente montadas, pode também ser alterada conforme necessário. Tendo em conta esta característica, muitos projetos paralelos inspiram-se em cópias modificadas com placas de expansão, acabando por receber seus próprios nomes. Este facto deve-se também à disponibilização, por parte dos criadores, dos esquemas e dos planos de construção das placas.

Até 2013, venderam-se mais de 700 000 placas, tendo estas servido nos sectores da educação, em projetos independentes e até em arte.

2.1.2. Arquitetura

O Arduino base dispõe de um microcontrolador Atmel AVR de 8 *bits*. Os Arduinos originais utilizam a série de *chips* megaAVR, especialmente os ATmega8, ATmega168, ATmega328 e ATmega1280. Porém, muitos outros processadores foram utilizados por clones.

Na sua maioria incluem um regulador linear de 5V e um oscilador de cristal de 16 MHz e é pré-programado com um *bootloader* que simplifica o carregamento de programas para uma memória *flash* embutida no MCU (*Microcontroller Unit*).

As placas de expansão ligam-se através de uma conexão série, mas a forma como este processo é efetuado varia conforme a versão. As placas série contêm um simples circuito para converter entre os sinais dos níveis RS-232 (*Recommended Standard 232*) e TTL (*Transistor-Transistor Logic*). Atualmente, existem alguns métodos diferentes para realizar a transmissão dos dados, como por exemplo placas programáveis via USB, adicionadas através de um *chip* adaptador USB-para-Serial como o FTDI (*Future Technology Devices International*) FT232.

A maioria dos pinos de entrada e saída dos microcontroladores são para interligação a outros circuitos. A versão *UNO*, que substituiu a *Duemilanove*, por exemplo, disponibiliza 14 pinos digitais, 6 das quais podem produzir sinais PWM (*Pulse-Width Modulation*), além de 6 entradas analógicas.

2.1.3. Tipos de Arduino

Existem algumas variantes físicas no mercado. Algumas destas representam a evolução do Arduino original, enquanto que outras são variações que visam dar ao Arduino alguns fatores específicos, como se pode observar no parágrafo seguinte e na Tabela 1.

Arduino *Duemilanove*: o Arduino *Duemilanove* (“2009”) foi a primeira placa microcontrolador, baseada no Atmega168 ou Atmega328, a ter sucesso e resultou da evolução de uma placa inicial denominada *Diecimila*. Esta placa está descontinuada e já não se encontra à venda.

Arduino Uno: esta é a placa que se tornou popular em todo o mundo, sendo uma evolução direta do Arduino *Duemilanove*. Esta foi uma tentativa da empresa em criar uma referência às gerações futuras do Arduino. Diferencia-se das suas antecessoras, porque o *chip* FTDI foi substituído por um MCU, que estão disponíveis em cima da placa, através de conectores fêmeas de 0,1 polegadas (0,25 centímetros).

Arduino Mega 1280: esta placa é baseada no ATmega1280. Dispõe de 54 linhas de entrada/saída digital, 16 portas analógicas de *input*, 4 UARTs (*Universal Asynchronous Receiver/Transmitter*: portas série via *hardware*), um oscilador de cristal a 16MHz, conexão USB, conexão de energia, cabeçalho ICSP. Esta é no fundo uma versão musculada do Arduino UNO. A sua evolução direta é o Arduino Mega 2560

Arduino Nano: este Arduino é compacto e foi criado para ser compatível com as placas *breadboard*. É muito completo, sendo a sua funcionalidade muito semelhante ao Arduino *Duemilanove*.

Fornece conectores machos na parte de baixo da placa, para serem ligados em placas *protoboards*.

Tabela 01 - Comparação de diferentes tipos de Arduinos.

Arduino	**Processador**	**Flash KiB**	**EEPROM KiB**	**SRAM KiB**	**com PWM**	**Pinos digitais**	**Pinos analógicos**	**Tipo de interface USB**	**Dimensões mm**
Diecimila	ATmega168	16	0.5	1	6	14	6	FTDI	68.6 × 53.3
Duemilanove	ATmega168/328P	16/32	0.5/1	½	6	14	6	FTDI	68.6 × 53.3
Uno	ATmega328P	32	1	2	6	14	6	ATmega8U2	68.6 × 53.3
Mega	ATmega1280	128	4	8	14	54	16	FTDI	101.6 × 53.3
Mega2560	ATmega2560	256	4	8	14	54	16	ATmega8U2	101.6× 53.3
Nano	ATmega168/328	16/32	0.5/1	½	6	14	8	FTDI	43 × 18

Fonte: Adaptado de: https://pt.wikipedia.org/wiki/Arduino

2.1.4. Clones Arduino

Embora a política oficial enfatize que o projeto é aberto para a incorporação de trabalhos paralelos no produto original e apesar do *hardware/software* ser desenhado sob licenças *copyleft*, os desenvolvedores expressaram o desejo de que o nome Arduino (ou derivados) fosse exclusivo para o produto oficial e que não fosse usado para trabalhos de terceiros sem autorização.

Assim sendo, um grupo de utilizadores criou um projeto alternativo, baseado na versão *Diecimila*, chamado de *Freeduino*, sendo que o nome não infere nos direitos de autor, é livre para ser usado para qualquer fim.

Outros produtos, compatíveis não oficiais e que obtiveram êxito em lançamentos, possuem a terminação *duino* como forma de se referenciar ao dispositivo da qual derivaram.

Existem placas compatíveis, denominadas como clones de *bootloader* compatível, com o *software* do Arduino, mas não expansíveis, ou seja, não aceitam *shields*. Possuem também diferentes conectores para energia e E/S.

2.1.5. Programação

O Arduino dispõe de um Ambiente Integrado de Desenvolvimento (IDE), multiplataforma escrito em Java, derivado dos projetos *Processing* e *Wiring* (Arduino, 2014). Está definido para introduzir a programação a artistas e a pessoas não familiarizadas com o desenvolvimento de *software*. Inclui um editor de código, com recursos de realce de sintaxe, parênteses correspondentes e indentação automática, sendo capaz de compilar e carregar programas para a placa com um único clique.

Através da biblioteca *Wiring*, possui a capacidade de ser programado em C/C++, o que permite criar com facilidade as mais diversas operações, sendo apenas necessário definir duas funções no pedido para fazer um programa funcional:

- *setup*(): inserida no início, na qual pode ser usada para inicializar configuração;
- *loop*(): chamada para repetir um bloco de comandos ou esperar até que seja desligada.

Em seguida, pode ver-se um exemplo do IDE (Figura 01):

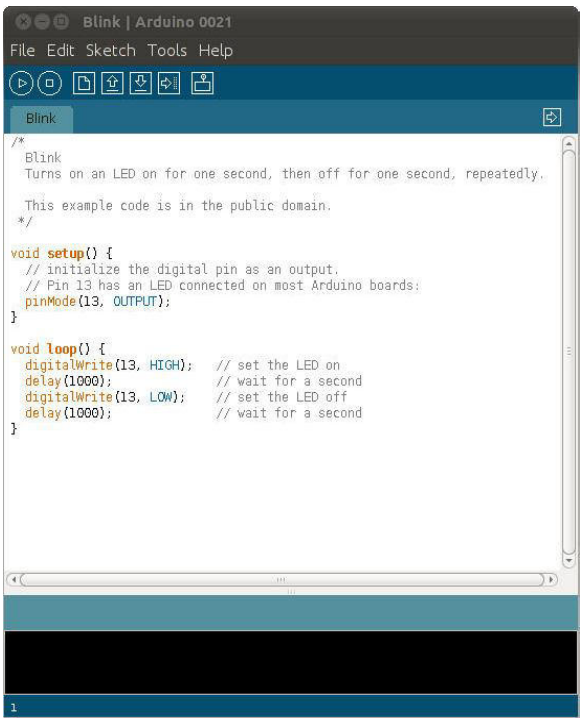


Figura 01 - Ambiente Integrado de desenvolvimento Arduino.

Fonte: Elaboração própria.

2.1.6. Algumas aplicações Arduino

Como esta plataforma é livre e versátil, existem inúmeros projetos académicos, artísticos e não comerciais, desenvolvidos e em desenvolvimento em diversos ramos.

2.1.6.1. Ideos c6o redux

Este projeto desenvolvido pela Ideos (HARTMANN, 2014) baseia-se no conceito das cassetes áudio dos anos 80 e 90. Contudo, visa a utilização de tecnologia moderna. O funcionamento deste dispositivo é simples, fazendo lembrar um disco de vinil, onde o utilizador pousa um cartão de papel, seleciona a música que pretende ouvir e passado um segundo esta toca. O cartão tem a ilustração de uma cassete áudio, representando assim uma lista de reprodução. Estes cartões dispõem de etiquetas RFID, que está conectado a um computador, dispõe internamente de uma série de antenas colocadas, para que os cartões deixados no dispositivo possam ser lidos sem que seja necessário o cartão ter uma posição padrão. Ainda dentro do dispositivo, existem alguns

Arduino Pro Mini que processam a informação e comunicam com o computador, que reproduz a lista pedida através de colunas de som conectadas. O protótipo pode ser visto na Figura 02.

Figura 02 - Ideos c6o redux em funcionamento.

Fonte: https://labs.ideo.com/2011/01/14/c60-evolution-of-an-idea/

2.1.6.2. Veículos voadores não tripulados

Existem diversos projetos que levaram ao surgimento de pequenas aeronaves, não tripuladas, que podem fazer voos de reconhecimento.

Entre eles, encontram-se os denominados *quadcopters* (RCvertt, 2014) que dispõem de 4 hélices e os *hexacopters,* com 6 hélices. Este tipo de veículos não tripulados são bastante estáveis, rápidos e manobráveis. O Arduino atua aqui como controlador de voo, monitorizando sensores, acelerómetros e giroscópios e atuando conforme necessário para aplicar a ação desejada.

Alguns exemplos, apresentados na Figura 03, são:

- Caspiquad (caspiquad, 2014)(a);
- AeroQuad (The open source quadcopter / multicopter, 2014)(b).

Outros, mais leves, como balões meteorológicos (Spacebits, 2014) e zepelins (Muñoz, 2014) dispõem também de Arduino. São utilizados para monitorizar sensores, medir temperaturas e registo de posicionamento global.

a)

b)

Figura 03 - Exemplos de *quadcopters* Arduino.

Fonte: https://code.google.com/archive/p/caspiquad
Fonte: http://aeroquad.com/content.php

2.1.6.3. Robótica

O Arduino também se encaixa no sector da robótica, já que a sua versatilidade permite que possa ser aplicado como plataforma de controlo, podendo manobrar elementos como um braço-robô (Larry, 2014).

Pequenos robôs podem ser produzidos, tendo como placa de controlo o Arduino. Este pode tomar decisões em relação às ações a efetuar, conforme os dados obtidos através dos sensores que lhe

estão a associados (BRENN, 2014). Atualmente, existem *kits* que servem de tutoriais simples e permitem aos entusiastas tomarem conhecimentos introdutórios sobre os conceitos e paradigmas da robótica. Contudo, existem projetos mais complexos como robôs hexápodes (Royer, 2014), conforme se pode ver na Figura 04.

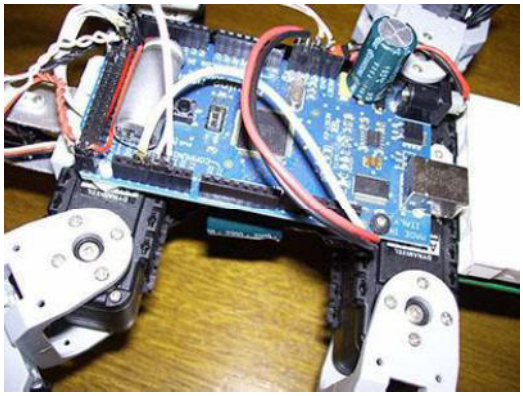


Figura 04 - Hexápode Arduino.

Fonte: https://blog.arduino.cc/2011/03/29/samsa-ii-the-hexapod

2.2. RFID

A tecnologia de identificação por radiofrequência, mais conhecida como RFID está largamente difundida nos tempos que correm. A sua existência em diversos sectores permite facilitar serviços à volta do globo.

2.2.1. Conceito

O conceito surgiu antes da 2ª Guerra Mundial e tem sofrido diversos desenvolvimentos e aplicações até aos dias de hoje. A sua funcionalidade baseia-se na identificação de diversos elementos, sem a utilização de fios ou a necessidade de contacto.
Para isso, os elementos a identificar dispõem de uma etiqueta (TAG) que ao aproximar-se à antena de um leitor é lida, recolhendo a informação que lhe está associada, para mais tarde ser processada por um elemento computorizado. Na Figura 05 pode ver-se o conceito de funcionamento.

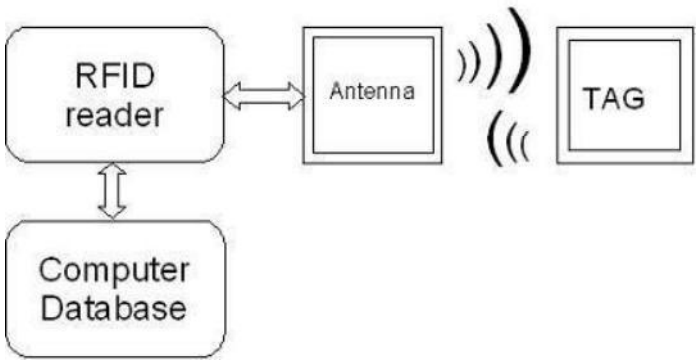


Figura 05 - Funcionamento RFID.

Fonte: http://www.scienceprog.com/wp-content/uploads/2013/01/RFID_tagging1.jpg

Atualmente existem dois sistemas de RFID:

- **Sistemas de baixa frequência a 125 kHz:** mais pequenos, mais baratos e transmitem a uma distância pequena (máximo cerca de 10 cm).
- **Sistemas de alta frequência a 13.56 MHz:** estes sistemas caracterizam-se pela leitura a uma distância considerável (até 1 metro). Os preços são mais elevados comparados com os sistemas anteriores.

2.2.2. Etiquetas

As etiquetas constituem o elemento crucial para o funcionamento desta tecnologia. As etiquetas RFID são constituídas por *transponders* (Transponder, 2014). O *transponder* ou transpondedor (abreviação de *Transmitter-Responder*) é um dispositivo de comunicação cujo, objetivo é receber, amplificar e retransmitir um sinal a uma frequência diferente ou transmitir uma mensagem pré-definida de uma determinada fonte.

A arquitetura do *transponder* é fortemente direcionada para a aplicação a que se destina, pelo que se verifica que não há um único modelo que seja indicável para todos os usos. Isso ocorre devido à interdependências de certas características, do impacto que têm umas sobre as outras e todas sobre o preço. Dos atributos do *transponder* podem salientar-se a presença de *chip*, uma antena, a forma de alimentação, as características da memória, e a frequência de operação.

Os *transponders* mais sofisticados dispõem de circuitos integrados que gerem o dispositivo, sendo responsáveis pela gestão do armazenamento de memória, gestão de colisões e gestão de funcionalidades de alto nível. Contudo, a simplicidade das etiquetas desprovidas de circuito integrado, que respondem apenas com uma resposta padrão, são normalmente as preferidas pelo seu preço reduzido.

As etiquetas podem também catalogar-se como ativas ou passivas. As etiquetas passivas são etiquetas desprovidas de bateria e cuja energia é disponibilizada pelo leitor. A energia é retirada do campo magnético criado na antena, associada à etiqueta, que recebe as ondas rádio provenientes do leitor. (YAN & LEE, 2009)

As vantagens das etiquetas passivas (Passive RFID Tag (or Passive Tag), 2014) são:

- O preço: são baratas de produzir;
- A longevidade: podem durar décadas, uma vez que não têm baterias;
- O tamanho reduzido, pode ter a dimensão de um grão de arroz, que lhes dá uma grande versatilidade, podendo ser utilizadas para diversos fins.

As desvantagens destas abrangem:

- As curtas distâncias para leitura, limitando a sua utilização em certas aplicações;
- A sua informação pode ser lida, mesmo depois de a sua utilização ser finalizada.

As etiquetas ativas (Active Tag , 2014) podem estar equipadas com baterias, podendo utilizar a sua energia na totalidade ou parcialmente. Algumas dispõem de baterias que podem ser substituídas, para que a sua utilização seja prolongada. É também possível conectar estas etiquetas a uma fonte de energia exterior.

Existem também as etiquetas que emitem as informações pretendidas em determinados intervalos de tempo, bem como as que apenas respondem a interrogações de leitores.

Existem algumas vantagens comuns entre todas as etiquetas ativas, nomeadamente:

- O longo alcance de comunicação;
- A capacidade de controlar e monitorizar de forma independente;
- A capacidade de iniciar comunicações;
- A capacidade de executar diagnósticos;

- A maior largura de banda;
- A possibilidade de estarem equipadas com redes autónomas, podendo determinar o melhor caminho de comunicação.
- Podem estar associados a sensores.

Podem enumerar-se diversas desvantagens destas etiquetas, designadamente:

- Necessidade de energia: as baterias têm vida limitada. Desta forma, as etiquetas têm vida limitada;
- São mais caras que as passivas;
- São fisicamente grandes, o que pode limitar algumas utilizações;
- A manutenção, a longo prazo, pode ser dispendiosa;
- Falhas de bateria podem levar a leituras erróneas.

No que concerne à memória, encontra-se uma grande diversidade de tamanhos, tipos e implementações. Há as que permitem somente leitura, vindo programadas de fábrica com um código único, atribuído aleatoriamente. Estas são predominantemente passivas e sem *chip*, estão entre as mais baratas e são largamente utilizadas na etiquetagem. Outras permitem que se escreva uma vez, fazendo com que, numa linha de produção, lhes seja atribuído um número, o qual não poderá ser apagado. As que implementam total possibilidade de leitura e escrita tem originado aplicações interessantes dada a capacidade de atualizar as informações, sempre que necessário, podem armazenar-se critérios de inspeção, andamento, características físicas entre outras, etc.

As faixas de operação distinguem-se, em termos de características passadas aos *transponders*, pelas taxas de transmissão atingidas, pela capacidade de operar próximo a metais e superfícies molhadas e pelo tamanho da antena num dispositivo passivo.

Quanto mais baixa a frequência de operação, melhor o desempenho, quando próximo de superfícies metálicas e menor a taxa de transmissão atingida. As etiquetas que operam até 125 kHz são as menos suscetíveis a degradações de performance, mas pecam pelo tamanho das antenas e um raio de leitura de menos de meio metro. Ainda assim, por ser uma tecnologia amadurecida, tem a maior base instalada no mercado. Na faixa dos 13.56 MHz encontramos *transponders* mais baratos, e com raio de leitura de até um metro, usados principalmente em cartões "inteligentes", rastreios a nível de *itens* por não necessitarem de múltiplas leituras à distância. Entre 868 e 915MHz

encontram-se as etiquetas com potencial para serem as mais baratas em grandes quantidades. Costumam ser utilizadas em cobrança automatizada de portagens por permitirem leituras até três metros de distância, porém alguns países proíbem ou restringem transmissões nessa banda, por ser reservada. As etiquetas no campo das micro-ondas permitem um sinal mais direcional e altas taxas de leitura, o que as torna ideais para certas aplicações. Por serem as mais suscetíveis a degradações devidas ao meio, o seu raio de leitura é reduzido e está próximo de um metro a 13,6 MHz.

2.2.3. O Leitor

O leitor RFID é a camada intermédia desta tecnologia, tendo as suas características ditadas pelas etiquetas, com que deve ser compatível, e pelas funcionalidades impostas pelo sistema de informação. Por serem em número relativamente menor dentro do sistema, ficam menos sujeitos às restrições de preço e tamanho que costumam guiar o projeto das etiquetas. A tendência do caminho seguido pela indústria é no sentido de melhorar a sua performance, compatibilidade, confiabilidade.

O sucesso da leitura depende de alguns fatores, particularmente :

- Da posição do *transponder* relativa à antena (polaridade);
- Da degradação provocada pelo meio; compatibilidade entre as partes;
- Das interferências eletromagnéticas.

O campo eletromagnético oscilante associado ao RFID tem direções fixas determinadas pela antena que o gera. Quanto mais alinhada a antena da etiqueta estiver com a direção de oscilação do campo elétrico, maior será a corrente induzida no circuito e melhor será a leitura. Variando a rotação, pode ir-se de um ponto de leitura ótima, até outro onde ela não ocorra, o que influencia a distância de leitura Figura 06.

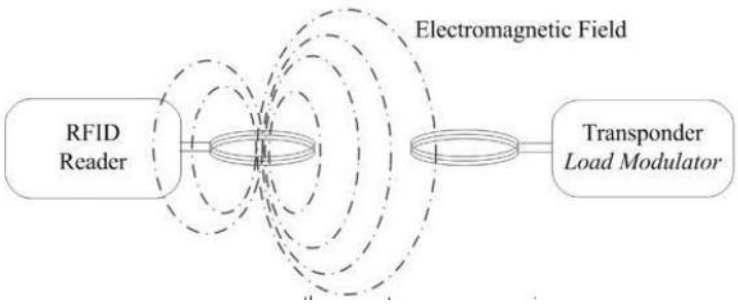


Figura 06 - Campo eletromagnético de leitura RFID.

Fonte: http://physics.stackexchange.com/questions/44037/why-is-near-field-communication-nfc-range-limited-to-about-20cm

O meio influencia a qualidade de transmissão do sinal. Dependendo do modelo e frequência de operação da etiqueta, esta pode tornar-se completamente inoperante quando próxima de uma superfície metálica. De forma semelhante, ondas eletromagnéticas têm, em geral, dificuldade em atravessar materiais que contenham água, a matéria orgânica em geral incluída. Quando tais situações são inevitáveis é necessário tomar medidas paliativas, como o emprego de frequências e potências adequadas. Ainda assim, é impossível ler através de metal.

Como ainda não foi adotado um padrão de comunicação RFID, cada classe de etiquetas opera de uma forma diferente. Há diferenças no protocolo de comunicação, na codificação, na modulação, e no modo de operação, isso considerando apenas uma mesma banda de operação. Em operações de busca há casos em que se tem que lidar com etiquetas de diferentes marcas e modelos, o que torna imperativo o emprego de modelos versáteis que permitam a operação. Os primeiros passos em direção à maior padronização estão a ser dados, porém, enquanto as diferenças existirem, a compatibilidade continuará a ser uma questão de suma importância.

Quanto mais uma onda se propaga, através de um meio maior, a probabilidade de interferir com outra existe, efeito que é especialmente grave quando estão todas na mesma frequência. Ao fazer a leitura simultânea de várias etiquetas semelhantes, as suas transmissões misturam-se no meio, impedindo que qualquer uma seja compreendida, o que origina uma colisão. Como o raio de leitura de uma etiqueta é limitado, pode fazer-se uma separação espacial da etiqueta sendo lida, mas isso nem sempre é possível. É necessário então, que seja implementado um protocolo anti colisão, que faça com que elas transmitam cada uma a seu tempo.

A pensar na finalidade a que se destinam, encontram-se no mercado desde pequenos leitores de mão, do tamanho de um telemóvel, nos dispositivos móveis que, em alguns casos incorpora a

variante RFID conhecida como NFC, até grandes leitores do tamanho de portais, que utilizam mais de uma antena, para garantir um nível aceitável de confiança. Disso depende também a arquitetura do posicionamento no ambiente. Num leitor manual, o utilizador pode variar a posição até que a leitura seja bem-sucedida. Para tal, uma pequena antena costuma bastar. Quando se trata de um ponto de leitura numa linha de produção, espera-se uma leitura precisa durante o tempo de passagem do *item*, o que costuma exigir o posicionamento estratégico de algumas antenas que ofereçam certeza de leitura, uma vez que o processo é influenciado pela posição angular da etiqueta.

O papel do leitor na automação industrial está em discussão atualmente: se o leitor deve ser mais uma entrada no esquema de automação ou considerado parte da infraestrutura de tecnologia da Informação presente na empresa.

Ao longo da sua existência entre as suas diversas aplicações foram também estudadas soluções que visam a integração desta tecnologia com as redes sem fios (SAMANO-ROBLES & A., 2009).

2.2.4. Algumas aplicações RFID

A difusão desta tecnologia é de tal ordem que pode ser encontrada em diversos sectores, nomeadamente:

2.2.4.1. Mineração e siderurgia

Este sector caracteriza-se por ambientes hostis, onde o campo visual é por vezes nulo. As características da tecnologia RFID dão-lhe vantagem para a sua utilização na identificação de toda a maquinaria envolvida, bem como, a identificação e catalogação dos processos de extração e produção. Pode ainda ser utilizada para a criação de zonas de segurança, evitando acidentes.

2.2.4.2. Transporte e tráfego

No que toca ao tráfego, esta tecnologia é utilizada para facilitar as cobranças nas vias, como a via verde, onde o trafego é intenso, tal como em autoestradas. Permite também, a identificação de veículos. A nível dos transportes rodoviários permite a utilização de cartões inteligentes como passes ou cartões de pagamento. No sector ferroviário pode também utilizar-se para identificação de vagões.

Nos transportes podem ver-se aplicações desta tecnologia, no que toca a bilhetes eletrónicos. Por exemplo, no metro de Lisboa e nos transportes rodoviários "Horários do Funchal".

O controlo de viaturas é um fator importante neste sector, não apenas para saber quais e quantas viaturas se encontram na estação ou em estaleiro, mas também como saber as que estão em serviço, bem como a deslocação no seu percurso. Há ainda a possibilidade de informar os utentes do tempo estimado para a passagem do veículo, através de paragens inteligentes.

2.2.4.3. Logística e armazenamento

Neste sector, esta tecnologia tem tido uma forte aceitação devido à sua versatilidade. A identificação de paletes e caixas com etiquetas RFID permite a automatização de todo o processo, evitando ao máximo a intervenção humana. Portais de leitura podem gerir a movimentação de produtos em tempo real, enviando as informações em redes Wi-Fi para os servidores da empresa.

2.2.4.4. Petróleo e gás

Os sistemas RFID têm ampla utilização nas empresas petrolíferas, distribuidoras de gás liquefeito de petróleo (GLP) e produtoras de gases especiais (atmosféricos). As etiquetas RFID são utilizadas para identificação de válvulas de controlo ou equipamentos ativos, como cilindros de gás. O uso de etiquetas ativas pode também ser visto em plataformas de prospeção para monitorização de pessoas e no controlo de movimentação de materiais. Em determinados casos, torna-se necessário o uso de leitores RFID homologados para uso em ambientes com risco de explosão.

2.2.4.5. Farmacêutica e hospitalar

Estes sectores, que lidam com a saúde e o bem-estar das pessoas, são altamente sensíveis e exigem controlo absoluto dos processos que lhe são associados, sob pena de causar grandes constrangimentos às empresas que lidam com esse mercado. Dentro deste contexto, a tecnologia RFID tem desempenhado um papel muito importante em várias aplicações, abrangendo desde a identificação de medicamentos controlados de laboratórios farmacêuticos até o auxílio no rastreamento de ativos, pacientes e profissionais em hospitais. Trata-se de um mercado com grande potencial, mas com pouca adoção desta tecnologia, até o momento.

2.2.4.6. Indústria automóvel

Monitorizar cada etapa de produção de um automóvel é vital para qualquer produtora mundial nos dias que correm. As etiquetas são aplicadas desde a inserção do monobloco na linha, acompanham o veículo ao longo do processo de montagem, passando pelo forno de pintura, até aos testes de controlo de qualidade no final da produção. Com o uso da tecnologia RFID, aumentou muito a flexibilidade de produção, podendo fabricar-se diversos modelos e configurações numa mesma linha de produção. A indústria de peças de automóveis também têm utilizado esta tecnologia em larga escala, na identificação em linhas de faróis, sistemas de ar condicionado, motores, entre outros.

2.2.4.7. Segurança de ativos

Entre as características da tecnologia RFID existe uma que a torna uma ferramenta ideal para o sector da segurança de bens, como equipamentos portáteis, instrumentos, obras de arte, peças de vestuário, bagagens e outros que são normalmente alvos da cobiça de marginais. A utilização de etiquetas RFID ativas pode considerar-se uma das medidas mais seguras e efetivas para proteger esses bens. Estas permitem monitorizar automaticamente movimentos dos bens ou a remoção destes.

2.2.4.8. Lavandaria e têxtil

A utilização de minúsculas etiquetas RFID em uniformes de empresas, enxovais de hotéis e hospitais é uma excelente opção para lavandarias industriais que precisam identificar cada peça de forma automática. Estas etiquetas suportam lavagens e a manipulação das peças nas várias etapas de uma lavandaria identificadas de forma única. Leitores especialmente desenhados permitem a leitura de uma ou várias peças em antenas tipo tubos ou painéis, interligados ao sistema central. Com isso, cada etapa de lavagem é monitorizada em tempo real.

2.2.4.9. Controlo da manutenção

A manutenção e calibração de equipamentos de forma periódica pode ser passível da utilização da tecnologia RFID para controlo *offline* ou *online* aplicadas a elevadores, máquinas copiadoras, impressoras, válvulas, automóveis, caminhões, máquinas, instrumentos de medição, aeronaves, embarcações, etc. Normalmente munidos de memória de armazenamento, essas etiquetas RFID

formatadas são lidas e programadas por leitores RFID portáteis que podem estar conectados em tempo real por meio de GPRS (*General Packet Radio Service*) ou Wi-Fi ou ainda serem operados em modo *batch* (*offline*).

2.3. ZigBee

A designação ZigBee é utilizada para especificar protocolos de comunicação de alto nível, com baixos consumos e baixo custo.

2.3.1. Conceito

Com base no padrão IEEE 802, mais especificamente no padrão IEEE 802.15.4 a operação do ZigBee baseia-se em sinais de radiofrequência, não licenciados.

Os dispositivos ZigBee, de baixo consumo, podem fazer passar a informação entre si de forma a fazê-la chegar a postos mais distantes, criando assim uma rede *mesh*. Esta rede caracteriza-se por ser descentralizada. Desta forma, estes dispositivos são muito utilizados devido à longa vida de bateria e à segurança.

A especificação ZigBee está pensada para utilização em áreas como:

- Redes de sensores;
- Controlo Industrial;
- Domótica;
- Automatização;
- Sensores embebidos;

Os dispositivos ZigBee podem dividir-se em 3 grupos, nomeadamente:

- Coordenador *(ZigBee Coordinator (ZC))*: serve como raiz de uma rede, que pode também conectar-se a outras redes. Existe apenas um em cada rede, podendo atuar como repositório de segurança e armazenando a informação da rede.
- Roteador (*ZigBee Router (ZR))*: pode apenas passar informação como roteador intermédio.

- Dispositivo Final *(ZigBee End Device (ZED))*: comunica apenas com dispositivos, superiores sem a capacidade de passar a informação de outros dispositivos podendo o dispositivo estar em modo dormente, o que permite poupar energia. Estes podem ser mais baratos que os outros dispositivos, por necessitarem de menos memória.

2.3.2. XBee

XBee é o nome dado pela Digi International a uma família de módulos de rádio ZigBee. Os primeiros módulos foram introduzidos no mercado pela MaxStream, em 2005, baseando-se na norma IEEE 802.15.4-2003, que convenciona as conexões, pelo ar, ponto a ponto e ponto a multiponto e a taxas de 250kbps. Foram lançados dois modelos: o primeiro, o XBee de 1mW e o segundo, o XBee-PRO, a 100mW. Após esta introdução, os XBee passaram a ser comercializados pela empresa Digi.

O XBee pode ser acoplado ao Arduino através de uma placa de expansão. Sendo o seu alcance de cerca de 3 m em interior e de 9 m no exterior. Pode ser utilizado em modo de linha de comandos e configurar uma variedade de opções de conexões desde *broadcast* a *mesh*.

2.3.3. Algumas aplicações XBee

Existem diversos projetos que têm o seu desenvolvimento sobre a base do XBee.

2.3.3.1. Openenergymonitor

A monitorização dos gastos dos recursos energéticos é cada vez mais uma tendência quotidiana. Assim, surgem soluções nesse sentido, tais como este projeto, que visa monitorizar e registar a energia despendida por determinado dispositivo numa habitação. (functional home energy monitor, 2014) (Figura 07).

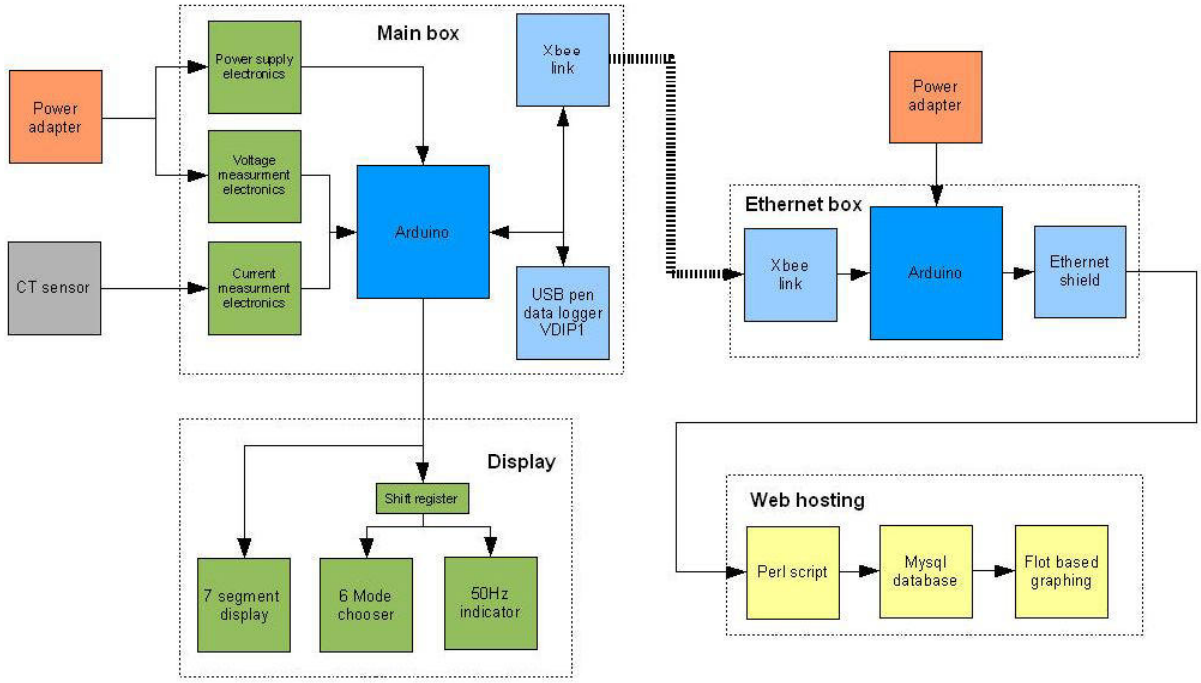


Figura 07 - Esquema do *Open Energy Monitor*.

Fonte: http://openenergymonitor.org/emon/

2.3.3.2. LCD *Messaging*

Este projeto visa a criação de uma solução que permita a exposição de mensagens em diversos pontos e que estas possam sofrer alterações frequentes, através de uma rede sem fios.

A arquitetura deste sistema, dispõe de um servidor de mensagens que comunica com um cliente. Esta comunicação é efetuada através de XBee e resume-se à transmissão de uma mensagem que será exposta pelo cliente. Para tal, o cliente utiliza um XBee, conectado a um Arduino, que por sua vez projeta uma mensagem num ecrã LCD (TACTICAL TEXTING IN PUBLIC SPACES, 2014).

2.3.3.3. The Pool

Este projeto visa a criação de um ambiente interativo, constituído por círculos concêntricos. Ao entrar nesta atmosfera os utilizadores podem interagir de forma colaborativa no sentido de obter diversos padrões de cores (Meet Jen Lewin and "The Pool", 2014).

Cada círculo está equipado com um dispositivo sem fios e que comunicam entre si (Figura 08). Não existe um dispositivo mestre, nem rotas ou conexões de controlo (Gomba, 2010), isto é

possível porque cada círculo é independente e interage com o ambiente, segundo a ação do utilizador.

Figura 08 - Projeto" The Pool" em funcionamento.

Fonte: http://lumigeek.com/2013-the-pool-by-jen-lewin/

2.4. Wi-Fi

Wi-Fi designa a tecnologia utilizada para a transmissão de dados, através do protocolo TCP/IP, sem a utilização de qualquer tipo de suporte físico, ou seja, para a conectividade sem fios utilizando as normas de redes cabladas TCP/IP (*Transmission Control Protocol/Internet Protocol*).

Atualmente, esta tecnologia está difundida em equipamentos como televisores, consolas de videojogos, *smartphones*, câmaras de vídeo, entre outros. Estes podem conectar-se a pontos de acesso bem como, rotadores para o acesso à rede ou *Internet* a grandes velocidades. Em interiores, as distâncias de acesso limitam-se a 20 metros e no exterior o limite é muito superior.

2.4.1. Padrões

O IEEE (*Institute of Electrical and Electronics Engineers*) definiu diversos padrões para a comunicações sem fios, representados na Tabela 02:

Tabela 02 - Padrões IEEE mais conhecidos.

Padrão	Frequências em GHz	Taxas de transmissão (max.)
802.11	2,4	2 Mb/s
802.11a	5	54 Mb/s
802.11b	2,4	11 Mb/s
802.11g	2,4	54 Mb/s
802.11n	2,4	300 Mb/s

Fonte: Adaptado de https://pt.wikipedia.org/wiki/Wi-Fi

2.4.2. Segurança

Tendo em conta a envergadura atual e a utilização deste tipo de redes, surge a necessidade de proceder a métodos de controlo para proteger as redes e os seus utilizadores de acessos não autorizados. Os principais mecanismos são:

- **WEP (*Wired Equivalent Privacy*)**: Existe o padrão 802.11 original, consistindo num mecanismo de autenticação que, basicamente, funciona de forma fechada ou aberta através do uso de chaves. Ao ser definida uma chave, o dispositivo terá que fornecer a mesma. Esse sistema pode trabalhar com chaves de 64 *bits* e de 128 *bits* (pode encontrar-se também 256 *bits*), tendo assim, diferentes níveis de segurança, sendo a última a mais segura. Todavia, não se recomenda a utilização do WEP devido as suas potenciais falhas de segurança.

- **WPA:** Face ao problema de segurança no WEP, a *Wi-Fi Alliance* criou o formato *Wired Protected Access* (WPA), que é mais seguro que o WEP por se basear num protocolo chamado *Temporal Key Integrity Protocol* (TKIP) e que ficou conhecido como WEP2. Sendo assim, ao contrário do WEP, neste sistema a chave é trocada periodicamente, sendo a sequência definida na configuração da rede e por essa razão, recomenda-se a sua utilização.

- **WPA2 (AES):** O WPA2 é uma variação do WPA que se baseia no protocolo *Advanced Encryption Standard* (AES), sendo conhecida também como 802.11i, que é um conjunto de especificações de segurança. Esse mecanismo oferece um alto grau de segurança mas tem como deficiência a alta exigência de processamento.

2.5. *Web Service*

O *Web Service* foi criado com o intuito de interligar diferentes tipos de tecnologias, servindo como ferramenta de integração de sistemas, podendo utilizar diferentes linguagens que transformam a informação em XML.

Essencialmente, o *Web Service* faz com que os recursos da aplicação do *software* estejam disponíveis na rede de uma forma normalizada. Outras tecnologias fazem a mesmo coisa, como por exemplo, os navegadores da *Internet* que acedem às páginas *Web* disponíveis, baseadas por norma nas tecnologias da *Internet*, HTTP (*Hypertext Transfer Protocol*) e HTML (*HyperText Markup Language*). No entanto, estas tecnologias não são bem-sucedidas na comunicação e integração de aplicações. Existe uma grande motivação para o uso de tecnologia *Web Service,* pois possibilita que diferentes aplicações comuniquem entre si e utilizem recursos diferentes.

Muitas pessoas consideram que os *Web Services* corrigem um grande problema da informática: a falta de integração de sistemas. Os *Web Services* permitem que a integração de sistemas seja realizada de maneira compreensível, reutilizável e padronizada. É uma tentativa de organizar um cenário cercado por uma grande variedade de diferentes aplicativos, fornecedores e plataformas.

2.5.1. Vantagens e Desvantagens

Os *Web Services* são modelos que surgiram para o desenvolvimento de aplicações típicas de negócio eletrónico, envolvendo e suportando o estabelecimento da colaboração e negociação de forma aberta, distribuída e dinâmica entre distintos parceiros.
Os *Web Services* poderão representar um sucesso significativo devido ao enorme esforço, por parte da maioria dos parceiros industriais, na normalização das tecnologias envolvidas.
As tecnologias subjacentes aos *Web Services* (tais como HTTP, SOAP, WSDL, UDDI, XML) são abertas, amplamente divulgadas e consensuais. Por outro lado, existe potencial para haver uma real independência das linguagens de programação (Java, C++, VB, Delphi, C#), das arquiteturas de computadores e sistemas operativos, o que permite uma evolução mais suave e económica para este modelo computacional.
No entanto, existem críticas que demonstram medos ou falsas expectativas que os investimentos em *Web Services* podem suscitar. Uma dessas críticas diz respeito ao facto do SOAP (*Simple Object Access Protocol)* ser menos eficiente do que os sistemas de RPC (*Remote Procedure Call*) existentes. Por exemplo, as mensagens (com os respetivos envelopes e descrição de tipos) trocadas entre as partes são descritas em formato de texto/XML enquanto nos sistemas clássicos de RPC são trocadas em formato binário.
No entanto, esta desvantagem é compensada significativamente pela facilidade de interoperação entre os serviços, sem os problemas conhecidos de segurança/*firewalls*, e pela facilidade de se esconder os detalhes proprietários das infraestruturas de suporte.

2.5.2. Tecnologias Utilizadas

Para a representação e estruturação dos dados nas mensagens recebidas/enviadas é utilizado o XML (*eXtensible Markup Language*). As chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas no protocolo SOAP (baseado em XML). Os serviços (operações, mensagens, parâmetros, etc.) são descritos usando a linguagem WSDL (*Web Services Description Language*). O processo de publicação/pesquisa/descoberta de *Web Services* utiliza o protocolo UDDI (*Universal Description, Discovery and Integration*).

2.5.2.1. XML

Extensible Markup Language é a base em que os *Web Services* são construídos. O XML fornece a descrição, o armazenamento, o formato da transmissão para trocar os dados através dos *Web Services* e também para criar tecnologias *Web Services* para a troca dos dados.
A sintaxe de XML usada nas tecnologias dos *Web Services* específica, como os dados são representados genericamente, define como e com que qualidades de serviço os dados são transmitidos, pormenoriza como os serviços são publicados e descobertos. Os *Web Services* decodificam as várias partes de XML para interagir com as diversas aplicações.

2.5.2.2. SOAP

O SOAP (*Simple Object Access Protocol*) baseia-se numa invocação remota de um método e, para tal, necessita especificar o endereço do componente, o nome do método e os argumentos para esse método. Estes dados são formatados em XML, com determinadas regras e enviados normalmente por HTTP para esse componente. Não define ou impõe qualquer semântica, quer seja o modelo de programação, quer seja a semântica específica da implementação. Este especto é extremamente importante, pois permite que, quer o serviço quer o cliente que invoca o serviço, sejam aplicações desenvolvidas sobre diferentes linguagens de programação. Por esta razão, o SOAP tornou-se uma norma aceite para se utilizar com *Web Services*. Desta forma, pretende garantir-se a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização da linguagem XML e do mecanismo de transporte HTTP ou outro como, por exemplo, o SMTP (*Simple Mail Transfer Protocol*). O SOAP permite que os documentos XML de envio e de receção na *Web* suportem um protocolo comum de transferência de dados, para uma comunicação de rede eficaz, ou seja, o SOAP providencia o transporte de dados para os *Web Services*.
Em relação à *Web*, o SOAP é um protocolo de RPC que funciona sobre HTTP (ou SMTP, ou outro) de forma a ultrapassar as restrições de segurança/*firewalls* normalmente impostas aos sistemas clássicos de RPC (RMI, DCOM, CORBA/IIOP) suportando mensagens XML. Em vez de usar HTTP para pedir uma página HTML, visualizada num *browser*, o SOAP envia uma mensagem de XML através do pedido HTTP e recebe uma resposta, se existir, através do HTTP. Para assegurar corretamente a transmissão da mensagem de XML, o servidor de HTTP, tais como Apache ou IIS

(*Microsoft Internet Information Server*), recebe mensagens SOAP e deve validar e compreender o formato do documento XML definido na especificação SOAP v1.1.

2.5.2.3. WSDL

É a sigla de *Web Services Description Language*, padrão baseado em XML para descrever o serviço como no COM, que traz os métodos do *Web Service*. Funciona como uma espécie de "TypeLibrary" do *Web Service*, além de ser usado para a validação das chamadas dos métodos.
O WSDL é uma especificação desenvolvida pelo W3C que permite descrever os *Web Services* segundo um formato XML.
O WSDL é extensível para permitir a descrição dos serviços e suas mensagens, independentemente dos formatos de mensagem e dos protocolos de rede que sejam usados. No entanto, é comum usar-se o MIME (*Multipurpose Internet Mail Extensions*) e o HTTP://SOAP.
O WSDL descreve os serviços disponibilizados à rede, através de uma semântica XML, este providencia a documentação necessária para se chamar um sistema distribuído e o procedimento necessário para que esta comunicação se estabeleça. Enquanto o SOAP especifica a comunicação entre um cliente e um servidor, o WSDL descreve os serviços oferecidos.

2.5.2.4. UDDI

O UDDI (*Universal Description Discovery and Integration*) é uma iniciativa em desenvolvimento no âmbito do consórcio industrial UDDI, promovido originalmente pela IBM, Microsoft e Arriba, com objetivo de acelerar a interoperabilidade e utilização dos *Web Services*, através da proposta de um serviço de registro de nomes de organizações e de descrição do serviço.
Um registro UDDI contém três tipos de informação:

- Informação geral de cada organização, tais como o nome, morada, telefone e contactos;
- Informação de organizações e serviços por categorias de negócios;
- Informação técnica sobre os serviços providenciados pelas organizações.

No que concerne às principais funções UDDI providencia/permite designadamente:

- Publicação: permite que uma organização divulgue o(s) seu(s) serviço(s);
- Descoberta: permite que o cliente do serviço procure e encontre um determinado serviço;
- Ligação (*bind*): permite que o cliente do serviço possa estabelecer a ligação e interagir com o serviço.

2.5.2.5. WS-i

É o consórcio de diversas empresas como a IBM, a Oracle e a Microsoft que garante a integração entre os *Web Services* para que estes possam "conversar entre-si".

2.5.3. Segurança

A segurança dos *Web Services* é um dos pontos fracos desta tecnologia devido ao temor das empresas à exposição de dados, embora possam publicar serviços de forma simples e que são totalmente isolados da base de dados. O problema não é a falta de mecanismos de segurança mas sim a falta de consenso em relação ao mecanismo a ser adaptado pela tecnologia *Web Service*,
Sobre este aspeto levantam-se diversas questões, sobretudo:

- Na autenticidade (ter a certeza que uma transação do *Web Service* ocorreu entre o servidor e seu cliente;
- Na privacidade (todas as mensagens trocadas entre o servidor e o cliente não são intercetadas por uma entidade não autorizada);
- Na integridade (as mensagens enviadas tanto pelo servidor ao cliente, como o contrário, devem permanecer inalteradas).

De seguida são descritos os principais mecanismos de segurança utilizados.

2.5.3.1. SSL

O SSL (*Secure Sockets Layer*) quando aplicado a pequenos dispositivos oferece autenticação, integridade de dados e privacidade de serviços. Assim, tornou-se possível enviar informação

confidencial utilizando um mecanismo de segurança SSL sob HTTP, também conhecido como HTTPS. Este mecanismo protege informações confidenciais e é fácil de ser configurado. Tem como desvantagem ser mais lento do que as transações HTTP não cifradas, pelo que não é adequado para taxas de transferências de dados elevadas. Por ser um mecanismo de proteção no nível de transporte, apresenta restrições para ser aplicado em aplicações *Web Services*, pois o SSL não permite criptografia da informação, nem o uso de sessões seguras entre mais de duas partes, uma vez que o seu funcionamento se baseia numa arquitetura de transporte ponto-a-ponto.

2.5.3.2. XML *Signature*

A XML *Signature* [IETF e W3C 2000] é uma iniciativa do W3C para especificar uma sintaxe XML e regras de processamento para criação e representação de assinatura digital. As vantagens na utilização da XML *Signature*, ao contrário de outras normas de assinaturas digitais, estão baseadas na independência da linguagem de programação, fácil interpretação humana e independência do fabricante. Esta tecnologia também permite assinar digitalmente subconjuntos de um documento XML.

2.5.3.3. XML *Encryption*

A XML *Encryption* [IETF e W3C 2002] especifica um processo para cifra de dados e sua representação em formato XML. Os dados podem ser dados arbitrários (incluindo um documento XML), elementos XML ou conteúdos de elementos XML. Um documento XML que utiliza a XML *Encryption* pode ser visto por qualquer utilizador, mas apenas o proprietário da chave de descodificação conseguirá compreender o conteúdo codificado.

2.5.3.4. *WS-Security*

O *WS-Security* (*Web Services Security*) é uma iniciativa conjunta de empresas como Microsoft, IBM e Verisign destinada ao uso da *XML-Signature* e da *XML-Encryption* para fornecer segurança às mensagens SOAP. O *WS-Security* é um esforço destinado a fazer com que os *Web Services*

trabalhem melhor num ambiente global. O *WS-Security* também inclui alguns importantes componentes como encaminhamento, confiança e tratamento de transações.

2.5.3.5. SAML

O SAML (*Security Assertion Markup Language*) é um padrão emergente para a troca de informação sobre autenticação e autorização. O SAML soluciona um importante problema para as aplicações da próxima geração, que é a possibilidade de utilizadores transportarem seus direitos entre diferentes *Web Services*. Isto é importante para aplicações que tencionam integrar um número de *Web Services* para formar uma aplicação unificada.

Apesar da tecnologia *Web* ser tratada como uma tecnologia normalizada existem algumas incompatibilidades entre os WSDL´s disponibilizados entre os diferentes fornecedores. A exemplo da especificação, ao que se refere ao *binding*, podem ser implementados de diferentes maneiras, causando um conflito ao nível da interpretação a fazer. Alguns cumprem a especificação, relacionando e declarando todos os métodos e objetos complexos de forma explícita, enquanto outros fornecedores não o fazem desta forma, tornando-os assim, incompatíveis.

2.5.4. Evolução dos *Web Services*

A nível de novos modelos de negócio só o futuro dirá quem tem razão: se os céticos ou conservadores, se os que arriscam e concretizam a sua visão. Com o conceito dos *Web Services* talvez o mais importante nem seja a tecnologia em si, mas toda uma discussão à volta dos fatores económico-políticos que este paradigma poderá suscitar, bem como os modelos de negócio que poderão emergir.

Parece natural a emersão de novos portais, não para as pessoas consultarem e usarem, mas para as aplicações, i.e., para os serviços se registarem/publicarem de modo a tornarem-se conhecidos, descobertos e usados. Esses portais de serviços (tecnicamente consiste em serviços de registos UDDI e/ou ebXML) poderão ser definidos a nível global, regional, para domínios de negócio horizontais ou verticais.

No entanto e naturalmente, novos problemas e requisitos tecnológicos são colocados com o conceito dos *Web Services*. Desde logo, ao nível da modelação destes serviços e dos processos de

negócio em que aqueles participam. Aspetos como a composição de serviços, coordenação de fluxos de trabalho, identificação e privacidade, segurança, negociação, contratos e pagamentos, tratamento de exceções, categorização e taxonomias de serviços, entre outros etc., deverão ser adequadamente investigados e tratados para que este paradigma possa vir a apresentar um largo consenso e sucesso.

2.6. Bases de dados

As bases de dados são hoje um pilar importante no mundo informático pois sustentam informações muitas vezes de importância vital para o funcionamento de aplicações e serviços.

Desde os anos 60 que estas são alvo de investigação a nível académico e empresarial. Estas investigações focam-se em novas teorias e protótipos: têm-se destacado pesquisas em modelos, conceito de transações únicas e indivisíveis (atómicas), técnicas relacionadas com controlo de concorrências, linguagens e otimização de pesquisas (*querys*). Existem também diversos jornais académicos dedicados à investigação de bases de dados.

Um Sistema de Gestão de Base de Dados ou SGBD é um conjunto de programas que permitem armazenar, modificar e extrair informações de um banco de dados. Há muitos tipos diferentes de SGBD: desde pequenos sistemas que funcionam em computadores pessoais a sistemas enormes que estão associados a supercomputadores.

Um sistema de gestão de base de dados implica a criação e manutenção de bases de dados, elimina a necessidade de especificação de definição de dados, age como *interface* entre os programas de aplicação e os ficheiros de dados físicos e separa as visões lógica e de conceção dos dados.

Assim sendo, são basicamente três os componentes de um SGBD:

- Linguagem de definição de dados (especifica conteúdos, estrutura a base de dados e define os elementos de dados);
- Linguagem de manipulação de dados (para poder alterar os dados na base);
- Dicionário de dados (guarde definições de elementos de dados e respetivas características – descreve os dados, quem os acede, entre outros.

Um conceito muito importante em base de dados é o de transação. Uma transação é uma sequência de operações, num sistema de gestão de base de dados que, são tratadas como um bloco único e indivisível (atómico) durante uma recuperação de falhas e também fornecem isolamento entre acessos concorrentes na mesma massa de dados. A integridade de uma transação depende de 4 propriedades, conhecidas como ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

2.6.1. Atomicidade

Todas as ações que compõem a unidade de trabalho da transação devem ser concluídas com sucesso, para que seja efetivada. Se durante a transação qualquer ação que constitui a unidade de trabalho falhar, a transação inteira deve ser desfeita (*rollback*). Quando todas as ações são efetuadas com sucesso, a transação pode ser efetivada e mantida em banco (*commit*).

2.6.2. Consistência

Todas as regras e restrições definidas no banco de dados devem ser obedecidas. Relacionamentos por chaves estrangeiras, verificação de valores para campos restritos ou únicos devem ser obedecidos, para que uma transação possa ser completada com sucesso.

2.6.3. Isolamento

Cada transação funciona completamente à parte de outras estações. Todas as operações são parte de uma transação única. O princípio é que nenhuma outra transação, operando no mesmo sistema, possa interferir no funcionamento da transação corrente (é um mecanismo de controle). Outras transações não podem visualizar os resultados parciais das operações de uma transação em andamento (ainda em respeito à propriedade da atomicidade).

2.6.4. Durabilidade

Significa que os resultados de uma transação são permanentes e podem ser desfeitos somente por uma transação subsequente. Por exemplo, todos os dados e *status* relativos a uma transação devem ser armazenados num repositório permanente, não sendo passíveis de falha por um problema de hardware.

Na prática, alguns SGBDs relaxam na implementação destas propriedades procurando melhorar o desempenho. É preciso ter em conta que, pela sua importância, as bases de dados necessitam de sistemas de gestão para que os utilizadores possam interagir com as mesmas quer a nível estrutural, quer a nível de informação armazenada.

2.7. MySQL

O MySQL é um gestor de bases de dados utilizado em todo o mundo. Entre os seus utilizadores estão a NASA, o Banco Bradesco, a HP, a Sony, a Lufthansa, o U.S. Army, a Cisco Systems, a Google e o próprio Instituto Politécnico da Guarda

Sendo muito versátil, o mesmo dispõe de diversas interfaces gráficas e interfaces de linhas de comando. É suportado por inúmeros sistemas operativos, que lhe garantem portabilidade e possui também grande compatibilidade podendo associar-se a inúmeras linguagens de programação.

Apesar do seu excelente desempenho e estabilidade é simples de usar, não exige muitos recursos e é *software* livre, com licença GPL. Suporta também *triggers*, funções, procedimentos e cursores bem como diversos motores de armazenamento.

2.8. *Java Server Pages* (JSP)

As páginas JSP (*Java Server Pages*) são objetos Java para a criação de páginas HTML dinâmicas, com a lógica de poderem ser facilmente geridas por ferramentas para editar XML e HTML. O conceito das páginas surgiu tendo em conta a funcionalidade dos *servlets*, que eram a preferência dos programadores e a vocação dos *web-designers* da utilização de *Javascript* no lado do cliente.

A grande vantagem perante os *servlets* é a separação da parte da apresentação da parte do conteúdo, uma vez que qualquer alteração nos *servlets* implica que haja sempre uma recompilação.

A constituição das JSP baseia-se em instruções Java limitadas por marcas HTML, sendo depois identificado e interpretado cada vez que a página é lida.

A semelhança entre os *servlets* e as JSP deve-se ao facto de que a sua especificação é uma extensão da API (*Application Programming Interface*) dos *servlets*. Através do JSP *Engine*, as paginas JSP

são convertidas em *servlets* sempre que uma determinada página é utilizada pela primeira vez ou sofra alguma alteração.

JSP permite que o código Java e outras ações pré-definidas sejam intercalados com conteúdo de marcação *Web* estático, com a página resultante, sendo compilada e executada no servidor que irá entregar um documento. As páginas compiladas, assim como as bibliotecas Java dependentes, utilizam o *bytecode* Java em vez de um formato de *software* nativo. Como qualquer outro programa Java, elas devem ser executadas numa máquina virtual Java (JVM), que se integra com o sistema operativo do servidor para fornecer um ambiente de plataforma neutra.

3. Desenvolvimento

Este capítulo visa a documentação dos processos e metodologias que se adotaram durante o desenvolvimento do projeto em questão. Inicialmente, indicam-se os diferentes componentes utilizados e referenciam-se as ações realizadas com os mesmos, documentando-se a implementação das diferentes soluções.

Começou-se pela arquitetura inicial que prevê a existência de um módulo que se encontrará na bilheteira, um segundo módulo que se encontrará a nível do utilizador de nome módulo de estádio e um terceiro como módulo de comunicação.

Na Figura 09, é apresentado o diagrama de blocos idealizado para o sistema:

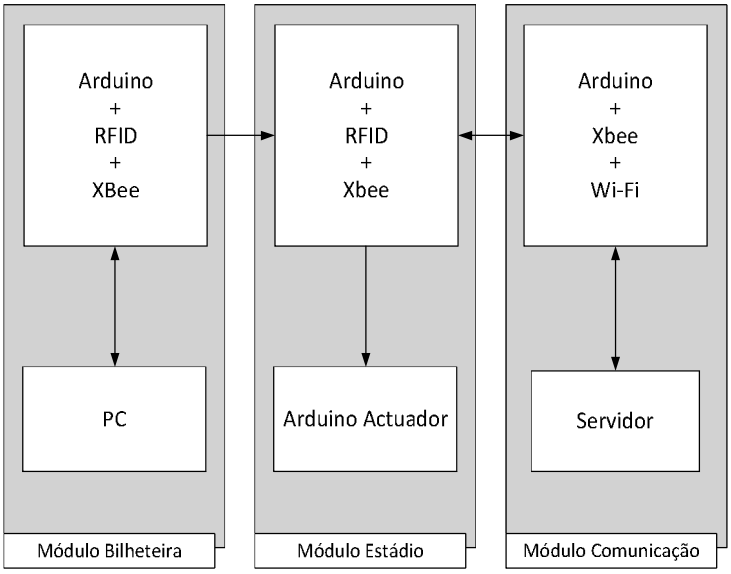


Figura 09 - Esquema de blocos representando a arquitetura do sistema MFootball.

Fonte: Elaboração própria

O esquema de blocos evidencia a interligação dos 3 módulos, designadamente:

- Módulo de Bilheteira: visa a possibilidade de interagir com o cartão RFID, de forma a proceder à venda de bilhetes. Esta interação baseia-se na leitura e na escrita de dados do cartão e encontrar-se-á nas bilheterias dos eventos, sendo facilmente acessível pelos utentes.
- Módulo de Estádio: terá a responsabilidade de ler os dados do cartão, comunicar com o módulo de comunicação e permitir a entrada ou não. Este situar-se-á junto das entradas dos

locais onde os eventos se irão realizar e estará sempre no raio de alcance do módulo de comunicação.

- Módulo de Comunicação: recebe a informação do módulo de estádio e está conectado à *Internet* para obtenção de resposta quanto ao pedido de entrada. Estará no raio de alcance do módulo de estádio e fora do alcance dos utentes.

Não contemplado, de forma explícita, no esquema anterior estão a página *Web*, o *Web Service* e o servidor de bases de dados.

A parte *Web* definida pela Figura 10 descreve a interação dos elementos que se encontram na rede, da seguinte forma:

- A página JSP será a plataforma *Web*;
- O *Web Service* irá interagir com a base de dados, sendo este o seu único canal para a manipulação de dados. A este, conectar-se-á o módulo de comunicação;
- A base de dados irá armazenar a informação pertinente ao funcionamento de sistema.

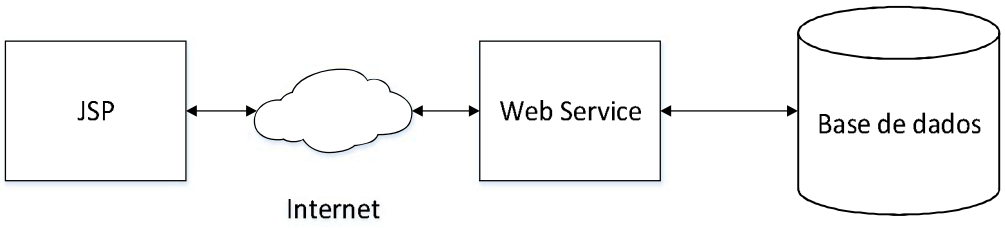


Figura 10 - Arquitetura *Web*.

Fonte: Elaboração própria

Existem hoje muitos projetos associados às tecnologias de comunicação sem fios, tendo a RFID potenciado a sua disseminação ao nível de objetos físicos, originando a chamada *Internet* das coisas. Dos projetos analisados, existem alguns cujas características são, de certo modo, semelhantes às que se procuram aplicar no presente projeto. Neste caso, destaca-se a solução da OTOT Electronics (Stadium Ticketing And Access Control, 2014), que disponibiliza também um sistema de controlo de acesso a eventos desportivos aplicada a estádios. Esta solução apresenta os seus componentes conectados através de TCP/IP LAN.

Outra área de grande aplicação da tecnologia RFID é no sector dos transportes e logística, onde tem tido um grande impacto ao nível da etiquetagem, identificação e localização dos produtos.

3.1. Componentes

Tendo a arquitetura básica planeada, decidiu passar-se à escolha dos componentes a utilizar.

3.1.1. Arduino

Como controlador, o Arduino mostra-se uma plataforma ideal para o desenvolvimento de diversos projetos, visando os mais diversos temas. Desta forma e como foi a plataforma introduzida na componente letiva do Mestrado de Computação Móvel, é a opção natural para integrar o projeto.

Atendendo às necessidades apresentadas, podemos referir que o módulo de bilheteira necessita de comunicar com o módulo RFID com o intuito de vender bilhetes e/ou registar utilizadores, o módulo de estádio necessita de comunicar através de um módulo XBee de forma a transmitir os dados vindo da leitura do cartão RFID, o módulo de comunicação irá interagir com um modulo XBee e com um módulo para comunicação por rede sem fios de forma a obter dados de um servidor. (CHEN, HUANG, M., & JAN, 2011)

Assim, optou-se pela utilização de um Arduino Mega 2560, uma vez que dispõe de um número de portas de série físicas suficientes para garantir as comunicações necessárias.

Como o módulo de bilheteira está conectado diretamente ao computador e utiliza uma única porta série para o módulo RFID, decidiu-se pela utilização de um Arduino UNO.

Havia a possibilidade da utilização de portas série via *software*, mas não foi adotada sendo o motivo apresentado posteriormente.

3.1.2. RFID

A nível de RFID, a escolha entre 13.56 MHz e os 125 kHz passou pelos seguintes fatores:

- Preços: os módulos RFID a 125 kHz são significativamente mais baratos que os módulos a 13.56 MHz.

- Distância: utilizando a velocidade 13.56 MHz, a interação com as etiquetas pode ser efetuada até um metro de distância, enquanto a 125 KHz a interação é efetuada a curta distância, cerca de 10 cm.

A distância de funcionamento, no presente caso, deverá ser curta e o preço é um fator também importante. Assim a escolha recai nas tecnologias a 125 KHz

No que toca à seleção das etiquetas, optou-se pela utilização de cartões de forma a facilitar a identificação do seu portador. Passou-se então à escolha dos cartões a utilizar, decidiu-se tendo por base duas alternativas apresentadas de seguida na Tabela 03:

Tabela 03 - Cartão EM4100.

Chip Emissor/Recetor:	Compatível EM4100
Frequência:	125 KHz
Tamanho de ID:	64 bit (Apenas Leitura)
Codificação:	Codificação Manchester
Material:	PVC
Temperatura:	-20°C / 50°C
Dimensões:	85.6 × 54 × 0.86 (mm)

Fonte: Adaptado de: http://www.apdanglia.org.uk/cardsandfobs.html

Tabela 04 - Cartão T5557

Chip Emissor/Recetor:	T5557
Frequência:	125 KHz
Tamanho de ID:	224 *bits* R/W (7 x 32 *bit*)
Codificação:	Manchester, FSK, PSK, Biphase, NRZ
Material:	ABS
Temperatura:	10°C - 50°C
Dimensões:	85.6 × 54 × 0.86 (mm)

Fonte: Adaptado de: http://www.apdanglia.org.uk/cardsandfobs.html

Passou-se, seguidamente, à escolha de componentes que permitissem a escrita de dados nas etiquetas. Assim sendo, cartões RFID T5557 (cujas características são apresentadas em cima na Tabela 04) foram a escolha efetuada uma vez que, estes são os únicos que preenchem esse requisito. Para interagir com estes cartões utilizou-se um módulo RFID 125 KHz.

3.1.3. XBee

No que toca aos módulos XBee, optou-se por comparar os módulos da Série 1 (S1) com os módulos da Série 2 (S2). Através da tabela seguinte (Tabela 05) pode efetuar-se uma avaliação das principais características de ambos. Esta resulta da análise da tabela (XBee S1 vs. XBee S2, 2014) onde se comparam as características do XBee S1 e do XBee S2.

Tabela 05 - Tabela comparativa de XBee Serie 1 e Serie 2

	XBee Series 1	**XBee Series 2**
Distância *Indoor*/Urbana	30 m	40 m
Distancia *Outdoor*/ Linha de vista	100 m	120 m
Potência de Saída	1 mW (0dbm)	2 mW (+3dbm)
RF Taxa de transmissão	250 Kbps	250 Kbps
Sensibilidade de Recetor	-92 dbm	-98 dbm
Alimentação elétrica	2.8 - 3.4 V	2.8 - 3.6 V
Consumo de corrente na transmissão	45 mA (@ 3.3 V)	40 mA (@ 3.3 V)
Consumo de corrente na Receção / Espera	50 mA (@ 3.3 V)	40 mA (@ 3.3 V)
Consumo em modo de espera (stand by)	10 uA	1 uA
Frequência	ISM 2.4 GHz	ISM 2.4 GHz

Fonte: Adaptado de: http://www.digi.com/support/kbase/kbaseresultdetl?id=2213

Comparados os elementos e uma vez que o custo de ambos é o mesmo, optou-se pela utilização do XBee S2 devido as distâncias de cobertura, quer internas quer externas, bem como à potência de saída.

3.1.4. Conectividade

No que toca a rede, o presente projeto divide-se em 3 partes.

3.1.4.1. Página *Web*

A página *Web* criada em JSP, visa a interação do utilizador, no que toca a dados pessoais, aquisição de *merchandising* e bilhetes. A escolha de JSP deve-se ao facto da possibilidade da inserção da página na plataforma da *Google Sites*.

3.1.4.2. Web Service

Pensado para servir como ponto integrador de todas as conexões *Web* a ser desenvolvido em C# devido à facilidade com que pode ser criado e a conhecimentos obtidos na licenciatura em Engenharia Informática e na parte letiva do mestrado de Computação Móvel.

3.1.4.3. Servidor de bases de dados

A base de dados, no presente projeto, serve de apoio através do qual é possível guardar a informação pertinente ao funcionamento da mesmo. O servidor de bases de dados será desenvolvido em MySQL. A sua simplicidade e a possibilidade da sua utilização gratuitamente, ditaram a sua escolha.

3.1.4.4. Conexão de módulo de comunicação

No módulo de comunicação, a conectividade foi pensada inicialmente para se efetuar via GPRS através de um modulo GSM/GPRS Sagem Hilo 9000. Em alternativa recorreu-se à rede Wi-Fi através de TCP/IP.

As redes Wi-Fi estão bem difundidas, estando disponíveis quer em estabelecimentos comerciais, serviços públicos até mesmo em transportes públicos.

Nas vantagens das redes Wi-Fi perante as redes GPRS enquadram-se a sua maior velocidade.

Como solução, pensou-se no RN171 da WiFly, por ser um módulo completo para comunicações TCP/IP e o facto de ser tão compacto e de ter um consumo baixo que o torna ideal para a

monitorização de sensores e para utilização em dispositivos alimentados por baterias. Este usa também o *shield* XBee para comunicar com o Arduino.

3.2. Funcionamento

O funcionamento do sistema pode ser explicado pela interação de 3 módulos diferentes que irão então permitir a entrada ou não do utilizador.

A ação será dividida por três unidades. A primeira unidade estará associada à bilheteira e será responsável pela escrita e leitura de cartões. A unidade seguinte, encontra-se a nível do utilizador, que irá ler os dados do cartão RFID e enviá-los a uma terceira unidade, que transmitirá essa informação, através de uma rede sem fios e obterá a resposta que permitirá a passagem do utilizador.

As ações destas unidades e a sua interligação podem ver-se no fluxograma apresentado na Figura 11.

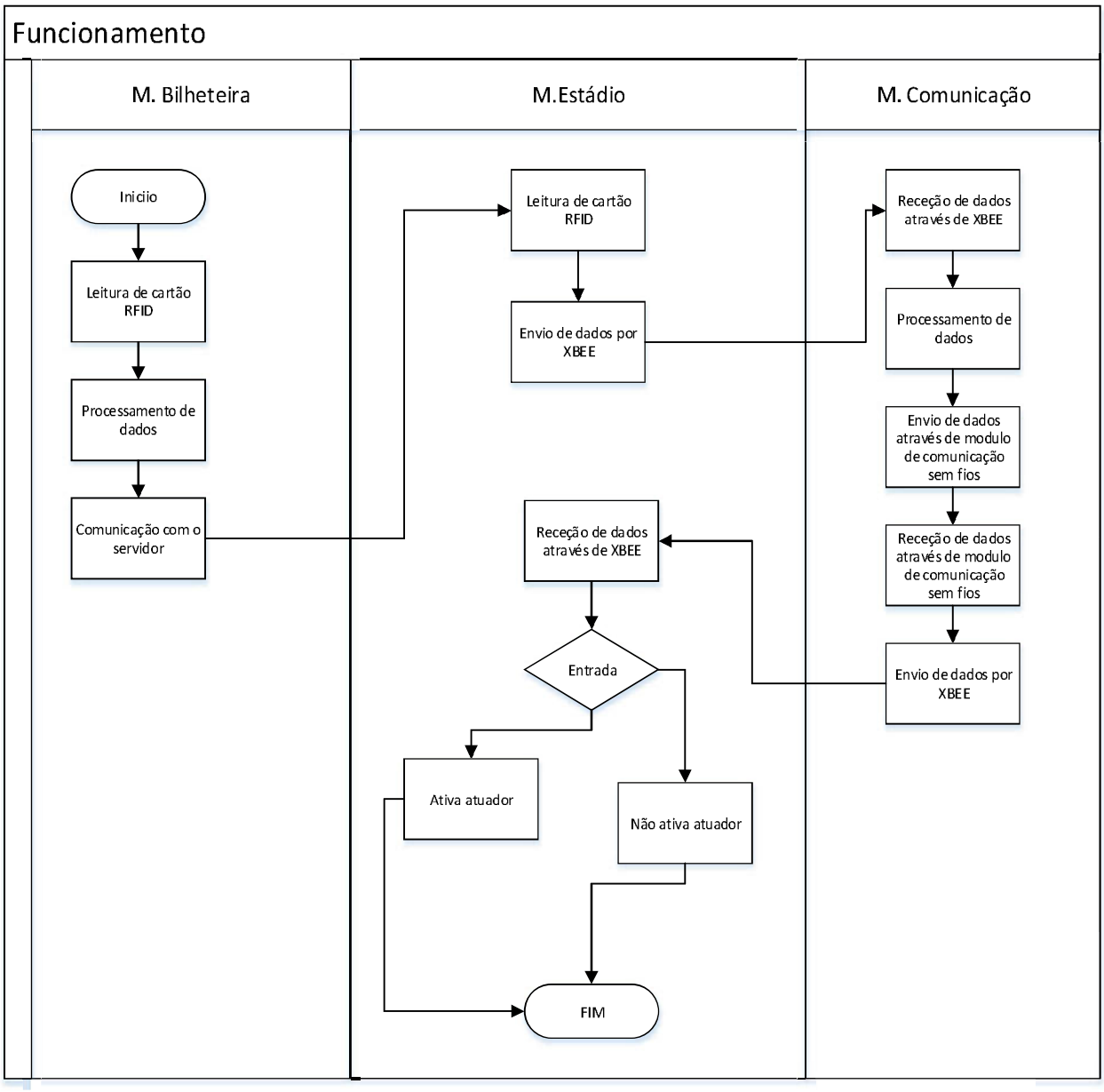


Figura 11 - Esquema básico de funcionamento.

Fonte: Elaboração própria

A comunicação entre o módulo de bilheteira e o servidor será responsável pela aquisição de dados do cartão RFID. Por sua vez, esses dados poderão transmitir-se através da conexão UART para o computador. Depois do tratamento de dados, a resposta pode ser enviada ou não, sendo que, se enviada será escrita no cartão.

O cartão ao passar pelo leitor do módulo do estádio, irá transferir informação através de ZigBee para o módulo de comunicação, o mesmo comunica com um servidor e a resposta é enviada de volta para o módulo de estádio que irá acionar um torniquete ou outro tipo de sistema de passagem.

3.3. Desenvolvimento

O desenvolvimento dos diferentes módulos foi realizado de uma forma faseada e sofrendo muitas alterações. Por isso, procurar-se-á dar a conhecer os processos, interpretações e as conclusões que foram encontradas.

Uma vez que o Arduino é peça central do desenvolvimento do projeto em questão foi então necessário ir à página *Web* e descarregar o ambiente de desenvolvimento Arduino (Download the Arduino Software, 2014) e instalar os controladores.

3.3.1. RFID

Em primeiro lugar, decidiu começar-se pelo estudo do funcionamento do módulo RFID. Desta forma, analisou-se a forma como a informação é armazenada nos cartões RFID e testar a leitura e a escrita dos dados no cartão.

Procedeu-se desta forma à acoplagem do módulo RFID a um Arduino, conforme representado na figura ou esquema seguinte (*Figura 12*).

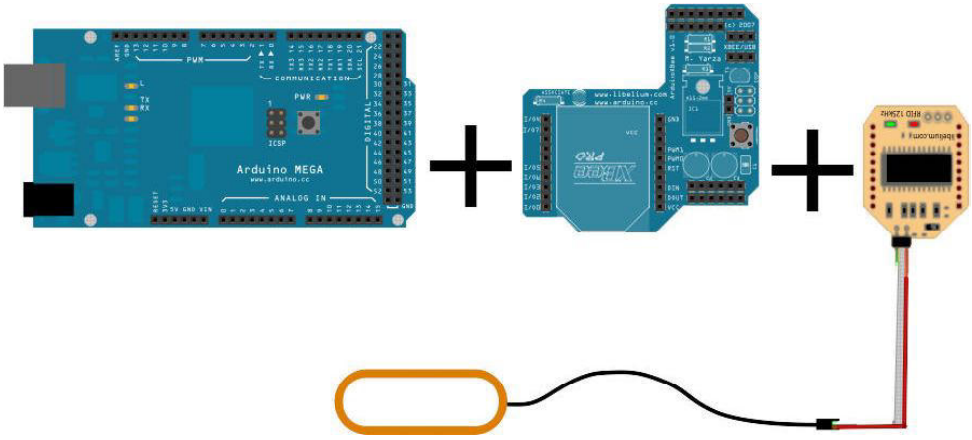


Figura 12 - Esquema de montagem de módulo de bilheteira.

Fonte: Elaboração própria

Cada cartão RFID dispõe de 8 blocos de 4 *bytes* (RFID 125kHz Module for Arduino Tutorial, 2014). Convém então saber, como é distribuída a informação no cartão, para podermos usufruir do cartão RFID (Tabela 06).

Tabela 06 - Disposição de dados no cartão T5557.

00	x	x	x	x	Aconselha-se que não seja programado podendo, no caso de o ser, deixar o cartão ilegível
01	52	58	8B	45	Pode ser utilizado para *Byte track*. Tipicamente detém informação previamente armazenada (52 58 8B 45)
02	[0-255]	[0-255]	[0-255]	[0-255]	Este campo pode ser escrito conforme a necessidade.
03	[0-255]	[0-255]	[0-255]	[0-255]	Este campo pode ser escrito conforme a necessidade.
04	[0-255]	[0-255]	[0-255]	[0-255]	Este campo pode ser escrito conforme a necessidade.
05	[0-255]	[0-255]	[0-255]	[0-255]	Este campo pode ser escrito conforme a necessidade.
06	[0-255]	[0-255]	[0-255]	[0-255]	Este campo pode ser escrito conforme a necessidade.
07	0	0	0	0	Este campo é normalmente utilizado para utilização de palavras-passe. A partir do momento que utilizadas as informações do cartão podem apenas ser acedidos através da utilização da palavra – passe de 4 *bytes* utilizada.

Fonte: Elaboração própria

Para teste usou-se o tutorial da *Cooking Hacks* (RFID 125kHz Module for Arduino Tutorial, 2014) que permite inicialmente a obtenção dos dados do cartão RFID e de seguida o tutorial que permite a escrita e leitura. Tendo esta informação, procedeu-se à utilização de algum código subjacente na leitura dos dados do cartão, começando pelas variáveis que irão armazenar os dados do cartão. São aqui exemplificadas as variáveis para o bloco de dados 2.

```
// Variáveis para leitura de dados de bloco 2

byte block2_byte1 = 0x00;
byte block2_byte2 = 0x00;
byte block2_byte3 = 0x00;
byte block2_byte4 = 0x00;

//-------------------------------------------------------
```

Como se pode ver, a informação dos blocos é composta por dados do tipo *byte* aqui representadas em hexadecimal.

Outra parte do código do tutorial que se mostrou útil foi a função *getTag*(), esta permite a leitura dos dados dos blocos e o seu armazenamento nas variáveis previamente definidas para o bloco 2:

```
void getTag(){

  val = Serial.read();    // we read ff

  if (val > 0){
     Serial.read();    // we read 01
     Serial.read();    // we read 0d
     Serial.read();    // we read 10

    .
    .
    .

     block2_byte1 = Serial.read();    // we read block 2 byte 1
     block2_byte2 = Serial.read();    // we read block 2 byte 2
     block2_byte3 = Serial.read();    // we read block 2 byte 3
     block2_byte4 = Serial.read();    // we read block 2 byte 4

    .
    .
    .

     Serial.read();    // we read checksum

      }
     }
```

Tendo isto, criou-se o pseudocódigo para a adaptação do código que será utilizado no programa:

```
Inicio getTag()

Variável val (byte) = Leitura de porta serie

Se val maior que 0

Leitura de porta serie
Leitura de porta serie
Leitura de porta serie

…
Variável block2_byte1 (byte) = Leitura de porta serie
Variável block2_byte2 (byte) = Leitura de porta serie
Variável block2_byte3 (byte) = Leitura de porta serie
Variável block2_byte4 (byte) = Leitura de porta serie
…
Leitura de porta serie

Fim de Se val maior que 0

Fim de getTag()
```

Neste seguimento, adaptou-se este código fonte para uma solução que servisse às necessidades do projeto.

Após isto, procedeu-se ao estudo referente ao módulo de bilheteira. Para este, decidiu-se que a interação direta do módulo RFID com o computador da bilheteira era a melhor solução. Para isso, o Arduino iria encontrar-se em modo *gateway* ou modo *reset* para que seja possível a leitura dos dados das portas série, diretamente pelo computador associado. A implementação pode ver-se na Figura 13.

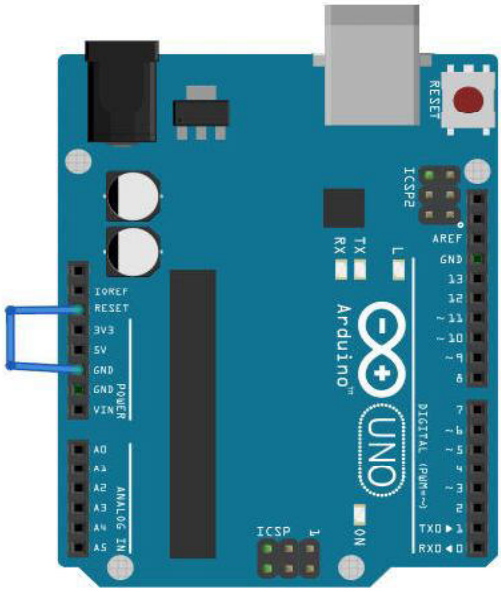


Figura 13 - Arduino em modo *reset* para comunicação com o computador.

Fonte: Elaboração própria

Este módulo irá depender de uma aplicação para proceder à interação com o cartão RFID, que será a responsável pela escrita e leitura de dados.

A aplicação deverá comunicar diretamente com as portas COM do computador onde se irá instalar. Esta desenvolveu-se em C# de forma a testar a possibilidade de obtenção de dados e da mesma se puderam tirar ilações, que permitem realizar a leitura e escrita de dados. A porta pode então definir-se:

```
SerialPort _porta = new SerialPort("COM10", 9600); //Definição de porta COM/velocidade
_porta.ReadTimeout = 10; // Timeout atribuído à porta
_porta.Open(); //Abrir porta
```

Desta forma, é possível obter dados na porta série, sendo todavia necessária a associação de um temporizador ou de um *thread* para a leitura de dados na porta série.

```
Thread _thread = new Thread((le_dados)); //Definição de Thread
```

Procedeu-se à leitura de dados que se consideraram pertinentes para um *array* sendo depois possível a manipulação dos dados

```
data[i] = _porta.ReadChar().ToString(); //leitura de dados na porta serie para uma posição de
array i
i++; //incrementação que leva ao avanço de posições no array.
```

Definiu-se então o pseudocódigo para perceber como é que o *thread* se pode associar à leitura dos dados:

```
Definir porta série
Definir thread(le_dados)

Inicio le_dados
Enquanto verdadeiro
Array de dados posição i = dados na porta
Incrementa i++
Fim de le_dados

Começa thread
```

Verificando-se a necessidade da escrita de dados no cartão, passou-se ao estudo de como o processo poderia ser feito. Para tal abre-se a porta e enviam-se os dados para a mesma.
Contudo, este envio deve ser pausado e ordenado. Mais uma vez, há a necessidade da utilização de um *thread* ou de temporizador.

```
porta.Write(datafield1, 0, 10); //Escrita de datafield1 na porta, com offset 0, número de intiros
10
Thread.Sleep(2000); //Pausa de 2 segundos
porta.Write(datafield2, 0, 10);
Thread.Sleep(2000);
porta.Write(datafield3, 0, 10);
Thread.Sleep(2000);
porta.Write(datafield4, 0, 10);
Thread.Sleep(2000);
porta.Write(datafield5, 0, 10);
Thread.Sleep(2000);
porta.Close(); // Fechar a porta após a escrita
```

Os *arrays datafield* funcionam como *buffers* de *bytes* que são enviados em intervalos de 2 segundos.

Tal como em procedimentos anteriores, definiu-se o pseudocódigo para adaptar o código que se irá aplicar:

```
Definir porta série
Definir thread(escrever_dados)

Inicio escrever _dados
Enquanto verdadeiro

Escrever na porta o buffer1
Esperar 2 segundos
Escrever na porta o buffer2
Esperar 2 segundos
Escrever na porta o buffer3
Esperar 2 segundos
Escrever na porta o buffer4
Esperar 2 segundos
Escrever na porta o buffer5
Esperar 2 segundos

Fechar porta serie

Fim de escrever _dados

Começa thread
```

Assim, culminou-se o estudo referente ao módulo RFID, uma vez que se pode ler e escrever no cartão se necessário.

3.3.2. XBee

Seguidamente procedeu-se ao estudo referente à comunicação via XBee. Os testes efetuados visaram encontrar a forma mais eficaz de transmissão de dados entre os módulos XBee. A documentação sobre os módulos XBee pode ser encontrada na página do fabricante (How To Use X-CTU, 2014), tal como o *software* que permite a sua configuração.

Para começar, conectaram-se duas placas Arduino a dois módulos XBee onde os Arduinos se encontram em *reset* (conforme a Figura 14).

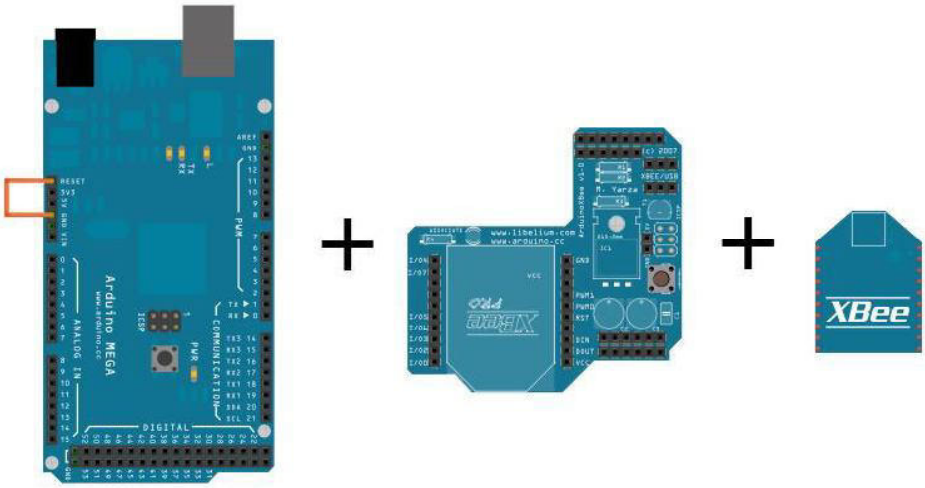


Figura 14 - Esquema de montagem para teste XBee.

Fonte: Elaboração própria

Procedeu-se à instalação do *software* X-CTU (XCTU Software, 2014), depois de descarregado do site da DIGI International. Ao abrir o X-CTU, tendo os XBee acoplados aos Arduino e devidamente conectados ao computador, deverão aparecer na primeira parte do *software* as portas COM ativas e associadas às placas Arduino. Ao escolher umas delas basta clicar em Test/Query como pode ver-se na Figura 15.

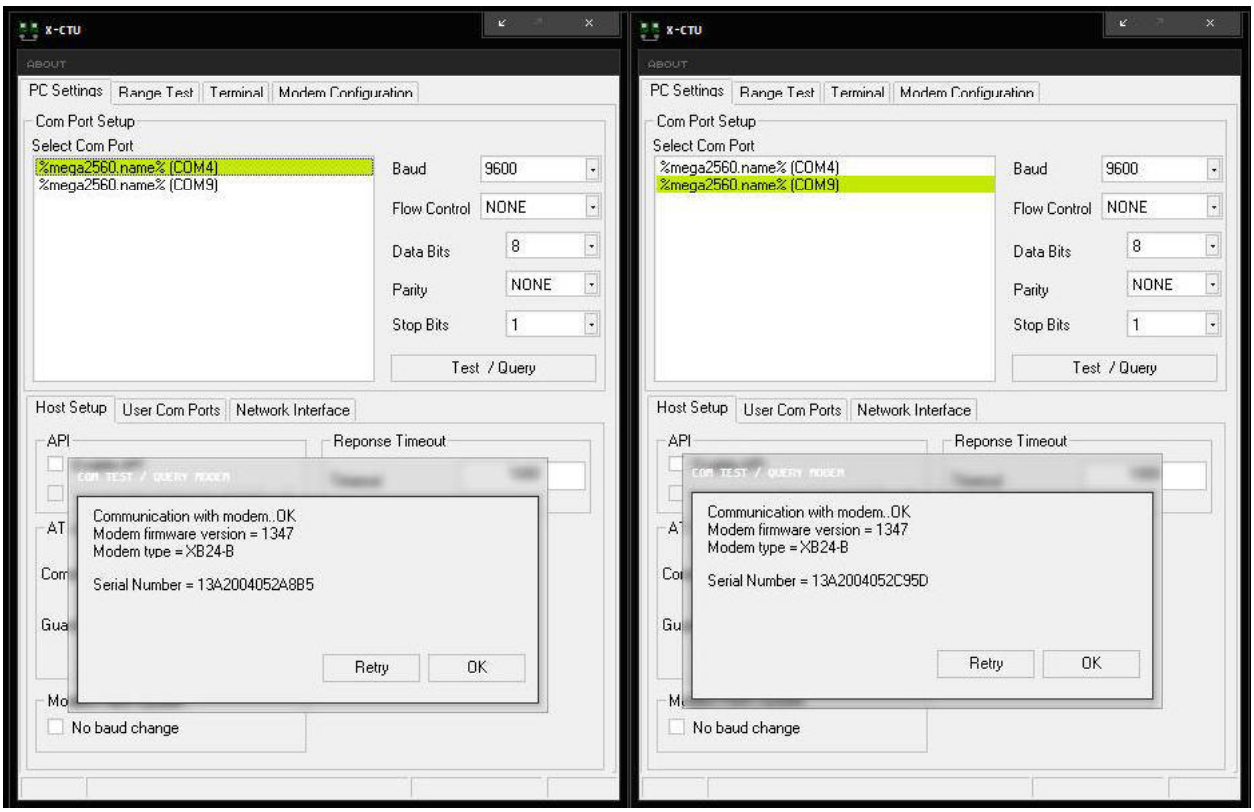


Figura 15 - Conexão a módulos XBee, com o *software* XCTU.

Fonte: Elaboração própria

Segue-se a resposta; se a conexão for bem-sucedida, deve surgir uma notificação de sucesso onde se pode ver a versão de *firmware*, o tipo de XBee (*Modem*) e número de série do mesmo. No caso da conexão não se efetuar é dada a notificação (Figura 16).

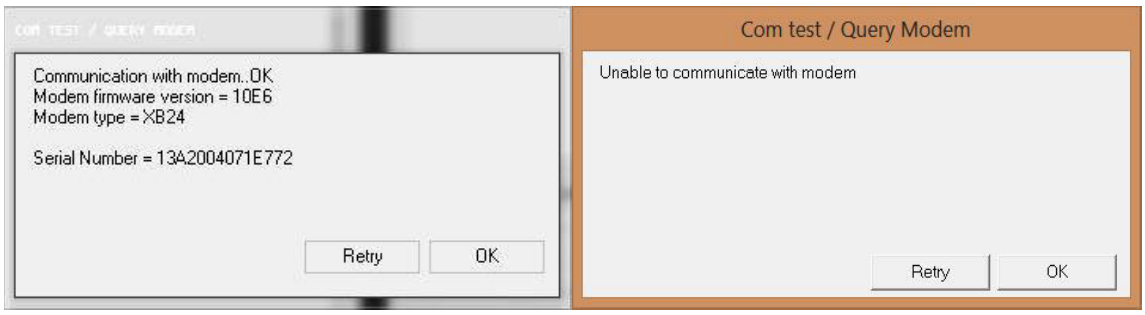


Figura 16 - Respostas possíveis à tentativa de acesso ao módulo XBee.

Fonte: Elaboração própria

No caso da conexão não se efetuar são também indicados possíveis passos para a pôr a funcionar, como demostrado na Figura 17.

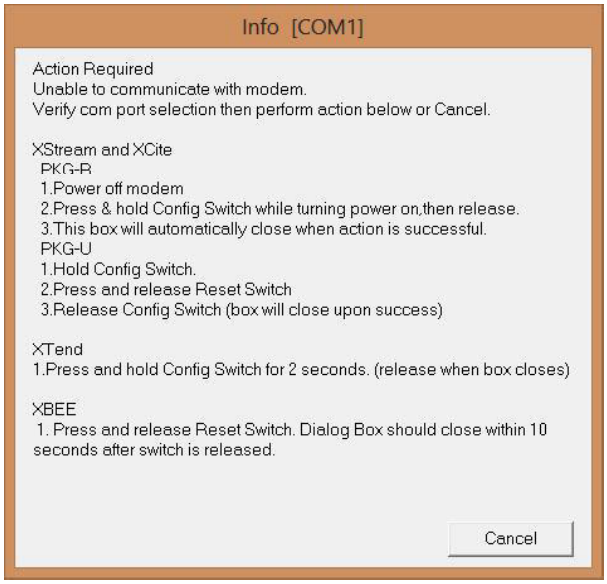


Figura 17 - Soluções para resolver o acesso da comunicação XBee.

Fonte: Elaboração própria

Contudo, muito frequentemente, a conexão pode resolver-se através do ajuste da velocidade (*baud rate*) e ativando ou não a opção de utilização de API.

Tendo sido a conexão bem-sucedida, ficam disponíveis opções para testes de distância (*Range Test*), utilização de consola de terminal (*Terminal*) e configuração (*Modem Configuration*).

Para proceder a um teste simples, podem abrir-se as duas janelas na lapela "Terminal" e escrever algo de forma a perceber se os módulos se encontram em modo *broadcast.* Este modo especifica que todas as mensagens enviadas são recebidas por todos os módulos ativos, sem a especificação de um destino. A transmissão ponto a ponto é denominada como *unicast*. Na Figura 18 pode ver-se o resultado de um dos primeiros testes.

Figura 18 - Teste de comunicação entre XBee's.

Fonte: Elaboração própria

Efetivamente, a mensagem escrita no terminal do XBEE1 (a azul no terminal COM4) foi recebida na totalidade pelo XBEE2 (a vermelho no terminal COM9) e vice-versa.
Contudo a operação em modo *broadcast* não é aconselhável, uma vez que não confere qualquer tipo de segurança. De uma forma sucinta, o modo *broadcast* define a transmissão de dados para todos os pontos disponíveis, sendo que qualquer um pode captar o seu conteúdo.
Os módulos XBee têm dois modos de funcionamento:

- Modo AT (*Transparent Mode*): neste modo o módulo XBee funciona como uma interface serie direta, ou seja, o que é recebido via UART através do pino D1 é enviado diretamente pelo pino D0. Este modo está ativo por defeito.
  Pode usar transmissão *broadcast* ou *unicast*.

- Modo API (*Application Programming Interface*): este modo é alternativo à utilização do modo transparente e atua até à extensão das capacidades de interação da rede, por parte do módulo (TITUS, 2012). Neste modo, a informação é encapsulada em pacotes que definem operações ou eventos do módulo.
  Os pacotes podem ser designadamente:
  - Transmissão: englobando pacotes de transmissão de dados (*RF Transmit Data Frame)* bem como pacotes de comandos (*Command Frame*), equivalentes a comandos AT;
  - Receção: engloba pacotes de transmissão de dados (*RF-received data frame*), resposta a comandos e de notificação de eventos (*reset, associate, disassociate, etc*).

  O modo API pode facilitar o funcionamento do XBee através de:
  - Transmissão de dados para destinos múltiplos sem a utilização de modo de comandos;
  - Receção de estados de sucesso ou falha de cada pacote;
  - Identificação de endereço da fonte de transmissão.

Tendo isto em conta decidiu-se explorar a utilização do modo API (Configuring XBees for API Mode, 2014) sendo para isso necessário proceder à configuração dos módulos.

A constituição dos pacotes do modo API (Tabela 07) segue regras específicas que, no caso de não serem rigorosamente cumpridas, não permitem a transmissão de dados.

Tabela 07 - Descrição de pacote API

Byte	Valor (hex)	Função	*Byte* para checksum
0	0x7E	Delimitador de pacote	
1	0x00	1º *byte* para tamanho	
2	0x13	2º *byte* para tamanho	
3	0x10	Identificador de API	01
4	0x00	Identificador de pacote	02
5	0x00	1º *byte* de endereço de 64 *bits* (1º byte endereço alto)	03
6	0x13	*byte* 6	04
7	0xA2	*byte* 7	05
8	0x00	*byte* 8 - Ultimo byte de endereço alto	06
9	partLA1	*byte* 9 - 1º byte de endereço baixo	07
10	partLA2	*byte* 10	08
11	partLA3	*byte* 11	09
12	partLA4	*byte* 12 - Ultimo byte endereço de 64 *bits* (ultimo *byte* de endereço baixo)	10
13	0xFF	*byte* 13 - 1º byte de endereço de 16 *bits*	11
14	0xFE	*byte* 14 - 2º byte de endereço de 16 *bits*	12
15	0x00	*byte* 15 - byte de aplicação opcional	13
16	0x00	*byte* 16 - byte de aplicação opcional	14
17	*checkbyte*	*byte* 17 primeiro de dados	15
18	*byte1*	*byte* 18 primeiro de dados	16
19	*byte2*	*byte* 19	17
20	*byte3*	*byte* 20	18
21	*byte4*	*byte* 21 ultimo de dados	19
22	*checksum*	*byte* 22	

Fonte: Elaboração própria

Existem também, duas somas de verificação, particularmente:

- A verificação de dados enviados (*checkbyte*):

  checkbyte= 0xFF-(*byte1*+*byte2*+*byte3*+*byte4*);

- A verificação de pacotes (*checksum*):

  checksum = 0xFF-(01+…+19)

Estas somas de verificação servem para garantir a integridade do pacote enviado, bem como os dados contidos no mesmo.
A diferenciação de cada módulo é feita através do endereço baixo, por exemplo (Tabela 08):

Tabela 08 - Endereços XBee.

Partes\módulos	XBEE1	XBEE2
partLA1	40	40
partLA2	52	52
partLA3	A8	C9
partLA4	B5	5D

Fonte: Elaboração própria

Seguidamente, passou-se à configuração dos módulos para proceder aos testes de comunicação. Para este processo, clica-se a lapela *Modem Configuration* seguida de *Modem XBee*; neste caso escolhe-se a opção XB24-B como tipo de *modem* a usar e, em *Function Set*, optou-se pela opção *ZNET 2.5 ROUTER/END DEVICE API,* para o modulo associado ao módulo de estádio, dando-lhe características de cliente e a opção *ZNET 2.5 COORDINATOR API* para o módulo de comunicação, para que este possa atuar como servidor. No campo *Destination* deve escrever-se o endereço do módulo de destino. Após esta opção configurou-se o campo API de 0 para 1 em ambos. Em *Modem Parameter and Firmware* clicou-se em *Write* para escrever as configurações nos módulos XBee (Figura 19).

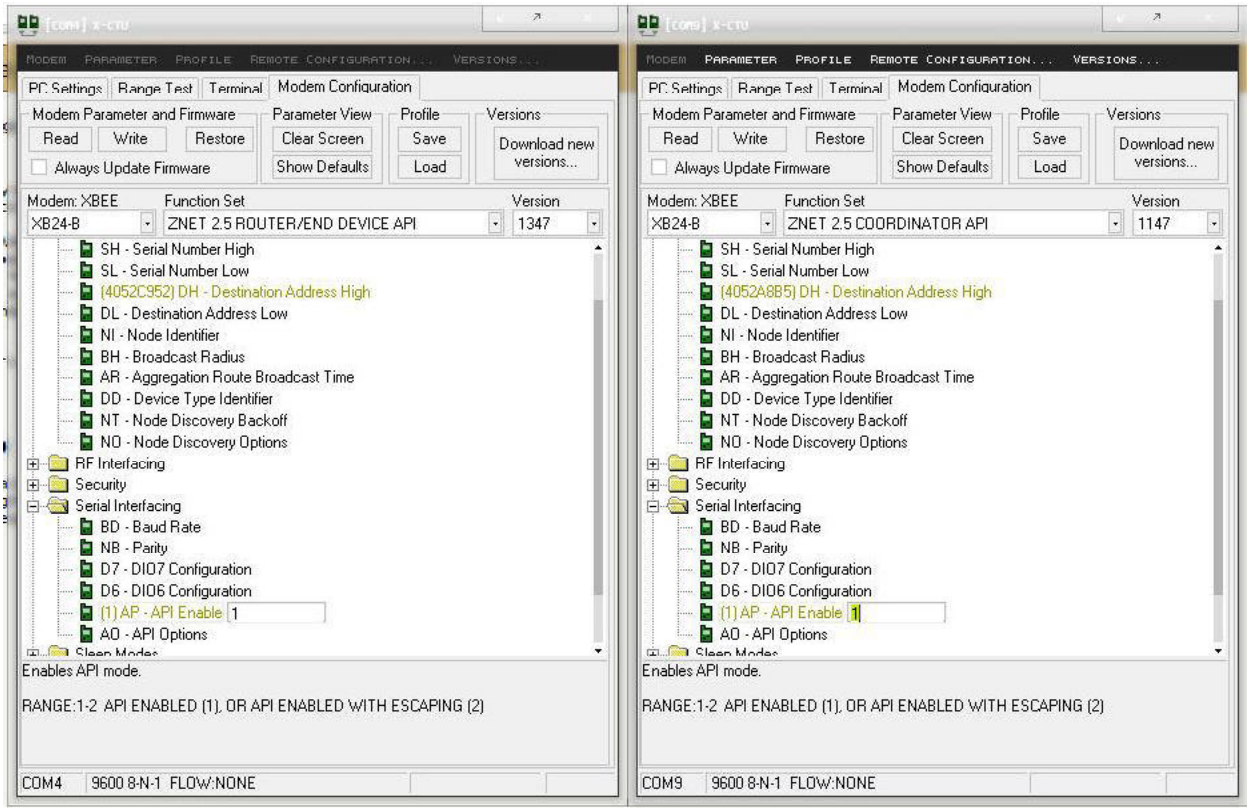


Figura 19 - Configurações de endereços.

Fonte: Elaboração própria

Após a configuração procedeu-se ao desenvolvimento de código Arduino, de forma a testar as comunicações.

Para o envio do pacote, os *bytes* devem ser escritos diretamente na porta onde o módulo XBEE1 está acoplado, neste caso, a porta serie 1.

Então para o módulo XBEE1 desenvolveu-se o pseudocódigo:

```
Iniciar configuração (void setup ())
Configurar porta serie 0 com o baud rate a 9600
Configurar porta serie 1 com o baud rate a 9600
 Definir variáveis de endereço
Fim de configuração

Iniciar ciclo (void loop())
Envio de Pacote para porta 1
Envio de Pacote para porta 0
Fim de ciclo
```

```
Função Verificação_dados
Soma Verificação = 0xFF – (dado1+dado2+dado3+dado4)
Fim de Verificação_dados

Aplicando este último ao código:

void setup(){
Serial.begin(9600);  //Comunicação PC
Serial1.begin(9600); //Comunicação XBee

_parte1=0x40;

 _parte2=0x52;

_parte3=0xC9;

_parte4=0x5D;

}

void loop(){
.
.
.
  Serial1.write(0x7E); // byte 0 - delimitador de pacote
  Serial.print(0x7E);
  Serial1.write(0x00); // byte 1 - 1º byte para tamanho
  Serial.print(0x00);
  Serial1.write(0x13); // byte 2 - 2º byte para tamanho
  Serial.print(0x13);
  Serial1.write(0x10);
  Serial.print(0x10); // byte 3 - Indentificador de API (01)
  be+= 0x10;
  Serial1.write(0x00); // byte 4 - Frame ID (02)
  Serial.print(0x00);
  be+= 0x00;
  Serial1.write(0x00); // byte 5 - 1º byte de endereço de 64 bits (1º byte endereço alto)(03)
  Serial.print(0x00);
  be+= 0x00;
  Serial1.write(0x13); // byte 6 (04)
  Serial.print(0x13);
```

```
  be+= 0x13;
  Serial1.write(0xA2); // byte 7(05)
  Serial.print(0xA2);
  be+= 0xA2;
  Serial1.write(0x00); // byte 8 - Ultimo byte de endereço alto(06)
  Serial.print(0x00);
  be+= 0x00;
  Serial1.write(0x40); // byte 9 - 1º byte de endereço baixo(07)
  Serial.print(0x40);
  be+= 0x40;
  Serial1.write(0x52); // byte 10(08)
  Serial.print(0x52);
  be+= 0x52;
  Serial1.write(0XC9); // byte 11(09)
  Serial.print(0XC9);
  be+=  0XC9;
  Serial1.write(0x5D); // byte 12 - Ultimo byte endereço de 64 bits (ultimo byte de endereço
baixo)(10)
  Serial.print(0x5D);
  be+= 0x5D;
  Serial1.write(0xFF); // byte 13 - 1º byte de endereço de 16 bit(11)
  Serial.print(0xFF);
  be+= 0xFF;
  Serial1.write(0xFE); // byte 14 - 2º byte de endereço de 16 bit(12)
  Serial.print(0xFE);
  be+= 0xFE;
  Serial1.write(0x00); // byte 15 -  byte de aplicação opcional(13)
  Serial.print(0x00);
  be+= 0x00;
  Serial1.write(0x00); // byte 16 -  byte de aplicação opcional(14)
  Serial.print(0x00);
  be+= 0x00;

  Serial1.write(_checkbyte); //byte 17 primeiro de dados. Sendo a soma de verificação de
dados(15)
  Serial.print(_checkbyte);
  be+= _checkbyte;

  Serial1.write(0x11); //byte 18 primeiro de dados(16)
  Serial.print(0x11);
  be+= 0x11;
```

```
  Serial1.write(0x98); // byte 19(17)
  Serial.print(0x98); // byte 19(17)
  be+= 0x98;

  Serial1.write(0x86); // byte 20(18)
  Serial.print(0x86);
  be+= 0x86;

  Serial1.write(0x38);   //byte 21 ultimo de dados(19)
  Serial.print(0x38);
  be+= 0x38;

  be=0xFF-be;
  Serial1.write(be); // byte 22
  Serial.print(be);
}


int doDataChecksum(int _byte1,int _byte2,int _byte3,int _byte4){

  int _check1= 0xFF-(_byte1+_byte2+_byte3+_byte4);

  return _check1;}
```

Para a receção de dados o seguinte pseudocódigo pode ser utilizado para a obtenção de dados no Arduino que está acoplado ao XBEE2 na porta serie 1.

```
Iniciar configuração (void setup ())
Configurar porta serie 0 com o baud rate a 9600
Configurar porta serie 1 com o baud rate a 9600
 Fim de configuração

Iniciar ciclo (void loop())
Se leitura da porta 1 for superior a 0
 Dados [i] = leitura da porta 1
Fim de leitura da porta 1
Escrita de Dados na porta 0
Fim de ciclo
```

Passando este conceito para código:

```
void setup(){
  Serial.begin(9600);  //Comunicação PC
  Serial1.begin(9600); //Comunicação XBee
}

void loop[]{

       _teste=Serial1.read();

       if (_teste>=0){

        if (i<21){ //A variável i varia até 20 porque o comprimento do pacote recebido pelo
       XBee é 20

          data1[i]=_teste;  //data1 recebe os dados do XBee
        if (i==21){
          // serve para ver a resposta recebida-----------------------------------
               Serial.println (data1[0]);
               Serial.println (data1[1]);
               Serial.println (data1[2]);
               Serial.println (data1[3]);
               Serial.println (data1[4]);
               Serial.println (data1[5]);
               Serial.println (data1[6]);
               Serial.println (data1[7]);
               Serial.println (data1[8]);
               Serial.println (data1[9]);
               Serial.println (data1[10]);
               Serial.println (data1[11]);
               Serial.println (data1[12]);
               Serial.println (data1[13]);
               Serial.println (data1[14]);
               Serial.println (data1[15]);
               Serial.println (data1[16]);
               Serial.println (data1[17]);
               Serial.println (data1[18]);
               Serial.println (data1[19]);
               Serial.println (data1[20]);
          //-------------------------------------------------------------------------
```

```
    }
  }
}
```

De forma a testar o código, decidiu-se que a utilização do *software TeraTerm* (Download Tera Term 4.85, 2014) seria apropriada. Após o *download* e a instalação *software* procedeu-se à conexão das portas COM. Como resultado:

Tabela 09 - Descrição de dados de testes API.

Enviado decimal	126 0 19 16 0 0 19 16 206 48 220 193 255 254 0 0 22 11 98 86 38 134
Enviado hexadecimal	0x7E 0x00 0x13 0x10 0x00 0x00 0x13 0xA2 0x00 0x40 0x52 0XC9 0x5D 0xFF 0xFE 0x00 0x16 0xFE 0x11 0x98 0x86 0x38 0x86
Recebido decimal	126 0 17 144 0 19 162 0 64 82 168 181 42 237 1 22 11 98 86 38 180
Recebido hexadecimal	0x7E 0x00 0x11 0x90 0x13 0xA2 0x00 0x40 0x52 0xA8 0xB5 0x2A 0xED 0x01 0x16 0x11 0x98 0x86 0x38 0xB4

Fonte: Elaboração própria

Como se pode verificar através da Tabela 09, o pacote de envio é maior que o pacote de receção. Contudo os dados recebidos pelo XBEE2 são os mesmos que enviados pelos XBEE1. Determinou-se assim, que a comunicação foi bem-sucedida, pelo que se obtém assim o código que permite o envio e receção de dados.

3.3.3. GPRS

No que toca à comunicação GPRS procedeu-se a testes de conectividade utilizando o módulo Hilo Sagem (Sim900) da Libellium. Este módulo permite comunicações GSM/GPRS com suporte total para *quadband* nas frequências 850/900/1800/1900 MHz.

Iniciou-se então a montagem dos componentes que permitam obter dados e proceder à adaptação deste módulo ao protótipo que se pretende. Assim sendo, seria necessária a verificação dos resultados recebidos pelo módulo. Partindo desta premissa, acoplou-se o módulo GPRS a um Arduino e a este, no pino TX da porta série, ligou-se um outro Arduino em modo *reset*, ligado através do pino RX, como pode ver-se na Figura 20.

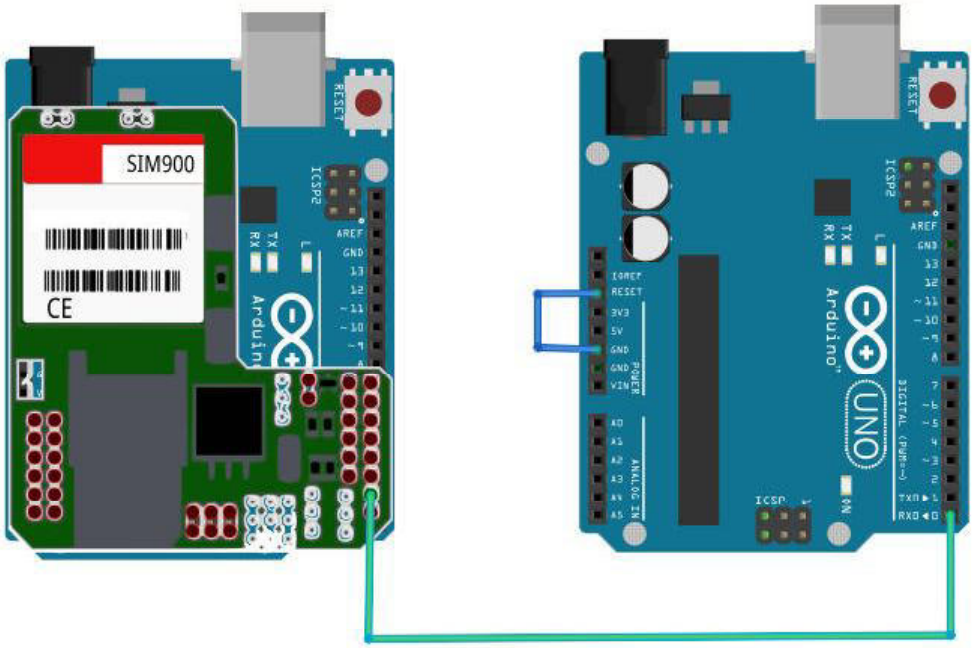


Figura 20 - Montagem de GPRS para testes.

. Fonte: Elaboração própria

É necessário então preparar módulo Hilo para proceder à comunicação, atendendo às operações (Figura 21):

1. Acoplagem de coluna (*speaker*);
2. Configuração para utilização de energia externa. (Vin pin);
3. Configuração da porta série para utilização através do Arduino.

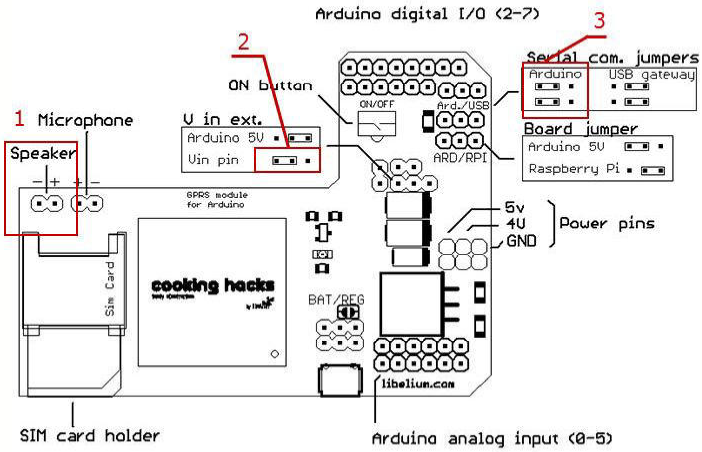


Figura 21 - Configurações de módulo GPRS.

Fonte: http://www.cooking-hacks.com/documentation/tutorials/arduino-gprs-gsm-quadband-sim900

Inicialmente, decidiu optar-se pela utilização de um tutorial existente na página da *cooking-hacks* dedicada ao *modem* GPRS em questão.

Efetuou-se um teste de conexão, tendo como base uma pesquisa na página da *Google*.

3.3.4. Comunicação TCP/IP

No que diz respeito à parte da rede TCP/IP, o desenvolvimento foi efetuado utilizando o módulo WiFly RN-171 da *Roving Networks*. Este módulo foi criado especialmente para aqueles que visam a migração da arquitetura 802.15.4 (ZigBee) para um padrão baseado em TCP/IP, sem a necessidade de redesenhar as soluções já existentes. Desta forma, este módulo incorpora a conexão rádio 802.11, utilizando a arquitetura planeada para a utilização de 802.15.4.

Tendo como base o tutorial da *Cooking Hacks* e no manual de utilizador fornecido pela Wifly, decidiu-se proceder a testes de forma a conectar o módulo RN171 ao *Web Service* associado ao projeto. Para tal, acoplou-se o módulo a um Arduino em modo *reset* de forma a poder-se testar a interação destes com os comandos especificados pela documentação estudada. Pode ver-se o esquema de montagem na Figura 22.

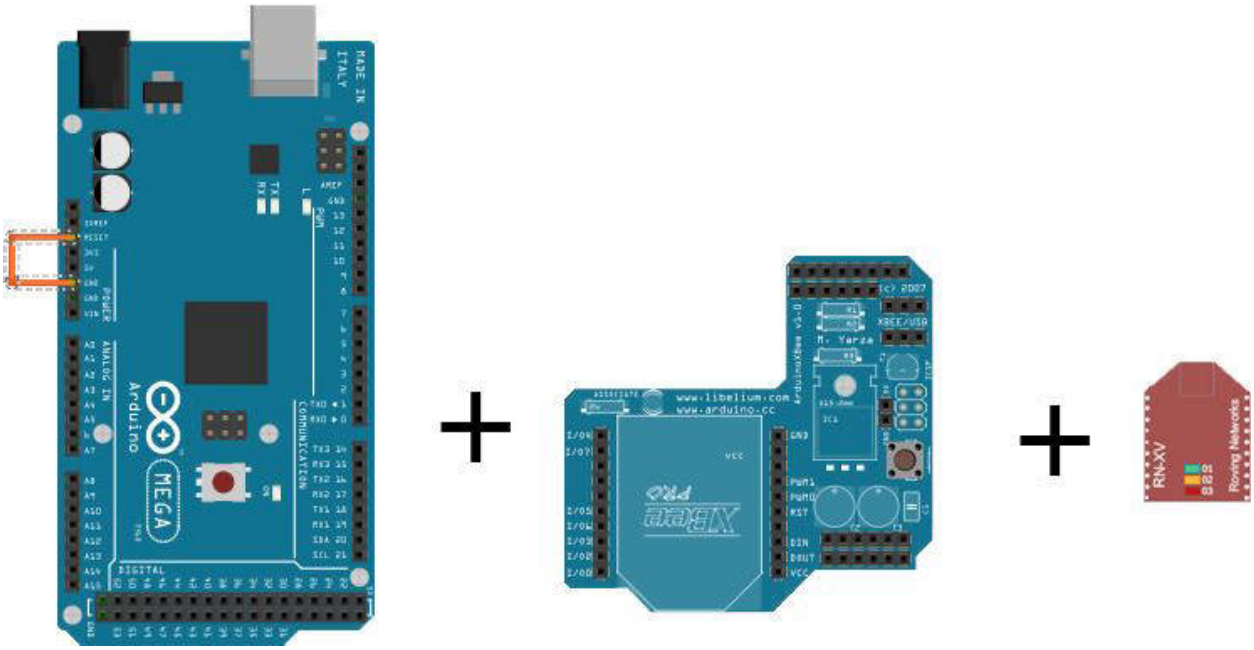


Figura 22 - Esquema de testes para o módulo RN171.

Fonte: Elaboração própria

De seguida, passou-se ao *TeraTerm*, onde para proceder ao modo de configuração se deve inserir o comando $$$ e, se formos bem-sucedidos, obter-se-á como resposta "CMD"
Depois disto, inseriram-se os comandos para configuração:

```
set wlan ssid <rede_a_conectar>      //definição de rede a conectar
set wlan pass <palavra_passe>//definição de palavra-passe
set ip proto 18                      //ativar cliente html
set ip host xxx.xxx.xxx.xxx  //definição de IP a conectar
set ip remote 80                     //definição de porta a ser utilizada
set com remote GET$/service1.asmx/testArduino?idteste=6487 //definição de método de Web Service
a aceder
set option format 1          //envio de dados HTML automaticamente
save
reboot
```

Como resultado, no *TeraTerm*, pode ver-se:

```
set wlan ssid <rede_a_conectar>
AOK
<2.30>
```

```
<2.30> set wlan pass <palavra_passe>
AOK
<2.30> set ip proto 18
AOK
<2.30>
<2.30> set ip host xxx.xxx.xxx.xxx
AOK
<2.30> set ip remote 80
AOK
<2.30> set com remote GET$/service1.asmx/testArduino?idteste=6487
AOK
<2.30> save
Storing in config
<2.30> reboot
*Reboot*WiFly Ver 2.30, 10-26-2011 on RN-171
MAC Addr=00:06:66:71:cf:83
Auto-Assoc 2do_drto chan=6 mode=WPA1 SCAN OK
Joining <rede_a_conectar> now..
*READY*
Associated!
DHCP: Start
DHCP in 27ms, lease=86400s
IF=UP
DHCP=ON
IP=192.168.1.137:2000
NM=255.255.255.0
GW=192.168.1.1
Listen on 2000
```

Após este resultado verificou-se que a configuração foi bem efetuada, ou seja, que a cada introdução de comandos se pode ver a resposta AOK, o que indica que o comando foi aceite.
Depois do reinício do módulo, pode ver-se na resposta que o módulo se associou à rede e ao mesmo foi-lhe atribuída uma configuração de rede e fica em modo de espera.
Inseriu-se então o comando $$$, de forma a poder dar-se mais indicações ao módulo. De seguida introduziu-se o comando *open* para aceder à página.

```
CMD
open
<2.30> *OPEN*HTTP/1.1 200 OK
Cache-Control: private, max-age=0
```

```
Content-Type: text/xml; charset=utf-8
Server: Microsoft-IIS/7.5
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Wed, 20 Mar 2013 23:53:37 GMT
Connection: close
Content-Length: 87

<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://merenthir.no-ip.org">0</int>*CLOS*
```

Como resposta obteve-se `*OPEN*HTTP/1.1 200 OK,` o que indica que a conexão foi efetuada com sucesso e como resultado obtiveram-se duas linhas XML. Esta era a resposta que se pretendia. Contudo, pode dar-se o caso em que a conexão não seja bem-sucedida e obter-se um resultado diferente, o que pode evidenciar erros de conexão, como se demonstra em seguida.

```
open
<2.30> *OPEN*HTTP/1.1 400 Bad Request
Content-Type: text/html; charset=us-ascii
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 21Connection: close
Content-Length: 311

<!DOCTYPE HTML PUBLIC "N""http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>Ba      HTTP-EQUIV="Content-Type"      Content="text/html;      charset=us-
ascii">equest</h2>
<hr><p>HTTP Error 400. The request is badly formed.<S*CMD
open
<2.30> *OPEN*HTTP/1.1 400 Bad Request
Content-Type: text/html; charset=us-ascii
Server: Microsoft-HTTPAPI/2.0
Date: ThuMT
Connection: close
Content-Length: 311

<!DOCTYPE HTML PUBL1//EN""http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLMETA     HTTP-EQUIV="Content-Type"     Content="text/html;     charset=us-ascad
Request</h2>
<hr><p>HTTP Error 400. The request is badly form
                                                *CLOS*CMD
```

Esta resposta pode ser obtida se, por exemplo, se fizerem pedidos de abertura (*open*) e estes forem efetuados num curto espaço de tempo. Neste caso, o pedido foi efetuado cerca de um segundo após a primeira conexão, que foi bem-sucedida. Assim, pode concluir-se que o tempo de pedidos de ligação será preponderante.

De forma a testar mais um pouco a conexão WI-FI, decidiu-se a sua introdução em código para implementação em Arduino. Para isso, há a necessidade associar o módulo à porta série 3 do Arduino Mega (Figura 23).

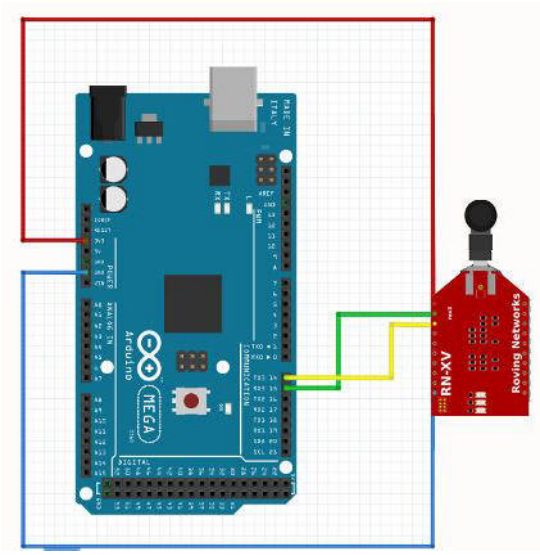


Figura 23 - Esquema de montagem de teste de conectividade TCP/IP.

Fonte: Elaboração própria

Pode verificar-se que o módulo WI-FI é ligado diretamente à porta série 3 para que esta seja utilizada para comunicação com o módulo e a porta série 0 poder servir para testar as mensagens que nos possam dar conta do que se passa, quando a informação é processada. Assim sendo a utilização dos *shields* é desnecessária. Procedeu-se à produção de código para Arduino para testar a conectividade do dispositivo:

```
Serial3.print("set wlan ssid <rede_a_aceder>\r");
check();
Serial3.print("set wlan phrase <palavra_passe>");
check();
Serial3.print("set ip proto 18\r");
check();
```

```
Serial3.print("set ip host XXX.XXX.XXX\r");
check();
Serial3.print("set ip remote 80\r");
check();
Serial3.print("set com remote GET$/service1.asmx/getEntrance2?idutilizador="+idsocio+"\r");
check();
Serial3.print("save\r");
check();
Serial3.print("reboot\r");
check();
Serial3.print("$$$");
check();
Serial3.print("open\r");
check();
```

Adaptaram-se as linhas de comando de comunicação ao módulo a ser estudado.

A ter em atenção está o facto de que as linhas de comando terem sido inseridas no inicio da função *setup*(). Contudo, os resultados não foram satisfatórios, sendo muitas vezes impossível a comunicação. Após isto, testou-se a inserção das mesmas na função *loop*() e a conexões testadas foram então bem sucedidas. Uma solução passa pela prévia configuração do módulo, sendo apenas necessária a utilização do pedido de ligação. A função *check*() é um adaptação da função com o mesmo nome vista em diversos tutoriais das *Cooking Hacks*.

```
void check()
{
       cont=0;
       delay(500);//original 1000
        // Receive the answer from the WIFI module.
        while (Serial3.available()>0)
       {
       recv[cont]=Serial3.read();
       Serial.print(recv[cont]);
       delay(100);
       cont++;
        }
  recv[cont]='\0';
}
```

Através da conexão via *TeraTerm*, observou-se de uma forma genérica que as respostas são recebidas após o envio das linhas de comando. Porém, existem falhas na escrita da resposta. Este fator pode dever-se a processamento do Arduino, associado também aos *delays,* vistos na função *check*() que foram reduzidos, de forma a obter o resultado desejado. Contudo, mesmo obtendo resultados de ligação positiva, o efeito obtido acabava por não mostrar as linhas .xml que se pretendiam utilizar como se pode verificar em baixo.

```
<2.30> *OPEN*HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Typ
```

Demorou algum tempo para se encontrar uma solução para este problema. De facto, os dados recebidos não eram mostrados na totalidade. Contudo, não implica necessariamente que não sejam recebidos pelo Arduino. Ou seja, o facto de as linhas .xml não serem mostradas pela porta série 0 não quer dizer que não sejam recebidas na porta série 3. Tendo isto em mente decidiu-se criar um teste para verificar a premissa acima mencionada. Para tal, criou-se um método no *Web Service*, cujo resultado recebido, pudesse ser processado.
O resultado recebido passou a ser algo como:

```
<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://merenthir.no-ip.org">#*#$0</int>*CLOS*
```

Utilizando o conjunto de caracteres especiais "#*#$" pôde criar-se um código que permitisse a obtenção do valor 0, que depois deveria mudar conforme necessário. Ou seja, o conjunto de caracteres resultante da comunicação seria percorrido até encontrar o símbolo cardinal, se isso acontecesse seria alterado o valor de uma variável de controlo que estaria designada anteriormente, por exemplo esse valor passaria de 0 para 1. Se após isto se encontrasse um asterisco e o valor da variável de controlo fosse 1, esse mesmo valor passaria a 2. Assim sendo, o segundo cardinal alteraria esse valor para 3 e o cifrão alterá-lo-ia para 4. Só após este último carater e que se encontraria a resposta do servidor que se pretende. Assim sendo, preparou-se um *sketch* onde as linhas XML seriam inseridas num *array* e através de uma máquina de estados se percorreriam os caracteres até ser encontrada a sequência pretendida.

O conceito pode ver-se no seguinte fluxograma (Figura 24):

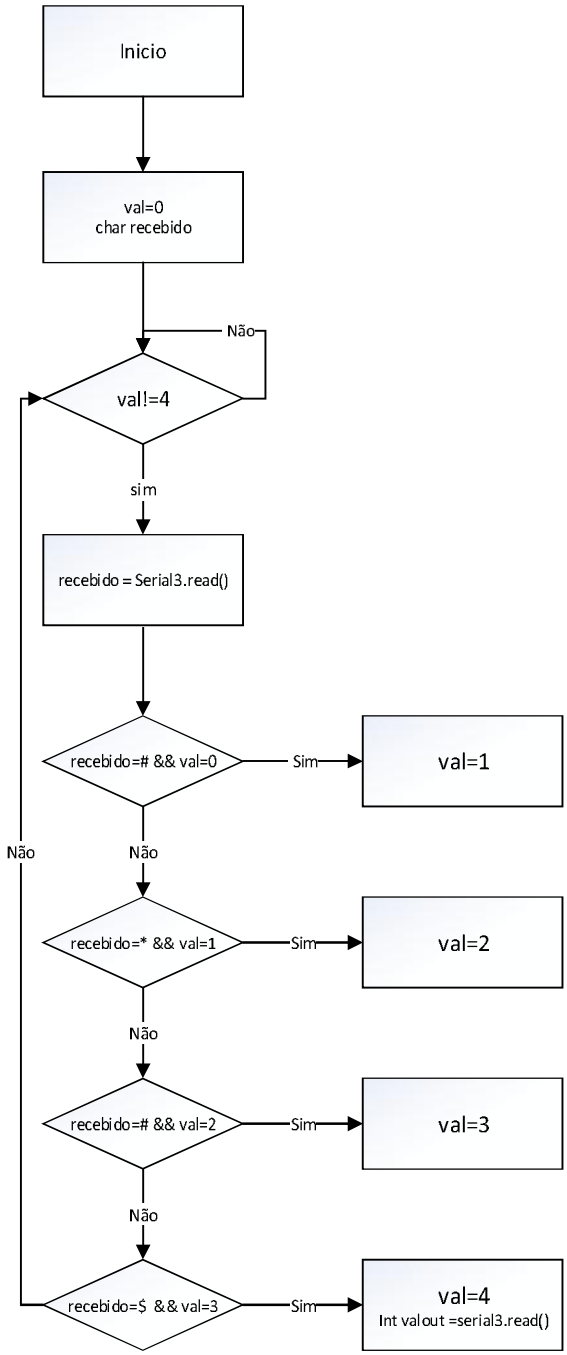


Figura 24 - Fluxograma que explica o conceito de obtenção de valor de linha XML.

Fonte: Elaboração própria

O pseudocódigo associado é o seguinte:

```
array resultado = linhas .xml
```

```
val = 0
enquanto  val diferente de 4  //variavel de controlo val

       recebido = leitura de porta 3
       se recebido=’#’ e val = 0 //captação de byte de controlo # associado à variável de
controlo
              imprimir “encontrou o 1º #”
              val=1
       fim de se recebido=’#’ e val = 0

       se recebido=’*’ e val = 1
              imprimir “encontrou o *”
              val=2
       fim de se recebido=’*’ e val = 1

       se recebido=’#’ e val = 2
              imprimir “encontrou o 2º #”
              val=3
       fim de se recebido=’#’ e val = 2

       se recebido=’$’ e val = 4
              imprimir “encontrou o *”
              val=2
              int valpretendido = leitura da porta 3
       fim de se recebido=’$’ e val = 4

fim de enquanto  val diferente de 4
```

Apresenta-se de seguida o código resultante:

```
while (val!=4)   {
    cont=0;
    recv[cont]=Serial3.read();
    if ( recv[cont]=='#' && val==0){
      Serial.println ("encontrou o 1 #");
      val=1;
      Serial.print ("val=1;");
    }
    else if ( recv[cont]=='*' && val ==1){
      Serial.println ("encontrou o *");
      val=2;
      Serial.println ("val=2;");
```

```
      //delay(100);
    }
    else if ( recv[cont]=='#' && val ==2){
      Serial.println ("encontrou o 2 #");
      val=3;
      Serial.println ("val=3;");
    }
    else if ( recv[cont]=='$' && val ==3){
      Serial.println ("encontrou o $");
      val=4;
      Serial.println ("val=4;"+cont);
      Serial.println (recv[cont]);
      inout1=Serial3.read();
      Serial.print ("valor requerido1: ");
      Serial.println (inout1);
      sendvalue(inout1);
    }
```

Pudemos então obter os dados do *Web Service*, de forma a serem aplicados convenientemente no projeto.

No que toca à utilização de LED*s* e do *buzzer,* que servem como sinalizadores, decidiu fazer-se um pequeno teste para verificar a sua funcionalidade. Existe já um exemplo no IDE Arduino que serviu de base, o *Blink_Test*, neste também se testou o botão que servirá de atuador.

```
int redled = 12; //led vermelho ligado à porta digital 12
int buzzer= 8;   //buzzer ligado à porta digital 8
int inPin = 7;   //   definição de porta como entrada
int val = 0;

void setup() {
  val = digitalRead(inPin); //Valor obtido do botão
  pinMode(redled, OUTPUT); //Definição de porta para saída (led vermelho)
  pinMode(buzzer, OUTPUT); //Definição de porta para saída (buzzer)
  pinMode(inPin, INPUT); //Definição de porta para saída (led vermelho)

}
void loop() {

  if(digitalRead(inPin)==HIGH){ //se na porta do botão o valor for HIGH
        digitalWrite(redled, HIGH); //ligar LED
```

```
      digitalWrite(buzzer, HIGH);//ligar buzzer
 }
else{

      digitalWrite(redled, LOW); //desligar LED
      digitalWrite(buzzer, LOW); //desligar buzzer
}}
```

De uma forma sucinta, o LED e o *buzzer* serão ligados se o botão for premido.

3.4. Montagem

Tendo em conta os testes efetuados e os resultados obtidos, procedeu-se à montagem final dos diferentes módulos. Apresentam-se de seguida os procedimentos associados à ação descrita.

3.4.1. Módulo Bilheteira: *Hardware*

A montagem inicial visava a montagem de um módulo RIFD num *shield* XBee e por sua vez num Arduino Mega 2560 como se pode ver na *Figura 25*.

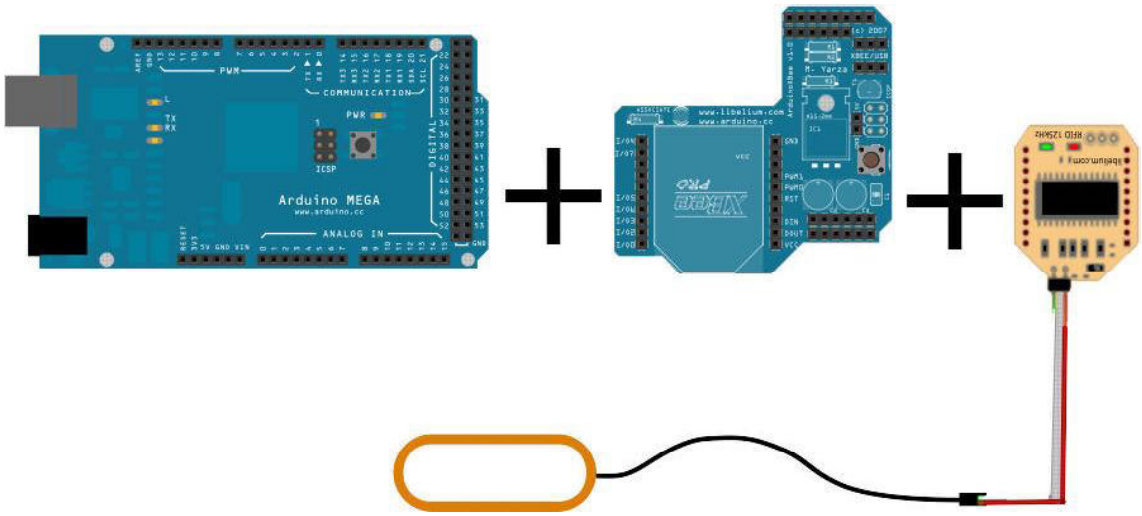


Figura 25 - Montagem inicial de módulo de bilheteira.

Fonte: Elaboração própria

Contudo, a ideia original da montagem visava a utilização das capacidades internas do Arduino Mega 2560 para tratamento de dados. Estes seriam utilizados no projeto, devido ao número das suas portas série físicas. No entanto, este módulo ao estar conectado diretamente a um computador, levaria a que utilização de um Arduino Mega 2560, o que seria um desperdício de capacidades do componente em questão.

Outro problema surgiu na montagem dos outros módulos, nomeadamente sobre a utilização dos *shields* e dos módulos de comunicação XBee, de RFID e ainda o módulo de comunicação sem fios. Ou seja, como utilizar dois *shields* em simultâneo no Arduino Mega 2560. A solução passa pela não utilização dos *shields* e a ligação direta dos componentes XBee e nos componentes semelhantes aos Arduinos utilizados nos módulos.

Para isso, teve-se em conta o diagrama de conexão no *shield* XBee e nos *datasheets* dos componentes. Assim, é apenas necessário conectar o pino 1 do XBee a 3.3 *volts*, o pino 2 (DC out) conectou-se ao RX da porta série, o pino 3 (DC *in*), conectou-se ao conector TX da porta série do Arduino e finalmente, ligar-se o pino 10 ao GND.

Entenda-se que esta conexão pode ser efetuada com módulos XBee e componentes semelhantes, que usam a mesma disposição dos pinos (como poder ver-se na Figura 26) para comunicar com os *shields* XBee.

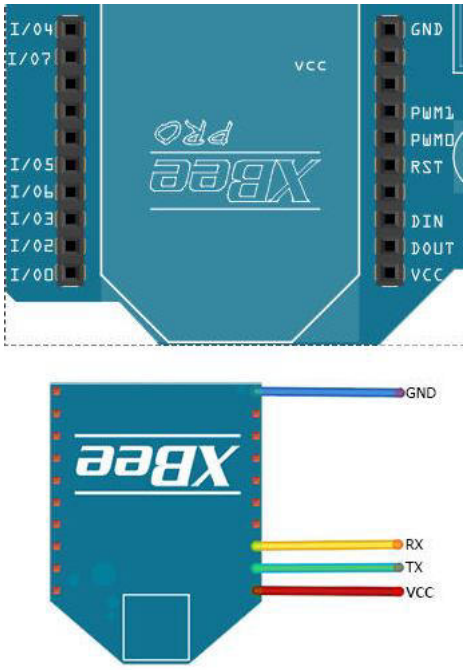


Figura 26 - Pinos a conectar no módulo XBee e semelhantes.

Fonte: Elaboração própria

Desta forma, para a montagem do módulo de bilheteira, acabou por decidir-se pela utilização de um Arduino UNO, uma vez que apenas é necessária uma porta série e este estará em modo *gateway*, ou modo *reset*. Este, permite assim, uma interação direta com o módulo RFID. Para que este modo esteja ativo, basta conectar o pino *reset* diretamente ao pino GND.

Pode verificar-se na Figura 27, o módulo RFID ligado diretamente ao Arduino, permitindo a interação direta com o computador, devido ao modo *reset* do Arduino.

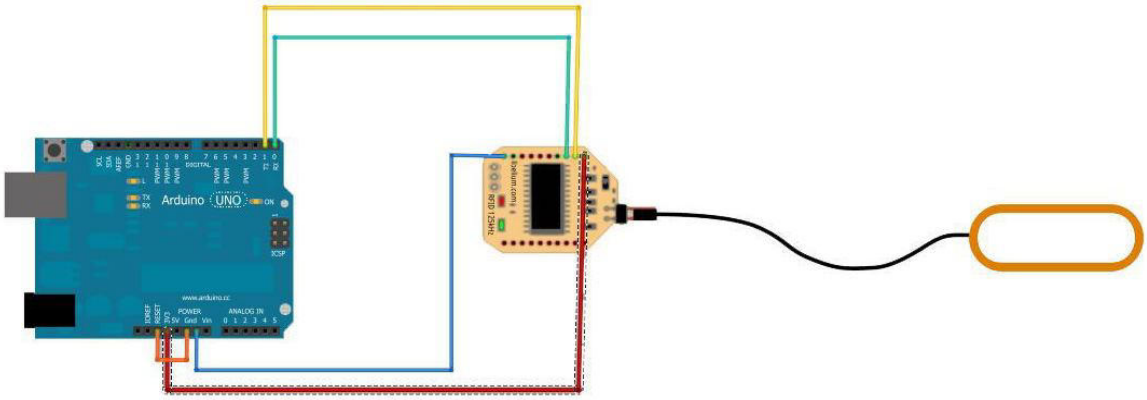


Figura 27 - Montagem final do módulo de bilheteira.

Fonte: Elaboração própria

3.4.2. Módulo Bilheteira: *Software*

O conceito do funcionamento interno do módulo de bilheteira pode ser visto no fluxograma apresentado na Figura 28.

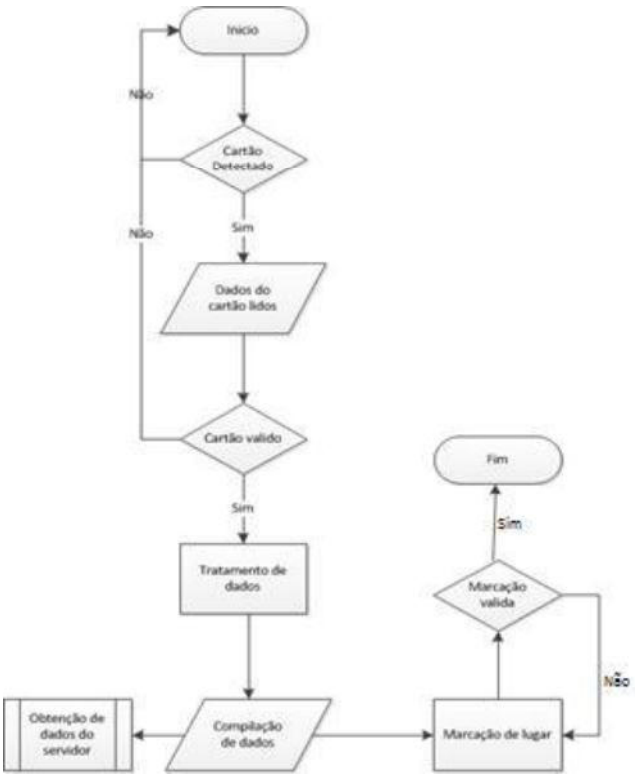


Figura 28 - Fluxograma de funcionamento interno de módulo de bilheteira.

Fonte: Elaboração própria

De forma a ser possível a obtenção de bilhetes e a interação com o módulo de bilheteira, desenvolveram-se duas aplicações.

- Aplicação bilheteira (Figura 29):

  Esta aplicação permitirá a obtenção de informação vinda do módulo de bilheteira e o tratamento da mesma, bem como a interação com os dados existentes no servidor.

Figura 29 - Interface da aplicação MFootball para a bilheteira.

Fonte: Elaboração própria

- Aplicação *Web* (*Figura 30*):
  A aplicação *Web* visa a possibilidade da aquisição de bilhetes, bem como a gestão de dados de utilizador e ainda aquisição de *merchadising*.

Figura 30 - Aplicação *Web*.

Fonte: Elaboração própria

3.4.3. Módulo de Estádio: *Hardware*

A montagem inicial, mostrada na Figura 31, visava a montagem de dois *shields* XBee, um primeiro com um módulo XBee e um segundo com um módulo RFID.

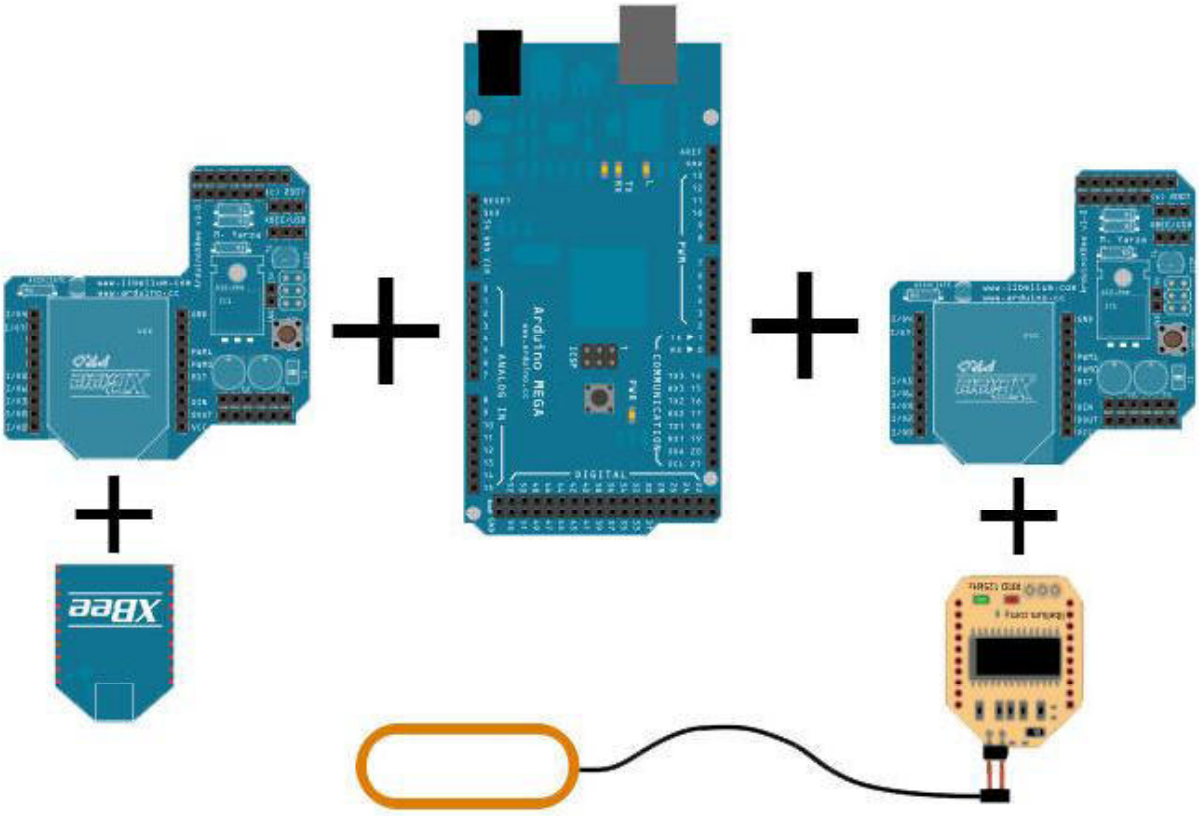


Figura 31 - Elementos de montagem inicial de módulo de estádio.

Fonte: Elaboração própria

Contudo, como referido anteriormente, ligar dois *shields* em simultâneo, mesmo num Arduino Mega 2560, não é algo simples.

Tendo essa mesma informação em mente, decidiu-se proceder à montagem do protótipo final alternativo ilustrado na Figura 32.

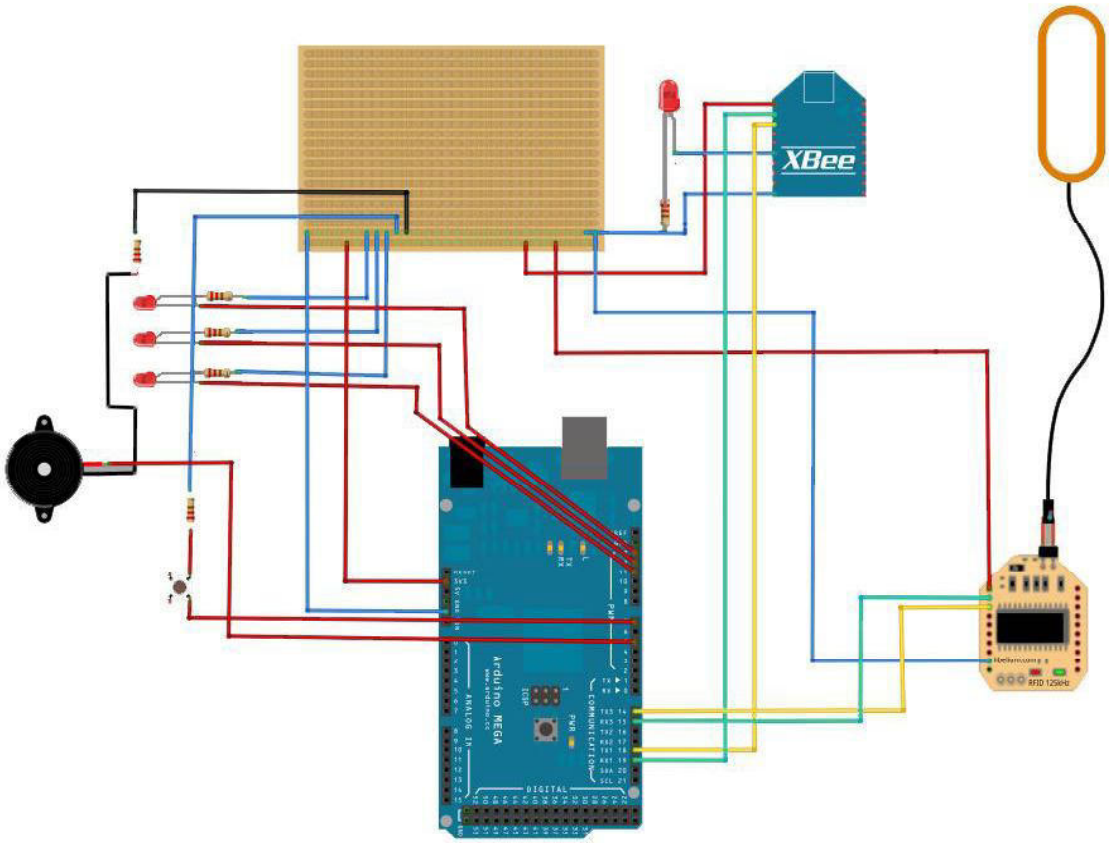


Figura 32 - Montagem final de módulo de estádio.

Fonte: Elaboração própria

A montagem do protótipo final, visa a ligação do módulo XBee diretamente à porta série 1 e do módulo RFID na porta série 3. A alimentação destes componentes é feita através da saída de 3,3V e GND.

Ao módulo XBee, foi acoplado um LED na porta 6, ligado diretamente ao pino GND de forma a perceber quando os dados passam por este componente.

Há que ter em conta que neste diagrama não se representa a alimentação do Arduino e, uma vez que o torniquete a ser utilizado não se encontra disponível, optou-se pela utilização de um botão ligado à porta 7. Pode também aplicar-se um motor, contudo neste caso, não foi aplicado para simplificar a montagem.

Na montagem podem ver-se LED's, representados a vermelho, que servirão para:

- LED *Power*: ligado à porta 13, este LED amarelo, irá notificar a existência de energia;

- LED Entrada negada: ligado à porta 12, este LED vermelho, acende quando a entrada é negada;
- LED Entrada permitida: ligado à porta 11, este LED verde, acende quando a entrada é permitida.

Por fim, na porta 5 encontra-se conectado um *buzzer*, que irá emitir um sinal sonoro com 1 *beep* para a entrada permitida e com para a entrada negada.

3.4.4. Módulo de Estádio: *Software*

O funcionamento lógico desta unidade começa pela leitura do cartão, depois os seus dados são enviados para a unidade de comunicação, dá-se então a receção dos dados e ativação, ou não do atuador.

O seguinte fluxograma (*Figura 33*), representa a lógica associada ao funcionamento da unidade de estádio.

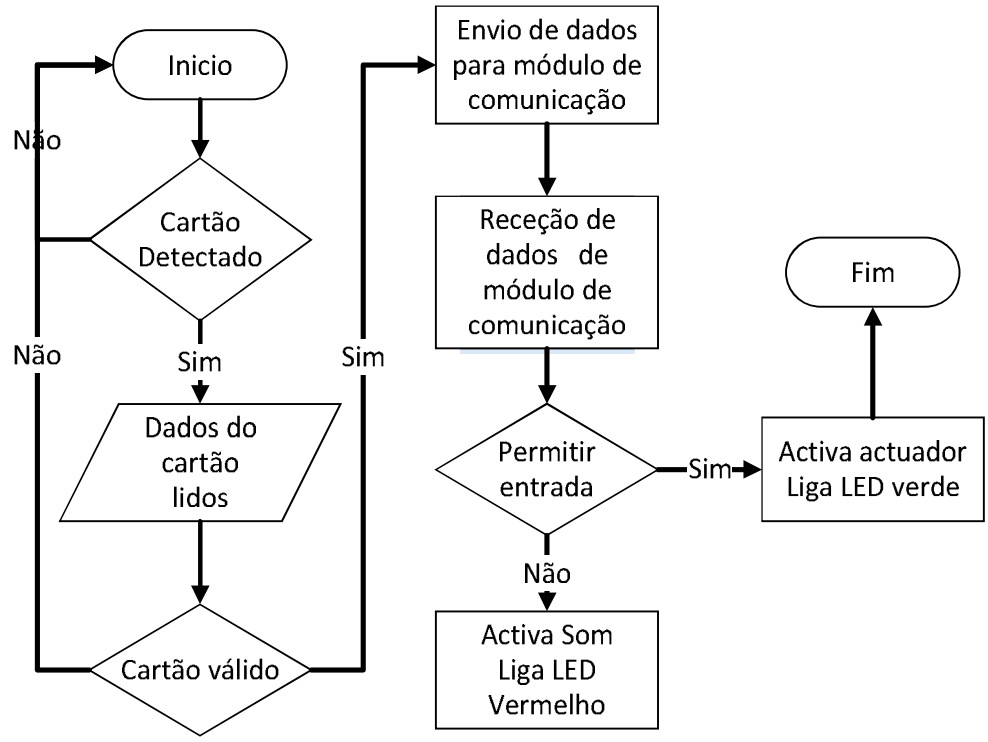


Figura 33 - Fluxograma do funcionamento do módulo do estádio.

Fonte: Elaboração própria

3.4.5. Módulo de Comunicação: *Hardware*

A montagem prevista inicialmente visava a utilização de um módulo GSM/GPRS baseada no *modem* GPRS Hilo 9000. Contudo, em testes efetuados chegou-se à conclusão de que o mesmo não era uma solução viável.

Assim, passou-se à utilização de um módulo, de redes sem fios, da WiFly RN171 da *Roving Networks*, acoplado num módulo com referência RN-XV. Após isto, começou uma nova fase de testes, de forma a tentar obter resultados que pudessem enquadrar-se nas necessidades para a construção dum protótipo final (Figura 34).

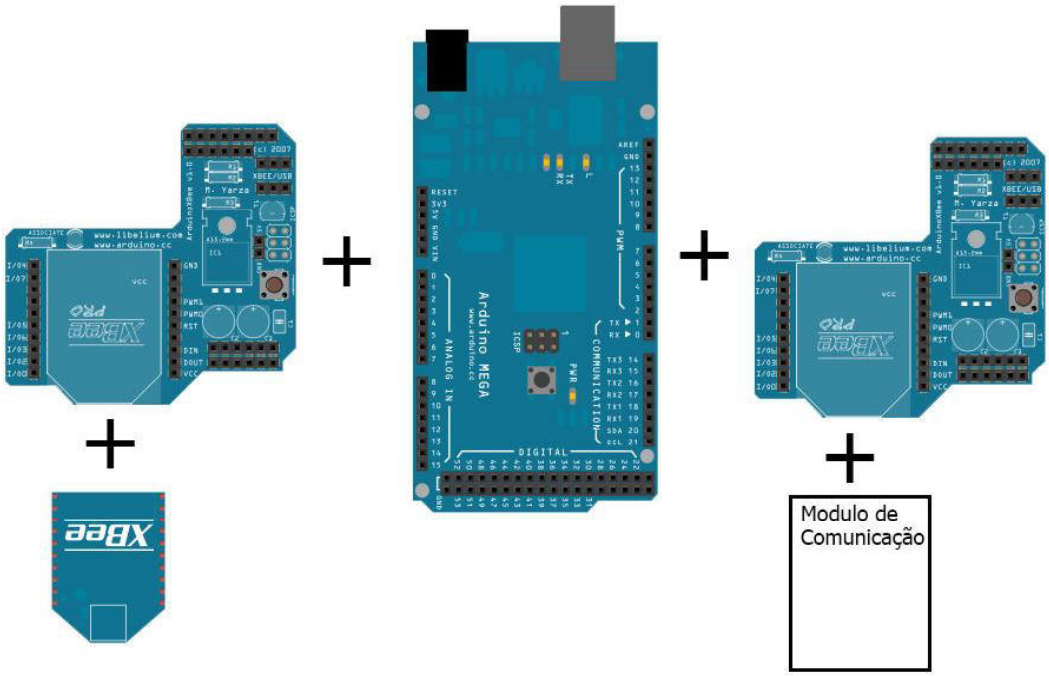


Figura 34 - Montagem inicial do módulo de comunicação.

Fonte: Elaboração própria

Deste modo, e de uma forma similar à unidade anterior, a montagem final irá ficar como o módulo de estádio, com o módulo XBee na porta série 1 e o módulo WiFly na porta de série 03 (*Figura 35*).

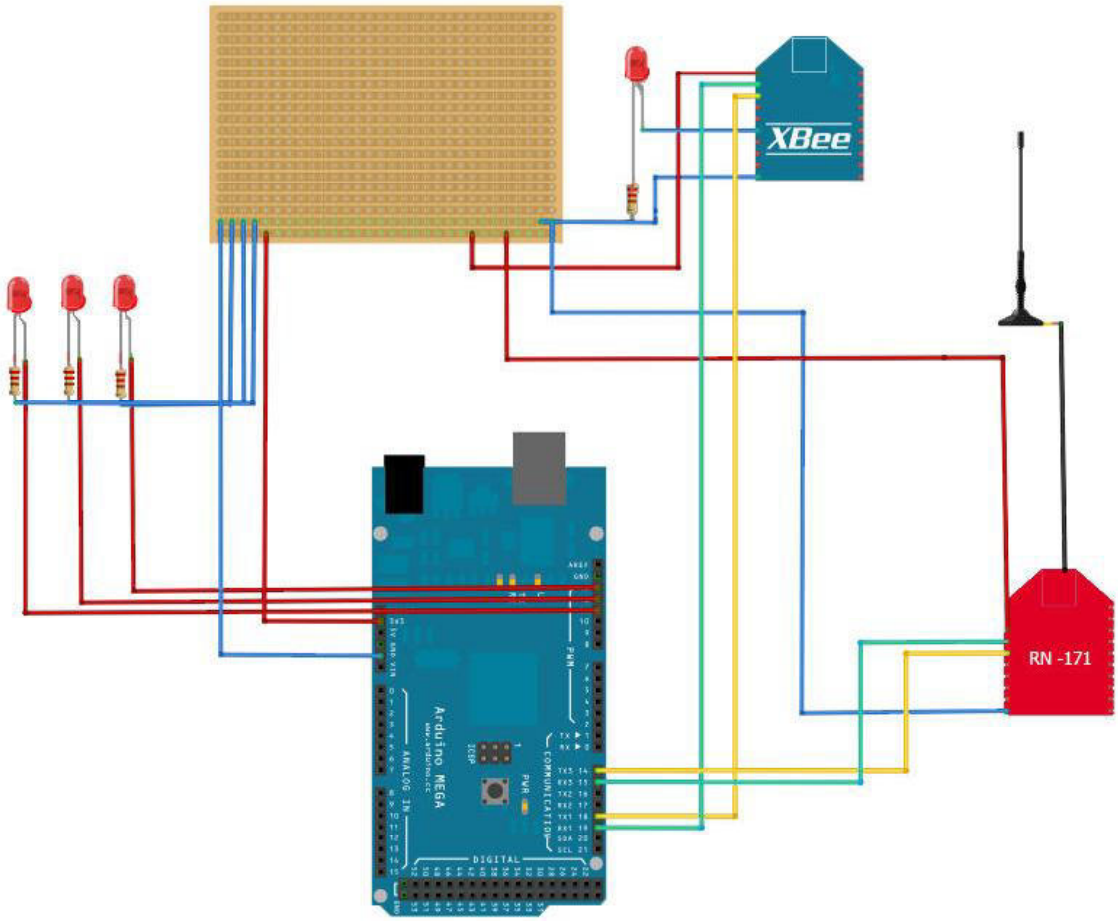


Figura 35 - Montagem final da unidade de comunicação.

Fonte: Elaboração própria

3.4.6. Módulo de Comunicação: *Software*

Na unidade de comunicação, os dados recebidos são enviados pela rede e a resposta do servidor é então enviada para a unidade de estádio.

A lógica pode ser vista no fluxograma na Figura 36.

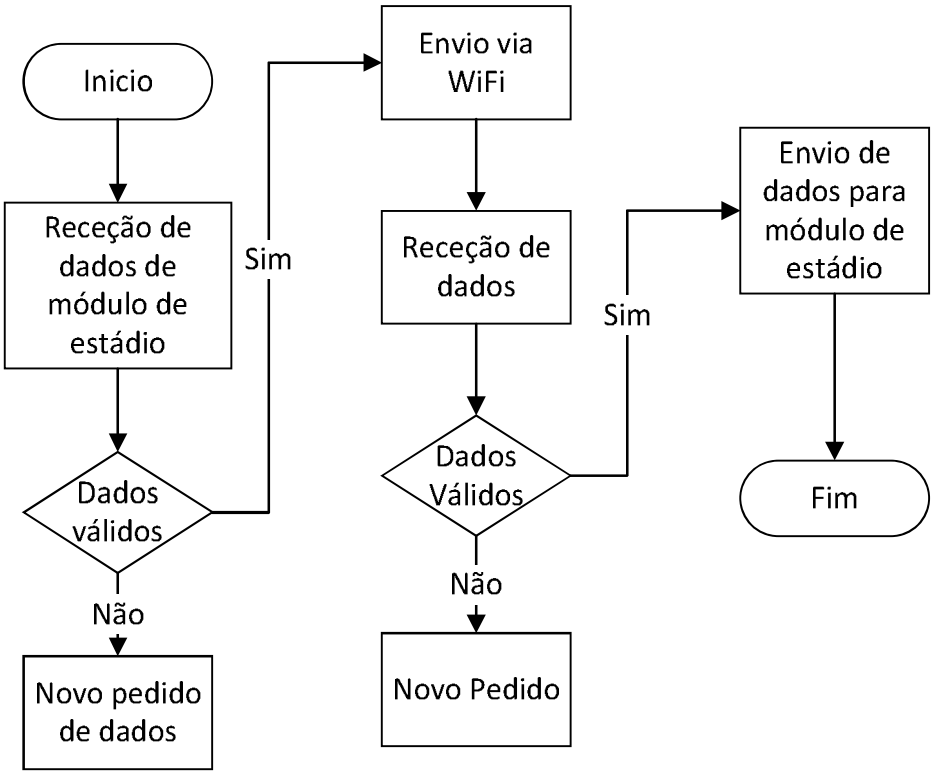


Figura 36 - Fluxograma de funcionamento de módulo de comunicação.

Fonte: Elaboração própria

3.5. Rede (*Web Service*)

Como foi referido anteriormente, os *Web Services* servem para integrar de tecnologias diferentes, permitindo a manipulação e a troca de dados entre si. No projeto apresentado, este serve como intermediário entre a base de dados e outros elementos do projeto que necessitam de dados nela contidos (Figura 37).

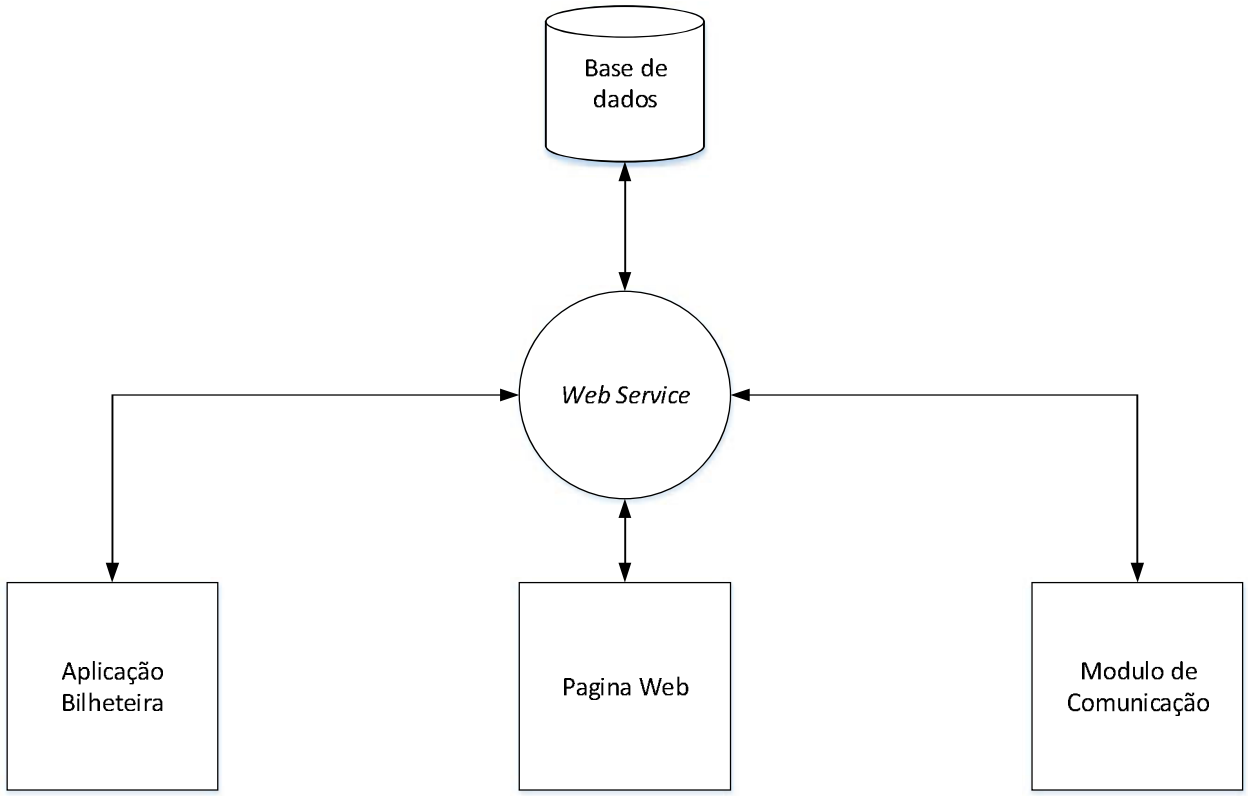


Figura 37 - Interação de *Web Service* e outros elementos do projeto.

Fonte: Elaboração própria

A programação do *Web Service* foi efetuada em *Microsoft Visual Studio* 2008, associado a C#. Uma vez que a escolha do gestor de base de dados caiu sobre o Oracle MySQL, é necessário o conector que irá permitir a comunicação do *Web Service* com a base de dados.
Começou-se então pela inserção das livrarias, referentes à utilização do conector MySQL:

```
using MySql.Data.MySqlClient;
using System.Data.SqlClient;
```

Após isto, é necessária a introdução de referência de MySQL. Para além do *namespace* e da classe os principais constituintes do *Web Service* são os métodos *Web*. Estes permitem a manipulação de dados pretendida e inúmeras ações sobre os mesmos. Para a interação com o servidor de base de dados é necessária a utilização de elementos associados à livraria do conector:

```
        private MySqlDataAdapter da = null;
        private DataSet ds = null;
        private MySqlDataReader dtreader = null;
```

Contudo, para permitir a ligação introduziu-se uma *connection string*, aqui representada na sua forma genérica.

```
    string constr = @"server=localhost;userid=root;password=root;database=test";
```

Após isto, partiu-se para a produção de métodos para a interação com o servidor de base de dados. Tendo em conta que a interação é efetuada através de procedimentos é necessário o envio dos dados que deverão ser tratados pelo servidor. Desta forma, especificou-se um modelo de método para o envio de dados.

```
        [WebMethod]//definição de WebMethod(sem esta o Web Service não implementa o
metodo)
        public string setInssocio(string idsocio, string nomesocio, string moradasocio,
int telefonesocio)
        {
            MySqlConnection conn = null; //Definição de conexão

            try
            {
                conn = new MySqlConnection(constr); //nova conexão soociada à
connection string
               //Comando que associa a conexão ao procedimento
               MySqlCommand cmd = new MySqlCommand("INSOCIO", conn);
                //Especificação de tipo de commando associado ao procedimento
                cmd.CommandType = CommandType.StoredProcedure;
               //Especificação de parametros de entrada de dados
                cmd.Parameters.Add(new MySqlParameter("@INIDSOCIO", idsocio));
                cmd.Parameters.Add(new MySqlParameter("@INNOME", nomesocio));
                cmd.Parameters.Add(new MySqlParameter("@INMORADA", moradasocio));
                cmd.Parameters.Add(new MySqlParameter("@INTELEFONE", telefonesocio));
                cmd.Parameters.Add(new MySqlParameter("@INDATA", date.Date));
```

```
            //Estabelecimento de conexão
             cmd.Connection.Open();

            //Especificação de DataAdapter
            MySqlDataAdapter adapter = new MySqlDataAdapter(cmd);

            //Execução de comando
             cmd.ExecuteNonQuery();

           //Encerramento de conexão
             cmd.Connection.Close();

             return "Inserido com sucesso."; //mensagem de retorno
         }
         catch (MySqlException ex)
         {
             System.Web.HttpContext.Current.Response.Write(ex.Message);
             return ex.Message; // mensagem de retorno de erro
         }


     }
```

Outro método foi criado tendo com o propósito de obter dados.

```
     [WebMethod]//definição de WebMethod(sem esta o Web Service não implementa o
metodo)
     public string getDatasocio(string idsocio)
     {
         string dtout = null; //Variável de saída
         MySqlConnection conn = null; //Definição de conexão
         MySqlDataReader rdr = null; //Definição de leitor de dados (DataReader)

         try
         {
             conn = new MySqlConnection(constr); //Nova conexão associada à
connection string
```

```
            //conexão associada ao procedimento OBTSOCIO
            MySqlCommand cmd = new MySqlCommand("OBTSOCIO", conn);
            //Especificação de tipo de commando associado ao procedimento
            cmd.CommandType = CommandType.StoredProcedure;
            //Especificação de parâmetro de entrada do procedimento
            cmd.Parameters.Add(new MySqlParameter("@INIDSOCIO", idsocio));
            //Estabelecimento de conexão
            cmd.Connection.Open();
            //Execução de comando associado ao leitor de dados
            rdr = cmd.ExecuteReader();
            //leitura de dados do objeto rdr (leitor de dados)
            while (rdr.Read())
            {
                  //Resultado é então introduzido na string de saida sendo cada
                  //dado separado por '##' para se tratar mais tarde
                  dtout=(rdr.GetInt32(0) + "##" + rdr.GetString(1) + "##" +
rdr.GetString(2) + "##" + rdr.GetString(3) + "##" + rdr.GetString(4));

             }}
           catch (MySqlException ex)
           {
               System.Web.HttpContext.Current.Response.Write(ex.Message);
               return ex.Message; //retorno de erro
           }
           finally
           {
               if (rdr != null)
               {
                   rdr.Close();
               }
               if (conn != null)
               {
                   conn.Close();
               } }
           return dtout; //retorno de variável de saida
       }
```

Verifica-se que o método é composto por diversas partes e que as mesmas têm uma função específica para proceder à sua inserção, neste caso concreto. Repare-se que, no caso de envio, existem diversas variáveis de entrada no método, que depois serão enviadas para o servidor separadamente, a sequência de entrada aguardada pelo procedimento de inserção de dados no servidor. De uma forma semelhante no segundo método, visando a obtenção de dados, há um parâmetro de entrada. A informação é recebida por um leitor de dados e é, de seguida, introduzidos numa *string* para ser retornada pelo método. Esta forma de fazer sair a informação será justificada na secção da pagina *Web*.

3.6. Página *Web*

A página *Web* foi inicialmente criada em JSP, de forma a ser inserida na plataforma *Google Sites*. Contudo, mais tarde verificou-se que a utilização de base de dados está associada à persistência de dados, através das linguagens JDO (*Java Data Objects*) e JPA (*Java Persistence API*).

A JDO é uma especificação da plataforma Java para persistência de objetos e esta linguagem também especifica a persistência de dados. Do outro lado a JPA é uma API padrão da linguagem Java, para persistência de dados que deve ser implementada por plataformas que desejem seguir tal padrão. A JPA define um meio de mapeamento objeto-relacional, para objetos Java simples e comuns. Contudo, estas mostraram-se difíceis de utilizar por falta de documentação referente à utilização de JDO. Devido à aquisição pela Oracle, a JPA encontra-se aparentemente descontinuada.

Aguardando que estas questões fossem resolvidas, decidiu-se pela utilização do MySQL, como gestor de base de dados pela simplicidade e pelas capacidades que detêm. Após isto, iniciou-se a pesquisa de um estilo a ser adaptado à plataforma *Web* a ser criada. Optou-se pela utilização do *NightVision* (caspiquad, 2014), uma vez que é gratuito e os seus autores permitem algumas alterações (Figura 38*).*

Tendo em conta isto, procedeu-se à alteração de formatação de forma obter o estilo da página.

Figura 38 - Estilo gratuito escolhido.

Fonte: http://templated.co/nightvision

A adaptação do *template* escolhido visou algumas alterações, como se pode verificar na Figura 39.

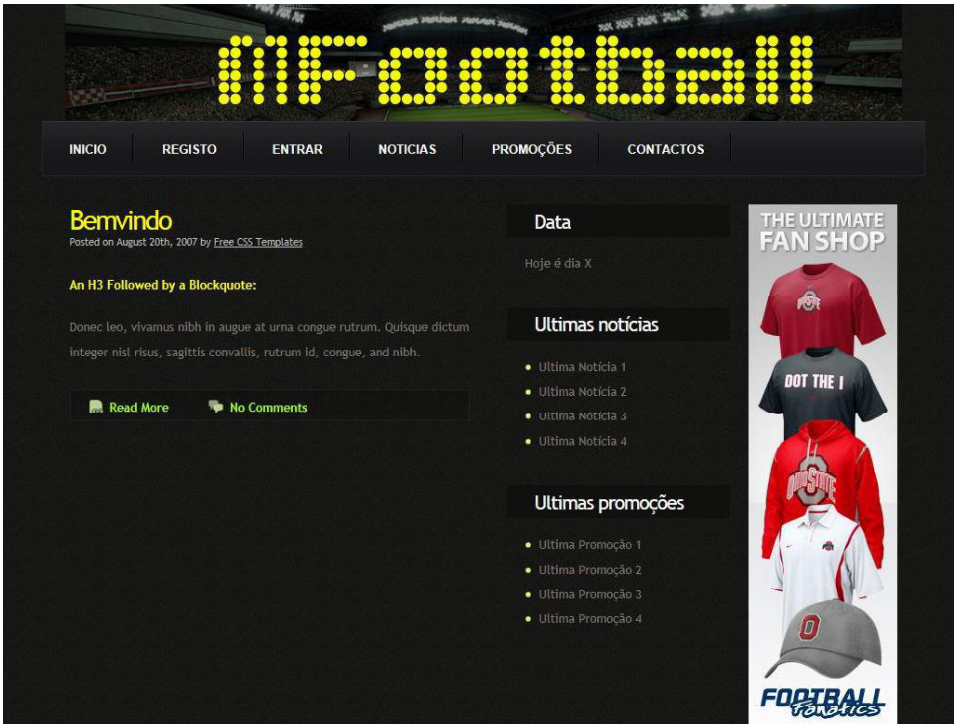


Figura 39 - Aplicação de estilo em página.

Fonte: Elaboração própria

A primeira página serve como menu, bem como para a disponibilização de notícias sobre o clube associado. Assim sendo, o código aplicado para a navegação é:

```
<div id="menu"><!— define o div atribuído ao menu de navegação
	<ul><!— menu horizontal
		<!— Pagina activa (current page item)
		<li class="current_page_item"><a href="">Inicio</a></li>
		<!— Link para pagina de registo
		<li><a href="register.jsp">Registo</a></li>
		<!— Link para pagina de entrada
		<li><a href="login.jsp">Entrar</a></li>
		<!— Link para pagina de noiticias
		<li><a href="news.jsp">Noticias</a></li>
		<!— Link para pagina de compras
		<li><a href="#">Promoções</a></li>
		<!— Link para pagina sobre
		<!--<li><a href="#">Sobre</a></li>-->
		<!— Link para pagina de contactos
		<li><a href="#">Contactos</a></li>
	</ul><!— Fim de definição de menu horizontal
</div><!— fim de definição de menu
```

Na página de registo (Figura 40), podem encontrar-se diferentes campos, que servem ao utilizador para a inserção de dados.

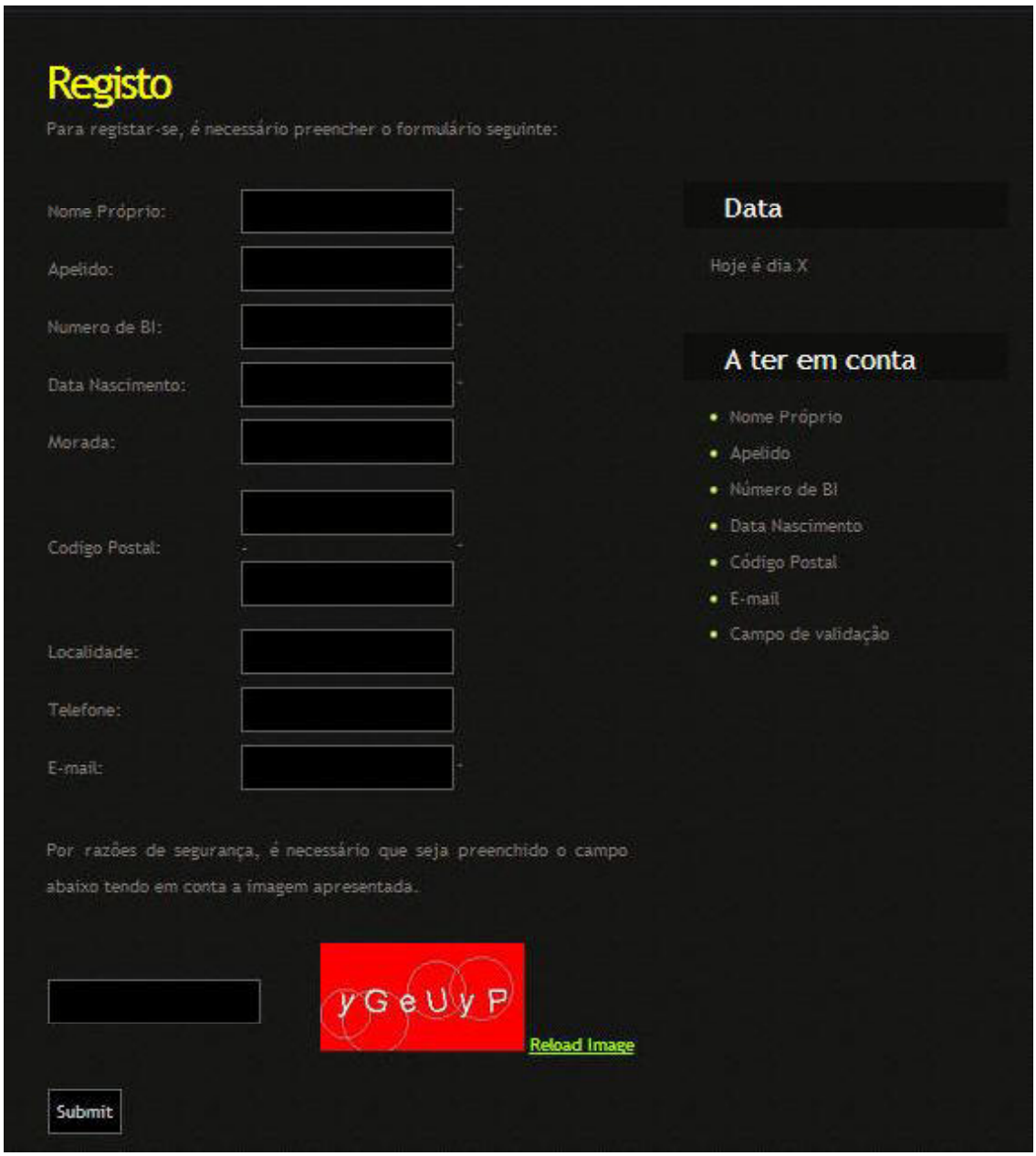


Figura 40 - Formulário de registo.

Fonte: Elaboração própria

Os campos a serem inseridos, que se encontram marcados por '*' são verificados através da utilização de *JavaScript* (Por exemplo, o campo Nome Próprio), os nomes não poderão ser numéricos, devem conter apenas dois caracteres e o campo não poderá deixar-se em vazio. Podem ver-se alguns exemplos na Figura 41 e na Figura 42.

Figura 41 - Validação de campo nome próprio.

Fonte: Elaboração própria

Figura 42 - Validação de nome.

Fonte: Elaboração própria

De forma a pôr em prática este tipo de validações, procedeu-se à utilização de funções em *JavaScript* para verificar os dados inseridos.

```
function checkNome()
       {
         //variável onde se irá armazenar os dados inseridos
         var nome = document.getElementById('tf_nproprio').value;
         //etiqueta que servirá para escrever mensagens na pagina
         var element = document.getElementById('labelUser');

            if(nome.length == 0)//se comprimento de nome=0
            {
                element.innerHTML = 'Por favor, insira um nome próprio';
                element.style.color = 'red';
            }
             else if(nome.length <=2) //se comprimento de nome menor ou igual a 2
            {
                element.innerHTML = 'Nome inválido.';
```

```
            element.style.color = 'red';
        }
        else
        {
            if (IsSpecial(nome)){
                element.innerHTML = 'Nome inválido.';
                element.style.color = 'red';
            }
            else {
                element.innerHTML = 'Nome próprio válido';
                element.style.color = 'green';
            }
        }
    }
```

Para que esta função seja ativada é necessário definir o campo a que se associa, bem como o evento onde será ativada através do código:

```
<input type='text' id="tf_nproprio" name="tf_nproprio" onblur='checkNome();'/>
```

O significado do código HMTL acima descrito, é:

- Campo tipo texto: `type='text'` – o tipo de dados a serem inseridos;
- Identificação de campo: `id="tf_nproprio"` – a identificação do ao se aplica a função de java script;
- Evento associado e função: `onblur='checkNome();` - Evento em que deve aplicar-se a verificação.

Foram também estudadas validações de outros campos, como por exemplo do número de bilhete de identidade. As mesmas podem ver-se na Figura 43.

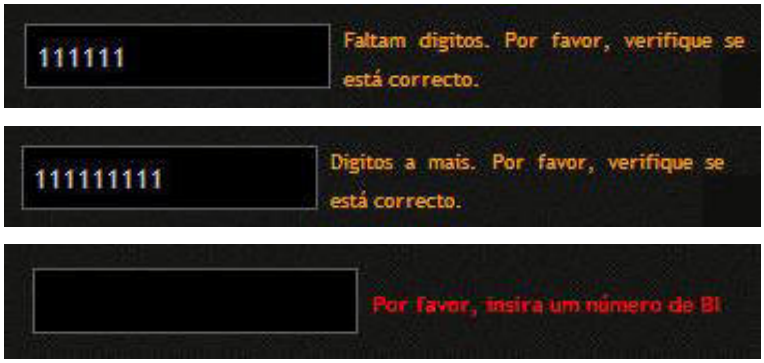


Figura 43 - Verificação de campo de bilhete de identidade.

Fonte: Elaboração própria

Podem ainda encontrar-se validações da data de nascimento (Figura 44).

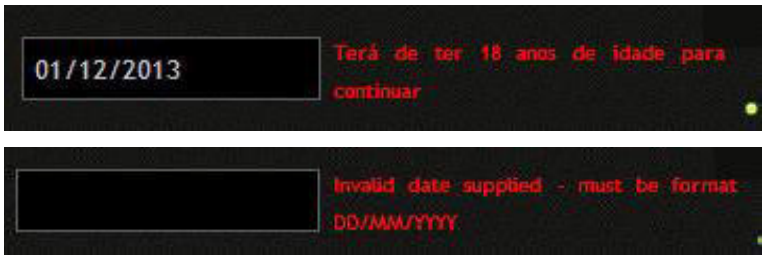


Figura 44 - Verificação de campo de data de nascimento.

Fonte: Elaboração própria

Entre as validações dos diferentes campos, decidiu-se pela utilização de um código de verificação ou *captcha* (Sample JSP Implementation, 2014) (Figura 45) para a inserção de dados.

Figura 45 - *Captcha* da página de registo.

Fonte: Elaboração própria

Este foi criado em JSP e foi chamado através do código:

```
<img src="indexa.jsp" id="img" /> //JSP que devolve captcha
<a href="#" onClick="return reloadImg('img');">Reload Image</a>
```

Para conectar a plataforma *Web* à base de dados optou-se pela ligação da mesma ao *Web Service* (.NET C# Web Service/JSP Client, 2014).

Para tal, deve selecionar-se a pasta "wsdl", na pasta "WEB-INF" que se encontra na pasta do projeto, (Figura 46) e clicar com o botão direito.

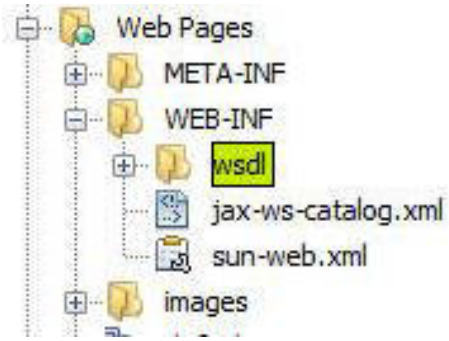


Figura 46 - Pasta WSDL.

Fonte: Elaboração própria

Selecionar a opção *Web Service from WSDL,* como na figura seguinte e escolher WSDL URL e inserir o endereço WSDL do *Web Service* e clicar em *finish* (Figura 47).

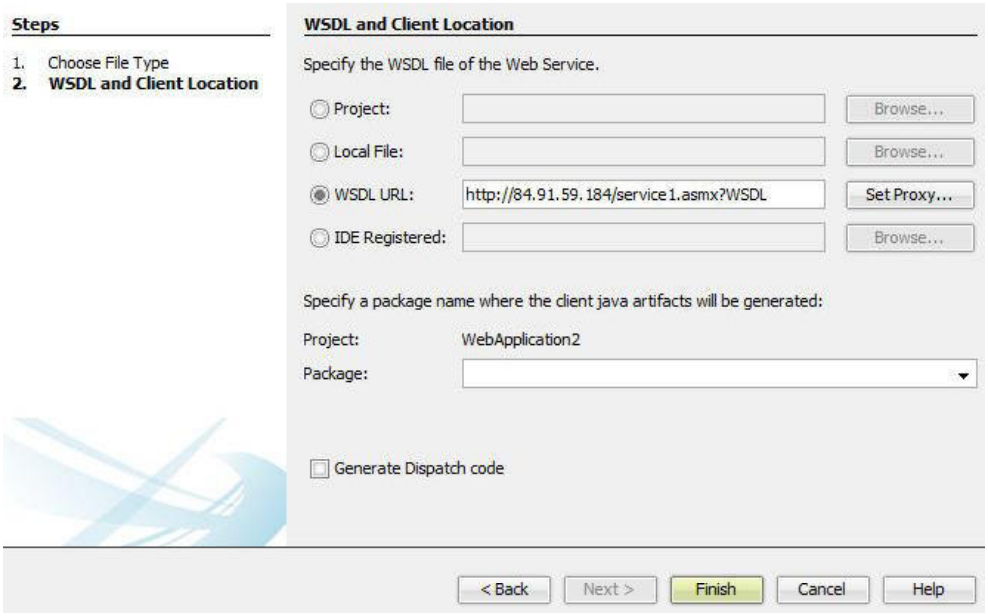


Figura 47 - Inserção de *link* do *Web Service*.

Fonte: Elaboração própria

Após isso, basta clicar no meio do código com o botão direito e escolher *Web Service Client Resources* e *Call Web Service Operation*, como se demonstra na Figura 48.

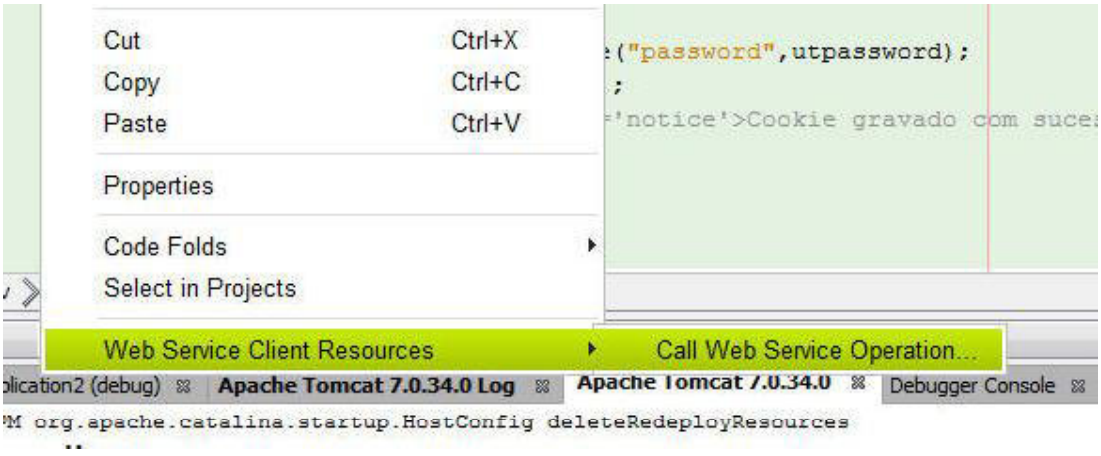


Figura 48 - Chamada de *Web Service* na página JSP.

Fonte: Elaboração própria

Por fim, escolhe-se o método *Web* do *Web Service* a ser utilizado, que neste caso será o método *getlogin* (Figura 49).

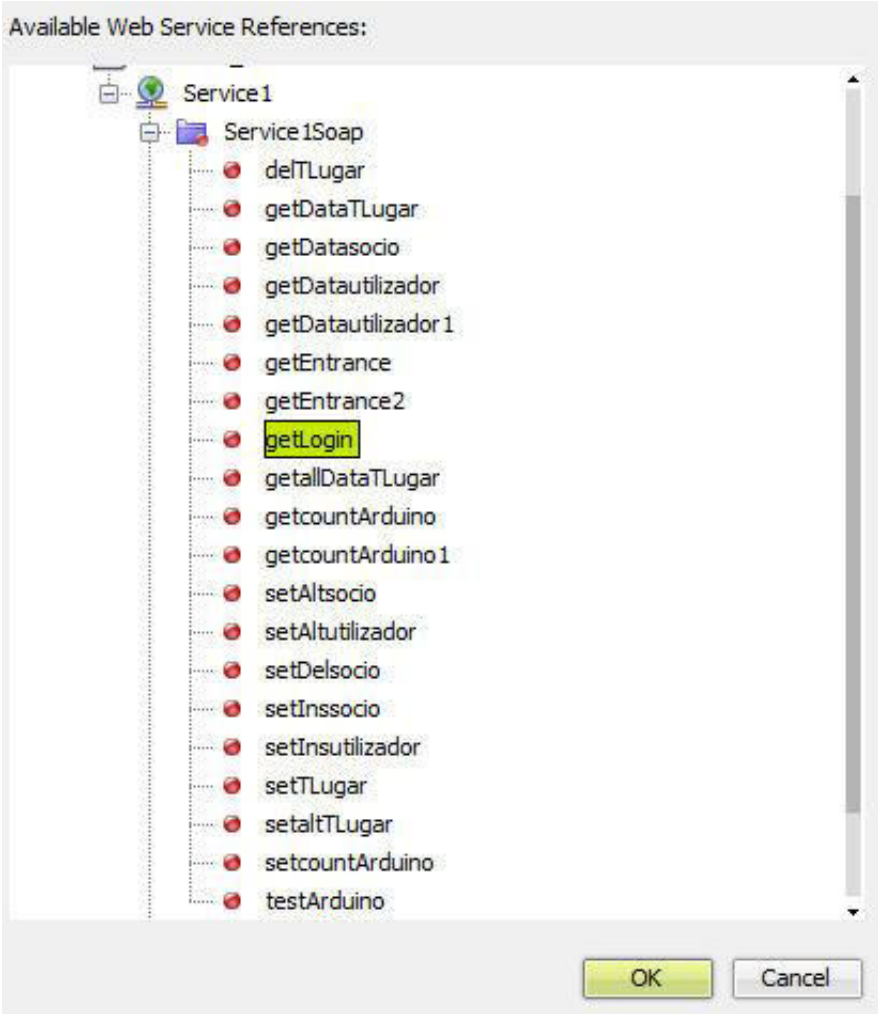


Figura 49 - Seleção de método *Web* a utilizar.

Fonte: Elaboração própria

Destes passos resulta o código que especifica a conexão.

```
org.no_ip.merenthir.Service1 service = new org.no_ip.merenthir.Service1();
org.no_ip.merenthir.Service1Soap port = service.getService1Soap();
// TODO initialize WS operation arguments here
java.lang.String idutilizador = request.getParameter("tf_nproprio");
java.lang.String utpassword = request.getParameter("tf_apelido");
// TODO process result here
```

```
int result = port.getLogin(idutilizador, utpassword);
```

A variável *result* é depois processada como conveniente. Neste caso, determina se o utilizador pode ou não proceder à entrada na plataforma, como se exemplifica na Figura 50.

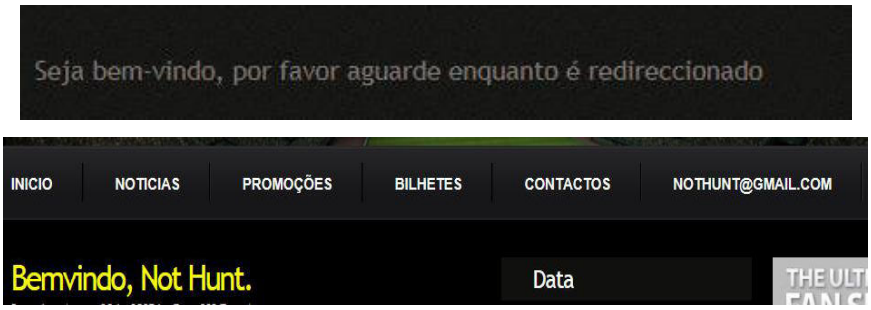


Figura 50 - Página após entrada.

Fonte: Elaboração própria

Para armazenar os dados de utilizador optou-se pela utilização de *cookies*.

```
Cookie nomeCok = new Cookie("username",idutilizador);
response.addCookie(nomeCok);

Cookie passCok = new Cookie("password",utpassword);
response.addCookie(passCok);
```

Quando necessário, pode aceder-se aos dados e manipulá-los, se necessário:

```
String[] _sep1 = new String[20];

     Cookie userCookie = null;
     Cookie passCookie = null;
           Cookie[]  meusCookies = request.getCookies();
           if(meusCookies != null){
              for (int i = 0; i < meusCookies.length; i++){
                 if (meusCookies[i].getName().equals("username")) {userCookie = meusCookies[i]; }
                 if (meusCookies[i].getName().equals("password")) {passCookie = meusCookies[i]; }
              }
           }
    try {
        org.no_ip.merenthir.Service1 service = new org.no_ip.merenthir.Service1();
        org.no_ip.merenthir.Service1Soap port = service.getService1Soap();
        java.lang.String result = port.getDatautilizador1(userCookie.getValue(), passCookie.getValue());
```

```
        _sep1=result.split("##");
        for (int i=0;i<=20;i++)
        {
            if (_sep1[i]!=null)
               System.out.println(_sep1[i]);
        }
    } catch (Exception ex) {
        System.out.printf("Erro: "+ex.getMessage());
    }
```

Como referido anteriormente, na parte do estudo do *Web Service,* os acessos à base de dados são devolvidos numa *String* e os dados são separados por '##'. Inicialmente, os dados eram devolvidos num *DataSet*, contudo, o processo reportava um erro:

src-resolve.4.2: Error resolving component 's:schema'. It was detected that 's:schema' is in namespace ..

Analisados alguns exemplos, denotou-se que se baseavam na utilização de dados do tipo inteiro, texto e booleano e sendo assim, a solução encontrada passa pela utilização de uma única *string* como solução. Os dados obtidos podem então ser utilizados como for pertinente (Figura 51).

Figura 51 - Dados de utilizador.

Fonte: Elaboração própria

3.7. Servidor *Web*

O servidor *Web* foi criado tendo em conta a necessidade de albergar a página e o *Web Service* embora ambos estejam preparados para funcionar em separado. Para o efeito, procedeu-se à instalação do *IIS 7* (*Internet Information Services versão 7*) sendo o processo efetuado num computador com sistema operativo *Microsoft Windows 7 Profissional*. Para iniciar o processo de instalação deve começar-se por aceder ao painel de controlo. Já no painel de controlo deve escolher-se a opção Programas. Aqui, deve escolher-se ativar ou desativar funcionalidades *Windows*. Na janela apresentada Figura 52, das funcionalidades, devem selecionar-se as seguintes funcionalidades.

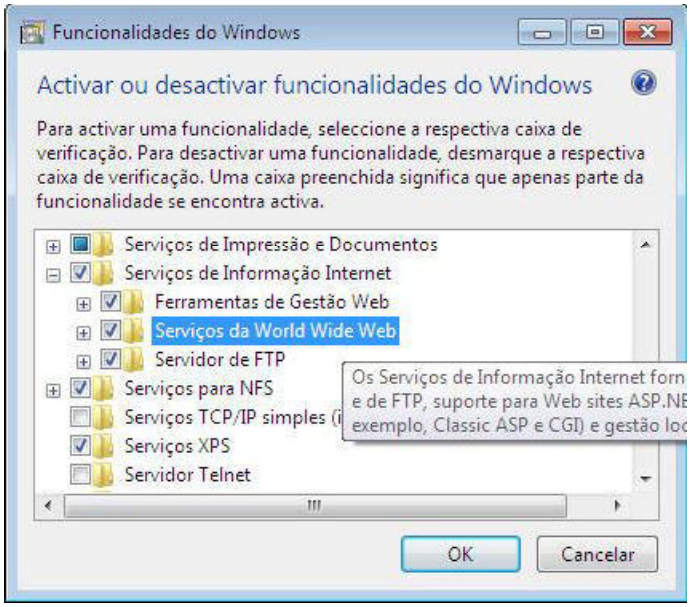


Figura 52 - Funcionalidades de IIS.

Fonte: Elaboração própria

Após o processo de instalação terminar, pode testar-se a mesma através do acesso a *localhost* no *browser*. Se a imagem de boas vindas do *IIS7* for mostrada então o servidor *Web* está a funcionar. Pode ver-se o exemplo na Figura 53.

Figura 53 - Boas-vindas de IIS.

Fonte: Elaboração própria

3.8. Servidor de Bases de dados

Como referido anteriormente, a base de dados será implementada em MySQL. Para tal descarregou-se e instalou-se o MySQL (Download MySQL Community Server, 2014).

3.8.1. Base de dados

Depois da instalação passou-se à implementação da base de dados e começou-se pela elaboração de um modelo de entidades relacionais ou modelo ER. Para proceder à modelação da base de dados utilizou-se o *PowerDesigner* 16 (PowerDesigner , 2014) da *Sybase* numa versão totalmente funcional de testes (*Figura 54*).

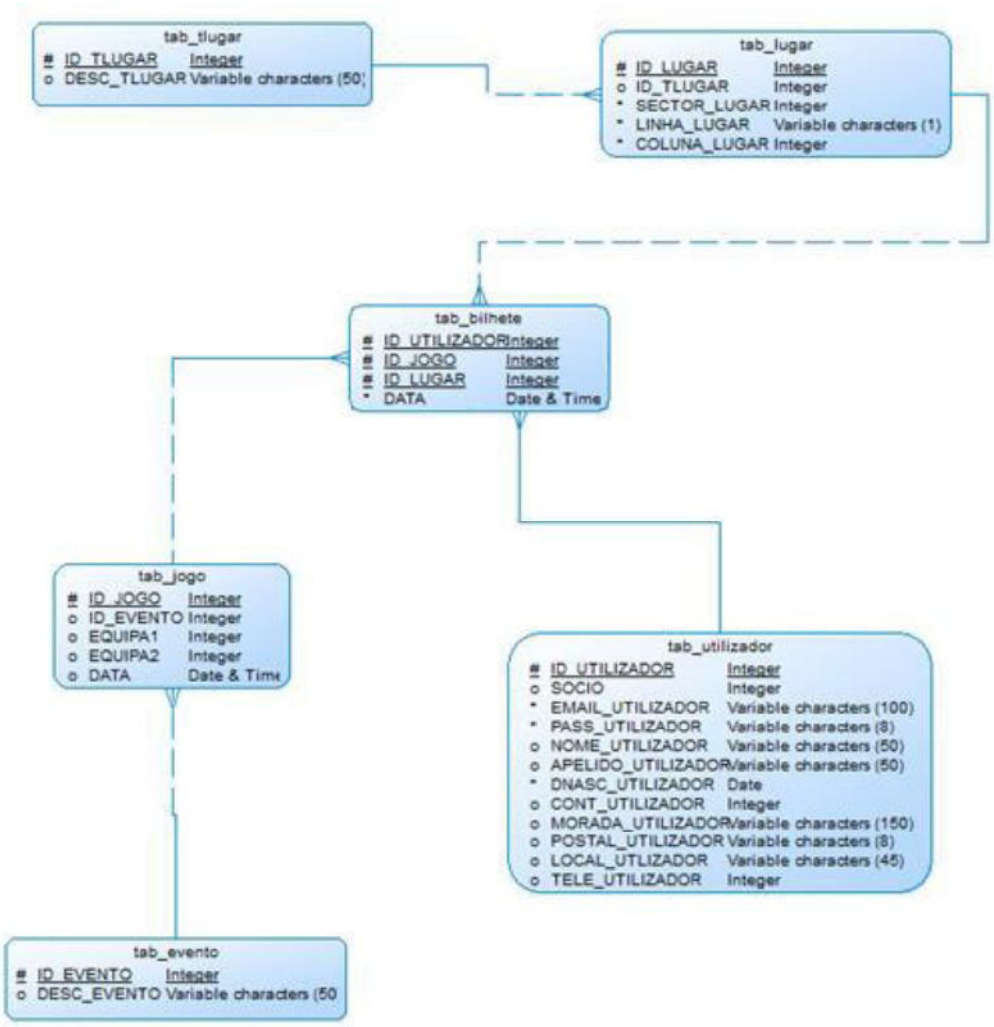


Figura 54 - Modelo ER de base de dados MFootball.

Fonte: Elaboração própria

A base de dados criada é pequena, mas serve bem o propósito que se pretende. Na verdade, o projeto visa a adaptação a plataformas já existentes. Desta forma, a base de dados utilizada serve como apoio à ação dos protótipos.

Existem então 5 tabelas que permitem armazenar as informações que permitirão o bom funcionamento do projeto e estão definidas como:

Tabela 10 - Definição de tab_bilhete.

Tabela: tab_bilhete				
Descrição: Esta tabela visa armazenar as informações referentes aos bilhetes. Informações como o jogo, o lugar, o utilizador e a data do evento que serve também como validade.				
Campo	Tipo	Chave primária	Chave estrangeira	Definição
ID_LUGAR	Inteiro	X		Numero identificador de lugar
ID_JOGO	Inteiro	X		Numero identificador de jogo
ID_UTILIZADOR	Inteiro	X		Numero identificador de utilizador
DATA	Data e tempo			Data associada ao evento / validade de bilhete

Fonte: Elaboração própria

Tabela 11 - Definição de tab_jogo.

Tabela: tab_jogo				
Descrição: esta tabela armazena informações referentes ao jogo tal como a identificação de jogo, o tipo de evento, as equipas e a data em que o jogo se realiza.				
Campo	Tipo	Chave primária	Chave estrangeira	Definição
ID_JOGO	Inteiro	X		Numero identificador de jogo
ID_EVENTO	Inteiro		X	Numero identificador de evento
EQUIPA1	Inteiro			Numero identificador de equipa
EQUIPA2	Inteiro			Numero identificador de equipa
DATA	Data e tempo			Data associada ao jogo

Fonte: Elaboração própria

Tabela 12 - Definição de tab_evento

Tabela: tab_evento				
Descrição: Armazenam-se aqui informações referentes ao tipo de evento descrevendo o jogo como um amigável ou mesmo um jogo de campeonato.				
Campo	Tipo	Chave primária	Chave estrangeira	Definição
ID_EVENTO	Inteiro	X		Numero identificador de evento
DESC_EVENTO	Texto			Descrição do evento

Fonte: Elaboração própria

Tabela 13 - Definição de tab_lugar.

Tabela: tab_lugar				
Descrição: Nesta tabela podem armazenar-se as informações do lugar. O tipo de lugar e a sua localização, o sector, a linha e a coluna.				
Campo	Tipo	Chave primária	Chave estrangeira	Definição
ID_LUGAR	Inteiro	X		Número identificador de lugar
ID_TLUGAR	Inteiro		X	Número identificador de tipo de lugar
SECTOR_LUGAR	Inteiro			Sector onde se localiza o lugar
LINHA_LUGAR	Texto			Linha onde se localiza o lugar
COLUNA_LUGAR	Inteiro			Coluna onde se localiza o lugar

Fonte: Elaboração própria

Tabela 14 - Definição de tab_tlugar.

Tabela: tab_tlugar				
Descrição: Esta tabela, armazena os tipos de lugar que podem ser atribuídos. Desde lugares *vip* a lugares para não sócios.				
Campo	Tipo	Chave primária	Chave estrangeira	Definição
ID_TLUGAR	Inteiro	X		Numero identificador de tipo de lugar
DESC_TLUGAR	Texto			

Fonte: Elaboração própria

Para começar o processo de criação do modelo relacional é necessário em *File* selecionar a opção *NewModel*. Na janela seguinte escolhe-se o tipo de modelo sendo neste caso, o *Logical Data Model*. A sequência pode ver-se nas figuras Figura 55 e Figura 56.

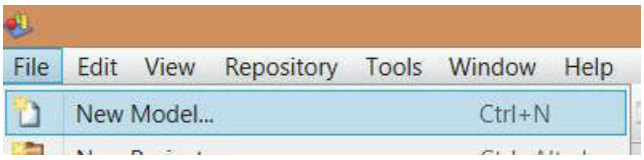


Figura 55 - Criação de novo modelo.

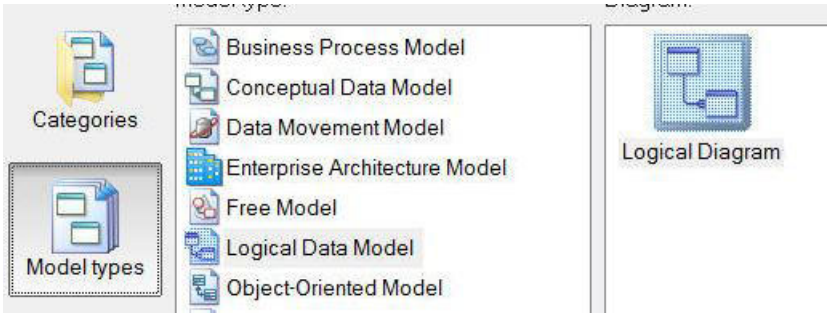


Figura 56 - Opção de modelo a criar.

Fonte: Elaboração própria

Após a atribuição do nome é criado um espaço de trabalho. Podem então, encontrar-se as ferramentas para a criação do modelo relacional. Começou-se pela seleção da entidade e a sua criação no espaço de trabalho (Figura 57).

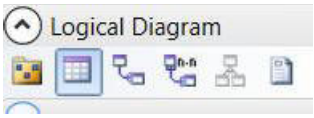


Figura 57 - Seleção de entidade.

Fonte: Elaboração própria

Podem especificar-se os campos da entidade que queremos criar, para isso com o botão direito e clicar em propriedades (Figura 58).

Figura 58 - Menu de contexto associado à entidade.

Fonte: Elaboração própria

É mostrada então, uma grelha, onde se podem definir os diferentes campos associados à entidade que estamos a criar. Existem para cada campo algumas propriedades que podem ser definidas (Figura 59):

1. Mandatório (*Mandatory*): Propriedade que define o campo como um dos campos cujo preenchimento é obrigatório ou não.
2. Chave Primária (*Primary key*): Define se o campo é chave primária ou não. A chave primária identifica cada registo na entidade.
3. Mostrada (*Displayed*): Define se o campo será mostrado ou não na entidade.

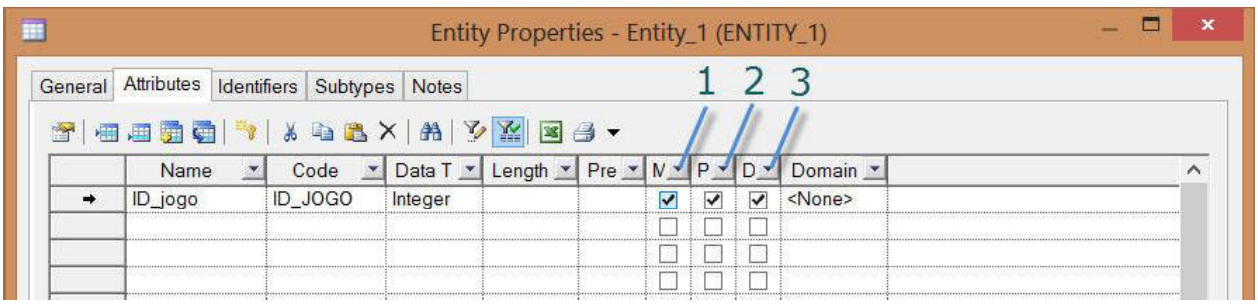


Figura 59 - Propriedades de entidade.

Fonte: Elaboração própria

Depois das entidades podem ainda definir-se as relações entre as diferentes entidades. As relações definem a forma como as entidades interagem entre si. Após a inserção, da relação ao clicar duas

vezes, com o botão esquerdo, podem definir-se as propriedades da relação, como se pode ver na Figura 60.

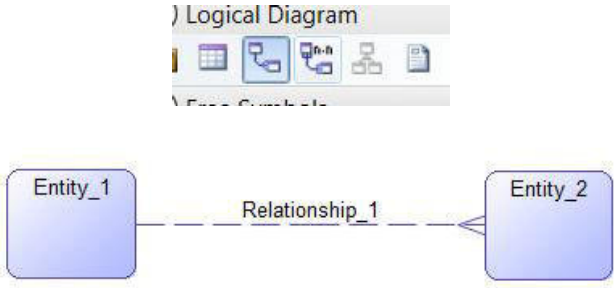


Figura 60 - Relação entre entidades.

Fonte: Elaboração própria

Na lapela de definições gerais podem definir-se o nome, o código e as entidades às quais esta relação está associada. Podem ainda editar-se propriedades de uma dada entidade, bem como adicionar nova entidade (Figura 61).

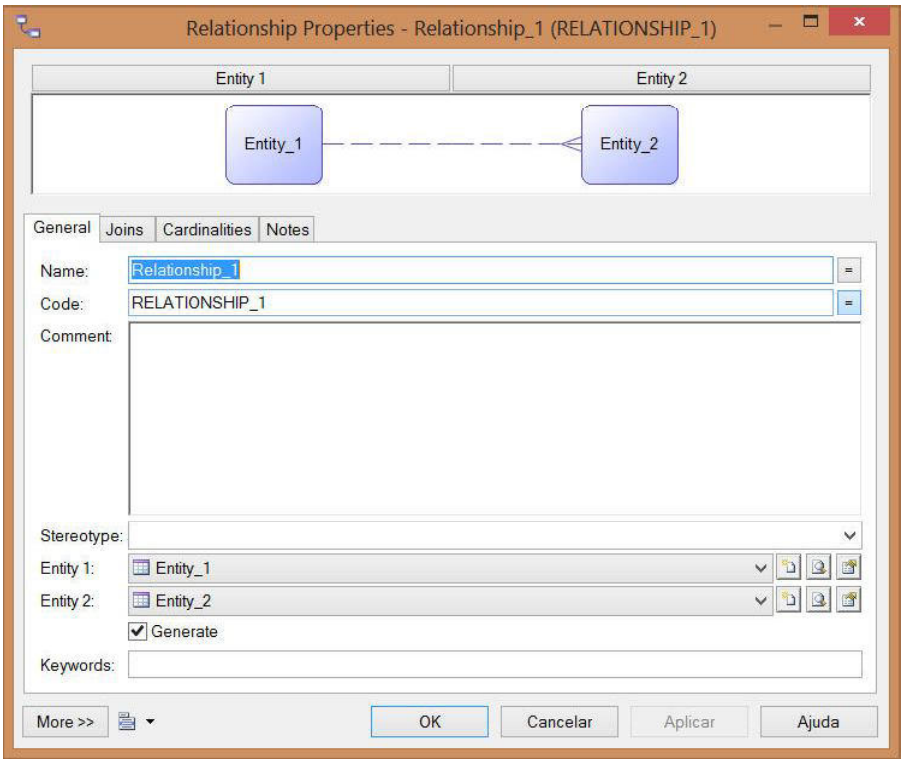


Figura 61 - Propriedades gerais de relação.

Fonte: Elaboração própria

Na lapela *Cardinlities* pode definir-se a multiplicidade e a obrigatoriedade entre as entidades (Figura 62).

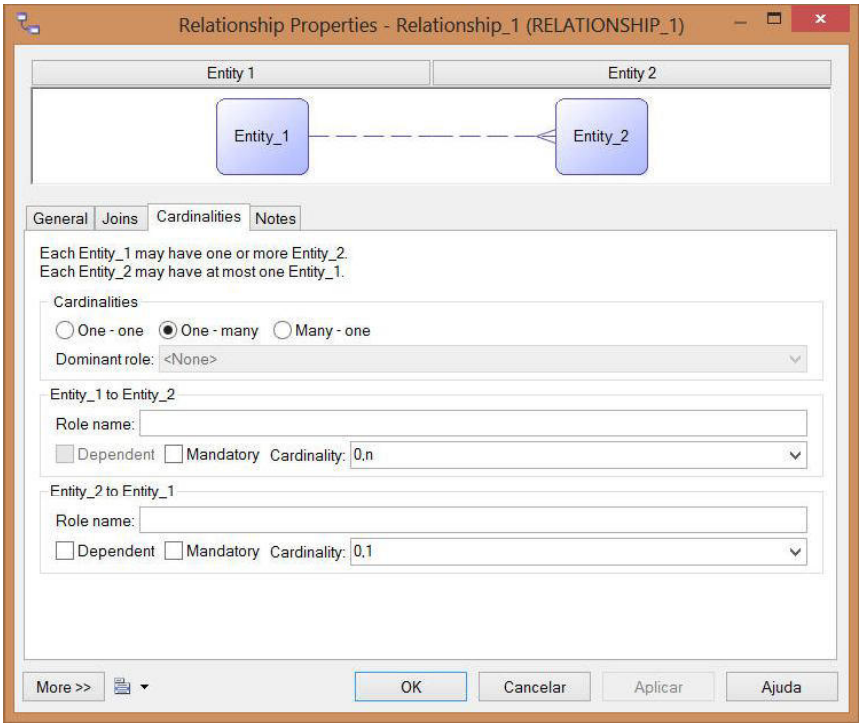


Figura 62 - Cardinalidades de relação

Fonte: Elaboração própria

Após a criação do modelo relacional, pode criar-se o modelo físico. Para isso, em *Tools* clicamos em *Generate Physical Data Model*, opção demonstrada na Figura 63. Pode então criar-se um *script* que nos permite criar a base de dados.

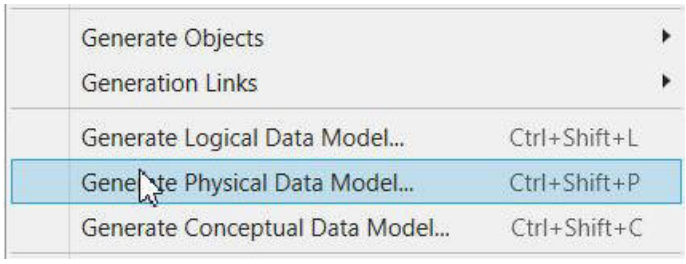


Figura 63 - Gerar modelo físico.

Fonte: Elaboração própria

Depois são mostradas as definições associadas ao *script* que será criado. Na lapela de definições gerais (*General*) podem aplicar-se definições como o caminho de saída, o nome dado ao ficheiro,

bem como o tipo de geração a utilizar. Na geração de *script* pode ainda definir-se se serão utilizados um ou mais ficheiros. A geração direta visa a coneção à base de dados para proceder à criação da base de dados (Figura 64).

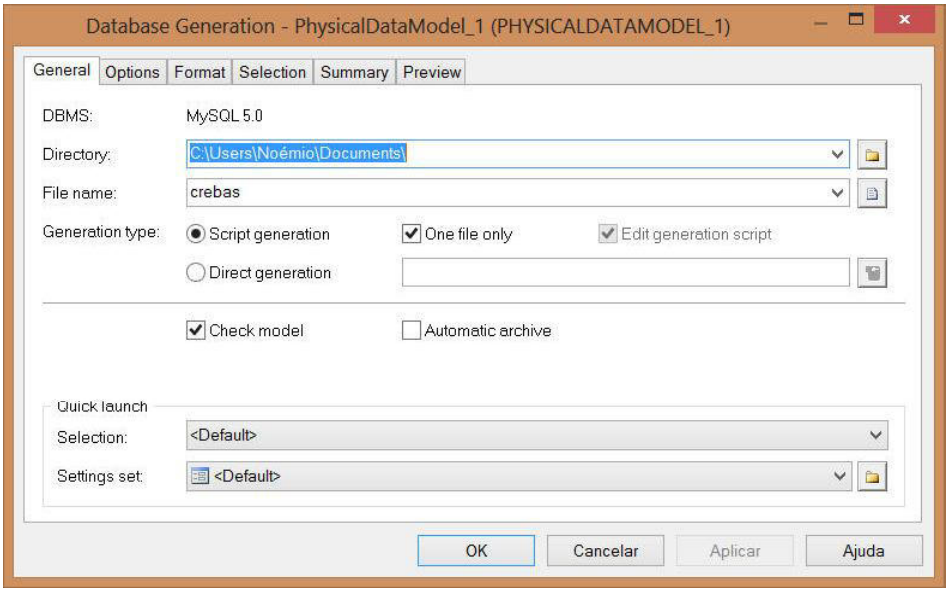


Figura 64 - Definições gerais de criação de script SQL.

Fonte: Elaboração própria

Na lapela *Selection* podem, definir-se as tabelas a serem inseridas. Atenção que, a não seleção de algumas tabelas podem mais tarde trazer problemas de criação do *script* SQL (Figura 65).

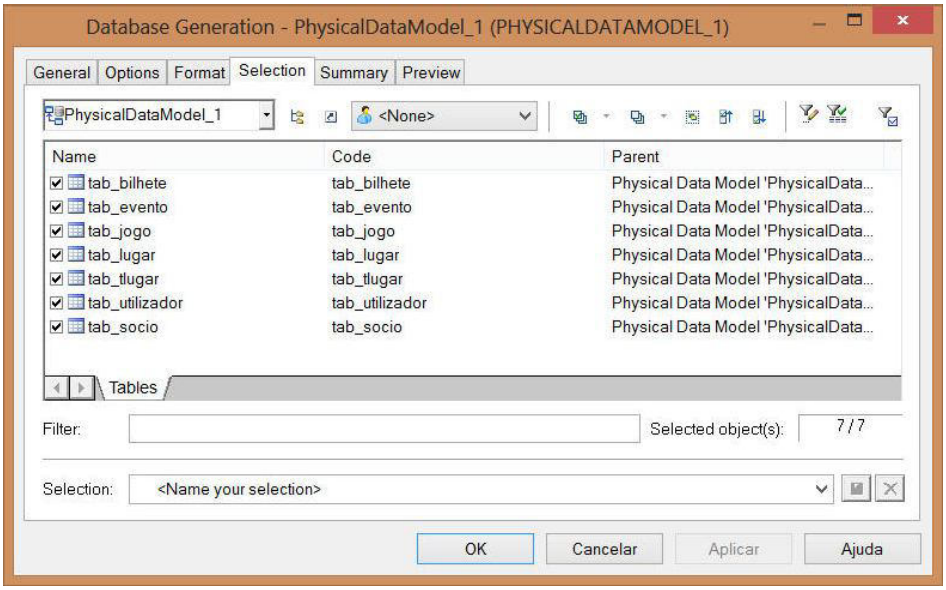


Figura 65 - Seleção de tabelas a serem inseridas no código do *script*.

Fonte: Elaboração própria

Na lapela *Preview* pode ver-se uma previsão do código SQL a ser criado (Figura 66).

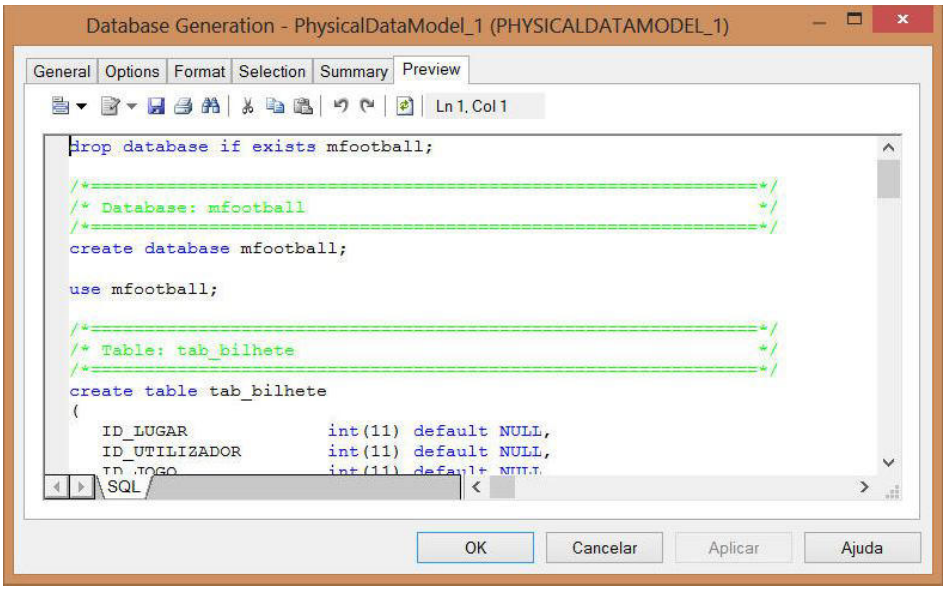


Figura 66 - Previsão do código SQL a ser armazenado pelo *script*.

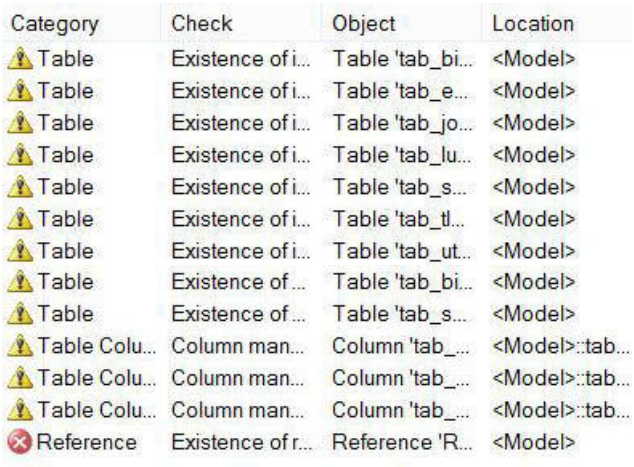
Fonte: Elaboração própria

Após prosseguir, é possivel que sejam encontrados erros e que sejam mostrados avisos antes do processo da criação do ficheiro. A mensagem pode ser vista na figura seguinte (Figura 67).

Figura 67 - Erro resultante da tentativa da criação do código SQL.

Fonte: Elaboração própria

São então mostrados os avisos e erros encontrados durante o processo de criação do código SQL (Figura 68).



Category	Check	Object	Location
Table	Existence of i...	Table 'tab_bi...	<Model>
Table	Existence of i...	Table 'tab_e...	<Model>
Table	Existence of i...	Table 'tab_jo...	<Model>
Table	Existence of i...	Table 'tab_lu...	<Model>
Table	Existence of i...	Table 'tab_s...	<Model>
Table	Existence of i...	Table 'tab_tl...	<Model>
Table	Existence of i...	Table 'tab_ut...	<Model>
Table	Existence of ...	Table 'tab_bi...	<Model>
Table	Existence of ...	Table 'tab_s...	<Model>
Table Colu...	Column man...	Column 'tab_...	<Model>::tab...
Table Colu...	Column man...	Column 'tab_...	<Model>::tab...
Table Colu...	Column man...	Column 'tab_...	<Model>::tab...
Reference	Existence of r...	Reference 'R...	<Model>

Figura 68 - Erros e avisos.

Fonte: Elaboração própria

Para corrigir, clicando sobre o elemento que se querem tratar e escolhem-se a opção *Correct*. Normalmente, as correções são de fácil resolução. Pode também, escolher-se a opção de correção automática (*Automatic Correction*). Esta ultima opção, é utilizada quando os problemas encontrados são muito simples de resolver, como pode ver-se na Figura 69.

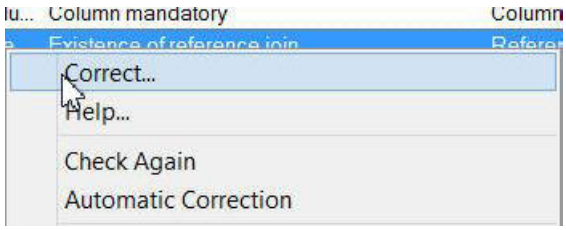


Figura 69 - Menu de contexto de erros ou avisos resultantes.

Fonte: Elaboração própria

Estando os problemas resolvidos, a criação do *script*, quando bem-sucedida, devolve a informação da criação do ficheiro (Figura 70).

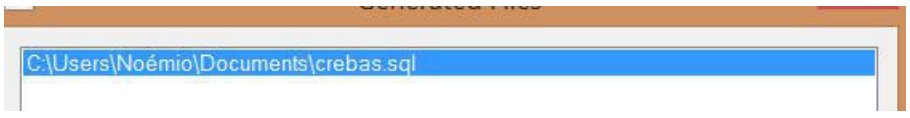


Figura 70 - Caminho da criação do ficheiro SQL.

Fonte: Elaboração própria

O resultado pode ser observado no campo *output*. Neste caso, a geração do ficheiro SQL foi bem-sucedida e pode ver-se a sua localização (Figura 71).

```
-> Reference: FK_RELATIONSHIP_7 (FK_RELATIONSHIP_7)
-> Reference: Reference_5 (REFERENCE_5)
Script Generation completed
Generation successful

Usage:
 (1) Start command prompt
 (2) Go to the directory C:\Users\Noémio\Documents\
 (3) Start the SQL interpreter:
    mysql.exe
 (4) Run the database creation script:
    mysql> source crebas.sql
```

Figura 71 - Resultado de criação de *script*.

Fonte: Elaboração própria

O passo seguinte é feito no MySQL *Workbench* e abrindo o ficheiro SQL (*open SQL script*) (Figura 72).

Figura 72 - Abrir ficheiro SQL.

Fonte: Elaboração própria

O ficheiro é então carregado para o painel, como mostra a Figura 73.

```
/*==============================================================*
/* Database name:  mfootball                                    *
/* DBMS name:      MySQL 5.0                                    *
/* Created on:     27/08/2013 01:08:13                          *
/*==============================================================*


drop database if exists mfootball;

/*==============================================================*
/* Database: mfootball                                          *
/*==============================================================*
create database mfootball;

use mfootball;

/*==============================================================*
/* Table: tab_bilhete                                           *
/*==============================================================*
create table tab_bilhete
(
   ID_LUGAR             int(11) default NULL,
   ID_UTILIZADOR        int(11) default NULL,
   ID_JOGO              int(11) default NULL,
   DATA                 datetime default NULL
)
ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figura 73 - Código SQL para criar base de dados.

Fonte: Elaboração própria

Depois clica-se em executar (Figura 74).

Figura 74 - Botão executar.

Fonte: Elaboração própria

Podem ver-se os resultados no painel de saída (*output*) (Figura 75).

	Time	Action	Message	Duration / Fetch
10	02:03:48	alter table tab_bilhete add constraint FK_REFERENCE_2 fo...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.686 sec
11	02:03:49	alter table tab_bilhete add constraint FK_REFERENCE_3 fo...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.889 sec
12	02:03:50	alter table tab_bilhete add constraint FK_REFERENCE_4 fo...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	1.108 sec
13	02:03:51	alter table tab_jogo add constraint FK_RELATIONSHIP_7 fo...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.531 sec
14	02:03:51	alter table tab_lugar add constraint FK_REFERENCE_5 fore...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.499 sec

Figura 75 - Resultado de criação de base de dados.

Fonte: Elaboração própria

A interação dos elementos exteriores é efetuada através de procedimentos (aqui denominados como *Procedures*) (Figura 76).

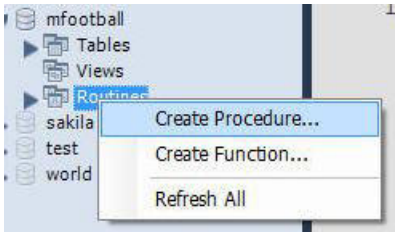


Figura 76 - Criar novo procedimento.

Fonte: Elaboração própria

Após este passo é mostrado parte do código que é utilizado para a criação de um procedimento pode ver-se um exemplo de como é criado um procedimento.

```
CREATE PROCEDURE `mfootball`.`new_procedure`
(
  IN NOMECAMPO1 TIPO(TAMANHO) -- Campo definido como entrada. O tamanho é opcional
  OUT NOMECAMPO2 TIPO(TAMANHO) -- Campo definido como saida. O tamanho é opcional
  IN OUT NOMECAMPO TIPO(TAMANHO) -- Campo definido como entrada e entrada. O tamanho
é opcional
)
BEGIN -- Inicio de operação do procedimento

--Aqui são excetuados os comandos de consultas, inserções, alterações e outras
END
```

Assim sendo pode ver-se na Figura 77, um exemplo existente na base de dados do projeto

```
CREATE DEFINER=`root`@`%` PROCEDURE `INSJOGO`(
    IN INIDJOGO INT(11), --
    IN INIDEVENTO INT(11),
    IN INEQUIPA1 INT(11),
    IN INEQUIPA2 INT(11),
    IN INDATA DATETIME
)
BEGIN

INSERT INTO
TAB_JOGO
(
    ID_JOGO,
    ID_EVENTO,
    EQUIPA1,
    EQUIPA2,
    DATA
)
VALUES
(
    INIDJOGO,
    INIDEVENTO,
    INEQUIPA1,
    INEQUIPA2,
    INDATA

)
;
```

Figura 77 - Exemplo de procedimento na base de dados.

Fonte: Elaboração própria

Para cada tabela, foram criados procedimentos para as operações:

- Inserir
- Alterar
- Obter
- Apagar

Existem também procedimentos para a entrada de utilizadores e para outras verificações. Na Figura 78 podem ver-se elementos da base de dados como tabelas e procedimentos.

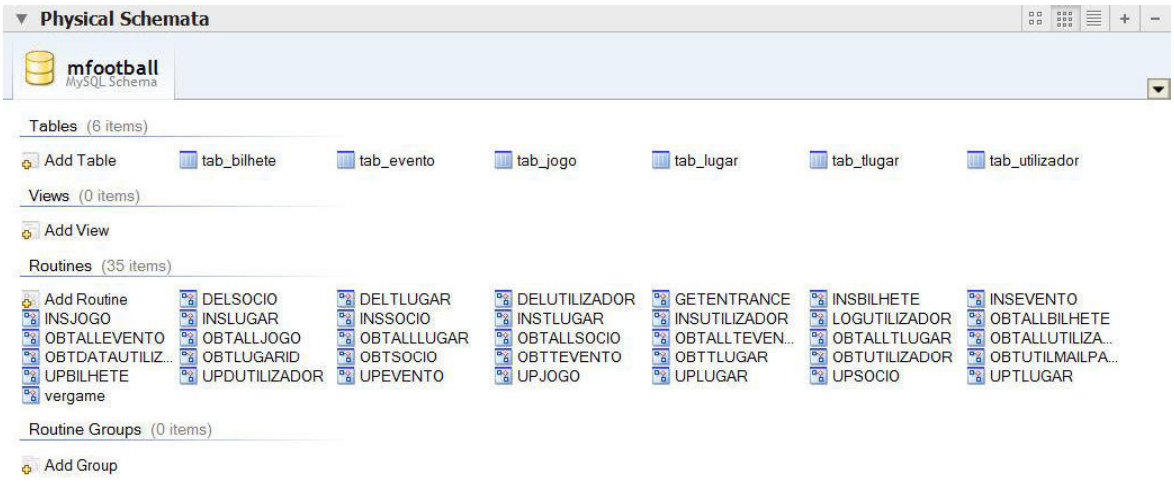


Figura 78 - Elementos da base de dados.

Fonte: Elaboração própria

3.9. Página JSP

Na preparação do servidor, para alocar-se a página JSP, é necessária a instalação do Apache Tomcat neste caso na versão 7 (Tomcat 7, 2014), com o pacote de instalação com instalador para *Windows* (Figura 79).

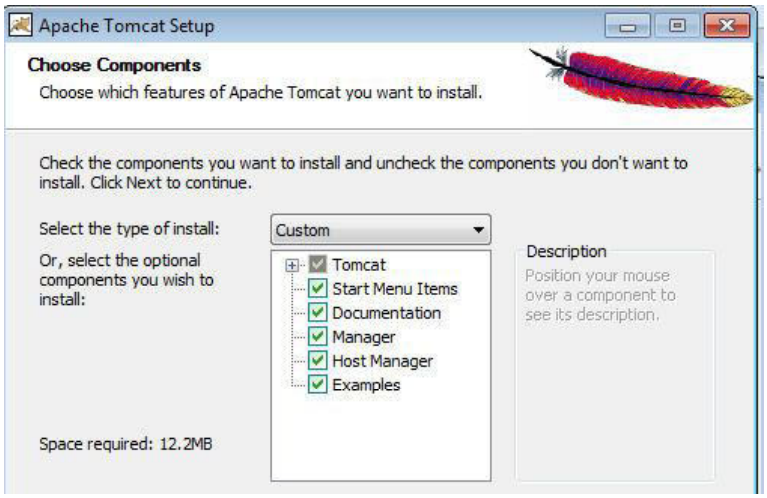


Figura 79 - Componentes a instalar.

Fonte: Elaboração própria

Podem então definir-se alguns parâmetros de configuração, como algumas portas, o nome do serviço, bem como as credenciais de administrador (Figura 80).

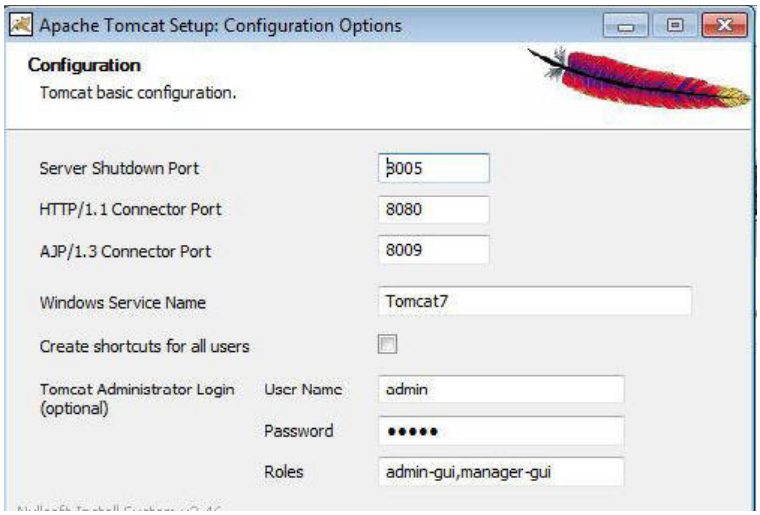


Figura 80 - Configurações básicas do *Tomcat*.

Fonte: Elaboração própria

Estando completo o processo de instalação o serviço *Tomcat* é iniciado Tendo este processo terminado definem-se as configurações para associar o *Apache Tomcat* 7 ao *IIS* 7.
Acedendo ao gestor de serviços de informação e *Internet* do *IIS*, pode aceder-se à definição de página, por omissão, através da opção enlaces (Figura 81).

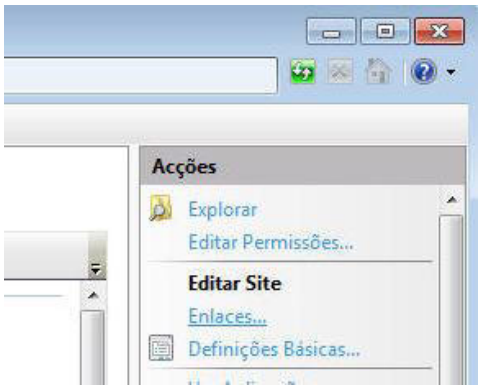


Figura 81 - Configurações por omissão da página inicial do IIS.

Fonte: Elaboração própria

Podem então definir-se as portas associadas ao *site*. Selecionando a linha pode editar-se a porta através do botão editar e neste caso mudar a porta existente para 82 (Figura 82).

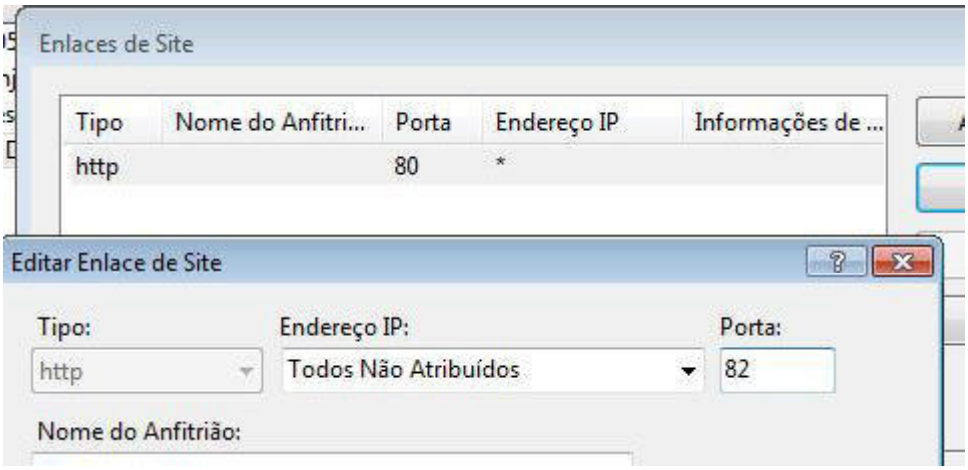


Figura 82 - Mudança de porta associada à página por omissão.

Fonte: Elaboração própria

Uma outra opção, no caso de não se querer definir-se uma porta, passa para a porta 80 por omissão. Neste caso, adicionou-se um novo *site* com o nome "*TomcatSite*" e ao mesmo associa-se a pasta "*webapps"* do *Tomcat*.

Após isto abriu-se a pasta *webapps* e na pasta da mesma copiou-se a pasta da plataforma JSP (Figura 83).

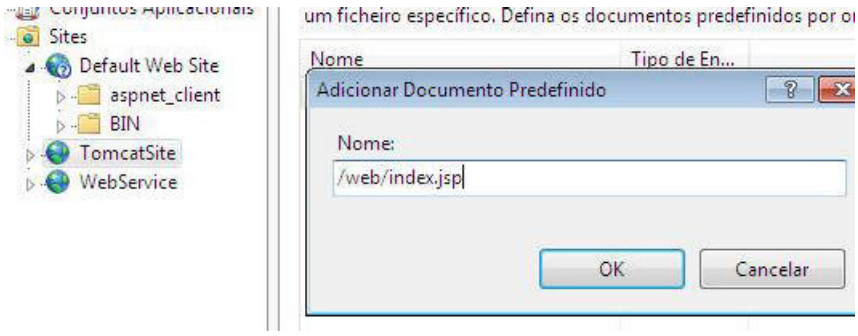


Figura 83 - Definição de página por omissão.

Fonte: Elaboração própria

Após este processo há ainda a necessidade de instalar um conector que ligue o *IIS* ao *Tomcat*. A escolha recai sobre o conector da *Riaforge* (IIS to Tomcat Connector , 2014) pela simples instalação e configuração (*Figura 84*).

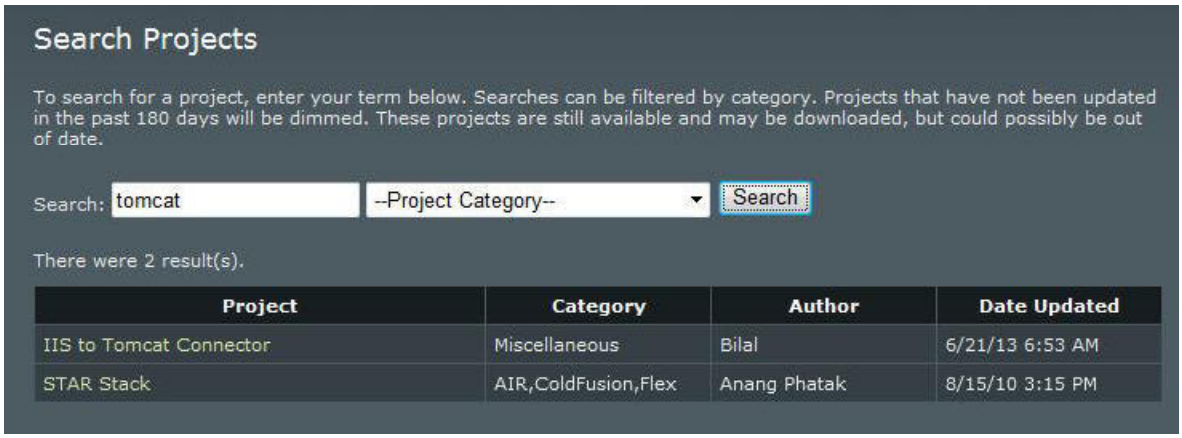


Figura 84 - Pagina de *downloads* de conector.

Fonte: Elaboração própria

Depois do descarregamento e o início de instalação é necessário proceder a configurações de instalação, começando pelo servidor e a porta associada ao mesmo. Neste caso mantém-se as definições por omissão como se pode verificar na Figura 85.

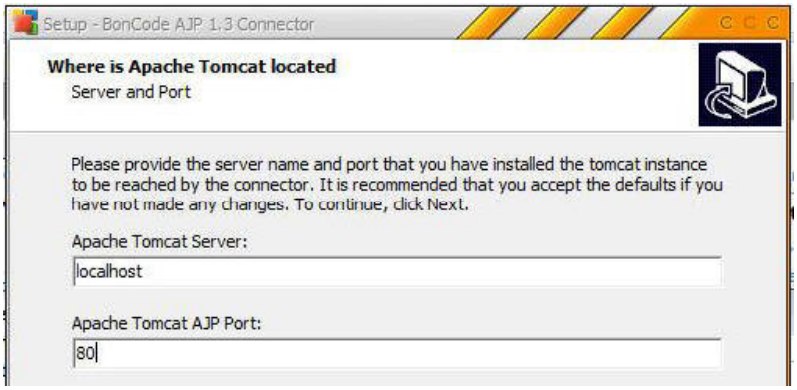


Figura 85 - Definição de parâmetros de servidor e porta.

Fonte: Elaboração própria

É dada a opção de alterar algumas pré-configurações. Contudo é aconselhado pelo produtor de *software* que se deixem estas configurações como se encontram, por omissão (Figura 86).

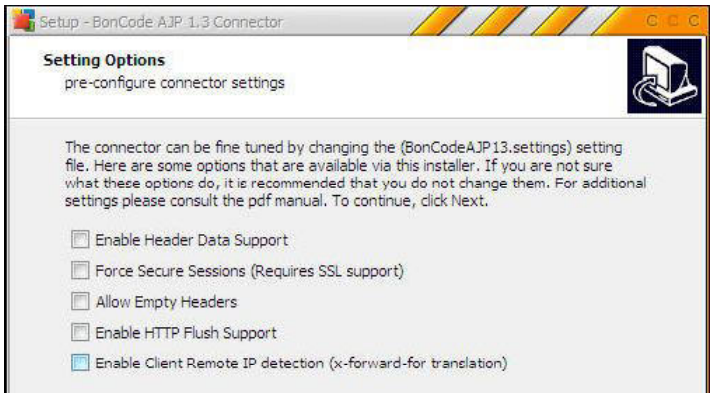


Figura 86 - Pré-configurações de conector.

Fonte: Elaboração própria

Depois passa-se à definição opções de *sites* e define-se a utilização de *sites* definidos pelo utilizador como se pode ver na Figura 87.

Figura 87 - Configuração para utilização de páginas personalizadas.

Fonte: Elaboração própria

Em seguida definem-se os tipos de pedidos de conexão de forma a serem encaminhados para o *Apache Tomcat* (Figura 88).

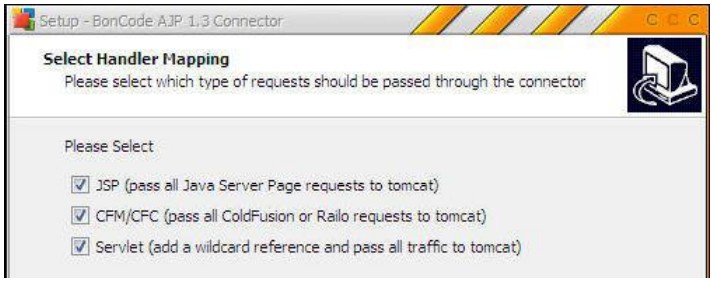


Figura 88 - Definição de pedidos a serem passados ao *Tomcat*.

Fonte: Elaboração própria

Na parte final pode ativar-se a configuração de diretórios (Figura 89).

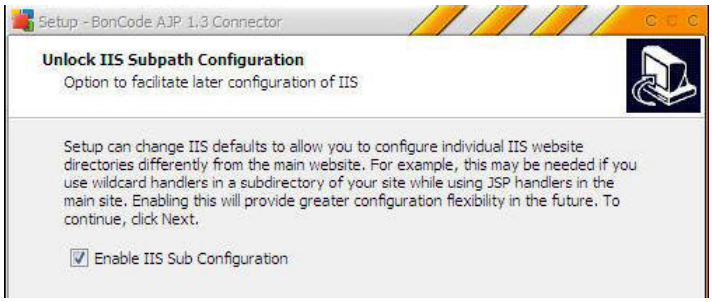


Figura 89 - Subconfiguração de *IIS*.

Fonte: Elaboração própria

Para testar se o conector está a funcionar basta fazer um acesso ao *localhost* e se a página local do *Tomcat* é mostrada, o processo culminado com sucesso, como se pode ver na Figura 90.

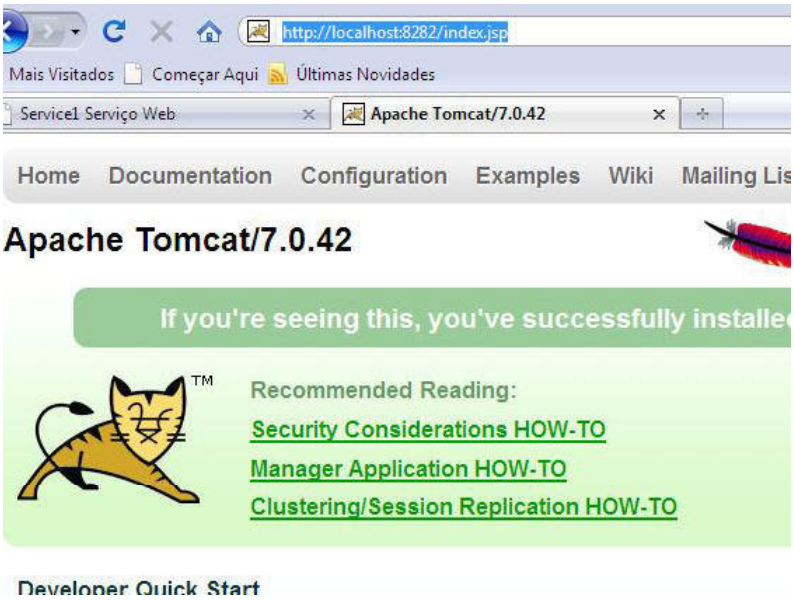


Figura 90 - Acesso a *localhost* associado a *Tomcat* na porta 8282.

Fonte: Elaboração própria

Salienta-se contudo, que o resultado, obtido após testes, é positivo. Pois a mudança da porta 8282, como se pode ver na Figura 90, para a porta 80, na Figura 91, foi bem-sucedida.

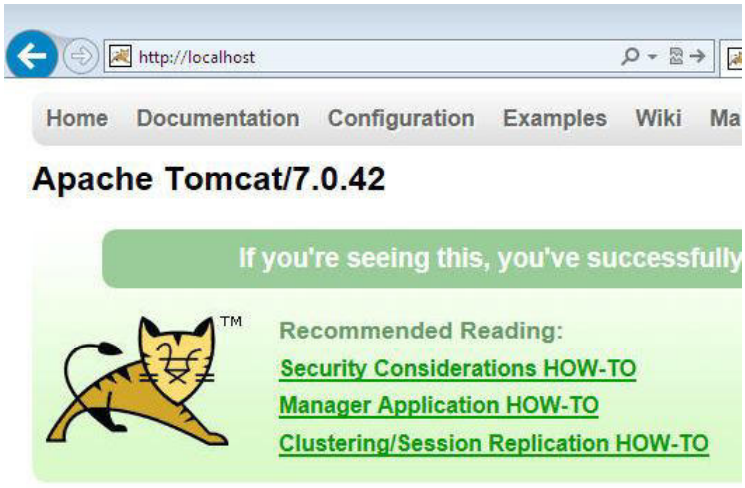


Figura 91 - Acesso a *localhost* associado ao *Tomcat* na porta 80.

Fonte: Elaboração própria

Finalmente, testou-se a página JSP inicial da plataforma do projeto e verificou-se que a mesma está acessível (Figura 92).

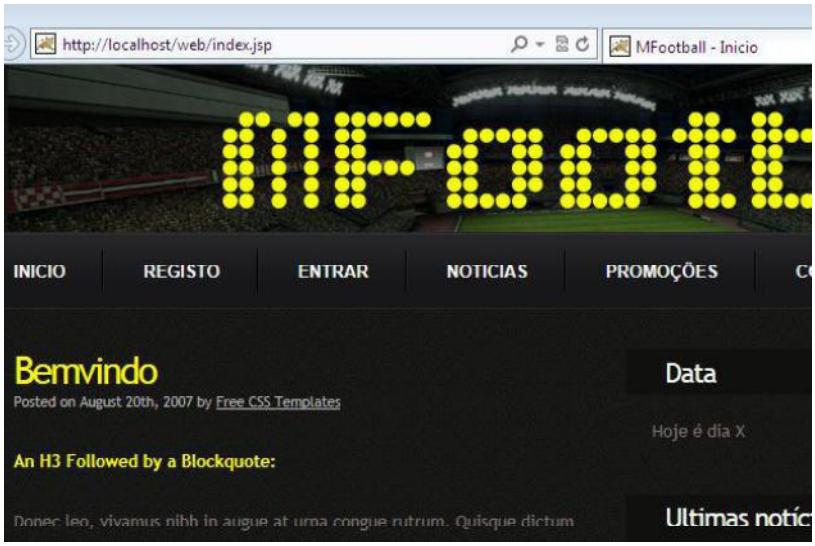


Figura 92 - Acesso a página JSP.

Fonte: Elaboração própria

3.10. *Web Service*

Tendo em conta o *Web Service* é necessária a instalação das funcionalidades de ASP/ASP .NET em programas e funcionalidades (Figura 93).

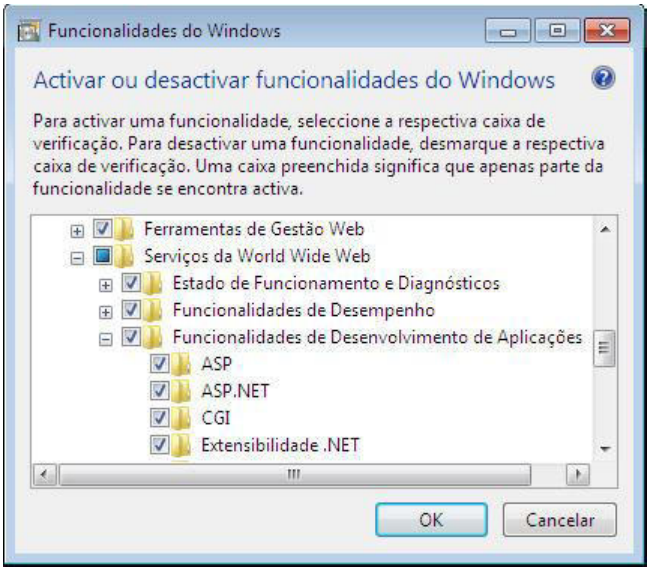


Figura 93 - Ativação de funcionalidades ASP/ASP.NET.

Fonte: Elaboração própria

No Gestor de Serviço de Informação e Internet clica-se com o botão direito sobre *Sites* para selecionar "Adicionar Web Site" (Figura 94)

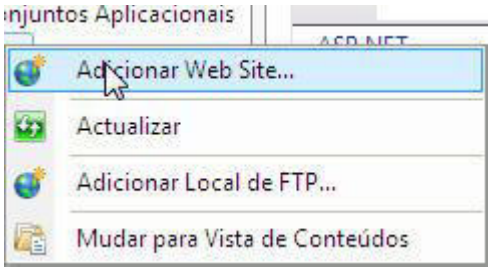


Figura 94 - Opção de adição de *Web Site*.

Fonte: Elaboração própria

Deu-se o nome de *Web Service*, definiu-se a pasta onde se encontra o mesmo e a porta de acesso (Figura 95).

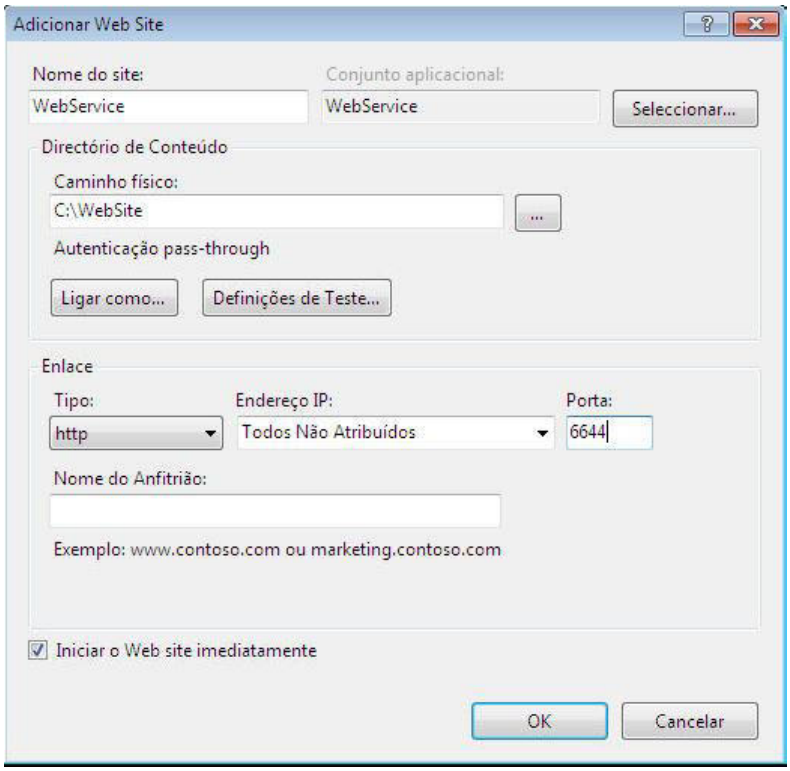


Figura 95 - Associação de pasta e porta de *Web Service.*

Fonte: Elaboração própria

Após isto, testou-se o acesso ao *Web Service* e pôde ver-se que o mesmo estava a funcionar e os seus métodos acessíveis (Figura 96).

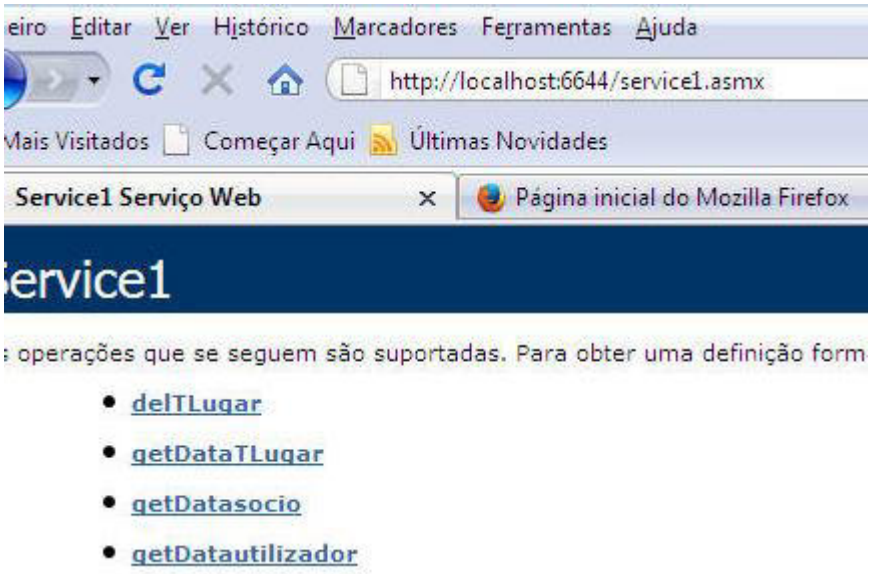


Figura 96 - Acesso a *Web Service*.

Fonte: Elaboração própria

3.11. Outras configurações

Estas configurações incluem configurações de portas de acesso associadas às páginas *Web*, ao *Web Service* e para o acesso o SGBD. Está também aqui documentada a criação de um domínio *online* que será associado ao projeto.

3.11.1. Portas de acesso

Tendo em conta o ambiente onde se situa o servidor *Web* houve a necessidade de configurar portas de acesso no *router* da rede onde o mesmo se situa. No projeto, o *router* em questão tem o endereço IP 192.168.1.1 e o servidor tem IP 192.168.1.14.

Tabela 15 - Tabela de configurações de portas.

Endereço IP entrada	Porta Entrada	Endereço IP saida	Porta Saida
192.168.1.1	80	192.168.1.14	80
192.168.1.14	80	192.168.1.1	80
192.168.1.1	6644	192.168.1.14	6644
192.168.1.14	6644	192.168.1.1	6644
192.168.1.1	3306	192.168.1.14	3306
192.168.1.14	3306	192.168.1.1	3306

Fonte: Elaboração própria

A porta 80 está destinada ao trafego HTTP para que a plataforma *Web* possa aceder-se do exterior, a porta 6644 destina-se ao acesso ao *Web Service* e a porta 3306.

3.11.2. Domínio *online*

Como dito anteriormente, devido ao ambiente onde se insere há a necessidade de permitir o acesso desde o exterior às páginas e tendo em conta este facto decidiu criar-se um domínio gratuito *online*. Para este fato optou-se pela sua associação ao domínio no-ip .org (noip.com, 2014).

Começou-se por aceder ao *site* e criar um perfil, depois é-nos possível gerir e adicionar *hosts* associados. Ao clicar em "*Add a Host"* como se pode ver nas opções apresentadas na Figura 97.

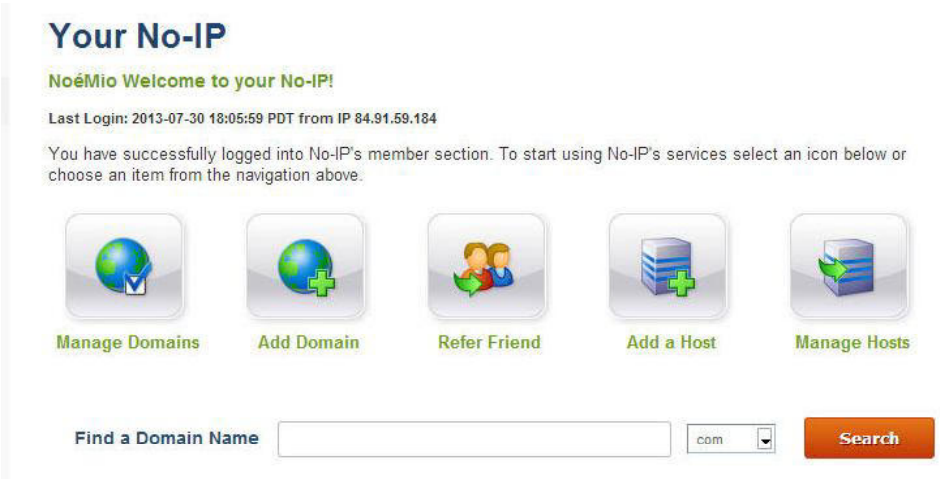


Figura 97 - Painel de entrada da página *no-ip.*

Fonte: Elaboração própria

No painel seguinte podem adicionar-se os dados do *host* , escolher o nome a adicionar e o seu domínio. É também necessário associar um grupo (Figura 98).

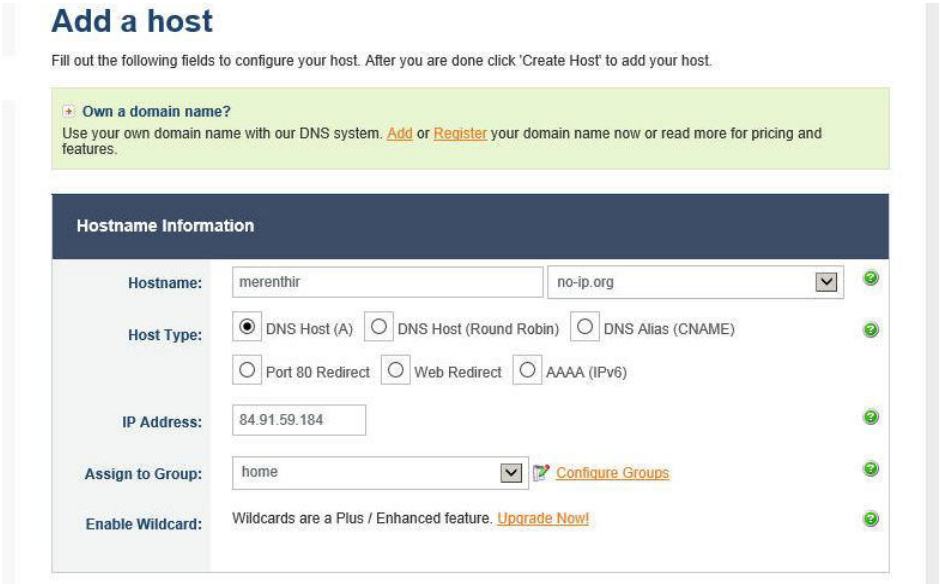


Figura 98 - Adicionar *host.*

Fonte: Elaboração própria

Após a adição de *host* é apresentado o *host* inserido, com a hipótese de uma nova inserção se necessário (Figura 99).

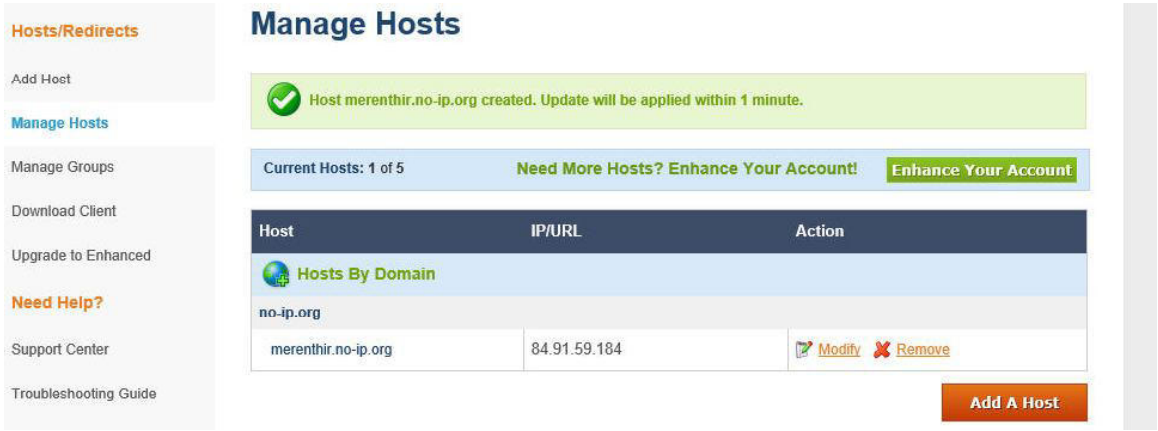


Figura 99 - *Host* inserido com sucesso.

Fonte: Elaboração própria

Tendo isto feito passou-se à instalação do cliente (Dynamic DNS Update Client (DUC) for Windows, 2014) que faz a gestão entre o nome da máquina e o endereço IP que normalmente é dinâmico (Figura 100).

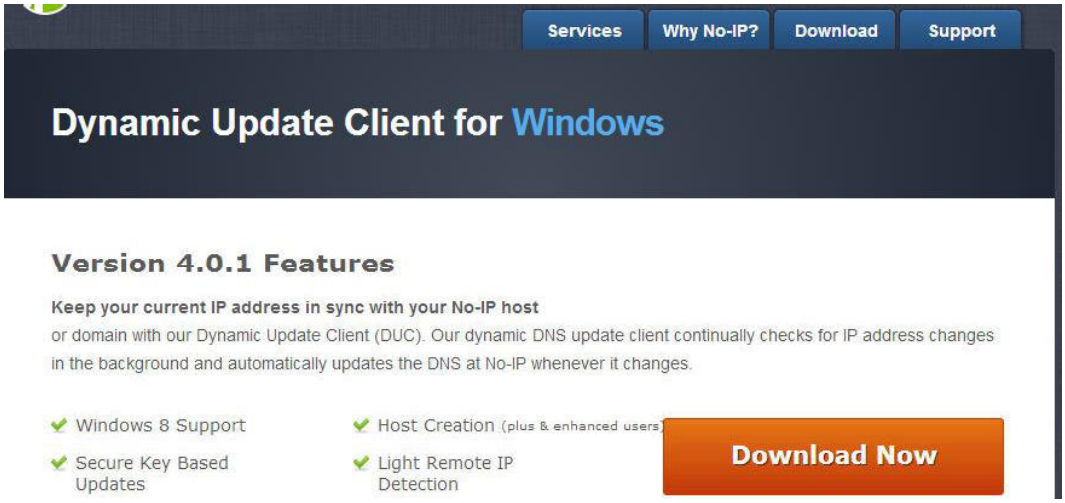


Figura 100 - Página para *download* de cliente.

Fonte: Elaboração própria

Após a instalação o cliente pede ao utilizador para proceder à configuração do mesmo e à sua associação ao servidor (Figura 101).

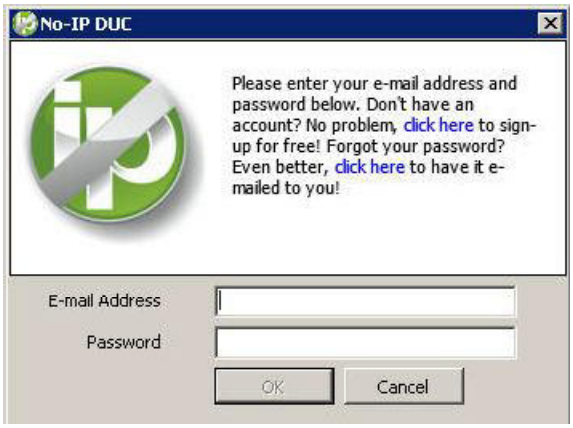


Figura 101 - Entrada para cliente.

Fonte: Elaboração própria

Seguidamente, o cliente abre e mostra o estado de conexão, o *host* associado e se o IP está associado ou não. Neste caso, ainda não se encontra nenhum *host* nem nenhum grupo e como tal o endereço IP não está todavia associado ao nome (Figura 102).

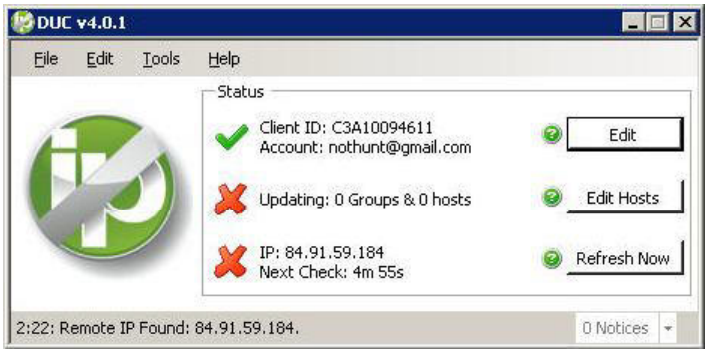


Figura 102- Software cliente sem associação de *host.*

Fonte: Elaboração própria

Ao premir em *Edit* podem ver-se os grupos e *hosts* disponíveis para associar ao processo que queremos concluir. Neste caso, como o grupo detém apenas o *host* que queremos, seleciona-se o grupo e guarda-se a definição (Figura 103).

Figura 103 - Gestão de grupos *e hosts.*

Fonte: Elaboração própria

Após está configuração, os elementos de configuração, na Figura 104, passam a estar a verde mostrando que se encontra a funcionar.

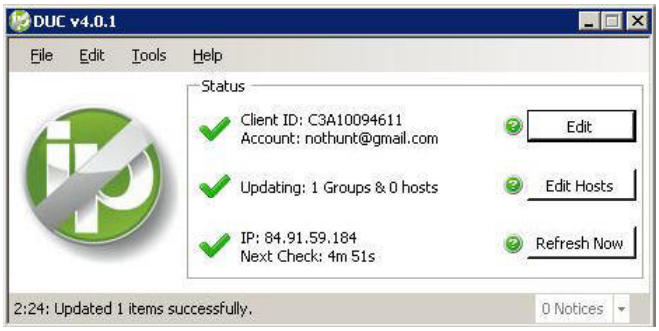


Figura 104 - Cliente conectado.

Fonte: Elaboração própria

Decidiu-se, então testar o acesso ao *Web Service* (Figura 105) para testar a conectividade.

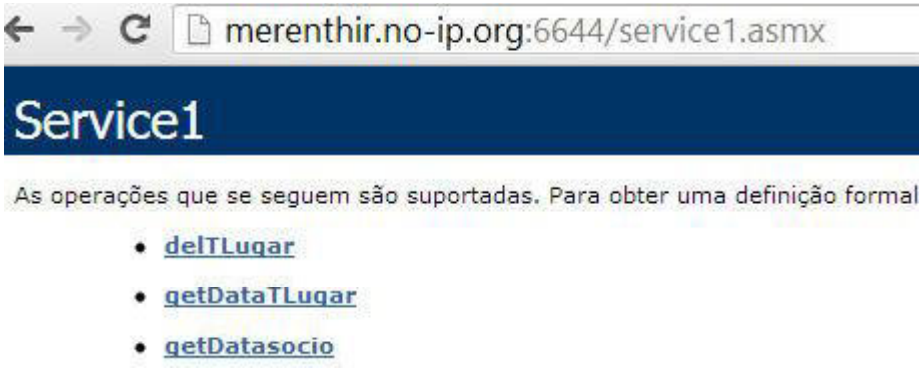


Figura 105 - Teste de acesso ao *Web Service* no domínio.

Fonte: Elaboração própria

4. Testes finais

Após os testes efetuados em separado de cada componente procedeu-se à sua junção dos componentes como já demonstrado e prepararam-se aos testes finais, de forma a ter uma ideia de quão funcionais ficaram os protótipos. Iniciou-se o processo com a escrita no cartão do número de identificação 11988638 no cartão T5557. Em baixo, podem ver-se os dados a serem escritos no cartão (Figura 106).

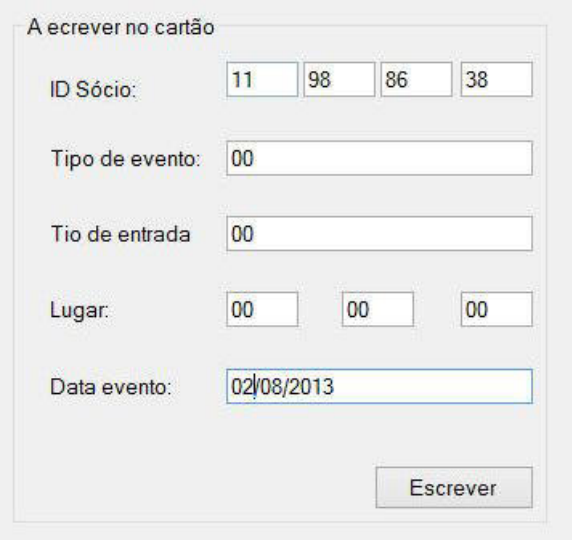


Figura 106 - Dados a inserir no cartão.

Fonte: Elaboração própria

Depois procedeu-se à passagem do cartão pelo módulo de estádio e pelo acompanhamento da informação no módulo de comunicação e o tratamento de informação em ambos. Para tal o *scketch* de cada Arduino foi programado para enviar para a porta série 0, informação que nos é pertinente. Inicialmente, testou-se o funcionamento no caso de o utilizador não ter entrada válida. Pôde acompanhar-se, através do *software TeraTerm*, o processamento da informação em cada um dos protótipos.

Iniciando-se no módulo de estádio, o mesmo deve devolver a frase "MEStadio Pronto" quando o *setup*() é carregado e fica a aguardar pela passagem do cartão RFID.

Após a passagem do cartão, deve ser mostrado "11988638" que é a identificação de utilizador existente no cartão.

Do protótipo de comunicação, é devolvida uma sequência de 4 zeros (“0000”), que deverá implicar que o LED vermelho acenda acompanhado de dois *bips* do *buzzer* (Figura 107).

Figura 107 - Receção de dados no protótipo de estádio.

Fonte: Elaboração própria

Do lado, do protótipo de comunicação o processo inicial é semelhante, sendo a primeira mensagem a indicação de que o mesmo está pronto e a aguardar.
De seguida, é mostrado o valor recebido sendo neste caso o valor pretendido “11988638”. Este valor aparece novamente ao ser inserido na *string* que define o valor a ser enviado para o *Web Service*.
Quando a função de envio de dados, via rede sem fios, é acedida, é mostrada a mensagem “dentro” querendo especificar que o processo de configuração do módulo WiFly RN-171 foi concluído e que o processo de transmissão irá iniciar-se.
De seguida, foi aplicado o comando “$$$” sendo que o módulo WiFly passou do modo de funcionamento automático ao modo de inserção de linhas de comando dai a resposta “CMD”. Logo de seguida foi introduzido o comando “*open*” que deu inicio à comunicação e depois recebeu-se a mensagem “enviou” (Figura 108).

Figura 108 - Resultado de receção de dados no protótipo de comunicação.

Fonte: Elaboração própria

Na receção de dados podem ver-se linhas que normalmente são interpretadas pelos navegadores de *Internet*, seguidas das linhas XML que nos são pertinentes.
As linhas são lidas até que a sequência “#*#$”. A cada caracter é anunciada a sua receção e depois da sequência encontrado o valor que será enviado ao protótipo de estádio. Neste caso o resultado é ‘0’ . Este valor é tratado em *byte* e na tabela ASCII (*American Standard Code for Information*

Interchange) tem o valor 48. Encontrado o valor, o mesmo é enviado e acaba a monitorização desta operação (Figura 109).

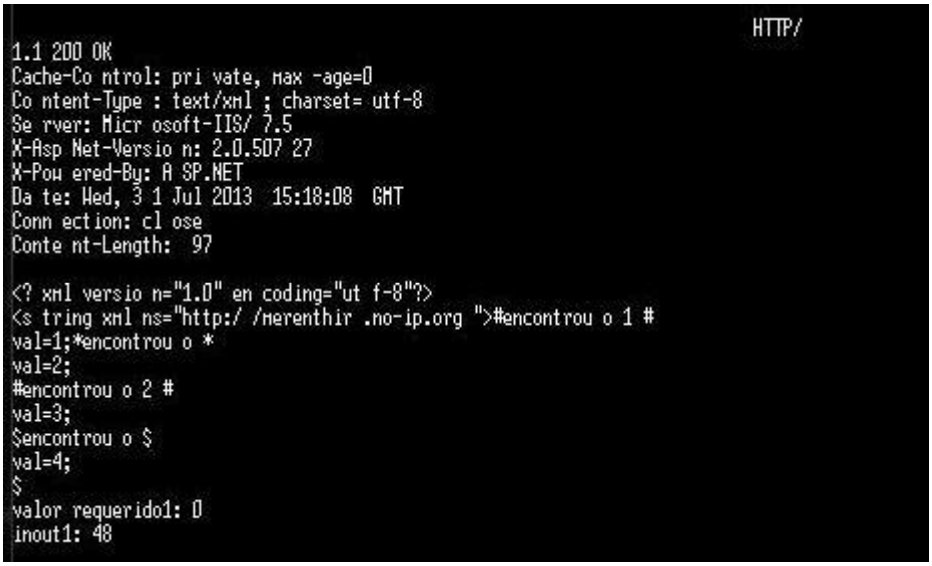


Figura 109 - Resposta de *Web Service*.

Fonte: Elaboração própria

Da mesma forma, procedeu-se a testes, sendo o bilhete previamente adquirido. Uma vez que, a inserção de dados de jogos não se encontra para já contemplada, o jogo foi inserido através do *Web Service*.

Acedendo ao painel de venda de bilhetes clicou-se em "*Ler dados*" e passou-se o cartão T5557 no leitor e a aplicação e esta devolve os dados do utilizador em questão.

Escolheu-se a data do jogo, o sector, a linha e a coluna do lugar e clicou-se em aplicar. A mensagem de êxito (Figura 110) é mostrada após o envio dos dados para a base de dados.

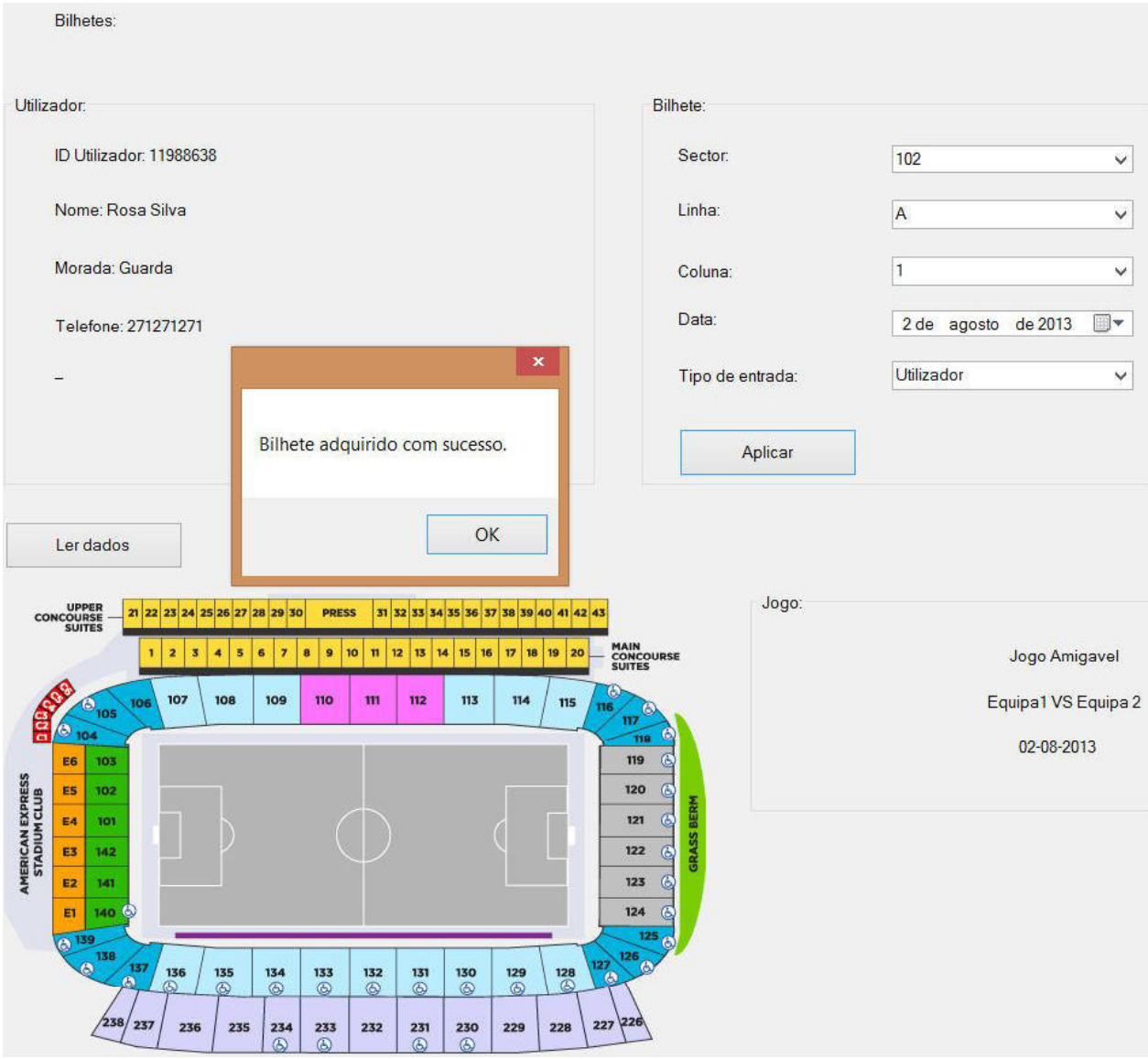


Figura 110 - Mensagem de sucesso na aquisição de bilhete.

Fonte: Elaboração própria

Da mesma forma, se testou a aquisição de um bilhete no mesmo lugar e a mensagem devolvida refere que o lugar pretendido já se encontra ocupado (Figura 111).

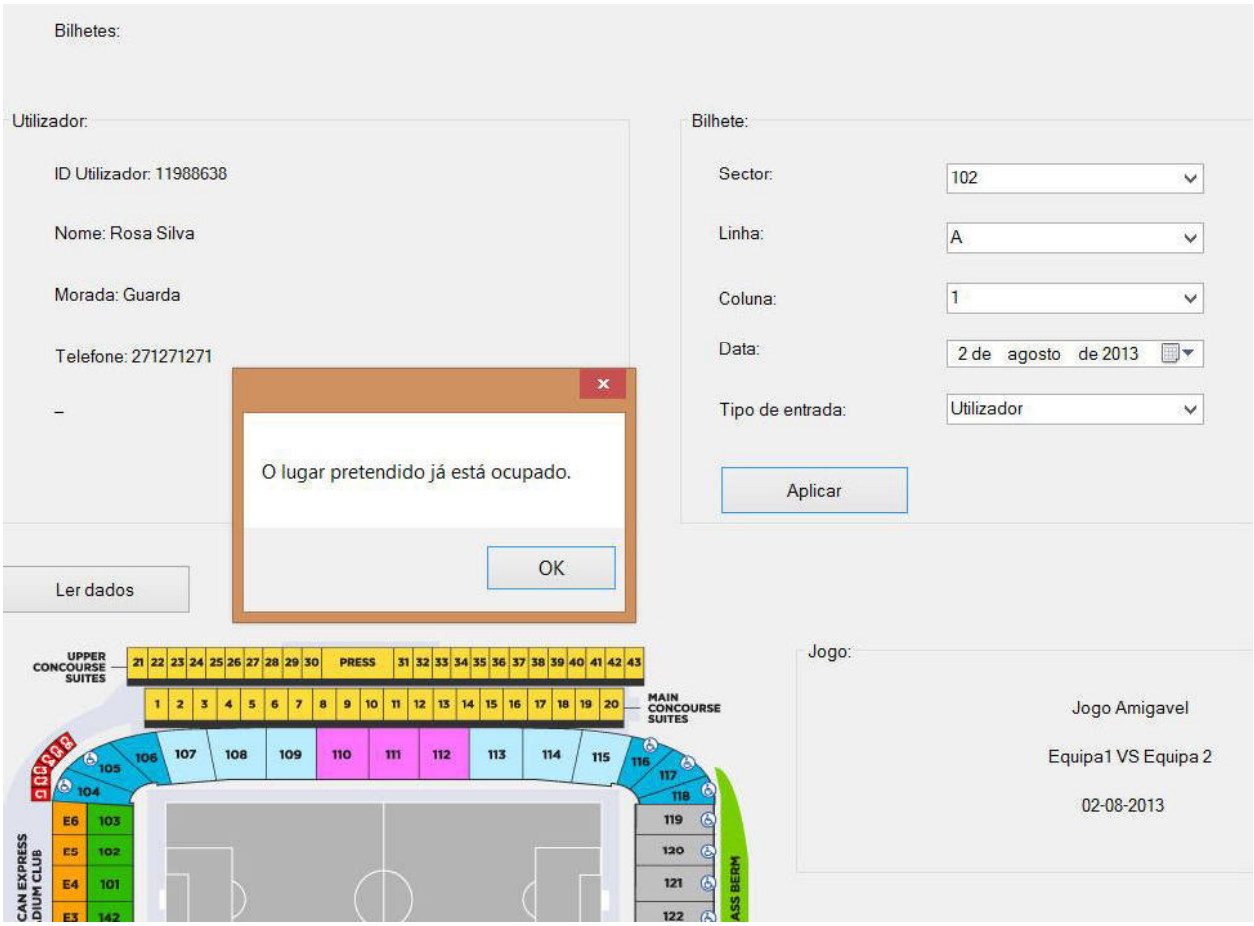


Figura 111 - Mensagem que dá a informação de ocupação de lugar.

Fonte: Elaboração própria

Embora não demonstrado aqui, a verificação de ocupação de lugar é testado para o utilizador para o lugar. Passou-se então ao teste com os protótipos físicos, passando cartão RFID no módulo de estádio e através do *TeraTerm* pôde ver-se no mesmo as mensagens que servem para perceber se os processos internos decorrem como planeado (Figura 112).

Figura 112 - Receção de mensagens de módulo de estádio.

Fonte: Elaboração própria

Como referido anteriormente, o módulo indica que está pronto após o carregamento. Após a passagem do cartão é mostrado o número de utilizador. Depois disto o módulo entra em modo de espera aguardando a resposta do protótipo de comunicação.

A resposta "0001" (Figura 112) indica que a entrada é válida e o LED verde acende. Basta então carregar no botão e o módulo passa ao modo normal de espera. No lado do protótipo de comunicação, a primeira parte é igual à apresentada anteriormente, diferindo no valor requerido que foi então enviado ao protótipo de estádio. Ou seja, no teste anterior recebeu-se o valor 0 e desta vez o valor 1 (Figura 113).

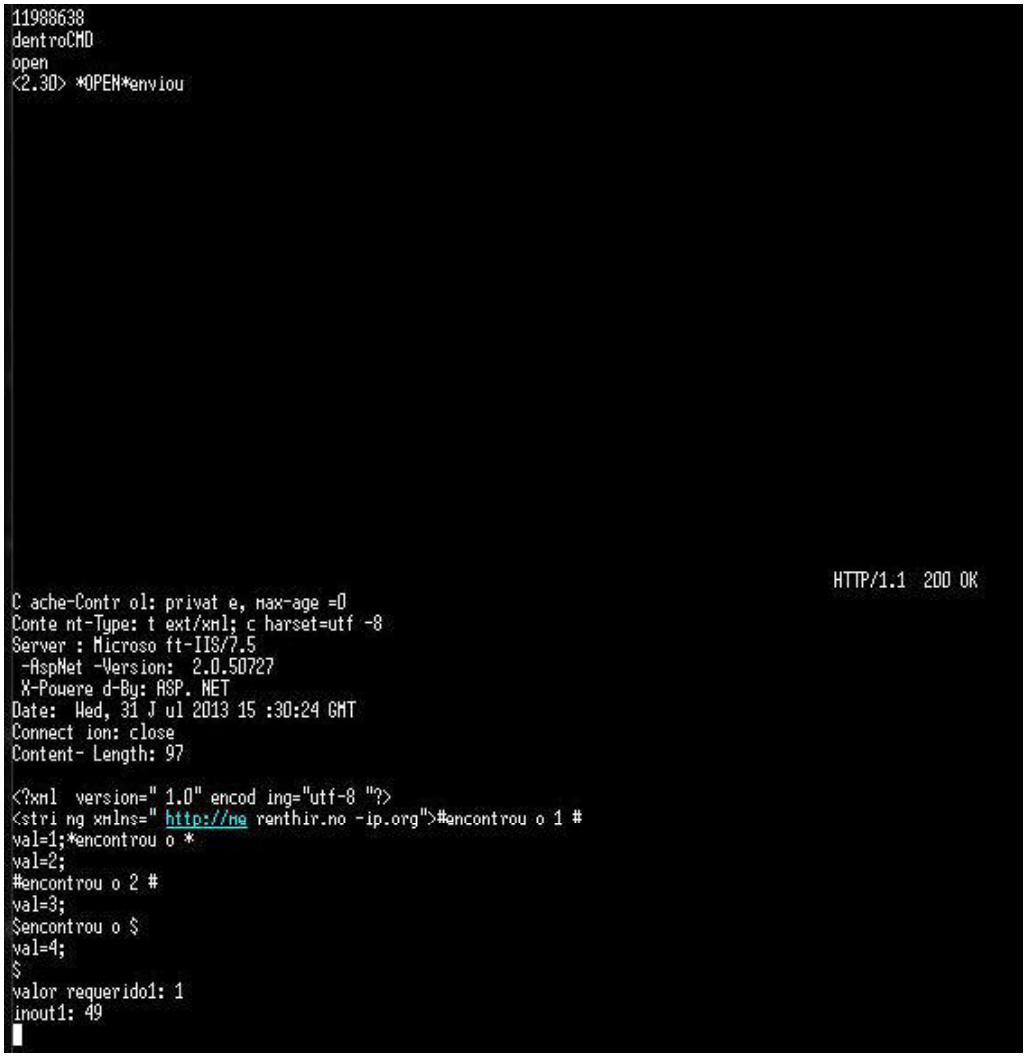


Figura 113 - Dados mostrados pelo prototipo de comunicação.

Fonte: Elaboração própria

Desta forma, como se constata na figura anterior, obtiveram-se resultados positivos. Contudo, por vezes o caso de um erro em que o protótipo de comunicação fica como que congelado à espera de dados vindo do *Web Service*, obrigando ao reinicio do módulo. Até a elaboração deste relatório, este problema está a ser estudado, à procura de uma solução.

5. Conclusão e trabalho futuro

Neste trabalho desenvolveu-se um sistema para a compra de bilhetes, integrando várias tecnologias de *hardware* e de *software*.

Após o desenvolvimento do projeto descrito pode concluir-se que encontrar novas formas de solucionar problemas pode levar o seu tempo, bem como ser um pouco desesperante quando nos encontramos com entraves que parecem elevar demasiado a fasquia.

Após este desabafo, pode dizer-se que o Arduino mostra ser uma plataforma que pode moldar-se a inúmeros cenários e, dentro dos mesmos, executar as mais diferentes funções. Quanto aos módulos utilizados nas diferentes montagens, pode concluir-se que:

- Os módulos XBee são versáteis e a relação qualidade/consumo é deveras apelativa para o desenvolvimento de soluções que necessitem redes sem fios para a transmissão de dados;
- O módulo RFID é de igual modo abrangente tendo também a vantagem de nos permitir moldar a informação que pode ser escrita e lida das etiquetas RFID;
- O módulo Wifly RN171 mostra bastante robustez nas conexões que executa em modo *gateway*. Todavia, parecem existir algumas questões no que toca à sua associação ao Arduino.

O desenvolvimento deste projeto mostrou-se um desafio, que confirma a necessidade de obtenção de novos conhecimentos e novas formas de pensar. Ainda assim, foi desenvolvido para dar um contributo nesta área, de forma a atingir objetivos como:

- Implementação não invasiva;
- A utilização de componentes de baixo custo.
- Capacidade de adaptação a estruturas pré-existentes.

Como trabalho futuro podem indicar-se os seguintes desenvolvimentos.

- Aumento de segurança: aplicação de metodologias e que permitam um aumento de segurança entre o *Web Service* e as suas aplicações;
- Ampliação da plataforma *Web*: aumento de funcionalidades bem como a adição de possibilidade de efetuar pagamentos através da mesma;

- Utilização de outras etiquetas: para além das etiquetas utilizadas a utilização de outras etiquetas poderá trazer uma maior expansão e a utilização de etiquetas RFID de papel, ou bilhetes de papel, pode aumentar receitas.

Finalmente, posso dizer que este desafio permitiu-me aprender imenso quer, sobre as tecnologias tratadas, quer das metodologias tratadas.

6. Bibliografia

(s.d.). Obtido de Make projects: http://makeprojects.com/Project/Build-your-own-Arduino-Controlled-Robot-/577/1

.NET C# Web Service/JSP Client. (janeiro de 2014). Obtido de Javaclaus' Blog: http://javaclaus.wordpress.com/2013/01/04/calling-a-net-c-web-service-with-a-javajsp-client/

Active Tag . (2014). Obtido de technovelgy.com: http://www.technovelgy.com/ct/Technology-Article.asp?ArtNum=21

AHSON, S. A. (2008). *RFID Handbook: Applications, Technology, Security, and Privacy.* CRC Press.

Arduino. (2014). Obtido de http://pt.wikipedia.org/wiki/Arduino: http://pt.wikipedia.org/wiki/Arduino

BANZI, M. (2008). *Getting Started with Arduino, Massimo Banzi.* O'Reilly Make Magazine.

BRENN, N. (janeiro de 2014). *http://makeprojects.com/Project/Build-your-own-Arduino-Controlled-Robot-/577/1*. Obtido de http://makezine.com/.

caspiquad. (2014). Obtido de https://code.google.com/p/caspiquad/: https://code.google.com/p/caspiquad/

CHEN, Y., HUANG, D., M., T., & JAN, J. (2011). A Design of Tamper Resistant Prescription RFID Access Control System. *Springer Science+Business Media, LLC*, 2795-2801.

Configuring XBees for API Mode. (janeiro de 2014). Obtido de http://www.instructables.com: http://www.instructables.com/id/Configuring-XBees-for-API-Mode/step3/XCTU-Reconfiguring/

Download MySQL Community Server. (janeiro de 2014). Obtido de MySQL the world's most popular open source database: http://dev.mysql.com/downloads/mysql/

Download Tera Term 4.85. (janeiro de 2014). Obtido de http://logmett.com: http://logmett.com/index.php?/download/tera-term-485-freeware.html

Download the Arduino Software. (janeiro de 2014). Obtido de http://arduino.cc/: http://arduino.cc/en/Main/Software

Dynamic DNS Update Client (DUC) for Windows. (janeiro de 2014). Obtido de http://www.noip.com: http://www.noip.com/downloads.php?page=win

functional home energy monitor. (janeiro de 2014). Obtido de http://openenergymonitor.blogspot.com: http://openenergymonitor.blogspot.com/2010/01/finished-working-functional-home-energy.html

GISLASON, D. (2008). *Zigbee Wireless Networking.* Newnes.

Gomba, D. (26 de julho de 2010). *THE POOL – A JEN LEWIN INTERACTIVE INSTALLATION*. Obtido de http://blog.arduino.cc: http://blog.arduino.cc/2010/07/26/the-pool-a-jen-lewin-interactive-installation/

HARTMANN, B. (2014). *C60 – Evolution of an Idea*. Obtido de http://labs.ideo.com/2011/01/14/c60-evolution-of-an-idea/: http://labs.ideo.com/2011/01/14/c60-evolution-of-an-idea/

How To Use X-CTU. (janeiro de 2014). Obtido de https://sites.google.com/site/xbeetutorial: https://sites.google.com/site/xbeetutorial/xctu

HUNT, V. D. (2010). *RFID: A Guide to Radio Frequency Identification.* Wiley-Interscience.

IIS to Tomcat Connector . (janeiro de 2014). Obtido de http://tomcatiis.riaforge.org/: http://tomcatiis.riaforge.org/

KARVINEN, T., & KARVINEN, K. (2011). *Make: Arduino Bots and Gadgets Six Embedded Projects with Open Source Hardware and Software.* Maker Media, Inc.

Larry, L. (2014). *Arduino – Modifying a Robot Arm*. Obtido de http://luckylarry.co.uk/arduino-projects/arduino-modifying-a-robot-arm/: http://luckylarry.co.uk/arduino-projects/arduino-modifying-a-robot-arm/

LEE, B. G. (2008). *Broadband Wireless Access & Local Networks: Mobile Wimax and Wifi.* Artech House Publishers.

MARGOLIS, M. (2011). *Arduino Cookbook.* O'Reilly Media Inc.

Meet Jen Lewin and "The Pool". (janeiro de 2014). Obtido de http://www.sparkfun.com/: http://www.sparkfun.com/news/392

Muñoz, J. (2014). *ArduBlimp (minimum Blimp) V1*. Obtido de http://diydrones.com/profiles/blogs/705844:BlogPost:27481: http://diydrones.com/profiles/blogs/705844:BlogPost:27481

nightvision. (janeiro de 2014). Obtido de http://www.freecsstemplates.org/: http://www.freecsstemplates.org/preview/nightvision/

noip.com. (janeiro de 2014). Obtido de http://www.noip.com/: http://www.noip.com/

OXER, J., & BLEMMINGS, H. (2009). *Practical Arduino: Cool Projects for Open Source Hardware.* aPress.

Passive RFID Tag (or Passive Tag). (janeiro de 2014). Obtido de technovelgy.com: http://www.technovelgy.com/ct/Technology-Article.asp?ArtNum=47

PowerDesigner . (janeiro de 2014). Obtido de Sybase: http://www.sybase.com/detail?id=1095981

RCvertt. (2014). *Quaduino-Part2 (Arduino QuadCopters)*. Obtido de http://www.rcgroups.com: http://www.rcgroups.com/forums/showthread.php?t=1066119

RFID 125kHz Module for Arduino Tutorial. (janeiro de 2014). Obtido de http://www.cooking-hacks.com/: http://www.cooking-hacks.com/index.php/documentation/tutorials/arduino-rfid

Royer, D. (janeiro de 2014). *SPIDEE-1*. Obtido de https://www.youtube.com/watch?v=J3ySNng9vsg.

Samano-Robles R., G. A. (2009). Integration of RFID Readers into Wireless Mobile. *Wireless VITAE'09*, p. 5.

SAMANO-ROBLES, R., & A., G. (2009). Integration of RFID Readers into Wireless Mobile Telecommunication Networks. *Wireless VITAE'09*, 327-331.

Sample JSP Implementation. (janeiro de 2014). Obtido de http://captchas.net/: http://captchas.net/sample/jsp/

SEN, P., SEN, D., & M., D. A. (2009). *RFID for Energy & Utility Industries.* PennWell Corp.

SoftwareSerial documentation could mention AltSoftSerial library. (janeiro de 2014). Obtido de Arduino: https://code.google.com/p/arduino/issues/detail?id=1111

Spacebits. (2014). Obtido de http://blog.spacebits.eu/: http://blog.spacebits.eu/

Stadium Ticketing And Access Control. (2014). Obtido de http://www.otot.ws/: http://www.otot.ws/en/systems/stadium-ticketing-and-access-control

TACTICAL TEXTING IN PUBLIC SPACES. (janeiro de 2014). Obtido de http://www.brokencitylab.org: http://www.brokencitylab.org/blog/arduino-php-lcds-xbees-tactical-texting-in-public-spaces/

The open source quadcopter / multicopter. (2014). Obtido de http://aeroquad.com/content.php: http://aeroquad.com/content.php

TITUS, J. A. (2012). *The Hands-on XBee Lab Manual: Experiments that Teach you XBee Wirelesss Communications.* ELSEVIER SCIENCE TECHNOLOGY.

Tomcat 7. (janeiro de 2014). Obtido de Apache Tomcat: http://tomcat.apache.org/download-70.cgi

Transponder. (janeiro de 2014). Obtido de Wikipedia: http://en.wikipedia.org/wiki/Transponder

TURCU, C., & C., M. S. (2011). *Current Trends and Challenges in RFID.* Romania: University of Suceava.

XBee S1 vs. XBee S2. (janeiro de 2014). Obtido de http://www.digi.com: http://www.digi.com/support/kbase/kbaseresultdetl?id=2213

XCTU Software. (janeiro de 2014). Obtido de http://www.digi.com: http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities

YAN, B., & LEE, D. (2009). Design of Sight Spot Ticket Management System Based on RFID . *International Conference on Networks Security, Wireless Communications and Trusted Computing*, 496-499.