Università degli Studi di Modena e Reggio Emilia

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Informatica

Tesi di Laurea

Progetto e Svilluppo di un Sistema per la Gestione e il Tracciamento di Dispositivi GPS

Candidato:
Andrea Selmini

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2011/2012

Ringrazio i miei familiari
e la mia fidanzata per il sostegno ricevuto.
Ringrazio inoltre i miei colleghi
della ditta Infomobility s.r.l. per l'aiuto
nello sviluppo di questo progetto.
Infine ringrazio il prof. Riccardo Martoglia
per la disponibilità e l'assistenza fornitami
per la compilazione di questa tesi.

Indice

Elenco delle figure

Elenco dei codici

Introduzione

In molti ambiti vi è il bisogno di possedere un sistema che permetta di rilevare la pozione di veicoli.
Alcuni esempi di imprese che si avvantaggiano di un sistema simile sono le imprese di spedizioni e di trasporti per rilevare se i propri autisti seguono i percorsi affidati, o nel caso di imprese di noleggio di autoveicoli per assicurarsi che i veicoli vengano restituiti secondo i termini pattuiti, o nel caso di imprese che fanno affidamento a macchine operatrici per rilevare se i propri operai stanno lavorando nei luoghi indicatigli, o nel caso di lavori in appalto per assicurarsi che la ditta appaltatrice segua i termini dell'accordo, oppure nel caso delle forze di polizia per conoscere la pozione di determinate persone. Un sistema simile inoltre risulta utilizzato anche come sistema di antifurto in modo da poter rintracciare un veicolo rubato.
L'utilizzo non è limitato ai soli autoveicoli, ma viene utilizzato anche in tutti i tipi di veicoli come navi, aerei ed elicotteri.

Per rendere possibile un'infrastruttura simile, si rendono necessarie diverse componenti: un dispositivo con funzioni di rilevamento gps, un sistema in grado di gestire la comunicazione con simili dispositivi e un sistema per permettere a un utente di gestire l'infrastruttura e avere accesso alle informazioni elaborate.

Nell'ambito di questo progetto il sistema che gestisce la comunicazione con i dispositivi è stato sviluppato come un'applicazione server con un metodo di comunicazione master/slave; invece il sistema per gestire l'infrastruttura e avere accesso alle informazioni è stato progettato come un sito web.

La struttura di questa tesi è organizzata in tre parti.
Nella prima parte, divisa in cinque capitoli, verranno presentati i linguaggi di programmazione utilizzati per lo sviluppo del progetto. La prima sezione tratta del C# utilizzato per l'applicazione server e per il codice lato server del sito web, la seconda tratta dell'Asp.NET utilizzato per la creazione del sito web, la terza e la quarta descrivono il Javascripe e il jQuery utilizzati per il codice lato client del sito web, infine la quinta sezione tratta del MySql

utilizzato per lo sviluppo del database.
Nella seconda parte verrà discusso il percorso che ha portato alla progettazione dell'intero sistema.
Nella terza parte verrà mostrata la parte implementativa del progetto, descrivendo le varie funzioni e mostrando porzioni di codice.

Parte I

Il Caso di Studio

Capitolo 1

Visual C#

1.1 Introduzione

Visual C# è un'implementazione del C# per Visual Studio sviluppata dalla Microsoft. E' basato sulle specifiche del C# e il termine Visual, presente anche negli altri linguaggi disponibili in Visual Studio, indica l'utilizzo di un IDE grafico per lo sviluppo delle applicazioni. Visual C# viene anche utilizzato da parte di ASP.NET per il codice lato server.

C# è un linguaggio di programmazione a oggetti sviluppato dalla Microsoft e standardizzato dall'ECMA nel 2001 e dall'ISO nel 2003. La versione più recente è C# 4.0 rilasciata il 12 Aprile 2010.
Il nome è stato ispirato dalla notazione musicale dove il simbolo ♯ indica che una nota deve essere di un semitono più alta, a causa delle limitazioni dei font e dal fatto che il vero e proprio simbolo ♯ utilizzato dalla notazione musicale non è presente sulle tastiere, è stato scelto di utilizzare il simbolo # durante i normali testi, ma dove è possibile Microsoft utilizza il simbolo musicale. Il suffisso # inoltre è comune anche ad altri linguaggi .NET che sono una variazione di altri linguaggi esistenti, alcuni esempi sono J# (derivato dal Java), A# (derivato da Ada) e F#, inoltre è utilizzato anche in alcune librerie come GTK# (un'implementazione di GTK+), Cocoa# (un'implementazione di Cocoa) e Qt#(una libreria per Qt Toolkit).
Il C# è il linguaggio che descrive meglio di tutti il CLI(Common Language Infrastructure), in quanto sviluppato da Microsoft proprio per la programmazione con il framework .NET, infatti i suoi tipo di dato primitivi hanno una corrispondenza univoca con i tipi implementati nel CLI.

Possiede elementi in comune con il C++ e una sintassi molto simile al Java ma per alcune caratteristiche si differenzia dai linguaggi precedenti.
Rispetto al C++ vi sono sostanziali differenze:

- i puntatori possono essere utilizzati solo in alcune porzioni di codice marcati come unsafe
- durante le operazioni aritmetiche vengono controllati automaticamente gli errori di overflow
- non possiede variabili o funzioni globali, per sostituirle si utilizzano variabili o funzioni statiche
- non è possibile deallocare esplicitamente della memoria, questo compito è dedicato ad un garbage collector
- non è possibile ereditare da più classi, ma è possibile implementare un numero indefinito di interfacce
- le uniche conversioni implicite permesse sono quelle dove non vi è il rischio di una perdita di dati, per tutte le altre si deve ricorrere a conversioni esplicite

Rispetto al Java inoltre, all'interno del C# tutto è veramente un oggetto.

1.2 Sintassi

C# possiede un sistema di tipi unificato chiamato Common Type System(CTS). Questo comporta che tutti i tipi, compresi quelli primitivi come gli integer, sono una sottoclasse di *SYstem.Object* e condividono svariati metodi.
Secondo il CTS i tipi sono suddivisi in due categorie: Valori e Riferimenti. I Valori non possiedono alcuna identità di riferimento e derivano da *System.ValueTypes*, devono avere sempre un valore di default, possono essere creati e copiati in ogni momento, non possono derivare tra di loro ma possono implementare interfacce e non possono avere un costruttore senza parametri. Alcuni esempi di Valori sono tutti i tipi primitivi, gli *enum* e le *struct*. I Riferimenti al contrario possiedono un'identità di riferimento, quindi ogni istanza di un'oggetto è differente dalle altre istanze. In generale non è possibile creare un'istanza dei Riferimenti né copiare un'istanza esistente, ma essi dispongono di diversi metodi per fornire tali funzioni esponendo un costruttore pubblico o implementando le interfacce necessarie. Alcuni esempi di Riferimenti sono le *String* e gli *Array*.

C# ha una sintassi simile al C++ e possiede tutte le sue forme di iterazione. Inoltre permette l'uso del costrutto *foreach*, che simile al *for* utilizza un enumeratore dell'oggetto per le iterazioni.
Le varie sezioni di codice vanno contenute all'interno di *namespace* (l'equivalente dei package del java), ed è possibile richiamare determinati namespace

tramite il comando *using*; le funzioni basilari sono contenute all'interno del namespace *System*.
Gli array derivano dall'oggetto base *Array* e, a differenza del C++, sono gestiti dinamicamente, inoltre possono essere multi-dimensionali.
E' possibile creare delle strutture tramite il comando *struct* e hanno molti punti in comune con le classi, sebbene abbiano alcune limitazioni:

- i campi non possono essere inizializzati a meno che non siano dichiarati *const* o *static*,
- non può avere costruttore o distruttori senza parametri,
- sono tipi di valore mentre le classi sono tipi di riferimento,
- è possibile creare nuove istanze senza usare la parola chiave new
- non può ereditare da altre struct o classi
- è possibile assegnare un valore null a una struttura che può essere utilizzata come tipo nullable.

Oltre al costrutto try...catch possiede il costrutto try...finally per eseguire determinate porzioni di codice anche in caso di errore, permette di intercettare diverse condizioni di errore tramite la classe *Exception* ed è possibile implementarne di nuove derivando dalla classe *Exception* e utilizzando la parola chiave *throw*.

Codice 1.1: Exception

```
throw new NotImplementedException();

try
{
    // Codice che potrebbe innescare l'eccezzione
    ...
}
catch (Exception ex)
{
    // Eccezzione catturata e gestita
    ...
}
finally
{
    // Codice eseguito in ogni caso dopo i blocchi try/catch
    ...
}
```

1.3 Classi

C# possiede diverse classi al suo interno adatte a semplificare la scrittura di programmi.

List

La classe *List* rappresenta l'equivalente generico della classe ArrayList e permette di creare un elenco di oggetti fortemente tipizzato e accessibile per indice, fornisce diverti metodi per la ricerca,l'insirimento, l'ordinamento e la modifica degli elenchi.

Codice 1.2: Esempio List

```
using System;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        List<string> dinosaurs = new List<string>();

        Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);

        dinosaurs.Add("Tyrannosaurus");
        dinosaurs.Add("Amargasaurus");
        dinosaurs.Add("Mamenchisaurus");
        dinosaurs.Add("Deinonychus");
        dinosaurs.Add("Compsognathus");

        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }

        Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);
        Console.WriteLine("Count: {0}", dinosaurs.Count);

        Console.WriteLine("\nContains(\"Deinonychus\"): {0}",
            dinosaurs.Contains("Deinonychus"));

        Console.WriteLine("\nInsert(2, \"Compsognathus\")");
        dinosaurs.Insert(2, "Compsognathus");

        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }
```

```
        Console.WriteLine("\ndinosaurs[3]: {0}", dinosaurs[3]);

        Console.WriteLine("\nRemove(\"Compsognathus\")");
        dinosaurs.Remove("Compsognathus");

        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }

        dinosaurs.TrimExcess();
        Console.WriteLine("\nTrimExcess()");
        Console.WriteLine("Capacity: {0}", dinosaurs.Capacity);
        Console.WriteLine("Count: {0}", dinosaurs.Count);

        dinosaurs.Clear();
        Console.WriteLine("\nClear()");
        Console.WriteLine("Capacity: {0}", dinosaurs.Capacity);
        Console.WriteLine("Count: {0}", dinosaurs.Count);
    }
}
```

Thread

Per la gestione del multithreading è presente la classe *Thread* che permette di eseguire porzioni di codice associati al processo. La classe fornisce diversi metodi che permettono di conoscere lo stato di un thread e di impostarne la priorità e tramite il metodo *ParameterizedThreadStart* è possibile passare dati alla routine del thread.

Codice 1.3: Esempio Thread

```
using System;
using System.Threading;

// Simple threading scenario:  Start a static method running
// on a second thread.
public class ThreadExample {
    // The ThreadProc method is called when the thread starts.
    // It loops ten times, writing to the console and yielding
    // the rest of its time slice each time, and then ends.
    public static void ThreadProc() {
        for (int i = 0; i < 10; i++) {
            Console.WriteLine("ThreadProc: {0}", i);
            // Yield the rest of the time slice.
            Thread.Sleep(0);
        }
    }

    public static void Main() {
```

```
        Console.WriteLine("Main thread: Start a second thread.");
        // The constructor for the Thread class requires a ThreadStart
        // delegate that represents the method to be executed on the
        // thread.  C# simplifies the creation of this delegate.
        Thread t = new Thread(new ThreadStart(ThreadProc));

        // Start ThreadProc.  Note that on a uniprocessor, the new
        // thread does not get any processor time until the main thread
        // is preempted or yields.  Uncomment the Thread.Sleep that
        // follows t.Start() to see the difference.
        t.Start();
        //Thread.Sleep(0);

        for (int i = 0; i < 4; i++) {
            Console.WriteLine("Main thread: Do some work.");
            Thread.Sleep(0);
        }

        Console.WriteLine("Main thread: Call Join(), to wait until ThreadProc ends.");
        t.Join();
        Console.WriteLine("Main thread: ThreadProc.Join has returned.  Press Enter to
        end program.");
        Console.ReadLine();
    }
}
```

TcpListener e TcpClient

La classe *TcpListener* fornisce dei metodi semplici per accettare connessioni in ingresso attraverso una socket in modalità sincrona. Per stabilire una connessione è possibile utilizzare la classe *TcpClient*. Una classe *TcpListener* viene creata indicando un indirizzo IP locale e una porta, oppure soltanto una porta, in modo da attendere la connessione sugli indirizzi indicati durante la creazione. Utilizzando il metodo *Start* si attenderanno le richieste di connessione in ingresso che verranno accodate fino alla chiamata del metodo *Stop* o al raggiungimento del numero massimo di connessioni stabilite; con il metodo *AcceptTcpClient* verrà estratta una connessione dalla coda con cui si potrà instaurare una comunicazione.
La classe *TcpClient* fornisce dei metodi semplici per la connessione, l'invio e la ricezione di un flusso di dati attraverso una socket in modalità sincrona.

Codice 1.4: Esempio TcpListener

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;

class MyTcpListener
```

```
{
  public static void Main()
  {
    TcpListener server=null;
    try
    {
      // Set the TcpListener on port 13000.
      Int32 port = 13000;
      IPAddress localAddr = IPAddress.Parse("127.0.0.1");

      // TcpListener server = new TcpListener(port);
      server = new TcpListener(localAddr, port);

      // Start listening for client requests.
      server.Start();

      // Buffer for reading data
      Byte[] bytes = new Byte[256];
      String data = null;

      // Enter the listening loop.
      while(true)
      {
        Console.Write("Waiting for a connection... ");

        // Perform a blocking call to accept requests.
        // You could also user server.AcceptSocket() here.
        TcpClient client = server.AcceptTcpClient();
        Console.WriteLine("Connected!");

        data = null;

        // Get a stream object for reading and writing
        NetworkStream stream = client.GetStream();

        int i;

        // Loop to receive all the data sent by the client.
        while((i = stream.Read(bytes, 0, bytes.Length))!=0)
        {
          // Translate data bytes to a ASCII string.
          data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
          Console.WriteLine("Received: {0}", data);

          // Process the data sent by the client.
          data = data.ToUpper();

          byte[] msg = System.Text.Encoding.ASCII.GetBytes(data);

          // Send back a response.
          stream.Write(msg, 0, msg.Length);
          Console.WriteLine("Sent: {0}", data);
        }
```

```
        // Shutdown and end connection
        client.Close();
      }
    }
    catch(SocketException e)
    {
      Console.WriteLine("SocketException: {0}", e);
    }
    finally
    {
      // Stop listening for new clients.
      server.Stop();
    }


    Console.WriteLine("\nHit enter to continue...");
    Console.Read();
  }
}
```

Codice 1.5: Esempio TcpClient

```
static void Connect(String server, String message)
{
  try
  {
    // Create a TcpClient.
    // Note, for this client to work you need to have a TcpServer
    // connected to the same address as specified by the server, port
    // combination.
    Int32 port = 13000;
    TcpClient client = new TcpClient(server, port);

    // Translate the passed message into ASCII and store it as a Byte array.
    Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

    // Get a client stream for reading and writing.
   //  Stream stream = client.GetStream();

    NetworkStream stream = client.GetStream();

    // Send the message to the connected TcpServer.
    stream.Write(data, 0, data.Length);

    Console.WriteLine("Sent: {0}", message);

    // Receive the TcpServer.response.

    // Buffer to store the response bytes.
    data = new Byte[256];

    // String to store the response ASCII representation.
    String responseData = String.Empty;
```

```
32
      // Read the first batch of the TcpServer response bytes.
34    Int32 bytes = stream.Read(data, 0, data.Length);
      responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);
36    Console.WriteLine("Received: {0}", responseData);

38    // Close everything.
      stream.Close();
40    client.Close();
    }
42  catch (ArgumentNullException e)
    {
44    Console.WriteLine("ArgumentNullException: {0}", e);
    }
46  catch (SocketException e)
    {
48    Console.WriteLine("SocketException: {0}", e);
    }
50
    Console.WriteLine("\n Press Enter to continue...");
52  Console.Read();
  }
```

NamedPipeServerStream e NamedPipeClientStream

Le *Named Pipe* sono un sistema per la comunicazione interprocessuale in locale o su una rete, un solo nome di pipe può essere condiviso da più oggetti NamedPipeClientStream.
La classe *NamedPipeServerStream* permette di instaurare una comunicazione tra named pipe in modalità sincrona o asincrona. Una volta instanziata attende la connessione di una classe *NamedPipeClientStream* con cui comunicare.
La classe *NamedPipeClientStream* permette anch'essa di instaurare una comunicazione attraverso le named pipe e si connette a una classe *NamedPipeServerStream* in attesa.
Alla creazione delle due classi si possono indicare dei parametri che andranno a incidere sul funzionamento della comunicazione:

- il nome della pipe su cui comunicare
- la direzione della comunicazione, che può essere In, Out o InOut a seconda che vi sia una comunicazione solo in uscita, in entrate o in entrambe le direzioni
- il metodo di trasmissione, che può essere Byte o Message a seconda che i dati siano letti come un flusso di byte o un flusso di messaggi

- altri opzioni che permettono di indicare se la comunicazione è asincrona o se il sistema deve ignorare cache intermedie ed accedere direttamente alla pipe

Codice 1.6: Esempio NamedPipeServerStream

```
using System;
using System.IO;
using System.IO.Pipes;

class PipeServer
{
    static void Main()
    {
        using (NamedPipeServerStream pipeServer =
            new NamedPipeServerStream("testpipe", PipeDirection.Out,
        PipeTransmissionMode.Message, PipeOptions.Asynchronous))
        {
            Console.WriteLine("NamedPipeServerStream object created.");

            // Wait for a client to connect
            Console.Write("Waiting for client connection...");
            pipeServer.WaitForConnection();

            Console.WriteLine("Client connected.");
            try
            {
                // Read user input and send that to the client process.
                using (StreamWriter sw = new StreamWriter(pipeServer))
                {
                    sw.AutoFlush = true;
                    Console.Write("Enter text: ");
                    sw.WriteLine(Console.ReadLine());
                }
            }
            // Catch the IOException that is raised if the pipe is broken
            // or disconnected.
            catch (IOException e)
            {
                Console.WriteLine("ERROR: {0}", e.Message);
            }
        }
    }
}
```

Codice 1.7: Esempio NamedPipeClientStream

```
using System;
using System.IO;
using System.IO.Pipes;

class PipeClient
```

```
{
    static void Main(string[] args)
    {
        using (NamedPipeClientStream pipeClient =
            new NamedPipeClientStream(".", "testpipe", PipeDirection.In,
        PipeTransmissionMode.Message, PipeOptions.Asynchronous))
        {

            // Connect to the pipe or wait until the pipe is available.
            Console.Write("Attempting to connect to pipe...");
            pipeClient.Connect();

            Console.WriteLine("Connected to pipe.");
            Console.WriteLine("There are currently {0} pipe server instances open.",
              pipeClient.NumberOfServerInstances);
            using (StreamReader sr = new StreamReader(pipeClient))
            {
                // Display the read text to the console
                string temp;
                while ((temp = sr.ReadLine()) != null)
                {
                    Console.WriteLine("Received from server: {0}", temp);
                }
            }
        }
        Console.Write("Press Enter to continue...");
        Console.ReadLine();
    }
}
```

Capitolo 2

Asp.Net

2.1 Introduzione

Asp.Net è un Web application framework sviluppatoda Microsoft per sviluppare siti web dinamici, applicazione web e web service. Rilasciato nel Gennaio 2002 e basato sul CLR (Common Language Runtime) è il successore dell'ASP (Active Server Page) e permette di scrivere codice utilizzando qualsiasi linguaggio di alto livello supportato dal framework .Net. Asp.Net semplifica la migrazione degli sviluppatori da applicazione Windows ad applicazione web permettendogli di utilizzare controlli widget del tutto simili a quelli disponibili nello sviluppo di interfacce su Windows.

2.2 Caratteristiche

Asp.Net permette di inserire codice dinamico che verrà eseguito sul server in due diversi modi.
Il primo metodo consiste nell'inserire il codice all'interno della pagina aspx all'interno di un determinato blocco.

```
1  <$\%$ @ Page Language = "C #"$\%$>
   <Script runat = "server">
3    //codice C#
   </ Script>
```

Il secondo metodo, disponibile a pertire da Asp.Net Framework 2.0 , permette di inserire il codice all'interno di un file esterno con estensione diversa a seconda del linguaggio utilizzato, per esempio nel caso del C# il file avrà estensione .aspx.cs, ma indicando nel file aspx in quale file ricercare il codice.

```
<%@ Page Language="C#" CodeFile="SampleCodeBehind.aspx.cs" Inherits="
      Website.SampleCodeBehind" AutoEventWireup="true" %>
```

Le applicazioni Asp.Net sono ospitate su un Web server e sono accessibili tramite il protocolo HTTP. Come tale, se un'applicazione accede alla informazioni sullo stato, deve implementarne la gestione. Per questo Asp.Net fornisce diversi oggetti per la gestione dello stato, questi sono Application, Session, ViewState e Cache, inoltre supporta anche altri sistemi come cookie, cache e query.
Lo stato applicazione è detenuto da un insieme di variabili condivise definite dall'utente ed accessibili dall'oggetto *Application*. Queste sono generate quando viene innescato l'evento Application_OnStart sul caricamento della prima istanza e restano disponibili fino alla chiusura dell'ultima istanza.
Lo stato della sessione è detenuto da un insieme di variabili persistenti durante l'intera sessione; queste variabili sono contenuto all'interno dell'oggetto *Session* fornito da Asp.Net e sono uniche per ogni istanza della sessione. E' possibile impostare il sistema in modo che le variabili di sessione vengano eliminate dopo un determinato tempo di inattività anziché attendere il termine della sessione utente.
Lo stato della visualizzazione è riferito al meccanismo di gestione dello stato a livello di pagina, usato dal codice html generato dalle applicazioni Asp.Net per gestire controlli web e widgets. Lo stato del controllo viene inviato al form tramite un campo nascosto chiamato *_VIEWSTATE*, accessibile dal server tramite l'oggetto *ViewState*.
Asp.Net fornisce anche un oggetto *Cache* per memorizzare diversi dati; l'oggetto mantiene i dati solo per un periodo di tempo definito e viene automaticamente eliminato ad ogni termine di sessione.

2.3 Controlli Asp.Net

Tra i molti controlli resi disponibili da Asp.Net ve ne sono alcune molto simili alla loro controparte html, altre invece permettono di utilizzare widget molto più complessi. I controlli Asp.Net sono riconoscibili dal fatto che il nome identificativo è preceduto da *"asp:"*.

asp:UpdatePanel Il controllo *UpdatePanel* è una parte centrale della funzionalità Ajax in Asp.Net. Esso viene utilizzato assieme al controllo *ScriptManager* per attivare la funzione di rendering a pagina parziale, cioè permette di aggiornare dinamicamente solo determinate parti della pagina web evitando di dover eseguire il postback dell'intera pagina.

asp:ScriptManager Il controllo *ScriptManager* consente di registrare gli script di rendering a pagina parziale e di fornire accesso ai metodi di web service e di servizi da parte di script registrando i metodi allo ScriptManager.

asp:Login Il controllo *Login* crea una seria di campi per inserire utente e password per una procedura di login e permette di gestire in modo semplice i controlli per la verifica dei dati inseriti.

asp:GridView Il controllo *GridView* permette di creare una tabella da diverse origini di dati, oppure generarne il contenuto dinamicamente e possiede incorporate diverse funzionalità come la possibilità di ordinare i dati, di eliminarli o modificarli, di selezionare diverse righe, di suddividere i dati in diverse pagine, di spostarli all'interno della tabella, di personalizzare l'aspetto della tabella e di gestire diversi eventi.

asp:DropDownList Il controllo *DropDownList* permette di creare un elenco a discesa a selezione singola di cui è possibile personalizzarne l'aspetto. L'origine dei dati può essere definita nel codice tramite il controllo *ListItem* oppure definita dinamicamente durante l'esecuzione.

asp:ListItem Il controllo *ListItem* rappresenta un singolo elemento di dati all'interno di un controllo elenco associato ai dati. Possiede sia una proprietà Value che una proprietà Text in modo da poter assegnare un valore differenza da ciò che viene visto da un utente.

asp:CheckBoxList Il controllo *CheckBoxList* permette di creare un gruppo di caselle a selezione multipla di cui è possibile associare l'origine dei dati dinamicamente. Tramite la proprietà Selected è possibile selezionare da codice determinati elementi o verificare quali elementi sono stati selezionati. E' anche possibile impostare l'elenco sia in orizzontale che in verticale. A differenza di una serie di singoli oggetti *CheckBox* che permettono maggior controllo nel layout, con *CheckBoxList* è più semplice gestire l'origine dei dati.

asp:Button Il controllo *Button* crea un pulsante di comando a cui è possibile associare un evento lato Server per rispondere al postback generato. Può generare anche un evento script lato Client che verrà gestito prima dell'invio della pagina o che addirittura genererà o bloccherà l'invio della pagina.

asp:TextBox Il controllo *TextBox* consente di creare un oggetto di input per l'inserimento di testo. E' anche possibile associare diversi eventi al controllo.

asp:Label Il controllo *Label* consente di inserire testo all'interno della pagina in modo dinamico durante l'esecuzione; a differenza del controllo Label dell'html non consente di visualizzare tag aggiuntivi.

asp:HiddenField Il controllo *HiddenField* consente di memorizzare un valore all'interno della pagina senza renderlo visibile ad un utente ma accessibile al codice lato Server.

asp:Image Il controllo *Image* consente di utilizzare delle immagini direttamente dal codice lato server. E' possibile impostare il testo da visualizzare nel caso in cui l'immagine non sia disponibile.

Capitolo 3

Javascript

3.1 Introduzione

Javascript è un linguaggio di scripting multi paradigma, che supporta uno stile di programmazione a oggetti, imperativo e funzionale ed è comunemente utilizzato nei siti web. E' stato originariamente sviluppato da Brendan Eich della Netscape con il nome di Mocha, poi rinominato in LiveScript e infine in JavaScript (soprattutto perchè è stato influenzato dal Java). Non vi è una vera relazione tra Java e JavaScript, entrambi utilizzato una sintassi simile al C ma le loro semantiche e il loro approccio alla modellazione a oggetti sono profondamente diversi.
Javascript è un linguaggio interpretato, solitamente utilizzato in ambito web come linguaggio di scripting lato client, questo perché il codice Javascript viene eseguito direttamente sul client e non sul server, alleggerendo in questo modo il carico di lavoro del server.
Di contro, questo approccio implica che l'utilizzo di sorgenti di dati di grandi dimensioni possa impiegare parecchio tempo per lo scaricamento, inoltre per l'utilizzo di dati contenuti in un database remoto si è costretti a ricorrere ad altri linguaggi per l'accesso alle informazioni per poi riportarli a variabili Javascript.
Javascript è un linguaggio debolmente tipizzato con variabili tipizzate dinamicamente, in gran parte è basato sugli oggetti dove gli oggetti sono array associativi, inoltre visto che è basato sui prototipi l'ereditarietà è fra oggetti e non fra classi (anche perché non ha classi) e supporta gran parte della struttura di programmazione del C.

Oltre a poter essere interpretato da tutti i vari browser web, Javascript permette l'uso delle API di Google per creare e interagire con una mappa di Google Map, inoltre vi è la possibilità di estendere le capacità del linguaggio tramite l'utilizzo di jQuery di cui tratteremo successivamente; per questi motivi si è scelto di utilizzare Javascript, nonostante la limitazione di non

poter accedere direttamente ai dati su database.

3.2 Sintassi

La sintassi del Javascript è simile a quella del C, per poter inserire codice all'interno di pagine web bisogna inserirlo all'interno del tag *<script></script>*. Per facilitare il riconoscimento degli script solitamente si specifica il linguaggio utilizzato all'inteno del tag *<script type=text/javascript></script>*. Oltre a inserire gli script direttamente nel codice della pagine è possibile inserirli all'interno di file .js per poi indicare alla pagina web quali file caricare, per fare questo bisogna indicare dove reperire il file *<script type=text/javascript src=Scripts/Codice.js></script>*.
Come detto in precedenza, le variabili sono definite dinamicamente e basta semplicemente assegnare loro un valore o usare il comando *var*. Le variabili definite all'esterno di una funzione sono accessibili da tutta la pagina web, mentre per passarne i valori tra pagine diverse si possono utilizzare i cookie o utilizzare frame nascosti oppure memorizzarli in finestre in background.
Ogni cosa o è un valore primitivo o è un oggetto; gli oggetti sono entità uniche che associano nomi di proprietà ai valori, questo significa che gli oggetti sono in realtà vettori associativi.
Esistono diversi oggetti predefiniti come Array, Boolean, Date, Function, Math, Nmber, Object, RegExp, String, ecc. Altri oggetti detti *oggetti ospite* sono definiti dall'ambiente di esecuzione del codice, quindi all'interno di un browser si ha accesso agli oggetti DOM come window, form, link, document, ecc.
Le funzioni sono istanze dell'oggetto Function e possono essere create ed assegnate come ogni altro oggetto. Solitamente vengono definite con il comando *function* e possono restituire un valore tramite il comando *return*, non devono necessariamente avere un nome ed è possibile richiamarle utilizzando un numero di argomenti inferiore a quello indicato nella definizione della funzione.

Codice 3.1: Funzioni

```
function(arg1, arg2, arg3, argn)
{
  istruzioni;
  return espressione;
}

var myFunc1 = new Function("alert('Hello')");
var myFunc2 = myFunc1;
myFunc2();
```

Le versioni più recenti di Javascript includono la possibilità di utilizzare i costrutti *try..catch..finally* per la gestione degi errori. Inoltre Javascript

possiede diversi eventi, solitamente definiti come funzioni anonime all'interno dei tag html. Vi sono diversi tipi di eventi: attivabili da movimenti o tasti del mouse, attivabili da tastiera, attivabili da modifiche dell'utente, attivabili dal caricamento di determinati oggetti, attivabili dal movimento delle finestre e vari altri tipo.

3.3 Classi

Nonostante Javascript non abbia le classi, è possibile simularle grazie all'elasticità dell'oggetto *function*.
La creazione di una classe può essere effettuata tramite l'assegnazione ad una variabile di una funzione e l'utilizzo dell'oggetto *this* per associare i metodi e le proprietà inserite al nome della variabile.
Vediamo un esempio:

Codice 3.2: Classi

```
var Persona = new function(param_nome, param_cognome)
{
  this.nome = param_nome;
  this.cognome = param_cognome;
  this.risultato = function()
  {
    return "Nome: "+ this.nome+"\nCognome: "+this.cognome;
  }
}

var p = new Persona("","Selmini");
p.nome = "Andrea";
alert(p.risultato());
```

Ora le proprietà nome e cognome ed il metodo risultato possono essere utilizzati come in una normale classe.

3.4 Google Maps API

Google possiede delle API (Application Programming Interface) che permettono di creare ed interagire con una mappa all'interno di un sito web.

Per caricare le API è necessario indicare il caricamento degli script all'interno della head del codice html.

Codice 3.3: Caricamento API

```
<script type="text/javascript"
 src="http://maps.googleapis.com/maps/api/js?sensor=false">
```

```
</script>

HTTPS:
<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?sensor
    =false">
</script>
```

Inoltre è possibile caricare alcune librerie per aggiungere determinati comandi, per utilizzarli bisogna aggiungere il parametro *libraries=NAME_LIBRARY* durante il caricamento delle API.

Per visualizzare una mappa all'interno di una pagina web è necessario inserire un oggetto div all'interno del codice html e istanziare un oggetto *Map*. Solitamente questa operazione viene effettuata al caricamento della pagina tramite l'evento *onload*.

Codice 3.4: Inserimento Mappa

```
function initialize() {
  var myOptions = {
    center: new google.maps.LatLng(-34.397, 150.644),
    zoom: 8,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  };
  var map = new google.maps.Map(document.getElementById("map_canvas"),
      myOptions);
}

window.onload=function(){
  initialize();
}
```

Tramite le API è possibile inserire linee, marcatori e figure geometriche all'interno della mappa e poter interagire con esse. Per creare simili oggetti grafici sono resi disponibile diversi oggetti all'interno del codice.

Oggetto Marker L'oggetto Marker permette di creare un marcatore per indicare una posizione. Il simbolo del marcatore può essere caricato da un'immagine ed è possibile assegnarli diversi eventi per permetterne l'interazione con il puntatore del mouse.

Codice 3.5: Marker

```
var myLatlng = new google.maps.LatLng(-25.363882,131.044922);
var marker = new google.maps.Marker({
  position: myLatlng,
  map: map
});
```

Oggetto Polyline L'oggetto Polyline permette di creare una linea composta da una serie di segmenti, consiste in un array di posizioni LatLng collegate e ordinate tra loro. E' possibile impostare il colore, la trasparenza e lo spessore di una Polyline, inoltre le si possono assegnare degli eventi per interagire con essa tramite il mouse.

Codice 3.6: Polyline

```
var flightPlanCoordinates = [
  new google.maps.LatLng(37.772323, −122.214897),
  new google.maps.LatLng(21.291982, −157.821856),
  new google.maps.LatLng(−18.142599, 178.431),
  new google.maps.LatLng(−27.46758, 153.027892)
];
var flightPath = new google.maps.Polyline({
  map: map,
  path: flightPlanCoordinates,
  strokeColor: "#FF0000",
  strokeOpacity: 1.0,
  strokeWeight: 2
});
```

MVCArray Gli oggetti Polyline sono composti da array di array, dove gli array sono oggetti MVCArray. Gli MVCArray sono array composti da oggetti LatLng contenente le coordinate che insieme compongono una spezzata.
Per accedere a questi array si utilizza il metodo *getPath()* e manipolando gli MVCArray tramite i metodi forniti si può modificare la spezzata in tempo reale.
Nonostante gli MVCArray siano array è impossibile accedere agli oggetti con *mvcArray[i]* ma si deve ricorrere al metodo *getAt(i)*, oppure tramite *getArray()* accedere agli array base che compongono l'oggetto ma in questo modo non vi saranno modifiche alla Polyline visualizzata.

Oggetto Polygon Gli oggetti Polygon sono simili agli oggetti Polyline tranne che invece che identificare una spezzata identificano un poligono. Permettono di impostare il colore, la trasparenza, lo spessore e sono composti da MVCArray.

Codice 3.7: Polygon

```
var triangleCoords = [
  new google.maps.LatLng(25.774252, −80.190262),
  new google.maps.LatLng(18.466465, −66.118292),
  new google.maps.LatLng(32.321384, −64.75737),
```

```
  new google.maps.LatLng(25.774252, −80.190262)
];

var bermudaTriangle = new google.maps.Polygon({
  map: map,
  paths: triangleCoords,
  strokeColor:"#FF0000",
  strokeOpacity:0.8,
  strokeWeight:2,
  fillColor: "#FF0000",
  fillOpacity: 0.35
});
```

Oltre all'oggetto generico Polygon vi sono gli oggetti Circle e Rectangle che semplificano la costruzione di cerchi e rettangoli o quadrati.

Oggetto Circle Gli oggetti Circle pur essendo simili come proprietà a un Polygon non è composto da *paths* di LatLng, ma da *center* specificato da un LatLng per il centro e da *radius* indicato in metri per il raggio. Inoltre Circle ha anche la proprietà *editable* che indica se si potrà modificare il cerchio in tempo reale direttamente dalla mappa.

Codice 3.8: Circle

```
var circle = new google.maps.Circle({
  map: map,
  center: new google.maps.LatLng(−34.397, 150.644),
  radius: 2500,
  editable: true
});
```

Oggetto Rectangle Gli oggetti Rectangle sono simili a un Polygon ma non è composto da *paths* di LatLng, bensì da *bouds*. Così come Circle, Rectangle possiede la proprietà *editable* per poterlo modificare direttamente dalla mappa in tempo reale.

Codice 3.9: Rectangle

```
var bounds = new google.maps.LatLngBounds(
  new google.maps.LatLng(44.490, −78.649),
  new google.maps.LatLng(44.599, −78.443)
);
var rectangle = new google.maps.Rectangle({
  map: map;
  bounds: bounds,
  editable: true
});
```

Capitolo 4

jQuery

4.1 Introduzione

jQuery è una libreria di funzioni (un cosiddetto software framework) javascript, cross-browser per le applicazioni web, che si propone come obiettivo quello di semplificare la programmazione lato client delle pagine HTML riducendo la sintassi di javascript e rendendola simile ai selettori del CSS.

Ideata per la prima volta il 22 Agosto del 2005 da John Resig, viene pubblicata una prima release Alpha il 30 Giugno 2006 e il 26 Agosto 2006 viene rilasciata la prima versione stabile.
Attualmente ha raggiunto la versione 1.7.1.

jQuery permette con poche righe di codice di cambiare completamente la visualizzazione di un elemento e di assegnargli effetti particolari, permette di gestire completamente gli eventi e possiede funzioni per utilizzare in maniera semplice AJAX e la connessione l'invio di dati.
Un altro dei punti forti di jQuery e la presenza di numerosissimi plugin che permettono di estenderne ulteriormente le funzionalita , il tutto seguito da una community molto attiva che fornisce ogni tipo di documentazione necessaria e plugin per qualsiasi necessita.

L'idea dietro jQuery viene espressa molto bene dal suo motto "Write Less, Do More".

4.2 Caratteristiche

jQuery ruota interamente intorno all'oggetto/funzione $ che è a sua volta un'abbreviazione di *jQuery*.
Il core di jQuery fornisce:

- I costruttori per l'utilizzo della libreria stessa

- I metodi e le proprietà per accedere agli elementi contenuti in un oggetto
- I metodi per creare e utilizzare liste e code (di oggetti o anche funzioni)
- I metodi per estendere il framework mediante plugin
- I metodi per eseguire della animazioni

jQuery riconosce gli elementi tramite i selettori del CSS, è anche possibile indicare più elementi utilizzando una classe oppure filtrare alcuni elementi specifici indicando attributi o contenuti.
Dato che utilizza i selettori della sintassi CSS è possibile anche concatenare più selettori, inoltre dato che ciascuna funzione ritorna lo stesso oggetto jQuery è possibile concatenare più funzioni.

4.3 PlugIn

Per sviluppare il sistema di cui parleremo successivamente si sono utilizzati svariati plugin disponibili in rete.

4.3.1 jQueryUI

jQueryUI è un plugin ufficiale che permette l'utilizzo di svariati oggetti ed effetti grafici integrandoli semplicemente nel codice.
Gli oggetti utilizzati sono *Datepicker* e *Slider*

Datepicker L'oggetto *Datepicker* permette di visualizzare un calendario interattivo. E' inoltre possibile impostare due oggetti Datepicker in modo da indicare un intervallo temporale, impedendo al secondo oggetto Datepicker di selezionare una data precedente alla data indicata nel primo Datepicker e impedendo al primo oggetto Datepicker di selezionare una data successiva alla data nel secondo Datapicker.

Figura 4.1: Esempio Datepicker

Codice 4.1: Esempio Utilizzo Datepicker

```
$(function() {
  var dates = $( "#from, #to" ).datepicker({
    defaultDate: "+1w",
    changeMonth: true,
    onSelect: function( selectedDate ) {
      var option = this.id == "from" ? "minDate" : "maxDate",
        instance = $( this ).data( "datepicker" ),
        date = $.datepicker.parseDate(
          instance.settings.dateFormat ||
          $.datepicker._defaults.dateFormat,
          selectedDate, instance.settings );
      dates.not( this ).datepicker( "option", option, date );
```

```
13      }
    });
15  });
```

Slider L'oggetto *Slider* permette di visualizzare una barra scorrevole di cui è possibile impostare valore minimo e massimo e di cui è possibile ottenere il valore selezionato al momento.

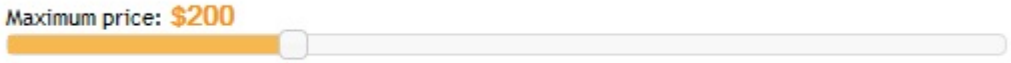

Figura 4.2: Esempio Slider

Codice 4.2: Esempio Utilizzo Slider

```
1  $(function() {
     $( "#slider-range-min" ).slider({
3      range: "min",
       value: 10,
5      min: 1,
       max: 700,
7      slide: function( event, ui ) {
         $( "#amount" ).val( "$" + ui.value );
9      }
     });
11   $( "#amount" ).val( "$" + $( "#slider-range-min" ).slider( "value" ) );
   });
```

4.3.2 Dynatree

Il plugin *Dynatree* (reperibile sul sito http://code.google.com/p/dynatree/) permette di creare un menu di navigazione ad albero dinamicamente direttamente da javascript.
Caratteristiche:

- Ottimizzato per alberi dinamici molto grandi (gli oggetti DOM vengono creati solo nel momento in cui necessitano)
- Programmabile attraverso una ricca interfaccia a oggetti.
- Supporto per Ajax e per il caricamento di parti dell'albero solo quando visualizzate
- Checkbox e possibilità di selezione gerarchica

- Supporto per drag and drop
- Supporto per la persistenza dei dati
- Possibilità di utilizzare la tastiera per navigare nell'albero
- Possibilità di inizializzare l'albero da codice HTML, JSON o oggetti Javascript.

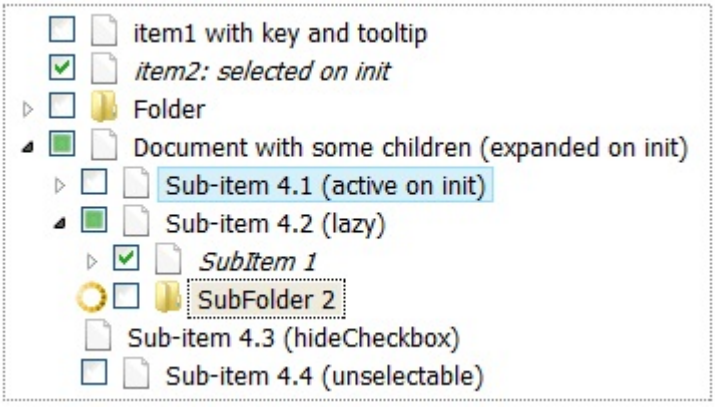

Figura 4.3: Esempio Dynatree

Codice 4.3: Esempio Dynatree

```
$(function(){
  // Attach the dynatree widget to an existing <div id="tree"> element
  // and pass the tree options as an argument to the dynatree() function:
  $("#tree").dynatree({
    onActivate: function(node) {
      // A DynaTreeNode object is passed to the activation handler
      // Note: we also get this event, if persistence is on, and the page is reloaded.
      alert("You activated " + node.data.title);
    },
    children: [
      {title: "Item 1"},
      {title: "Folder 2", isFolder: true, key: "folder2",
        children: [
          {title: "Sub-item 2.1"},
          {title: "Sub-item 2.2"}
        ]
      },
      {title: "Item 3"}
    ]
  });
});
```

4.3.3 ContexMenu

Il plugin *ContextMenu* (reperibile sul sito http://www.abeautifulsite.net/blog/2008/09/jquery-context-menu-plugin/) permette di creare un piccolo menu visualizzabile facendo click destro su determinati elementi.

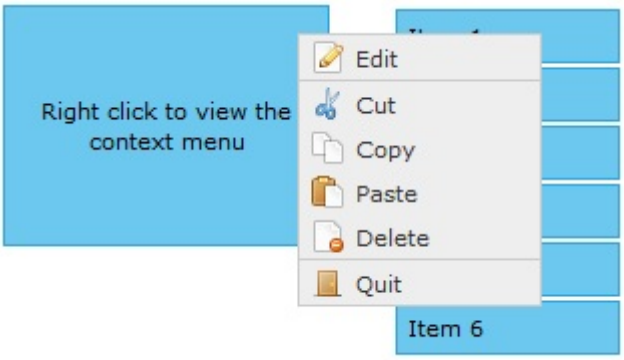

Figura 4.4: Esempio ContextMenu

Codice 4.4: Esempio ContextMenu

```
<ul id="myMenu" class="contextMenu">
    <li class="edit">
        <a href="#edit">Edit</a>
    </li>
    <li class="cut separator">
        <a href="#cut">Cut</a>
    </li>
    <li class="copy">
        <a href="#copy">Copy</a>
    </li>
    <li class="paste">
        <a href="#paste">Paste</a>
    </li>
    <li class="delete">
        <a href="#delete">Delete</a>
    </li>
    <li class="quit separator">
        <a href="#quit">Quit</a>
    </li>
</ul>

function() {
    $("#selector").contextMenu({
        menu: ''myMenu''
    },
        function(action, el, pos) {
        alert(
```

```
            ''Action: '' + action + ''\n\n'' +
            ''Element ID: '' + $(el).attr(''id'') + ''\n\n'' +
            ''X: '' + pos.x + ''  Y: '' + pos.y + '' (relative to element)\n\n'' +
            ''X: '' + pos.docX + ''  Y: '' + pos.docY+ '' (relative to document)''
            );
    });
});
```

4.3.4 TimeEntry

Il plugin *TimeEntry* (reperibile sul sito http://keith-wood.name/timeEntry.html) permette di impostare un oggetto input field in modo da poter inserire un orario, utilizzando un controllo posizionato a lato.

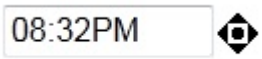

Figura 4.5: Esempio TimeEntry

Codice 4.5: Esempio TimeEntry

```
$('#defaultEntry').timeEntry().change(function() {
    var log = $('#log');
    log.val(log.val() + ($('#defaultEntry').val() || 'blank') + '\n');
});
```

4.3.5 jqGrid

Il plugin *jqGrid* (reperibile sul sito http://www.trirand.com/blog/) permette di creare una tabella caricata dinamicamente.
Caratteristiche:

- La grafica della tabella è compatibile con i temi di jQueryUI
- Possibilità di suddividere il contenuto della tabella in più pagine
- Possibilità di organizzare il contenuto in sotto-tabelle
- Possibilità di ordinare e ridimensionare colonne e righe
- Possibilità di editare, salvare il contenuto delle singole celle e di modificare le righe e le colonne
- Possibilità di caricare il contenuto della tabella solo quando lo si visualizza

Basic Example

Inv No	Date	Client	Amount	Tax	Total	Notes
13	2007-10-06	Client 3	1000.00	0.00	1000.00	
12	2007-10-06	Client 2	700.00	140.00	840.00	
11	2007-10-06	Client 1	600.00	120.00	720.00	
10	2007-10-06	Client 2	100.00	20.00	120.00	
9	2007-10-06	Client 1	200.00	40.00	240.00	
8	2007-10-06	Client 3	200.00	0.00	200.00	
7	2007-10-05	Client 2	120.00	12.00	134.00	

Page 1 of 2 10 View 1 - 10 of 13

Figura 4.6: Esempio jqGrid

Codice 4.6: Esempio jqGrid

```
$(function(){
  $("#list").jqGrid({
  data: table,
    datatype: 'local',
    colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
    colModel :[
      {name:'invid', index:'invid', width:55},
      {name:'invdate', index:'invdate', width:90},
      {name:'amount', index:'amount', width:80, align:'right'},
      {name:'tax', index:'tax', width:80, align:'right'},
      {name:'total', index:'total', width:80, align:'right'},
      {name:'note', index:'note', width:150, sortable:false}
    ],
    pager: '#pager',
    rowNum:10,
    rowList:[10,20,30],
    sortname: 'invid',
    sortorder: 'desc',
    viewrecords: true,
    gridview: true,
    caption: 'My first grid'
  });
});
```

4.3.6 MouseWheel

Il plugin *MouseWheel* (reperibile sul sito http://brandonaaron.net/code/mousewheel/docs) permette di intercettare il movimento della rotella del mouse sopra un oggetto presente nella pagina web. Questo plugin è utilizzato all'interno del plugin *TimeEntry* per modificare l'orario tramite la rotella del mouse.

Codice 4.7: Esempio mousewheel

```
// using bind
$('#my_elem').bind('mousewheel', function(event, delta) {
    console.log(delta);
});

// using the event helper
$('#my_elem').mousewheel(function(event, delta) {
    console.log(delta);
});
```

Capitolo 5

MySql

5.1 Introduzione

MySql è un sistema per la gestione di database relazionali disponibile sia per sistemi UNIX come GNU/LINUX che per Windows e supporta la maggior parte della sintassi SQL.
MySql permette di sfruttare diversi sistemi di tabelle (o storage engine) alcuni dei quali non ufficiali e prodotti da terze parti. Gli storage engine ufficiali sono:

- InnoDB
- MyISAM
- Memory
- CSV
- Archive
- Blockhole
- Merge
- Federated

MySql fornisce solo un'interfaccia accessibile da una shell, per avere un'interfaccia grafica è presente il software MySql Workbench che permette di creare e gestire connessioni con il database, modellare schemi di tabelle graficamente, amministrare il database e le sue connessioni e fornisce un editor sql con cui poter interagire direttamente con le tabelle.

5.2 MySql e C#

Per poter interfacciare MySql con vari linguaggi di programmazione sono disponibili diversi driver ufficiali, in particolare per l'utilizzo con il C# si è utilizzato il file Connector/NET che rende disponibili la dll MySql.Data.
La dll, che implementa un'interfaccia ADO.NET ed è interamente scritta in C#, rende disponibili diverse classi all'interno del C# tra cui *MySqlConnection* che permette di collegarsi al database, *MySqlCommand* che permette di creare un comando da eseguire attraverso una stringa contenente una query, *MySqlDataAdapter* che permette di creare un oggetto tableadapter in cui inserire il risultato di una query e *MySqlException* che permette di intercettare un'eccezione causata da un componente che interagisce con il database.

5.3 Tipi di Dati

MySql supporta diversi tipi di dato divisi in tipi numerici, tipi di stringhe (di caratteri o di byte) e tipi di tempo.
I tipi di dato numerici sono:

- Tinyint: numero intero di 1 byte
- Smallint: numero intero di 2 byte
- Int: numero intero di 4 byte
- Bigint: numero intero di 8 byte
- Decimal: tipo di dato numerico utilizzato quando vi è la necessità di grande precisione, durante la creazione viene indicato la precisione e la scala del valore
- Float: numero in virgola mobile
- Double: numero in virgola mobile con doppia precisione
- Bit: numero binario
- Bool o Boolean: sono un sinonimo di Tinyint in quanto 0 viene considerato false e un numero diverso da 0 viene considerato vero, ma sono accettati anche i valori *TRUE* e *FALSE*

I tipi di stringhe sono:

- Char: stringa di caratteri con dimensione fissa
- Varchar: stringa di caratteri con dimensione variabile

- Binary: identico a Char ma contiene stringhe binarie
- Varbinary: identico a Varchar ma contiene stringhe binarie
- Blob: stringa di dati binari che non richiedono un set di caratteri
- Text: stringa di dati testuali che non richiedono un set di caratteri
- Enum: contiene un elenco di elementi permessi enumerati
- Set: contiene una lista di oggetti stringa permessi che può anche essere vuota

I tipi di data sono:

- Date: contiene una data con giorno, mese e anno
- Time: contiene un orario con ore, minuti e secondi
- Year: contiene un anno che può essere indicato in 2 o 4 cifre
- Datetime: contiene una data con giorno, mese, anno, ore, minuti e secondi
- Timestamp: contiene una data come Datetime, ma sono permessi i valori tra *1970-01-01 00:00:01* e *2038-01-19 03:14:07*

5.4 Sintassi

La sintassi di MySql è identica alla sintassi del Sql. Con i comandi *CREATE*, *DROP*, *ALTER* è possibile creare, eliminare e modificare tabelle o database.
Tramite il comando *INSERT* è possibile inserire righe all'interno di una tabella, indicando *INTO* si può indicare il nome della tabella su cui effettuare l'operazione, con *VALUES* vengono indicati i valori da inserire nei diversi campi della tabella.
Con il comando *UPDATE* è possibile modificare le righe di una tabella, tramite *WHERE* si possono specificare delle condizioni per modificare solo determinate righe mentre con *SET* si indicano i valori da sostituire.
Con il comando *DELETE* è possibile eliminare le righe, con *FROM* si può indicare in quale tabella effettuare l'operazione e tramite *WHERE* selezionare solo determinate righe che rispettano la condizione desiderata.
Infine con il comando *SELECT* è possibile visualizzare il contenuto delle righe; il comando *SELECT* permette alcuni l'uso dei seguenti parametri: con *ALL* si visualizzano tutte le righe della tabella, con *FROM* si indicano da quali tabelle cercare i dati, con *WHERE* si possono inserire delle condizioni per specificare i dati da visualizzare, con *GROUP BY* è possibile raggruppare i dati, con *HAVING* si possono selezionare dati particolare tra quelli

raggruppati da *GROUP BY*, mentre con *ORDER BY* si possono ordinare i dati indicando *DESC* o *ASC* per indicare un ordinamento discendente o ascendente.

Parte II

Progetto del Sistema

Capitolo 6

Database

In questo capitolo analizzeremo i passaggi che hanno portato dalla creazione di uno schema ER alla creazione delle tabelle vere e proprie che compongono il database utilizzato dal sistema.

6.1 Schema ER

Questa è la rappresentazione dello schema ER da cui si è partiti per sviluppare il database vero e proprio. Gli attributi per le entità Permessi e Configurazione sono stati raggruppati negli attributi *Valori Configurazione* e *Valori Permessi* per questioni di spazio ma saranno elencati nella loro completezza successivamente.

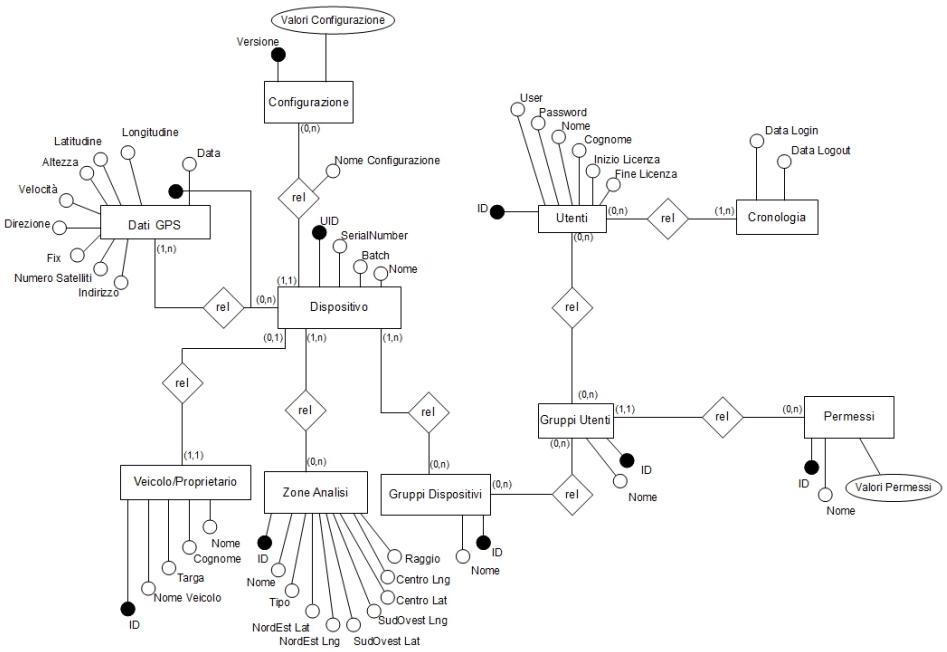

Figura 6.1: Schema ER

6.2 Traduzione Progetto Logico

Ora si analizzeranno i passaggi per la traduzione dello schema ER nel progetto logico.
Per ogni entità è stata creata una tabella che possa contenere i dati.

L'entità *Dati GPS* è stata convertita nella tabella *Datigps* che contiene i dati delle posizioni registrate da ogni dispositivo.
Datigps(<u>UID</u>,<u>Data</u>, Longitudine, Latitudine, Altezza, Velocità, Direzione, Fix, Numero Satelliti, Indirizzo)
FK: UID REFERENCES Dispositivo NOT NULL

L'entita *Veicolo/Proprietario* è stata convertita nella tabella *Veicolo/-Proprietario* che contiene i dati dei veicoli su cui è montato un dispositivo.
Veicolo/Proprietario (ID, Nome Veicolo, Targa, Cognome, Nome, UID)
AK: UID
FK: UID REFERENCES Dispositivo NOT NULL

L'entita *Zone Analisi* è stata convertita nella tabella *ZoneAnalisi* che contiene i dati per identificare una zona, utilizzata in alcune analisi dei dati.
ZoneAnalisi (ID, Nome, Tipo, Latitudine NordEst, Longitudine NordEst, Latitudine SudOvest, Longitudine SudOvest, Latitudine Centro, Longitudine Centro, Raggio)

La relazione tra *ZoneAnalisi* e *Dispositivo* è stata trasformata nella tabella *Dispositivi_ZoneAnalisi*.
Dispositivi_ZoneAnalisi (UID,ID_ZoneAnalisi)
FK: UID REFERENCES Dispositivo NOT NULL
FK: ID_ZoneAnalisi REFERENCES ZoneAnalisi NOT NULL

L'entita *Configurazione* è stata convertita nella tabella *Configurazione* che contiene i valori di una configurazione di un dispositivo.
Configurazione (VersioneConfig, VersioneSoftware, NomeApn, LoginApn, PasswordApn,IpServerConfig, PortServerConfig, IpServerData, PortServerData, IpServerUpgrade, PortServerUpgrade, DirectoryFtp, CentroServizi, NumSmsAlarm, NumSmsAlarm2, NumSmsAlarm3, NumCall, SensAccelerometro, FreqCampionamento, ModoAquisizione)

La relazione tra *Configurazione* e *Dispositivo* è stata trasformata in una tabella.
Dispositivi_Configurazioni (NomeConfig, VersioneConfig)
FK: VersioneConfig REFERENCES Configurazione NOT NULL

L'entita *Dispositivo* è stata convertita nella tabella *Dispositivo* che contiene i dati dei dispositivi.
Dispositivo (UID, SerialNumber, Batch, Nome, NomeConfig)
FK: NomeConfig REFERENCES Dispositivi_Configurazioni NOT NULL

L'entita *Gruppi Dispositivi* è stata convertita nella tabella *GruppiDispositivi* che contiene i dati dei gruppi di dispositivi.
GruppiDispositivi (ID, Nome)

La relazione tra *Dispositivo* e *GruppiDispositivi* è stata trasformata nella tabella *Dispositivi_GruppiDispositivi*.
Dispositivi_GruppiDispositivi (UID, ID)
FK: UID REFERENCES Dispositivo NOT NULL

FK: ID REFERENCES GruppiDispositivi NOT NULL

L'entita *Gruppi Utenti* è stata convertita nella tabella *GruppiUtenti* che contiene i dati dei gruppi di utenti.
GruppiUtenti (ID, Nome, ID_Permessi)
FK: ID_Permessi REFERENCES Permessi NOT NULL

La relazione tra *GruppiDispositivi* e *GruppiUtenti* è stata trasformata nella tabella *GruppiDispositivi_GruppiUtenti*.
GruppiDispositivi_GruppiUtenti (ID_GruppiDispositivi, ID_GruppiUtenti)
FK: ID_GruppiDispositivi REFERENCES GruppiDispositivi NOT NULL
FK: ID_GruppiUtenti REFERENCES GruppiUtenti NOT NULL

L'entita *Permessi* è stata convertita nella tabella *Permessi* che contiene le impostazioni dei livelli di permessi.
Permessi (ID, Nome, Edit_Dispositivo, Read_Dispositivo, Edit_Configurazione, Read_Configurazione, Edit_Veicolo, Read_Veicolo, Edit_Utenti, Read_Utenti, Edit_GruppiUtenti, Read_GruppiUtenti, Edit_GruppiDispositivi, Read_GruppiDispositivi, Edit_Associazioni, Read_Associazioni, Edit_DatiGps, Read_DatiGps, Edit_Permessi, Read_Permessi, Cronologia)

L'entita *Utenti* è stata convertita nella tabella *Utenti* che contiene i dati degli utenti.
Utenti (ID, User, Password, Nome, Cognome, InizioLicenza, FineLicenza, ID_GruppiUtenti)
FK: ID_GruppiUtenti REFERENCES GruppiUtenti NOT NULL

L'entita *Cronologia* è stata convertita nella tabella *Cronologia* che contiene i dati degli accessi.
Cronologia (ID_Utenti, DataLogin, DataLogout)
FK: ID_Utenti REFERECENS Utenti NOT NULL

6.3 Tabelle Database Finale

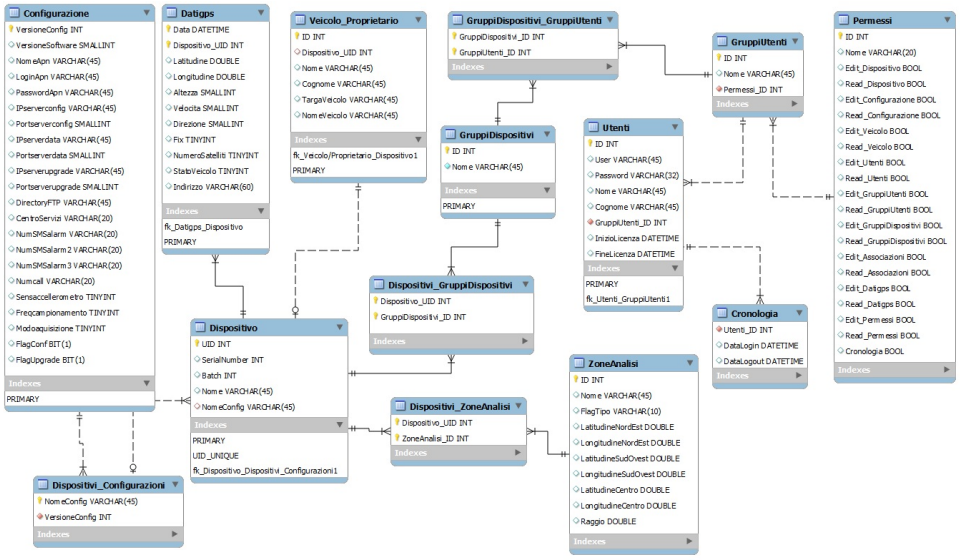

Figura 6.2: Tabelle Database

Capitolo 7

Protocollo di Comunicazione

Introduzione

Per la comunicazione tra il server e i dispositivi si è reso necessario sviluppare un protocollo per interpretare i messaggi correttamente.
La comunicazione è di tipo master/slave, con i dispositivi (nel ruolo di master) che inviano un messaggio e il server (nel ruolo di slave) che elabora il messaggio e ne invia uno in risposta con la conferma dell'avvenuta ricezione ed eventuali altre informazioni richieste dal comando.

7.1 Protocollo di Comunicazione Server

Definizione del protocollo corrispondente ai messaggi inviati dal server a un dispositivo:

- Header [3 byte]
- Lunghezza Messaggio [2 byte]
- ID Dispositivo [4 byte]
- Comando [1 byte]:
  - Saluto: 0
  - Dati GPS: 1
  - Dati GPS Storico: 2
  - Configurazione: 3
- Risposta [1 byte]:
  - KO: 0
  - OK: 1
- Notifica [1 byte]:
  - Nessuna Notifica: 0
  - Nuova Configurazione: 1
- Dati Comando (vedi seguito)
- Checksum [4 byte]

Dati Comando:

Dati GPS: Nessun Dato

Dati GPS Memorizzati: Nessun Dato

Configurazione:

- Versione Configurazione [4 byte]
- Versione Software [2 byte]
- Lunghezza Nome Apn [2 byte]
- Nome Apn [stringa]
- Lunghezza Login Apn [2 byte]

- Login Apn [stringa]
- Lunghezza Password Apn [2 byte]
- Password Apn [stringa]
- Indirizzo IP Server Configurazione [4 byte]
- Porta Server Configurazione [2 byte]
- Indirizzo IP Server Dati [4 byte]
- Porta Server Dati [2 byte]
- Indirizzo IP Server Aggiornamento [4 byte]
- Porta Server Aggiornamento [2 byte]
- Lunghezza Directory File Aggiornamento [2 byte]
- Directory File Aggiornamento [stringa]
- Sensibilità Accelerometro [1 byte]
- Frequenza Campionamento Dati GPS [1 byte]
- Modalità Acquisizione Dati GPS [1 byte]:
    - Disabilitato: 0
    - Temporale: 1
    - Accelerometro: 2

7.2 Protocollo di Comunicazione Dispositivo

Definizione del protocollo corrispondente ai messaggi inviati da un dispositivo al server:

- Header [3 byte]
- Lunghezza Messaggio [2 byte]
- ID Dispositivo [4 byte]
- Comando [1 byte]:
  - Saluto: 0
  - Dati GPS: 1
  - Dati GPS Storico: 2
  - Configurazione: 3
- Errori/Allarmi [2 byte]:
  - Nessun Errore Allarme: 0
- Dati Comando (vedi seguito)
- Checksum [4 byte]

Dati Comando:

Dati GPS:

- Fix [1 byte]:
  - Fix Non Valido: 0 e 1
  - Fix 2D: 2
  - Fix 3D: 3
- Longitudine [4 byte] (valore $/10^6$)
- Latitudine [4 byte] (valore $/10^6$)
- Altitudine [2 byte]
- Velocità km/h [2 byte]
- Direzione [1 byte] (valore dimezzato)
- Giorno [1 byte]
- Mese [1 byte]

- Anno [1 byte]
- Ore [1 byte]
- Minuti [1 byte]
- Secondi [1 byte]
- Numero Satelliti [1 byte]
- Rete GPRS [1 byte]:
  - Rete Non Disponibile: 0
  - Rete Disponibile: 1
- Stato Veicolo [1 byte]:
  - Fermo: 0
  - In Movimento: 1
- Roaming [1 byte]:
  - Non In Roaming: 0
  - In Roaming: 1
- Spare [1 byte](utilizzo futuro)

Dati GPS Memorizzati:

- Numero Sequenze Dati GPS Inviati [2 byte]
- Sequenze Dati GPS

Configurazione:

- Versione Configurazione [4 byte]

Capitolo 8

Progetto Servizio Server

8.1 Introduzione

Data la necessità di avere sempre un sistema attivo che comunichi con i dispositivi, il software server che interagisce con i dispositivi è stato progettato come un servizio windows. Questo per evitare il bisogno che vi sia un utente loggato nel sistema.
La necessità che un utente possa interagire col servizio richiede di sviluppare un software che attraverso un'interfaccia grafica permetta di attivare o sospendere il server, o visualizzare eventuali informazioni riguardanti la comunicazione con i dispositivi utili per rilevare possibili errori e per il debugging, e che inoltre fornisca la possibilità di modificare alcune impostazioni di configurazione utilizzate sia dal servizio che dal sito web; suddette informazioni saranno mantenute all'interno di un file di configurazione.
Il servizio dovrà implementare un sistema per la comunicazione in rete, detto socket, in modo da poter interagire con i dispositivi. La gestione delle comunicazioni con i dispositivi dovranno avvenire all'interno di thread per permettere la connessione con più dispositivi in contemporanea, e le informazioni ricevute dovranno essere salvate all'interno di un database. La comunicazione dovrà inoltre seguire un determinato protocollo, che è stato mostrato in precedenza. Per quanto riguarda la comunicazione tra il servizio e il software di controllo, tra i diversi sistemi di inter-process communication quello che è stato ritenuto più performante e sicuro nella comunicazione, e verrà quindi implementato, saranno le named-pipe.

8.2 Requisiti Funzionali

Ora andremo ad illustrare le principali funzionalità che saranno implementate successivamente nel servizio server.

Creazione Socket

Introduzione: il servizio dovrà aprire un sistema di comunicazione socket per comunicare con i dispositivi.

Input: indirizzo della porta su cui controllare i tentativi di accesso alla socket.

Processing: creazione e apertura di una socket in modalità server.

Output: controllo della socket per eventuali connessioni.

Connessione con un Dispositivo

Introduzione: il servizio dovrà riconoscere un tentativo di connessione da parte di un dispositivo alla socket creata

Input: ricezione segnale di collegamento.

Processing: creazione di un thread per la comunicazione con il dispositivo connesso.

Output: avvio di una funzione per la ricezione dei dati.

Ricezione Dati da un Dispositivo

Introduzione: il servizio dovrà raccogliere i dati inviati dal dispositivo.

Input: segnale ricezione dati.

Processing: i dati vengono raccolti un in buffer.

Output: avvio di una funzione di conversione del buffer contente i dati.

Conversione Dati Ricevuti

Introduzione: il servizio dovrà interpretare i dati ricevuti secondo un protocollo definito in precedenza.

Input: buffer contenete i dati.

Processing: conversione dei dati presenti nel buffer secondo il protocollo.

Output: avvio di una funzione per l'elaborazione della funzione corrispondente al comando ricevuto.

Elaborazione e Invio Risposta

Introduzione: il servizio dovrà interpretare il messaggio ricevuto e inviare una risposta al dispositivo.

Input: messaggio convertito.

Processing: elaborazione risposta in base al comando ricevuto, invio della risposta al dispositivo.

Output: ritorno alla funzione di ricezione dei dati.

Salvataggio Dati

Introduzione: il servizio dovrà salvare i dati contenenti delle informazioni riguardo la posizione.

Input: dati inerenti la posizione.

Processing: elaborazione e salvataggio dati su database.

Output: conferma salvataggio dati o segnalazione errore attraverso un'eccezione.

Creazione Named-Pipe

Introduzione: il servizio dovrà creare un sistema di comunicazione tramite named-pipe per comunicare con il software di controllo.

Input: nome della named-pipe da creare.

Processing: creazione e apertura di una named-pipe in modalità server.

Output: controllo della named-pipe per eventuali connessioni.

Connessione con Software di Controllo

Introduzione: il servizio dovrà riconoscere i tentativi di connessione da parte del sotware di controllo tramite la named-pipe.

Input: ricezione segnale di collegamento.

Processing: creazione di un thread per la comunicazione attraverso la named-pipe.

Output: avvio di una funzione per la ricezione dei messaggi.

Ricezione Messaggi dal Software di Controllo

Introduzione: il servizio dovrà ricevere i messaggi inviati dal software di controllo.

Input: segnale ricezione messaggio.

Processing: il messaggio viene interpretato e viene richiamata la funzione richiesta.

Output: esecuzione della funzione richiesta.

Elaborazione Funzione

Introduzione: il servizio dovrà eseguire la funzione richiesta e inviare una risposta al software di controllo.

Input: eventuali dati utili alla funzione.

Processing: esecuzione della funzione richiesta ed elaborazione della risposta corrispondete.

Output: invio della risposta corrispondente alla funzione richiesta, ritorno alla funzione di ricezione messaggi.

8.3 Activy Diagram

Ora andremo a mostrare l'Activity Diagram inerente al servizio con funzioni di server.

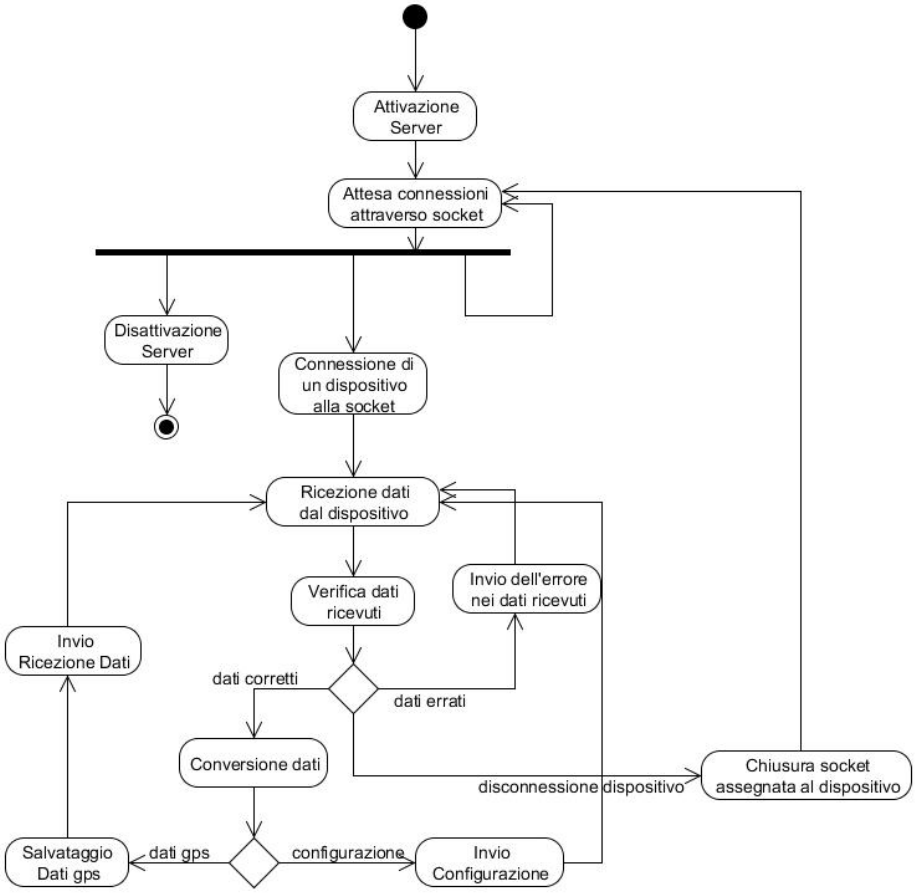

Figura 8.1: Activity Diagram Server

Capitolo 9

Progetto Software di Controllo

9.1 Introduzione

Per permettere ad un utente di interagire con il servizio si dovrà sviluppare un software di controllo composto da un'interfaccia grafica. Il software di controllo permetterà di attivare e disattivare il server, monitorare il funzionamento del server e inoltre di poter modificare i dati utilizzati dal server per il suo funzionamento che risiedono nel file di configurazione come i dati di accesso al database o la porta do comunicazione delle socket.

9.2 Requisiti Funzionali

Ora andremo ad illustrare le principali funzionalità che saranno implementate successivamente nel software di controllo del servizio.

Connessione al Servizio

Introduzione: il software di controllo deve potersi collegare al servizio all'avvio, oppure successivamente se viene persa la connessione.

Input: all'avvio o click sul pulsante riconnetti.

Processing: connessione al servizio attraverso la named-pipe.

Output: abilitazione comandi, impostazione stato connesso

Accensione Server

Introduzione: viene inviato un comando di accensione del server socket al servizio.

Input: click su pulsante accendi.

Processing: invio del comando di accensione.
Output: messaggio di conferma accensione o di essere già acceso.

Spegnimento Server

Introduzione: viene inviato un comando di spegnimento del server socket al servizio.
Input: click su pulsante spegni.
Processing: invio del comando di spegnimento.
Output: messaggio di conferma spegnimento o di essere già spento.

Richiesta Stato del Server

Introduzione: viene inviato un comando per richiedere lo stato del server (acceso/spento).
Input: click su pulsante stato server.
Processing: invio del comando di richiesta dello stato e ricezione messaggio contenente l'informazione richiesta.
Output: messaggio contenente lo stato del server.

Visualizzazione Schermata Debug

Introduzione: viene aperto un form per la visualizzazione delle informazioni necessarie al debugging.
Input: click su pulsante Schermata Debug.
Processing: apertura form debug, invio del comando di abilitazione dei messaggi di debug.
Output: visualizzazione form.

Impostazione Opzioni Debug

Introduzione: è possibile selezionare quale tipo di informazioni visualizzare nel debug.
Input: selezione delle opzioni di debug.
Processing: invio del comando di attivazione dell'opzione di debug selezionata.
Output: visualizzazione messaggi relativi all'opzione scelta nel debug.

Salvataggio Debug

Introduzione: è possibile salvare su un file i messaggi di debug presenti su schermo.
Input: click su pulsante salva.
Processing: salvataggio su di un file dei messaggi presenti su schermo.
Output: conferma salvataggio file.

Reset File di Salvataggio

Introduzione: è possibile resettare il contenuto del file contenente il salvataggio del debug.
Input: click su pulsare Clear File.
Processing: cancellazione contenuto del file di salvataggio del debug.
Output: conferma avvenuta operazione.

Visualizzazione Schermata Configurazione Database

Introduzione: viene aperto un form per la configurazione del database.
Input: click su pulsante Configura Database
Processing: apertura form Configurazione Database.
Output: visualizzazione form.

Impostazione Configurazione Database

Introduzione: è possibile impostare i dati necessari per la connessione al database.
Input: modifica dei valori nel form.
Processing: modifica dei valori all'interno del file di configurazione con i valori inseriti.
Output: conferma avvenuta operazione.

Visualizzazione Schermata Configurazione Server

Introduzione: viene aperto un form per la configurazione del server socket.
Input: click su pulsante Configura Server
Processing: apertura form Configurazione Server
Output: visualizzazione form.

Impostazione Configurazione Server

Introduzione: è possibile impostare i dati necessari per la creazione e la connessione di un server socket.
Input: modifica dei valori nel form.
Processing: modifica dei valori all'interno del file di configurazione con i valori inseriti.
Output: conferma avvenuta operazione.

9.3 Activity Diagram

Ora andremo a mostrare l'Activity Diagram inerente al software di controllo del servizio.

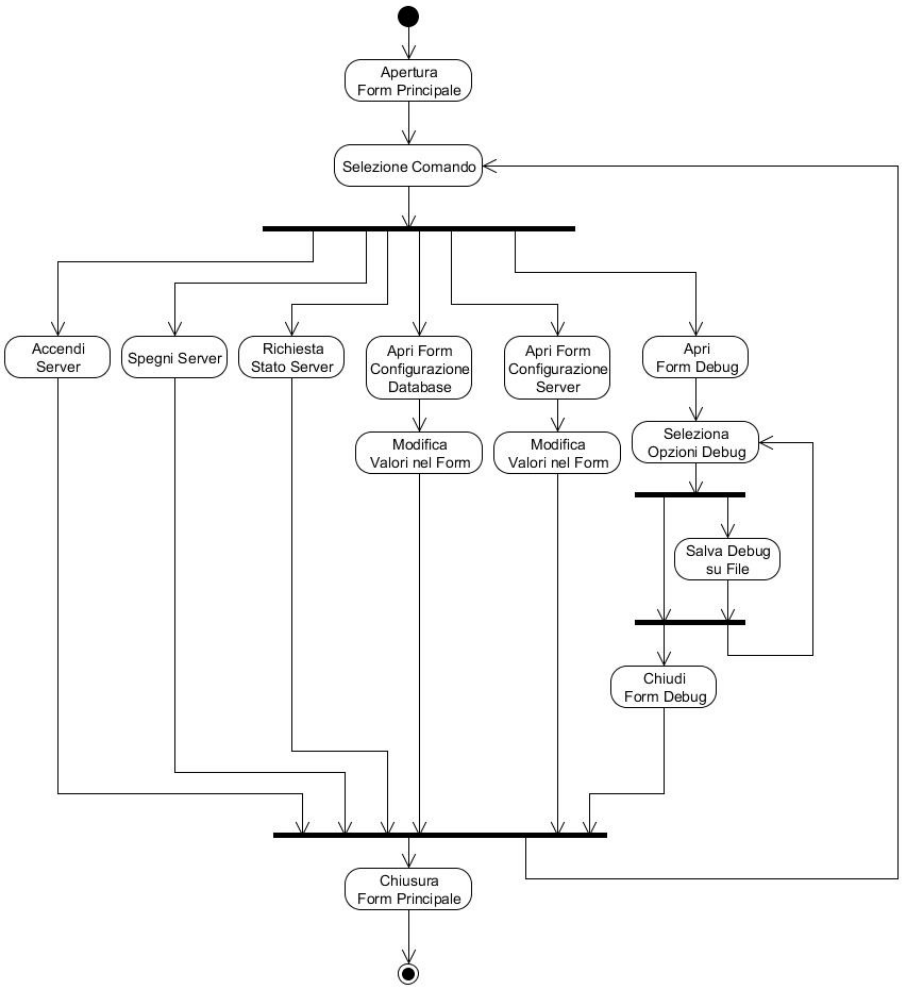

Figura 9.1: Activity Diagram Software di Controllo

Capitolo 10

Progetto Sito Web

10.1 Introduzione

Per permettere a un utente di interagire con le informazioni presenti sul database, e per rendere disponibile tali funzionalità in qualsiasi momento, esse sono state inserite all'interno di un sito web.
Il sito web è progettato per girare sul sistema IIS (Internet Information Service) interno di Windows ed essere presente su un server, inoltre potrà avere funzioni che saranno eseguite sia a lato server sia a lato client.
L'accesso al sito e le funzionalità a cui si avrà accesso saranno gestite da una fase di login tramite nome utente e password.
Il sito web è composto da:

- una schermata per effettuare l'accesso al sito,
- una schermata per la visualizzazione della posizione attuale di un dispositivo,
- una schermata per la visualizzazione del percorso effettuato da un dispositivo,
- una schermata per la visualizzazione di analisi più particolareggiate,
- una schermata per la visualizzazione del risultato delle analisi in un tabulato,
- una schermata per la visualizzazione di alcuni dati in forma tabellare,
- una schermata per l'amministrazione di tutte le informazioni.

10.2 Requisiti Funzionali

Ora andremo ad illustrare le principali funzionalità che saranno implementate successivamente.

Visualizzazione posizione attuale

Introduzione: è possibile visualizzare l'ultima posizione presente su database di uno o più dispositivi, e le relative informazioni collegate a quella determinata posizione.

Input: selezione dei dispositivi da un menu.

Processing: saranno elaborate le informazioni riguardo l'ultima posizione di un dispositivo, nel caso non siano presenti sarà informato l'utente dell'assenza di esse.

Output: inserimento all'interno di una mappa di una o più immagini nella posizione risultante; inserimento all'interno di una tabella delle informazioni correlate alle posizioni.

Visualizzazione percorso

Introduzione: è possibile visualizzare un percorso rappresentato da più coordinate, effettuato all'interno di un arco temporale.

Input: selezione dei dispositivi da un menu e inserimento di due date rappresentati un arco di tempo.

Processing: saranno ricercate su database le posizioni identificate dai dati in input per poi raggrupparle in una struttura che rappresenti un percorso.

Output: inserimento su una mappa di una spezzata che rappresenti il percorso indicando le posizioni di partenza e di arrivo; inserimento all'interno di una tabella delle informazioni correlate alle posizioni che compongono il percorso.

Visualizzazione Analisi su mappa

Introduzione: è possibile visualizzare un percorso presente all'interno di un'area indicata, effettuato durante un arco temporale.

Input: selezione dei dispositivi da un menu, inserimento delle date rappresentati un arco temporale e selezione di un'area attraverso una figura circolare o un quadrilatero.

Processing: saranno ricercate su database le posizioni identificate dai dati in input per poi raggrupparle in una struttura che rappresenti un percorso.

Output: inserimento su mappa di una spezzata che rappresenti il percorso risultante.

Visualizzazione Analisi su tabulato

Introduzione: è possibile visualizzare all'interno di un tabulato le informazioni relative alle posizioni che compongono il percorso risultante dalla funzionalità precedente.

Input: conferma visualizzazione tabulato.

Processing: sarà creato un tabulato sfruttando le informazioni già elaborate in precedenza.

Output: inserimento all'interno di una tabella delle informazioni elaborate.

Salvataggio Tabulato

Introduzione: è possibile creare un file contenente le informazioni presenti sul tabulato creato con la funzionalità precedente in modo da salvarlo in locale.

Input: conferma salvataggio.

Processing: sarà elaborato il tabulato creando un file contenente le stesse informazioni.

Output: file creato, con richiesta di salvataggio in locale.

Visualizzazione Tabelle

Introduzione: è possibile visualizzare il contenuto filtrato di alcune tabelle.

Input: selezione bottone corrispondente alla tabella desidera.

Processing: saranno filtrati i contenuti della tabella scelta ed elaborati in modo da permetterne una facile comprensione.

Output: tabella contenente i dati elaborati.

Imposta Configurazione

Introduzione: è possibile impostare una nuova configurazione nel database per i dispositivi scelti.

Input: compilazione form o selezione configurazione preesistente, click su pulsante imposta configurazione.

Processing: nel caso di una nuova configurazione verranno inseriti i dati presenti nel form nel database, altrimenti saranno associati i dispositivi con la configurazione indicata.

Output: conferma avvenuta operazione.

Eliminazione Configurazione

Introduzione: è possibile eliminare una configurazione dal database ed eventualmente selezionare una configurazione che la sostituisca nei dispositivi utilizzanti la configurazione eliminata.

Input: selezione configurazione, selezione eventuale configurazione alternativa, click su pulsante elimina.

Processing: eventuale riassegnazione di una configurazione dei dispositivi, eliminazione configurazione selezionata.

Output: conferma avvenuta operazione.

Modifica Configurazione

Introduzione: è possibile modificare una configurazione già presente nel database.

Input: selezione configurazione, modifica valori nel form, click su pulsante modifica.

Processing: modifica dei valori della configurazione con quelli ricevuti.

Output: conferma avvenuta operazione.

Inserimento Dispositivi

Introduzione: è possibile inserire nuovi dispositivi nel database.

Input: compilazione form, click su pulsante inserisci.

Processing: inserimento su database dei dati inseriti.

Output: conferma avvenuta operazione.

Eliminazione Dispositivi

Introduzione: è possibile eliminare un dispositivo presente sul database.

Input: selezione dispositivo, click su pulsante elimina.

Processing: eliminazione del dispositivo dal database.

Output: conferma avvenuta operazione.

Modifica Dispositivi

Introduzione: è possibile modificare un dispositivo presente su database.

Input: selezione dispositivo da modificare, modica valori nel form, click su pulsante modifica.

Processing: modifica dei valori nel database con i valori inseriti.

Output: conferma avvenuta operazione.

Inserimento Veicolo/Proprietario

Introduzione: è possibile inserire i dati di un nuovo veicolo e del proprietario nel database.

Input: compilazione form, click su pulsante inserisci.

Processing: inserimento dei dati inseriti nel database.

Output: conferma avvenuta operazione.

Eliminazione Veicolo/Proprietario

Introduzione: è possibile eliminare un veicolo e il suo proprietario dal database.

Input: selezione veicolo, click su pulsante elimina.

Processing: eliminazione del veicolo e del suo proprietario dal database.

Output: conferma avvenuta operazione.

Modifica Veicolo/Proprietario

Introduzione: è possibile modificare i dati di un veicolo e del suo proprietario nel database.

Input: selezione veicolo, modifica valori nel form, click su pulsante modifica.

Processing: modifica dei valori nel database con i valori inseriti.

Output: conferma avvenuta operazione.

Inserimento Gruppi Dispositivi

Introduzione: è possibile inserire un nuovo gruppo di dispositivi nel database.

Input: compilazione form, click su pulsante inserisci.

Processing: inserimento su database dei dati inseriti.

Output: conferma avvenuta operazione.

Eliminazione Gruppi Dispositivi

Introduzione: è possibile eliminare un gruppo di dispositivi selezionando un gruppo alternativo a cui riassegnare i dispositivi appartenenti al gruppo eliminato.

Input: selezione gruppo da eliminare, selezione gruppo alternativo, click su pulsante elimina.

Processing: riassegnazione del gruppo di dispositivi nei dispositivi interessati, eliminazione gruppo di dispositivi indicato.

Output: conferma avvenuta operazione.

Modifica Gruppi Dispositivi

Introduzione: è possibile modificare un gruppo di dispositivi presente su database.

Input: selezione gruppo di dispositivi da modificare, modifica valori nel form, click su pulsante modifica.

Processing: modifica dei valori su database con i valori inseriti.

Output: conferma avvenuta operazione.

Inserimento Gruppi Utenti

Introduzione: è possibile inserire un nuovo gruppo di utenti nel database.
Input: compilazione form, click su pulsante inserisci.
Processing: inserimento su database dei dati inseriti.
Output: conferma avvenuta operazione.

Eliminazione Gruppi Utenti

Introduzione: è possibile eliminare un gruppo di utenti, scegliendo se eliminare anche gli utenti appartenenti al gruppo.
Input: selezione gruppo utenti, scelta opzione se eliminare anche utenti, click su pulsante elimina.
Processing: eliminazione del gruppo di utenti e, se impostato, eliminazione degli utenti appartenenti al gruppo.
Output: conferma avvenuta operazione.

Modifica Gruppi Utenti

Introduzione: è possibile modificare un gruppo di utenti presente su database.
Input: selezione gruppo utenti da modificare, modifica valori nel form, click su pulsante modifica.
Processing: modifica dei valori su database con i valori inseriti.
Output: conferma avvenuta operazione.

Impostazione Associazioni

Introduzione: è possibile impostare le associazioni tra le tabelle del database collegate tra loro.
Input: selezione dell'associazione da modificare, modifica valori del form, click su pulsante imposta.
Processing: modifica delle associazioni nel database.
Output: conferma avvenuta operazione.

Inserimento Utenti

Introduzione: è possibile inserire un nuovo utente nel database.
Input: compilazione form, click su pulsante inserisci.
Processing: inserimento nel database dei dati inseriti
Output: conferma avvenuta operazione.

Eliminazione Utenti

Introduzione: è possibile eliminare un utente dal database.
Input: selezione utente, click su pulsante elimina.
Processing: eliminazione utente indicato dal database.
Output: conferma avvenuta operazione.

Modifica Utenti

Introduzione: è possibile modificare i dati di un utente presente sul database
Input: selezione utente, modifica valori nel form, click su pulsante modifica.
Processing: modifica dei valori su database con i valori inseriti.
Output: conferma avvenuta operazione.

Inserimento Permessi

Introduzione: è possibile inserire una nuova configurazione di permessi per gli utenti nel database.
Input: compilazione form, click su pulsante inserisci.
Processing: inserimento nel database dei dati inseriti.
Output: conferma avvenuta operazione.

Eliminazione Permessi

Introduzione: è possibile eliminare una configurazione di permessi presente su database, selezionando la configurazione di permessi con cui sostituire quella eliminata se associata a degli utenti.
Input: selezione configurazione di permessi da eliminare, selezione configurazione di permessi alternativa, click su pulsante elimina.
Processing: eliminazione della configurazione selezionata, sostituzione, riassegnazione della configurazione di permessi con quella selezionata negli utenti interessati.
Output: conferma avvenuta operazione.

Modifica Permessi

Introduzione: è possibile modificare una configurazione di permessi presente su database.
Input: selezione configurazione di permessi da modificare, modifica dei valori nel form, click su pulsante modifica.
Processing: modifica dei valori su database con i valori inseriti.
Output: conferma avvenuta operazione.

Eliminazione Posizioni

Introduzione: è possibile eliminare dal database le posizioni registrate da un dispositivo all'interno di un arco temporale.

Input: impostazione arco temporale, selezione dispositivo, click su pulsante elimina.

Processng: eliminazione dei dati corrispondenti alle opzioni indicate.

Output: conferma avvenuta operazione.

Visualizzazione Cronologia Accessi

Introduzione: è possibile visualizzare gli accessi al sito avvenuti durante un arco temporale da uno o più utenti.

Input: impostazione arco temporale, selezione utenti, click su pulsante mostra.

Processing: saranno ricercate su database i dati corrispondenti alle opzioni indicate e poi elaborate in una tabella.

Output: tabella contenente i dati elaborati.

Eliminazione Cronologia Accessi

Introduzione: è possibile eliminare dal database le informazioni riguardanti gli accessi al sito avvenuti all'interno di un arco temporale da parte di uno o più utenti.

Input: impostazione arco temporale, selezione utenti, click su pulsante elimina.

Processing: eliminazione dei dati corrispondenti alle opzioni inserite.

Output: conferma avvenuta operazione.

10.3 Activity Diagram

Ora andremo a mostrare l'Activity Diagram inerente alle varie funzionalità del sito.

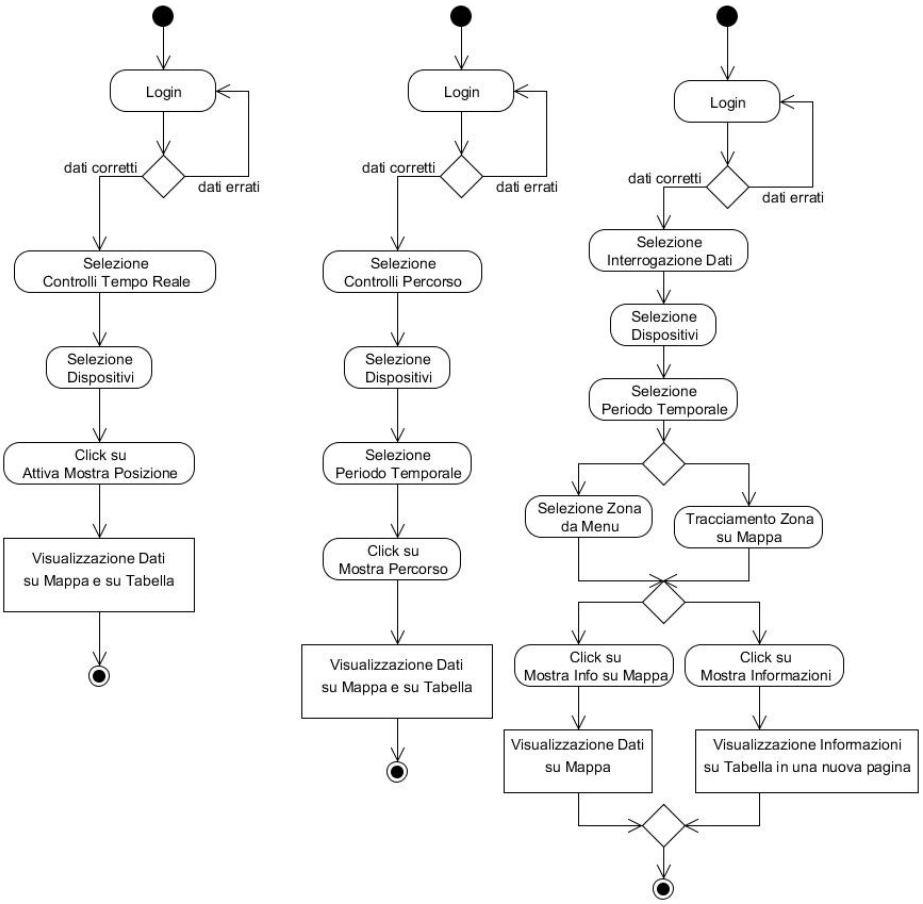

Figura 10.1: Activity Diagram Funzioni Real Time, Percorso e Interrogazioni

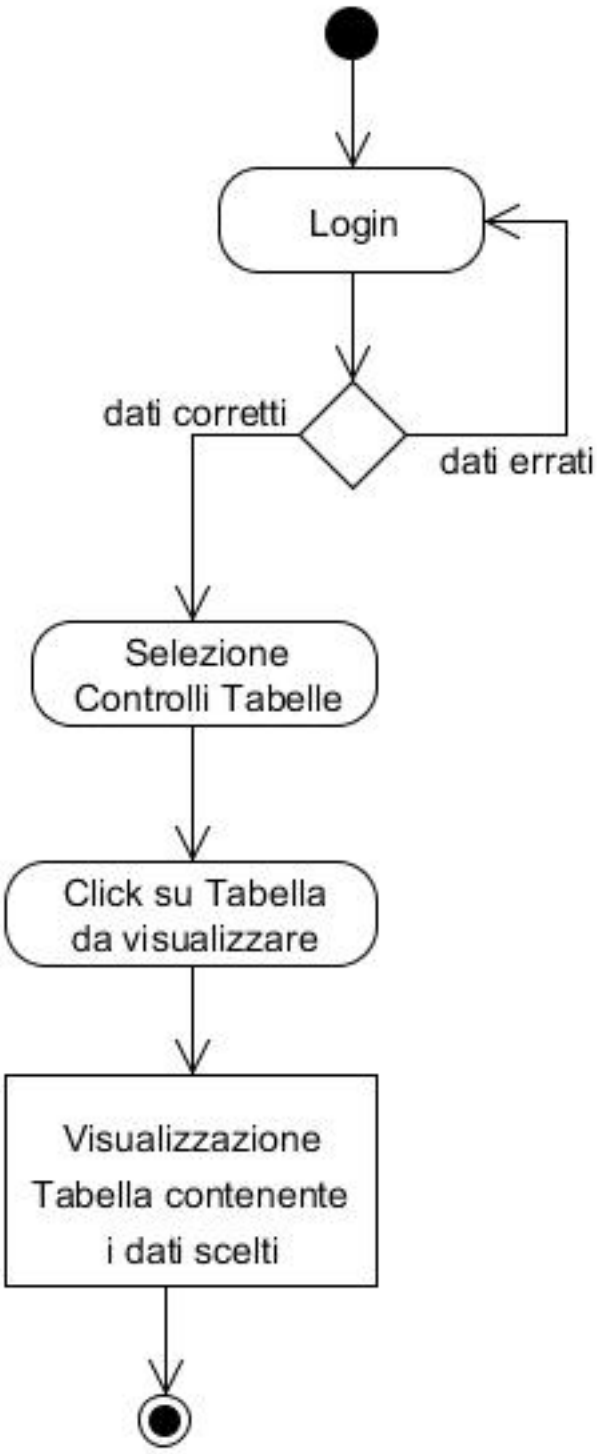

Figura 10.2: Activity Diagram Funzione di Visualizzazione Tabelle

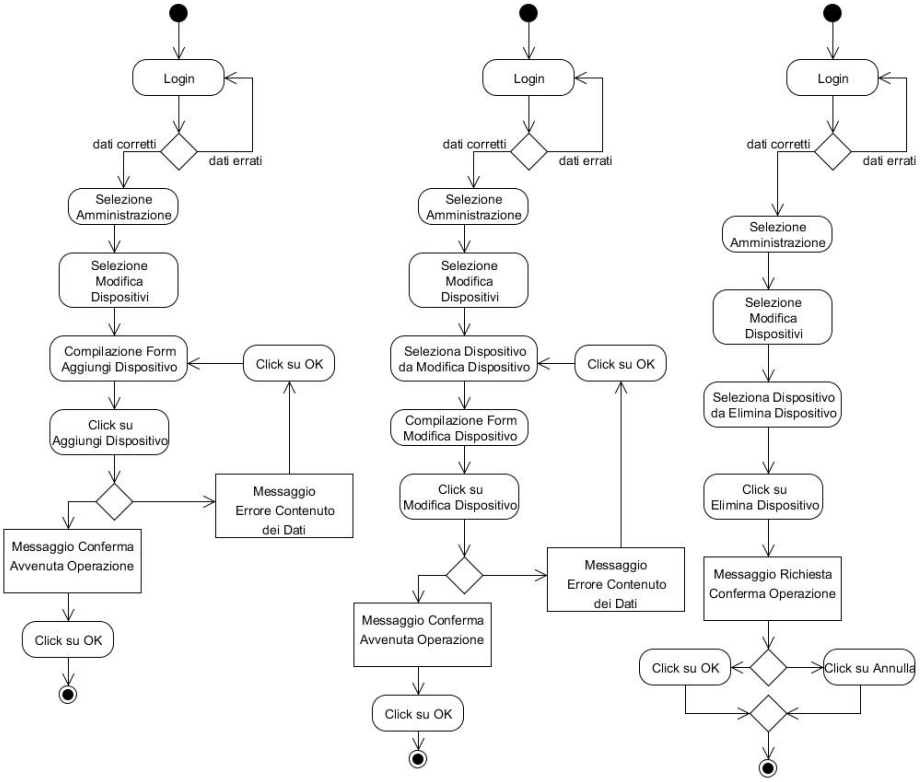

Figura 10.3: Activity Diagram Funzioni Aggiungi, Modifica ed Elimina Dispositivo

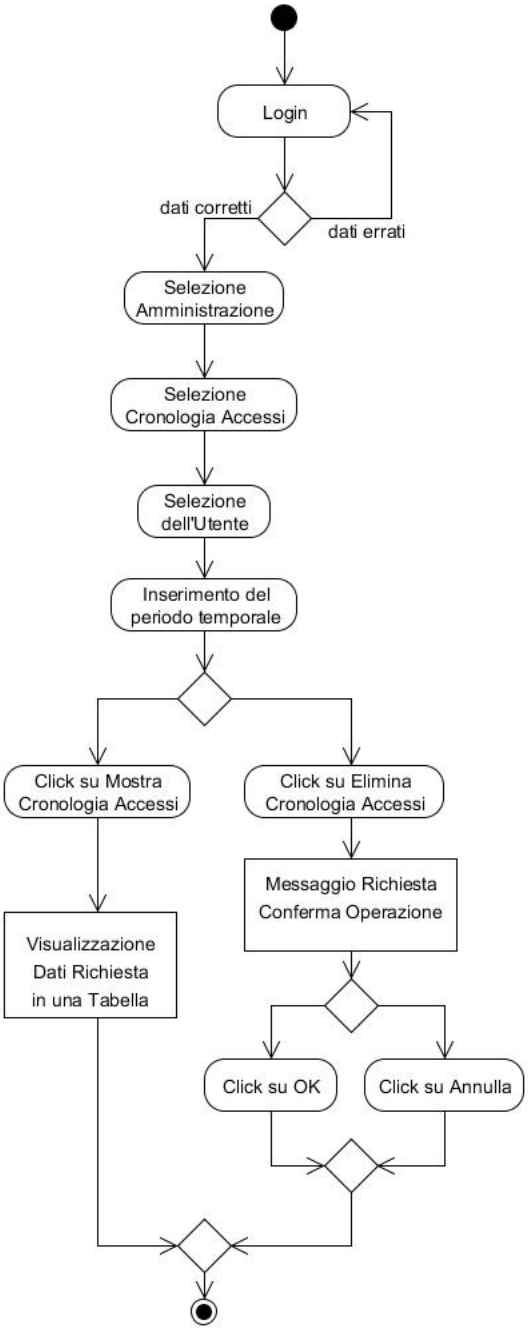

Figura 10.4: Activity Diagram Funzioni Cronologia Accessi

Parte III

Implementazione del Sistema

Capitolo 11

Implementazione del Servizio Server

Ora andremo ad analizzare il codice e le scelte di implementazione del servizio che funge da server. Data la grande quantità di codice presente saranno analizzate solo alcune parti ritenute di particolare interesse. Per svilupparlo si è utilizzato Visual Studio, un ambiente di sviluppo della Microsoft e si è utilizzato il C#.

11.1 Scelte di Implementazione

Per la comunicazione in rete tra il servizio e i dispositivi ci si è avvalsi delle classi *TcpListener* e *TcpClient* interne del c# che permettono una comunicazione attraverso le socket, mentre per la comunicazione interna tra il sevizio e il software di controllo si è utilizzato le classi *NamedPipeServerStream* e *NamedPipeClientStream*.
Per interagire con il database, visto che si è utilizzato *MySQL*per sviluppare e gestire il database, si è dovuto utilizzare le classi *MySqlConnection*, *MySqlCommand*, *MySqlDataAdapter* e *MySqlException*, rese disponibili dalla dll *MySql.Data* fornita da *MySQL*.

11.2 Classe Program

La classe *Program* è la classe di ingresso dell'applicazione.
Per poter utilizzare il sistema di debug interno di Visual Studio si è creata una maschera che permette di far partire il sistema server come se fosse un'applicazione (invece che come servizio), data l'impossibilità di utilizzare il debugger su servizi windows.

Codice 11.1: Classe Program

```
#define SERVICE
//#define PROGRAM
...

...
static class Program
{
  static void Main(string[] args)
  {
#if(PROGRAM)
    try
    {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new FormProgram());
      return;
    }
    catch (Exception)
    {
    }
#endif

#if(SERVICE)
      ServiceBase[] ServicesToRun;
      ServicesToRun = new ServiceBase[]
      {
        new Service()
      };
      ServiceBase.Run(ServicesToRun);
#endif
    }
  }
```

11.3 Classe Service

La classe *Service* definisce le operazioni compiute dal servizio. La funzione*OnStart()* definisce le operazioni che esegue all'avvio del servizio, mentre la funzione *OnStop* definisce le operazioni che esegue allo spegnimento del servizio. Per mantenere l'esecuzione del servizio sempre attiva viene creato

un thread che avvia il server socket. Allo spegnimento del servizio invece il server socket viene disattivato assieme alle funzionalità di debug e di log.

11.4 Classe FormProgram

La classe *FormProgram* viene utilizzata quando si avvia il programma in forma di applicazione e consiste in un semplice form vuoto che si occupa di attivare il server socket all'avvio e di disattivarlo alla chiusura del form, oltre a chiudere le funzioni di debug e di log.

11.5 Classe UtilThread

La classe *UtilThread* contiene le funzionalità che permettono di avviare e chiudere il server socket, e gestire la ricezione dei dati dai dispositivi. Il tutto viene gestito all'interno di thread per permettere la comunicazione in contemporanea con più dispositivi.

Funzione startServer

La funzione *starServer* avvia un server socket e un thread che, attraverso la funzione *AcceptClient*, attende ripetutamente la connessione di un client socket.

Funzione endServer

La funzione *endServer* interrompe il thread che attende la connessione dei client e chiude il server socket.

Funzione AcceptClient

La funzione *AcceptClient* consiste in un ciclo che si mantiene in attesa di un client socket, e una volta connesso crea un nuovo thread che, attraverso la funzione *getData*, riceve i dati dal cliet.

Codice 11.2: Funzione AcceptClient

```
public void AcceptClient()
  {
    int temp = 0;
    try
    {
```

```
        while (true)
        {
          infoThreads infothread;
          debug.WriteLog("AcceptClient", "attendo client", 1);
          TcpClient client = this.server.AcceptTcpClient();

          Thread streamThread = new Thread(new ParameterizedThreadStart(getData)
        );
          streamThread.Name = "Client " + temp;
          client.ReceiveTimeout = 60000;
          client.SendTimeout = 500;

          infothread.ClientThread = streamThread;
          infothread.ClientTcp = client;
          infothread.cont = listConnection.Count;

          listConnection.Add(infothread);
          streamThread.Start(infothread);
          temp++;

          if (debug != null && debug.ScreenDebug)
          {
            debug.WriteScreen("connesso " + streamThread.Name);
            debug.WriteScreen("connessi " + listConnection.Count + " client");
          }
          debug.WriteLog("AcceptClient", "connessione client " + temp, 1);
          debug.WriteLog("AcceptClient", "client attualmente connessi: " +
        listConnection.Count, 1);

        }
      }
      catch (ThreadInterruptedException t)
      {
        if (debug != null && debug.OptScrShowError)
        {
          debug.WriteScreen(t.ToString());
        }

        foreach (infoThreads connection in listConnection)
        {
          connection.ClientThread.Abort();
          connection.ClientTcp.Close();
        }
        listConnection.Clear();
        debug.WriteLog("AcceptClient", "chiusura totale thread Exception: " + t, 1);
        return;
      }
      catch (Exception e)
      {
        if (debug != null && debug.OptScrShowError)
        {
          debug.WriteScreen(e.ToString());
        }
```

```
58        foreach (infoThreads connection in listConnection)
          {
60          connection.ClientThread.Abort();
            connection.ClientTcp.Close();
62        }
          listConnection.Clear();
64
          debug.WriteLog("AcceptClient", "chiusura totale thread Exception: " + e, 1);
66        return;
        }
68    }
```

Funzione getData

La funzione *getData* si occupa di leggere i dati inviati dal client socket e di inserirli in un buffer, successivamente attraverso la classe Parser verranno convertiti i byte nel messaggio secondo il protocollo. La lettura dei dati avviene all'interno di un ciclo da cui si esce solo nel caso di errore o disconnessione del client. La disconnessione da un client avviene quando scade il tempo di timeout di ricezione o quando è il client a disconnettersi da server; questi casi vengono riconosciuti da server ricevendo in lettura 0 byte o facendo scattare un'eccezione. All'uscita del ciclo viene rimosso dalla lista l'elemento corrispondente al client associato alla funzione.

Codice 11.3: Funzione getData

```
public void getData(object info)
2     {
        Parser parsing = new Parser();
4       infoThreads infothread = (infoThreads)info;

6       TcpClient client = infothread.ClientTcp;
        client.ReceiveBufferSize = 100 * 1024;
8       NetworkStream stream = client.GetStream();
        byte[] bufferGlobal = new byte[1024];
10      int byteread = 0;

12      while (true)
        {
14        try
          {
16          try
            {
18
              byteread = stream.Read(bufferGlobal, 0, bufferGlobal.Length);
20
              if (debug != null && debug.OptScrShowCountByte)
22            {
                debug.WriteScreen("Letti " + byteread.ToString() + " byte");
24            }
```

```
26          if (byteread == 0)
            {
28            infothread.ClientTcp.Close();

30            if (debug != null && debug.OptScrShowOtherInfo)
              {
32              debug.WriteScreen("Scaduto timeout o nessun dato ricevuto: chiudo
        connessione client " + infothread.cont);
              }
34            debug.WriteLog("getdata", "timeout scaduto disconnessione thread: " +
        infothread.ClientThread.Name, 1);
              break;
36          }
          }
38        catch (IOException i)
          {
40          infothread.ClientTcp.Close();

42          if (debug != null && debug.OptScrShowError)
            {
44            debug.WriteScreen("Eccezione client" + infothread.cont + " chiusura thread:
        " + i.ToString());
            }
46          debug.WriteLog("getdata", "disconnessione thread: " + infothread.
        ClientThread.Name + " IOException: " + i, 1);
            break;
48        }
          if (debug != null && debug.OptScrShowDataReceived)
50        {
            string text = "";
52          for (int i = 0; i < byteread; i++)
            {
54            text += bufferGlobal[i].ToString("X2");
              text += " ";
56          }
            debug.WriteScreen("Ric: " + text);
58        }
          parsing.Reading(bufferGlobal, byteread, infothread.ClientThread.Name, stream);
60      }
        catch (ThreadAbortException x)
62      {
          if (debug != null && debug.OptScrShowError)
64        {
            debug.WriteScreen(x.ToString());
66        }
          infothread.ClientTcp.Close();
68        debug.WriteLog("getdata", "disconnessione thread: " + infothread.ClientThread
        .Name + " ThreadAbortException: " + x, 1);
          break;
70      }
        catch (Exception e)
72      {
          if (debug != null && debug.OptScrShowError)
```

```
74          {
              debug.WriteScreen(e.ToString());
76          }
            debug.WriteLog("getdata", "disconnessione thread: " + infothread.ClientThread
          .Name + " Exception: " + e, 1);
78          infothread.ClientTcp.Close();
            break;
80        }
        }
82      listConnection.Remove(infothread);
        if (debug != null && debug.OptScrShowOtherInfo)
84      {
          debug.WriteScreen("elimino thread da lista");
86      }
        debug.WriteLog("getdata", "eliminazione thread: " + infothread.ClientThread.
          Name + " da lista", 1);
88    }
```

11.6 Classe Parser

La classe Parser si occupa di interpretare il buffer dei dati ricevuti e di inviare la risposta al client socket in base ai comandi ricevuti.

Funzione Reading

La funzione *convertData* interpreta i byte dell'header e poi quelli della lunghezza del messaggio, e successivamente compone il messaggio finché non sono stati ricevuti tutti i byte indicati dalla lunghezza del messaggio. Una volta composto interamente il messaggio controlla il checksum; nel caso che il checksum corrisponda passa il buffer contenente il messaggio alla funzione *convertData*, altrimenti invia un messaggio indicante l'errore nella ricezione al client.

Funzione convertData

La funzione *convertData* controlla che l'id del dispositivo corrisponda a uno dei dispositivi presenti nel database, successivamente controlla il comando ricevuto e in base a questo controlla che la lunghezza del messaggio corrisponda alle informazioni che dovrebbe contenere il messaggio, infine, dopo aver inviato in risposta al client di aver ricevuto correttamente il messaggio, gestisce le operazioni da effettuare.

Funzione GetPos

La funzione *GetPos* si occupa di interpretare un messaggio contenente delle informazioni riguardo la posizione del dispositivo e del suo stato, una volta interpretati i byte li raccoglie in una posizione attraverso la classe *DatiGps* e per mezzo della classe *UtilDB* inserisce le informazioni al database.

Funzione GetPosHistory

La funzione *GetPosHistory* si occupa di interpretare un messaggio contenente una seria di informazioni che non è stato possibile ricevere in precedenza, questo avviene effettuando le stesse operazioni della funzione *GetPos* per il numero di posizioni indicate dal messaggio.

Funzione createResponseOK_KO

La funzione *createReponseOK_KO* si occupa di creare una risposta indicante se il messaggio è stato ricevuto correttamente o era corretto, oppure se il messaggio non è stato considerato valido. Inoltre indica le eventuali notifiche che devono essere segnalate al dispositivo.

Funzione createResponseConfig

La funzione *createResponseConfig* si occupa di creare un messaggio contenente i valori della configurazione di un dispositivo.

11.7 Classe UtilityDB

La classe *UtilityDB* contiene le funzionalità per leggere e inserire informazioni nel database. Di questa classe ne esiste una sola istanza che viene condivisa in modo da lasciare la gestione degli accessi al database alle varie funzioni.

Funzione ExecuteChangeOnDb

La funzione *ExecuteChangeOnDb* si occupa di effettuare un inserimento o una modifica nel database attraverso una query passata come stringa. E' la funzione che effettivamente interagisce col database.

Codice 11.4: Funzione ExecuteChangeOnDb

```
private bool ExecuteChangeOnDb(string queryString)
  {
    lock (lockoperation)
    {
      try
      {
        if (connection.State != ConnectionState.Open)
        {
          connection.Open();
        }
      }
      catch (MySqlException mysqlexcp)
      {
        debug.WriteLog("ExecuteChangeOnDb()", "MySqlException: " + mysqlexcp, 1)
      ;
        if (debug != null && debug.OptScrShowError)
        {
          debug.WriteScreen(mysqlexcp.ToString());
        }
        return false;
      }
      catch (Exception e)
      {
        debug.WriteLog("ExecuteChangeOnDb()", "Exception: " + e, 1);
        if (debug != null && debug.OptScrShowError)
        {
          debug.WriteScreen(e.ToString());
        }
        return false;
      }

      try
      {
        MySqlCommand ins = new MySqlCommand(queryString, connection);
        int ret = ins.ExecuteNonQuery();
        if (ret < 0)
        {
          return false;
        }
        return true;
      }
      catch (MySqlException mysqlexcp)
      {
        debug.WriteLog("ExecuteChangeOnDb()", "MySqlException: " + mysqlexcp, 1)
      ;
        if (debug != null && debug.OptScrShowError)
        {
          debug.WriteScreen(mysqlexcp.ToString());
        }
        return false;
      }
      catch (Exception e)
      {
```

```
52          debug.WriteLog("ExecuteChangeOnDb()", "Exception: " + e, 1);
            if (debug != null && debug.OptScrShowError)
54          {
              debug.WriteScreen(e.ToString());
56          }
            return false;
58        }
        }
60    }
```

Funzione ExecuteSelectOnDb

La funzione *ExecuteSelectOnDb* si occupa di effettuare l'estrazione di informazioni dal database attraverso una query passata come stringa. Le informazioni estratte vengono ottenute attraverso un DataSet, una struttura che può contenere diverse tabelle, ma dato che le informazioni richieste sono contenute all'interno di una sola tabella, è solo questa che viene ritornata dalla funzione. E' la funzione che effettivamente interagisce col database.

Codice 11.5: Funzione ExecuteSelectOnDb

```
  private DataTable ExecuteSelectOnDb(string queryString)
2   {
      DataSet table = new DataSet();
4     lock (lockoperation)
      {
6       try
        {
8         if (connection.State != ConnectionState.Open)
          {
10          connection.Open();
          }
12      }
        catch (MySqlException mysqlexcp)
14      {
          debug.WriteLog("ExecuteSelectOnDb()", "MySqlException: " + mysqlexcp, 1);
16        if (debug != null && debug.OptScrShowError)
          {
18          debug.WriteScreen(mysqlexcp.ToString());
          }
20        return null;
        }
22      catch (Exception e)
        {
24        debug.WriteLog("ExecuteSelectOnDb()", "Exception: " + e, 1);
          if (debug != null && debug.OptScrShowError)
26        {
            debug.WriteScreen(e.ToString());
28        }
          return null;
30      }
```

```
32        try
          {
34          MySqlDataAdapter tableadapter = new MySqlDataAdapter(queryString,
        connection);
            tableadapter.Fill(tableset);
36          return tableset.Tables[0];
          }
38        catch (MySqlException mysqlexcp)
          {
40          debug.WriteLog("ExecuteSelectOnDb()", "MySqlException: " + mysqlexcp, 1);
            if (debug != null && debug.OptScrShowError)
42          {
              debug.WriteScreen(mysqlexcp.ToString());
44          }
            return null;
46        }
          catch (Exception e)
48        {
            debug.WriteLog("ExecuteSelectOnDb()", "Exception: " + e, 1);
50          if (debug != null && debug.OptScrShowError)
            {
52            debug.WriteScreen(e.ToString());
            }
54          return null;
          }
56      }
      }
```

11.8 Classe DatiGps

La classe *DatiGps* viene utilizzata per raccogliere le informazioni riguardanti una posizione. Tutti i valori vengono gestiti che se fossero delle proprietà per leggere o assegnare un valore facilmente alle variabili, che restano private alla classe.

11.9 Classe ConfigDispositivo

La classe *ConfigDispositivo* viene utilizzata per raccogliere le informazioni riguardanti la configurazione di un dispositivo. Tutti i valori vengono gestiti che se fossero delle proprietà per leggere o assegnare un valore facilmente alle variabili, che restano private alla classe.

11.10 Classe UtilityDebug

La classe *UtilitDebug* contiene tutte le funzionalità utili per il debugging e per poter comunicare col servizio da un software esterno. Di questa classe ne esiste una sola istanza che viene condivisa da tutte le altre classi in modo che i vari thread non interferiscano tra loro nelle funzionalità. Le informazioni che possono essere intercettate e stampate sono gestite attraverso i valori di varie proprietà, i cui valori verranno modificati dal software esterno.

Funzione WriteLog

La funzione *WriteLog* scrive le principali operazioni e gli eventuali errori avvenuti in modo da tenere traccia di eventuali bug che vengono identificati durante l'utilizzo normale del software.Per fare ciò si appoggia alla classe *LogFile* di cui si parlerà successivamente.

Funzione WriteScreen

La funzione *WriteScreen* viene utilizzata per scrivere le varie informazioni di debug sul software esterno di controllo. Per permettere il passaggio sia di informazioni che di comandi tra i software viene passata una classe *ObjectMessage* serializzata che verrà trattata più avanti. Per fare ciò utilizza la classe *NamedPipeUtility* che gestisce le namedpipe di cui si parlerà successivamente.

11.11 Classe LogFile

La classe *LogFile* non fa altro che esporre una funzione statica per scrivere su un file, controllando che sia possibile accedere al file e gestendo i tentativi di scriverci all'interno da più thread contemporaneamente. Il file di log così creato viene archiviato e rinominato con la data odierna ogni giorno.

11.12 Classe NamedPipeWrapper

La classe *NamedPipeWrapper* è una classe atratta e contiene le funzioni base necessarie all'utilizzo delle named-pipe; verrà ereditata sia dalla classe che utilizza le named-pipe come server, cioè il servizio, sia dalla classe che utilizza le named-pipe come client, cioè il software di controllo. La classe

implementa le funzioni per far partire e per fermare il thread che gestisce la comunicazione e le funzioni di lettura e scrittura.

Codice 11.6: Classe NamedPipeWrapper

```
public abstract class NamedPipeWrapper<T> where T : PipeStream
 {
...
...

   public void Read()
   {
     try
     {
       if (Pipe != null)
       {
         Object msg = ReadPipe(Pipe);
         if (msg != null)
         {
           PipeReceivedData(msg);
         }
       }
     }
     catch (Exception e)
     {
     }
   }

   protected Object ReadPipe(PipeStream outStream)
   {
     try
     {
       if (outStream != null)
       {
         byte[] buffer = new byte[BUFFER_SIZE];
         int byteMessage = outStream.Read(buffer, 0, buffer.Length);
         if (byteMessage == 0)
         {
           Stop();
           return null;
         }
         Array.Resize(ref buffer, byteMessage);
         return ObjectSerializator.ByteArrayToObject(buffer);
       }
       return null;
     }
     catch (Exception e)
     {
       return null;
     }
   }

   public void Write(Object obj)
   {
```

```
      if (Pipe.IsConnected == true && Pipe.CanWrite == true && obj != null)
      {
        WritePipe(obj, Pipe);
      }
    }

    protected void WritePipe(Object outObject, PipeStream inStream)
    {
      try
      {
        Stream pipeWriter = inStream;
        byte[] buffer = new byte[BUFFER_SIZE];
        buffer = ObjectSerializator.ObjectToByteArray(outObject);
        pipeWriter.Write(buffer, 0, buffer.Length);
        pipeWriter.Flush();
      }
      catch (Exception e)
      {
        return;
      }
    }

  }
```

11.13 Classe NamedPipeUtility

La classe *NamedPipeUtility* eredita la classe *NamedPipeWrapper* e implementa le funzioni per la creazione di una named-pipe con funzione di server e le funzioni eseguite dal thread.

Codice 11.7: Classe NamedPipeUtility

```
class NamedPipeUtility : NamedPipeWrapper<NamedPipeServerStream>
  {
...

...
    protected override void PipeThreadWorking(Object data)
    {
      try
      {
        Pipe.WaitForConnection();
        Connected = true;
        while (true)
        {
          Read();
        }

      }
      catch (ThreadAbortException e)
      {
```

```
20          if (!closePipe)
            {
22            Start();
            }
24        }
          catch (Exception e)
26        {

28        }
        }
30    }
```

11.14 Classe ObjectMessage

La classe *ObjectMessage* costituisce un messaggio da passare tra il servizio e il software di controllo, dopo essere stata serializzata. La funzione *getObjectData* viene utilizzata automaticamente dal sistema per de-serializzare il messaggio ricevuto.

Codice 11.8: Classe ObjectMessage

```
[Serializable()] public class ObjectMessage : ISerializable
2  {
     private int _command;
4    private string _message;
     public string _AssemblyDest;
6
     public ObjectMessage(string dest)
8    {
       _AssemblyDest = dest;
10   }
     public ObjectMessage(SerializationInfo info, StreamingContext context)
12   {
       this.Command = info.GetInt32("_command");
14     this.Message = info.GetString("_message");
     }
16
     public int Command
18   {
       get { return _command; }
20     set { _command = value; }
     }
22   public string Message
     {
24     get { return _message; }
       set { _message = value; }
26   }
     public void GetObjectData(SerializationInfo info, StreamingContext context)
28   {
       info.AssemblyName = this._AssemblyDest;
30     info.AddValue("_command", _command);
```

```
      info.AddValue("_message", _message);
32    }
    }
```

11.15 Classe ObjectSerializator

La classe *ObjectSerializator* viene utilizzata per serializzare e deserializzare un oggetto. All'interno di questo sistema viene utilizzata per serializzare e deserializzare la classe *ObjectMessage* durante la comunicazione attraverso le named-pipe.

Codice 11.9: Classe ObjectSerializator

```
1  class ObjectSerializator
   {
3    public static byte[] ObjectToByteArray(Object obj)
     {
5      if (obj != null)
       {
7        BinaryFormatter binForm = new BinaryFormatter();
         MemoryStream memStream = new MemoryStream();
9        binForm.Serialize(memStream, obj);
         return memStream.ToArray();
11     }
       return null;
13   }

15   public static Object ByteArrayToObject(byte[] Bytes)
     {
17     MemoryStream memStream = new MemoryStream();
       BinaryFormatter binForm = new BinaryFormatter();
19     memStream.Write(Bytes, 0, Bytes.Length);
       memStream.Seek(0, SeekOrigin.Begin);
21     Object obj = (Object)binForm.Deserialize(memStream);
       return obj;
23   }
   }
```

Capitolo 12

Implementazione del Software di Controllo

Ora andremo ad analizzare il codice e le scelte di implementazione del software che permette di interagire col servizio. Data la grande quantità di codice presente saranno analizzate solo alcune parti ritenute di particolare interesse.

12.1 Struttura

Il software di controllo è composto da quattro schermate.
Una schermata principale che permette di riconnettersi al servizio server,

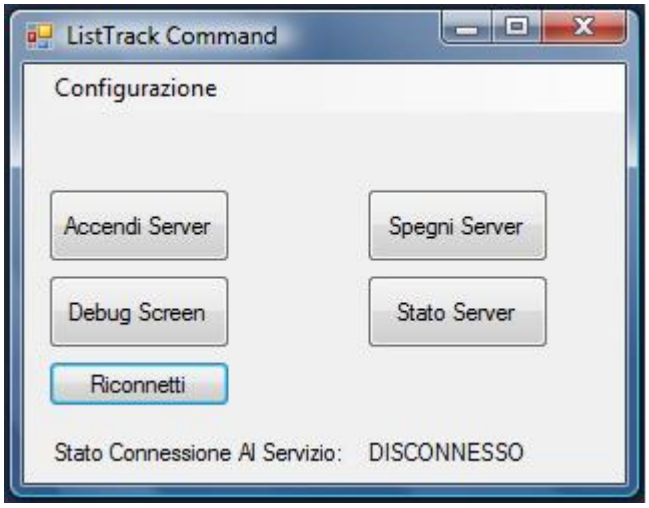

Figura 12.1: Schermata Principale

attivare o disattivare il server, conoscere lo stato di funzionamento del server e di accedere alle altre schermate dedicate ad altre funzioni.
Due schermate adibite all'inserimento dei parametri di connessione al database e ai parametri di connessione al server.
Una schermata in cui è possibile visualizzare le varie informazioni di debug raccolte durante il funzionamento del sito.

12.2 Classe MainForm

La classe *MainForm* costituisce la schermata principale del software. Essa permette di inviare i comandi per far partire e fermare il server socket e per visualizzarne lo stato, di aprire le schermate per configurare la porta di comunicazione del server e per configurare i dati di accesso al database, e infine di aprire una schermata su cui è possibile visualizzare tutte le operazioni effettuate dal servizio.

Funzione MainFormLoad

La funzione *MainFormLoad* viene eseguita al caricamento della schermata e si collega, attraverso la classe *NAmedPipeComunicator*, a una named

pipe in modalità client, inoltre crea degli eventi che scatteranno quando saranno ricevuti dei dati dal servizio e quando si ha un collegamento al servizio attraverso la named-pipe.

Codice 12.1: Funzione MainFormLoad

```
private void MainForm_Load(object sender, EventArgs e)
  {
    notifyIconHome.Icon = new Icon(@"img\black_server.ico");
    if (pipeClient == null)
    {
      pipeClient = new NamedPipeComunicator();
      pipeClient.PipeReceivedData += new NamedPipeComunicator.
      PipeReceivedDataEventHandler(ShowReceivedPipeData);
      pipeClient.PipeConnected += new NamedPipeComunicator.
      PipConnectedEventHandler(PipeChangeState);
      pipeClient.Start();
    }
  }
```

Funzione ShowReceivedPipeData

La funzione *ShowReceivedPipeData* viene richiamata alla ricezione di un messaggio attraverso la named-pipe. Nel caso che sia visibile la schermata di ricezione delle informazioni il messaggio viene mostrato su di essa attraverso la funzione *SetTextListBox_ThSafe*, altrimenti viene mostrato all'interno di un pop-up. Il messaggio è composta dalla classe *ObjectMessage* serializzata vista in precedenza nell'implementazione del servizio.

Codice 12.2: Funzione ShowReceivedPipeData

```
void ShowReceivedPipeData(Object obj)
  {
    if (obj != null)
    {
      ObjectMessage objMsg = (ObjectMessage)obj;
      if (DebugScreen != null && DebugScreen.listBoxOutput.Visible && !
      DebugScreen.IsDisposed)
      {
        SetTextListBox_ThSafe(DebugScreen.listBoxOutput, objMsg.Message);
      }
      else
      {
        MessageBox.Show(objMsg.Message);
      }
    }
  }
```

Funzione SetTextListBox_ThSafe

La funzione *SetTextListBox_ThSafe* permette di scrivere, anche quando più thread cercano di farlo contemporaneamente, all'interno di una listbox presente nella schermata di ricezione delle informazioni.

Codice 12.3: Funzione SetTextListBox_ThSafe

```
delegate void delThSafe_SetListBox(ListBox ctrl, string Text);
  private void SetTextListBox_ThSafe(ListBox ctrl, string Text)
  {
    Form ownerform = ctrl.FindForm();
    if (ownerform == null)
      return;

    if (ownerform.IsDisposed)
      return;

    if (ctrl.InvokeRequired)
    {
      delThSafe_SetListBox d = new delThSafe_SetListBox(SetTextListBox_ThSafe);
      try
      {
        ownerform.Invoke(d, new object[] { ctrl, Text });
      }
      catch (Exception)
      {
      }
    }
    else
    {
      int index = ctrl.Items.Add(Text);
      ctrl.SelectedIndex = ctrl.Items.Count - 1;
    }
  }
```

Funzione buttonDebugScreen_Click

La funzione *buttonDebugScreen_Click* è interessante in quanto, oltre ad aprire il form contenente la schermata di visualizzazione delle informazioni, crea degli eventi che vengono generati dalla classe *DebugScreen* e che vengono gestiti dalle funzioni *ChangeOption* e *CloseDebugScreen*.
Gli eventi permettono di riconoscere quali opzioni per la visualizzazione sono state selezionate e quindi cosa comunicare al servizio, e di riconoscere la chiusura della schermata per informare il servizio della disabilitazione di tutte le opzioni selezionate precedentemente.

12.3 Classe FormDebugScreen

La classe *FormDebugScreen* costituisce la schermata di visualizzazione delle informazioni ricevute attraverso la named-pipe e viene utilizzata principalmente per il debug del servizio. Le informazioni vengono inserite all'interno di una listbox ed è possibile salvarne il contenuto all'interno di un file di testo. E' possibile impostare quali informazioni si desidera visualizzare attraverso delle checkbox, che invieranno il comando relativo al servizio per indicargli di inviare i messaggi relativi all'opzione selezionata. La maggior parte delle funzioni presenti all'interno di questa classe non fanno altro che segnalare l'evento di una modifica delle opzioni alla classe *MainForm*, in quanto l'unica a comunicare con il servizio per mezzo della classe *NamedPipeComunicator*.

12.4 Classe NamedPipeComunicator

La classe *NamedPipeComunicator*, come la classe *NamedPipeUtility* vista in precedenza, eredita la classe *NamedPipeWrapper* e implementa le funzioni per la creazione di una named-pipe con funzione di client e le funzioni eseguite dal thread.

Codice 12.4: Classe NamedPipeComunicator

```
class NamedPipeComunicator : NamedPipeWrapper<NamedPipeClientStream>
 {
...

   protected override void PipeThreadWorking(object data)
   {
     try
     {
       Pipe.Connect();
       Connected = true;
       PipeConnected();
       while (true)
       {
         Read();
       }
     }
     catch (ThreadAbortException e)
     {
     }
     catch (Exception e)
     {
     }
   }
 }
```

Capitolo 13

Implementazione del Sito Web

Ora andremo ad analizzare il codice e le scelte di implementazione del sito web. Data la grande quantità di codice presente saranno analizzate solo alcune parti ritenute di particolare interesse.

13.1 Scelte di Implementazione

Parte del sito è stata sviluppata in Asp.NET e C# per poter sfruttare alcune potenzialità di Asp.NET e per mantenere alcune informazioni a lato server di difficile accesso da qualsiasi utente. Mentre una parte del sito è stata sviluppata in Javascript, sfruttando jQuery, per alleggerire il server da parte dell'elaborazione. Per la gestione della grafica delle pagine si è sfruttato css.
Il maggior sfruttamento di Asp.NET è dato dall'utilizzo degli UpdatePanel, un oggetto che permette di ricaricare solo una porzione di pagina, e non l'intero sito, ogni volta che si debbano modificare le informazioni visibili su schermo.
Come si vedrà successivamente le funzionalità principali del sito sono contenute in un solo file aspx che gestisce la visualizzazione delle varie schermate e l'esecuzione del codice richiesto da ogni funzione.
Il fatto di poter accedere al database solo tramite linguaggi d'alto livello come il C# e la necessità di avere accesso ad alcuni dati presenti sul database da lato client, ha portato a sviluppare un Web Service che gestisca tutte le comunicazioni con il database e sfruttare una classe SoapClient che tramite Ajax dà accesso alle funzioni del Web Service da javascript.

13.2 SoapClient

La classe SoapClient creata in javascript è stata reperita su *Guru4.net* e permette a lato client di richiamare le funzioni presenti sul Web Service richiamando la loro descrizione. Il client (mediante una funzione JavaScript) richiama il metodo *SOAPClient.invoke* specificando:

- l'URL del Web Service,
- il nome del metodo del Web Service da eseguire,
- i parametri da passare al metodo del Web Service (raccolti tramite *SOAPClientParameters*)
- la modalità di esecuzione della chiamata (sincrona o asincrona),
- il metodo di callBack (opzionale per chiamate sincrone) da eseguire al termine dell'operazione.

Il metodo *SOAPClient.invoke* esegue le seguenti operazioni (esempio riferito al caso di una chiamata asincrona):

- recupera la descrizione del servizio (WSDL). Se è la prima volta nel contesto corrente che viene invocato il Web Service la descrizione viene richiesta al server e poi archiviata per eventuali richieste future,
- costruisce una richiesta SOAP per l'invocazione del metodo (con i relativi parametri) e la invia al server,
- quando il server risponde con il risultato dell'operazione richiesta, il client elabora la risposta e, utilizzando la descrizione del servizio, costruisce gli oggetti JavaScript corrispondenti,
- se la chiamata è stata effettuata in modo asincrono viene invocato il metodo di callBack specificato; se invece il client era in attesa della risposta in modalità sincrona, viene restituito l'oggetto corrispondente.

Codice 13.1: Esempio HelloWorld Codice Client

```
function HelloWorld()
{
  var pl = new SOAPClientParameters();
  SOAPClient.invoke(url, "HelloWorld", pl, true, HelloWorld_callBack);
}
function HelloWorld_callBack(r)
{
  alert(r);
}
```

Codice 13.2: Esempio HelloWorld Codice WebService

```
[WebMethod]
public string HelloWorld()
{
  return "Hello World!";
}
```

13.3 Struttura

Ora vedremo la struttura del sito e le varie schermate che lo compongono.

13.3.1 Schermata Login

Figura 13.1: Schermata Login

Questa schermata permette l'accesso tramite un nome utente e una password. In base all'utente, si ha accesso alle sole funzionalità rese disponibili dal tipo di permessi associato all'utente e si ha accesso solo ai dispositivi appartenenti ai gruppi associati all'utente. La password viene criptata attraverso un algoritmo criptografico di hashing e confrontata con il relativo valore hashing presente nel database.

13.3.2 Schermata Tempo Reale

Questa schermata contiene una mappa dove è possibile visualizzare in tempo reale la posizione attuale di uno o più dispositivi (,e una tabella contenente le varie informazioni relative).
Il tasto Attiva Tempo Reale avvia uno script (richiamato ripetutamente) che ricerca l'ultimo dato presente relativo ai dispositivi selezionati, ne mostra la posizione sulla mappa e mostra tutte le informazioni in una tabella sottostante. E' facile riconoscere a quale dispositivo appartenga una posizione dalla differente icona visualizzata a seconda del dispositivo a cui si riferisce. Nel caso di un dispositivo in movimento, le informazioni relative al dispositivo, presenti nella tabella, vengono ripetutamente aggiornate e sulla mappa viene aggiornata la posizione mostrando anche una spezzata rappresentante il percorso fatto negli ultimi 30 minuti dal momento dell'attivazione dello

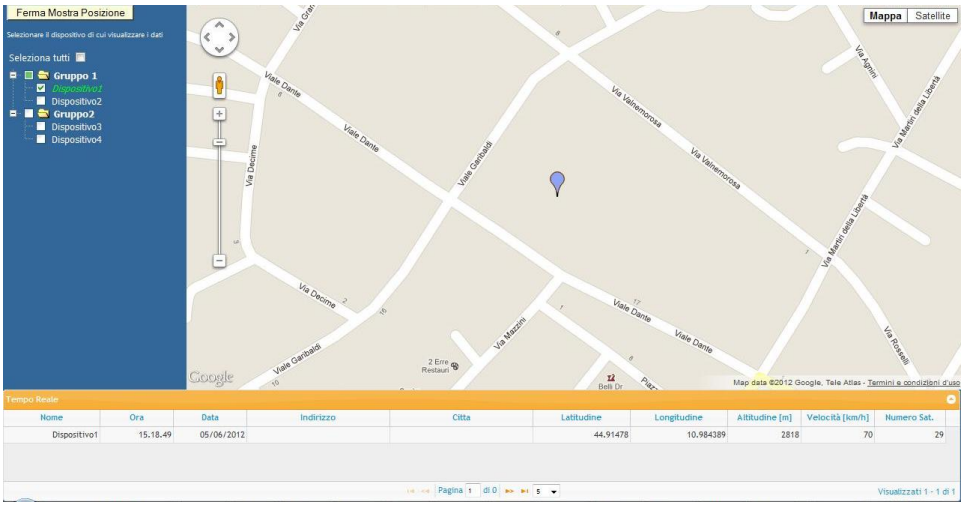

Figura 13.2: Schermata Tempo Reale

script, in modo da avere un'idea del tragitto recente effettuato e dell'attuale posizione del veicolo su cui è presente il dispositivo.

13.3.3 Schermata Percorso

Questa schermata contiene una mappa dove è possibile visualizzare il percorso effettuato da uno o più dispositivi e una tabella contenente tutte le posizioni che compongono il percorso.

E' possibile filtrare il percorso visualizzato indicando una data di inizio e

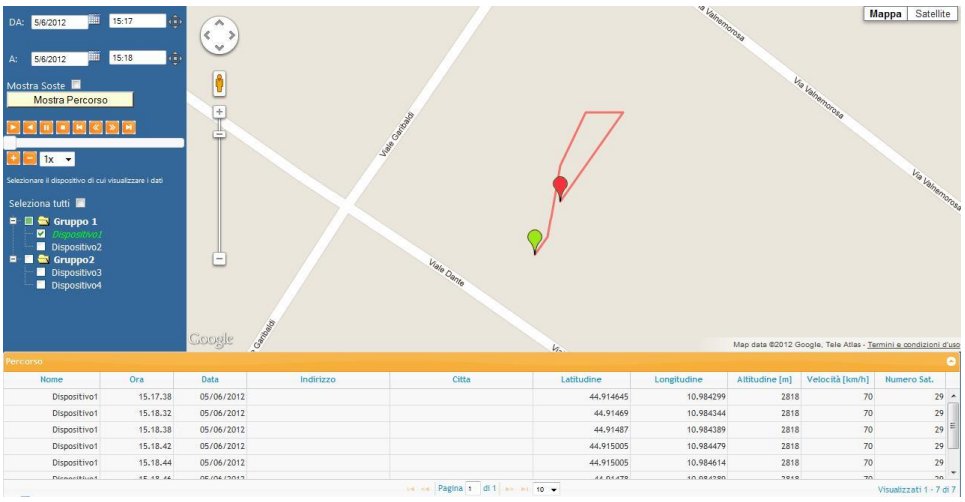

Figura 13.3: Schermata Percorso

una data di fine utilizzando il calendario che compare cliccando sul campo

dove è indicata la data. I campi con le date si impostano automaticamente sulla data della prima e dell'ultima posizione presente sul database dei dispositivi selezionati.
Il tasto Mostra Percorso attiva uno script che ricerca tutti i dati dei dispositivi selezionati compresi nel periodo indicato, visualizza i percorsi ottenuti disegnando una spezzata ottenuta unendo le posizioni così ottenute, e mostra tutte le informazioni sulle posizioni che compongono i percorsi in una tabella sottostante. I percorsi divisi per dispositivo e facilmente riconoscibili dal colore differente a seconda del dispositivo a cui si riferiscono.

Nel caso si sia scelto di visualizzare il percorso di un solo dispositivo, compaiono dei comandi aggiuntivi. Questi comandi permettono di visualizzare un'icona che simula il movimento del dispositivo sul tracciato, evidenziando il dato della tabella a cui si riferisce in quel momento la posizione.

Figura 13.4: Comandi Simulazione

13.3.4 Schermata Analisi

Questa schermata contiene una mappa dove è possibile visualizzare i risultati di un'analisi particolareggiata dei dati. Vengono visualizzati tutti i

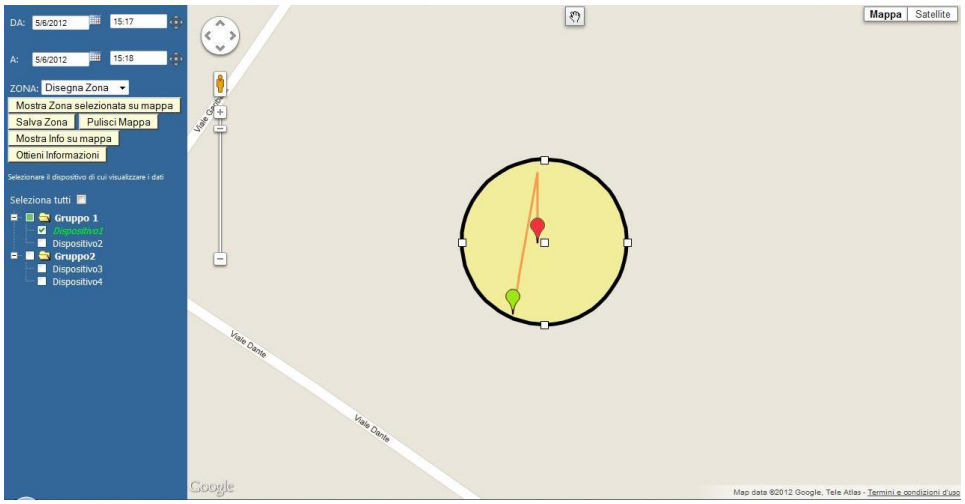

Figura 13.5: Schermata Analisi

dati dei dispositivi selezionati presenti all'interno di un'area evidenziata e

compresi nell'arco di tempo indicato. Il risultato è una serie di spezzate mostrate sulla mappa.

Inoltre è possibile richiedere il risultato dell'analisi sotto forma di tabulato che potrà poi essere esportato in un file Excel.

Salva su File

ID	Nome	Latitudine	Longitudine	Data	Velocita	Indirizzo	Satelliti	Sosta
1	Dispositivo1	44,914645	10,984299	05/06/2012 15.17.38	70	Non Elaborato	29	
1	Dispositivo1	44,91469	10,984344	05/06/2012 15.18.32	70	Non Elaborato	29	
1	Dispositivo1	44,91487	10,984389	05/06/2012 15.18.38	70	Non Elaborato	29	
1	Dispositivo1	44,915005	10,984479	05/06/2012 15.18.42	70	Non Elaborato	29	
1	Dispositivo1	44,915005	10,984614	05/06/2012 15.18.44	70	Non Elaborato	29	
1	Dispositivo1	44,91478	10,984389	05/06/2012 15.18.46	70	Non Elaborato	29	
1	Dispositivo1	44,91478	10,984389	05/06/2012 15.18.49	70	Non Elaborato	29	

Figura 13.6: Tabulato Analisi

13.3.5 Schermata Tabelle

Questa schermata permette di poter visualizzare alcuni dati presenti nel database in forma tabellare. Una serie di bottoni indica quali dati vedere rappresentati nella tabella.

Controlli Tempo Reale | Controlli Percorso | Interrogazione Dati | Controlli Tabelle | Amministrazione

Mostra Tabella Utenti
Mostra Tabella Dispositivi
Mostra Tabella Configurazioni
Mostra Tabella Veicoli/Proprietari
Mostra Tabella Gruppi di Utenti
Mostra Tabella Gruppi di Dispositivi
Mostra Tabella Permessi

User	ID	Nome	Cognome	InizioLicenza	FineLicenza	Gruppo
admin	1	ADMIN				Amministratore
andrea	2	Andrea	Selmini	11/03/2012 00:00:00	14/03/2014 00:00:00	Amministratore

Figura 13.7: Schermata Tabelle

13.3.6 Amministrazione

Questa schermata permette di avere aggiungere, modificare o eliminare le informazioni presenti sul database. I comandi visualizzati a cui si ha accesso dipendono dai permessi associati all'utente.
I comandi presenti sono:

- Configurazione Dispositivo:
  In questa schermata è possibile visualizzare quale configurazione ha

ogni dispositivo, assegnare una nuova configurazione scelta tra quelle già presenti, modificare una configurazione esistente o creare una nuova configurazione.

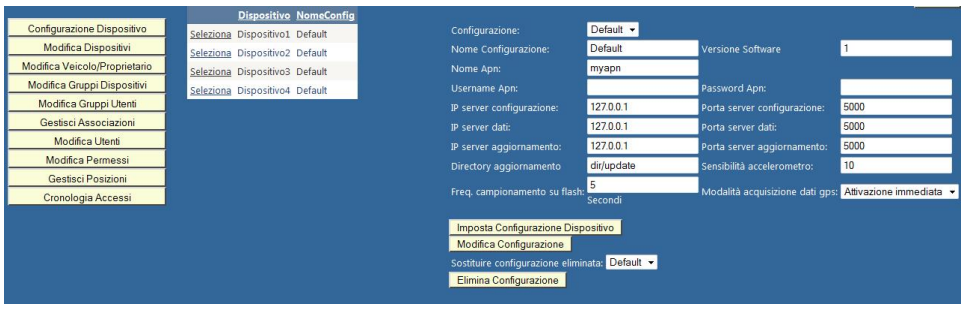

Figura 13.8: Configurazione Dispositivo

- Modifica Dispositivo:
  In questa schermata è possibile aggiungere un nuovo dispositivo al database, modificare i valori di uno già esistente o eliminarlo dal database. Nel caso dell'eliminazione di un Dispositivo è possibile anche eliminare tutte le posizioni registrate associate a quel dispositivo.

Figura 13.9: Modifica Dispositivo

- Modifica Veicolo/Proprietario:
  In questa schermata è possibile aggiungere i dati di un nuovo veicolo e del suo proprietario al database, assegnare un dispositivo al veicolo, modificare i dati di un veicolo già presente o eliminarlo dal database.

- Modifica Gruppi Dispositivi:
  In questa schermata è possibile aggiungere un nuovo gruppo di dispositivi, modificarne uno già esistente o eliminarlo dal database. Nel caso dell'eliminazione di un gruppo di dispositivi è possibile riassegnare i dispositivi appartenenti a quel gruppo ad un nuovo gruppo.

- Modifica Gruppo Utenti:
  In questa schermata è possibile aggiungere un nuovo gruppo di utenti e assegnargli un determinato tipo di permessi, modificare un gruppo già presente o eliminarlo dal database. Nel caso dell'eliminazione di un gruppo di utenti è possibile scegliere se eliminare o meno anche gli utenti appartenenti a quel determinato gruppo.

- Gestisci Associazioni:
  Questa schermata presenta un menu per permettere di gestire le varie associazioni tra le tabelle del database. Le informazioni sono volutamente ridondanti per rendere la visualizzazione dei dati il più semplice possibile per un utente.
  I comandi presenti sul menu sono:

  - Dispositivo -> Gruppi Dispositivi
    Qui è possibile associare un dispositivo scelto da un elenco, con un elenco a scelta multipla contenente i Gruppi Dispositivi. Selezionando un Dispositivo si selezionano automaticamente i Gruppi Dispositivi a cui è attualmente associato.
  - Gruppo Dispositivi -> Dispositivi
    Qui è possibile associare un Gruppo Dispositivi scelto da un elenco, con un elenco a scelta multipla contenente i Dispositivi. Selezionando un Gruppo Dispositivi si selezionano automaticamente i Dispositivi a cui è attualmente associato.
  - Gruppo Dispositivi -> Gruppi Utenti
    Qui è possibile associare un Gruppo Dispositivi scelto da un elenco, con un elenco a scelta multipla contenente i Gruppi Utenti. Selezionando un Gruppo Dispositivi si selezionano automaticamente i Gruppi Utenti a cui è attualmente associato.
  - Gruppo Utenti -> Gruppi Dispositivi
    Qui è possibile associare un Gruppo Utenti scelto da un elenco, con un elenco a scelta multipla contenente i Gruppi Dispositivi. Selezionando un Gruppo Utenti si selezionano automaticamente i Gruppi Dispositivi a cui è attualmente associato.
  - Gruppo Utenti -> Utenti
    Qui è possibile associare un Gruppo Utenti scelto da un elenco, con un elenco a scelta multipla contenente gli Utenti. Selezionan-

do un Gruppo Utenti si selezionano automaticamente gli Utenti a cui è attualmente associato.

- Modifica Utenti:
  Questa schermata permette di aggiungere un nuovo utente al database, assegnandolo a un Gruppo Utenti, dandogli un nome utente e una password e impostandogli un periodo di licenza dentro il quale è abilitato ad accedere al sito. E' possibile anche modificare i dati di un utente già esistente o eliminarlo dal database.

- Modifica Permessi:
  Questa schermata permette di aggiungere un nuovo tipo di permessi al database, modificarne uno già esistente o eliminarlo dal database. Nel caso dell'eliminazione di un tipo di permessi è possibile riassegnare gli utenti appartenenti a quel tipo ad un nuovo tipo di permessi.

Figura 13.10: Modifica Permessi

- Gestisci Posizioni:
  Questa schermata permette di eliminare le posizioni registrate sul database, di un determinato dispositivo. E' possibile filtrare l'eliminazione indicando un periodo tra due date. Selezionando un dispositivo, il periodo si aggiorna automaticamente indicando la data della posizione più recente e di quella più vecchia. Questa funzionalità è utile per accedere a determinate posizioni nel caso in cui si sia eliminato un dispositivo dal database, ma ne siano state mantenute le posizioni appartenenti.

- Cronologia Accessi:
  Questa schermata permette di visualizzare la cronologia degli accessi al sito. E' possibile filtrare la ricerca per utenti e per un periodo tra due date.
  E' anche possibile eliminare dal database i dati cronologici di accesso indicati dai filtri.

Figura 13.11: Cronologia Accessi

13.4 Script Asp.NET

Ora passeremo ad analizzare il codice Asp.NET del sito.

13.4.1 File Login.aspx

Il file *Login.aspx* contiene la schermata di login per accedere al sito. Il login avviene tramite un oggetto Asp.NET Login la cui gestione del codice risiede a lato server.

Codice 13.3: Login.aspx

```
...
    <asp:Login ID="Login1" runat="server" DisplayRememberMe="False"
     OnAuthenticate="Login1_Authenticate"
     Height="176px" Width="296px" ForeColor="White">
     <LoginButtonStyle CssClass="button_other" />
    </asp:Login>
...
```

13.4.2 File Default.aspx

Questa pagina contiene tutte le schermate che gestiscono le varie funzionalità del sito.
All'interno del corpo del file le varie strutture che compongono l'intero sito sono racchiusi in contenitori chiamati *div* in modo da poter impostare velocemente quali strutture visualizzare di volta in volta. All'interno del codice vi è l'utilizzo di un oggetto Asp.NET ScriptManager che permette l'utilizzo degli UpdatePanel e degli script lato server al loro interno.

Codice Menu Contestuale

Questa parte del codice viene utilizzata per la creazione di un menu contestuale visibile facendo click destro su uno dei dispositivi presenti nell'elenco. La creazione avviene sfruttando il plug-in *ContextMenu* di jQuery.

Codice 13.4: Menu Contestuale

```
<ul id="myMenu" class="contextMenu">
  <li class="center"><a href="#center">Centra</a></li>
  <li class="follow"><a href="#follow">Segui</a></li>
  <li class="zoom"><a href="#zoom">Zoom</a></li>
  <li class="info"><a href="#info">Info</a></li>
  <li class="quit separator"><a href="#quit">Annulla</a></li>
</ul>
```

Codice Header

Questa parte del codice definisce il div *Header* contenente i div *userinfo* e *menu*. Esso è posizionato nella parte alta del sito.
Il div *userinfo* contiene in modo nascosto alcune informazioni non delicate riguardo l'utente che necessitano in parte del codice javascript, inoltre contiene il bottone per effettuare il logout dal sito.
Il div *menu* contiene il menu principale che permette di passare tra le varie schermate del sito.

Codice Navigation

Il div *Navigation* contiene la definizione dei vari contenitori che contengono i comandi di controllo che vengono visualizzati a seconda della schermata. Esso è posizionato nel lato sinistro del sito.

Comandi Realtime Il div *div_realtime_command* contiene i pulsanti visualizzati nella schermata del real time.

Comandi Data e Ora Il div *div_data_time_command* contiene i controlli che permettono di seleziona un arco temporale indicando date e orari. Per selezionare una data viene utilizzato un calendario creato con il plug-in *DatePicker* di jQuery, mentre il controllo per selezionare un orario viene creato con il plug-in *TimeEntry*.

Comandi Percorso Il div *div_path_command* contiene i comandi per la schermata di visualizzazione di un percorso ed i comandi per simulare il movimento di un veicolo lungo il percorso.

Comandi Analisi Il div *div_interrogation_command* contiene i comandi per schermata di analisi. I comandi per permettere di selezionare un'area fanno parte direttamente della mappa.

Comandi Tabella Il div *div_table_command* contiene i comandi per visualizzare le varie tabelle.

Comandi Elenco Dispositivi Il div *div_inputtext* contiene la struttura che rappresenta l'elenco dei dispositivi, utilizzata in varie schermate. La struttura è rappresentata come un menu ad albero e viene creata sfruttando il plug-in *Dynatree* di jQuery .

Comandi Amministrazione Il div *div_administration_command* contiene i comandi per accedere alle varie schermate riservate all'amministrazione di tutto il sistema.

Codice Content

Il div *Content* contiene le strutture che compongono la parte centrale delle varie schermate. Esso è posizionato nella parte centrale, affianco al div *Navigation*.

Schermata Mappa Il div *map_canvas* è il div in cui verrà creata una mappa tramite le api fornite da Google.

Schermata Tabella Il div *Table_screen* contiene la schermata che mostra alcuni dati in forma tabellare. Le tabelle sono create per mezzo dell'oggetto Asp.NET *GridView*, inserito all'interno di un *UpdatePanel* per non dover ricaricare l'intera pagina ogni volta che viene assegnato un contenuto diverso alla tabella.

Schermate Amministrazione Il div *administration_screen* raccoglie tutti i contenitori che compongono le varie schermate dell'amministrazione. Esso è contenuto all'interno di un *UpdatePanel* per evitare di dover ricaricare l'intero sito ad ogni modifica delle varie schermate.

Schermata Configurazione Dispositivo Il div *div_configurazione_disp* raccoglie i contenuti che compongono la schermata per la configurazione di un dispositivo. Essa è composta da una tabella contenente i dispositivi e a quale configurazione sono associati, da un form contenente tutte le opzioni della configurazione e da una serie di bottoni per inserire, modificare ed eliminare una configurazione nel database.

Schermata Modifica Dispositivi Il div *div_edit_device* raccoglie i contenuti che compongono la schermata per la modifica dei dispositivi e delle loro informazioni. Essa è composta da una serie di form compilabili che permettono di aggiungere, modificare ed eliminare un dispositivo e le sue informazioni nel database.

Schermata Modifica Veicolo/Proprietario Il div *div_edit_vehicles* raccoglie i contenuti che compongono la schermata per la modifica dei veicoli e dei loro proprietari associati ai dispositivi. Essa è composta da una serie di form compilabili che permettono di aggiungere, modificare ed eliminare le informazioni sui veicoli e sui proprietari nel database.

Schermata Modifica Gruppi Dispositivi Il div *div_edit_groupdevice* raccoglie i contenuti che compongono la schermata per la modifica dei gruppi di dispositivi. Essa è composta da un form compilabile che permette di aggiungere, modificare ed eliminare un gruppo di dispositivi nel database.

Schermata Modifica Gruppi Utenti Il div *div_edit_groupuser* raccoglie i contenuti che compongono la schermata per la modifica dei gruppi di utenti con cui raggruppare i singoli utenti. Essa è composta da un form compilabile che permette di aggiungere, modificare ed eliminare un gruppo di utenti nel database.

Schermata Gestisci Associazioni Il div *div_edit_association* raccoglie i bottoni che permettono di passare nella schermata per modificare l'associazione indicata sul bottone. I dati nelle schermate sono volutamente ridondanti per rendere l'operazione più chiara e semplice da effettuare per un utente.

Schermata Associazione Dispositivo -> Gruppi Dispositivi Il div *div_association_device_groupsdevice* raccoglie i contenuti che compongono la schermata per l'impostazione delle associazioni tra un dispositivo ed i gruppi di dispositivi. Essa è composta da un'oggetto dropdown per selezionare un dispositivo e da una lista di checkbox per impostare i gruppi di dispositivi.

Schermata Associazione Gruppo Dispositivi -> Dispositivi Il div *div_association_groupdevice_devices* raccoglie i contenuti che compongono la schermata per l'impostazione delle associazioni tra un gruppo di dispositivi ed i dispositivi. Essa è composta da un'oggetto dropdown per selezionare un gruppo di dispositivi e da una lista di checkbox per impostare i dispositivi.

Schermata Associazione Gruppo Dispositivi -> Gruppi Utenti Il div *div_association_groupdevice_groupsuser* raccoglie i contenuti che compongono la schermata per l'impostazione delle associazioni tra un gruppo di dispositivi ed i gruppi di utenti. Essa è composta da un'oggetto dropdown per selezionare un gruppo di dispositivi e da una lista di checkbox per impostare i gruppi di utenti.

Schermata Associazione Gruppo Utenti -> Gruppi Dispositivi Il div *div_association_groupuser_groupsdevice* raccoglie i contenuti che compongono la schermata per l'impostazione delle associazioni tra un gruppo di utenti ed i gruppi di dispositivi. Essa è composta da un'oggetto dropdown per selezionare un gruppo di utenti e da una lista di checkbox per impostare i gruppi di dispositivi.

Schermata Associazione Gruppi Utenti -> Utenti Il div *div_association_groupuser_users* raccoglie i contenuti che compongono la schermata per l'impostazione delle associazioni tra un gruppo di utenti e gli utenti. Essa è composta da un'oggetto dropdown per selezionare un gruppo di utenti e da una lista di checkbox per impostare gli utenti.

Schermata Modifica Utenti Il div *div_edit_user* raccoglie i contenuti che compongono la schermata per la modifica degli utenti. Essa è composta da una serie di form compilabili che permettono di aggiungere, modificare

ed eliminare un utente nel database.

Schermata Modifica Permessi Il div *div_edit_permits* raccoglie i contenuti che compongono la schermata per la modifica di una configurazione di permessi da assegnare ad un gruppo di utenti. Essa è composta da un form compilabile che permette di creare, modificare ed eliminare una nuova configura di permessi dal database.

Schermata Gestisci Posizioni Il div *div_edit_position* raccoglie i contenuti che compongono la schermata per eliminare le posizioni e le informazioni correlate di un dispositivo. Essa è composta da degli oggetti per selezionare uno spazio temporale e da un oggetto dropdown con cui selezionare il dispositivo di cui si desidera eliminare le posizioni dal database.

Schermata Cronologia Accessi Il div *div_cronologia* raccoglie i controlli che compongono la schermata per la visualizzazione degli accessi al sito effettuati dai vari utenti e la gestione di queste informazioni. Essa è composta da degli oggetti per selezionare uno spazio temporale e da un oggetto dropdown con cui selezionare l'utente di cui si desidera eliminare la cronologia dal database.

Codice InformationPanel

Il div *information_panel* è posizionato nella parte bassa del sito e contiene i div *infopanel_real_time* e *infopanel_path* che contengono le tabelle su cui verranno visualizzate le informazioni riguardanti le posizioni presenti sulla mappa. Le tabelle vengono create per mezzo del plug-in *jqGrid* di jQuery.

13.4.3 File AnalisiTabella.aspx

Il file *AnalisiTabella.aspx* contiene la tabella su cui è possibile visualizzare i risultati ottenuti attraverso la schermata di Analisi dei dati. La pagina è composta da una oggetto *GridView* che conterrà i risultati e da un bottone per salvare il contenuto della tabella su di un file excel, successivamente al salvataggio viene data la possibilità di aprire o di salvare il file in locale.

13.5 Script C#

Ora passeremo ad analizzare il codice C# del sito richiamato dagli oggetti Asp.NET. Per la visualizzazione del codice completo si rimanda all'appendice.

13.5.1 File Logis.aspx.cs

Il file *Login.aspx.cs* contiene il codice richiamato dalla pagina *Login.aspx*.

Login1_Authenticare L'unica funzione rilevante di questo file è la funzione *Login1_Authenticare* che viene utilizzata dall'oggetto Asp.Net Login per effettuare il controllo nel database del nome utente e della password inserita e per controllare se il periodo di licenza assegnato ai dati inseriti è ancora valido.

13.5.2 File Default.aspx.cs

Il file *Default.aspx.cs* contiene il codice richiamato dalla pagina *Default.aspx*.

Funzione Page_Load La funzione *Page_Load* viene richiamata al caricamento della pagina e controlla i permessi assegnati all'utente per impostare quali comandi possono essere visualizzati.

Funzione Button_logout_Click La Funzione Button_logout_Click viene richiamata premendo il bottone Logout e attraverso la chiusura della Session permette di richiamare la funzione *Session_End*

Codice 13.5: Funzione Button_logout_Click

```
protected void Button_logout_Click(object sender, EventArgs e)
{
  Session.Abandon();
  Response.Redirect("Login.aspx");
}
```

Funzione Session_End La funzione Session_End viene richiamata automaticamente alla chiusura della Session, un evento che si verifica dopo un

certo periodo di inattività o forzandone da codice la chiusura. L'evento viene sfruttato dal sito per registrare l'orario l'orario di accesso al sito da parte di un utente.

Codice 13.6: Funzione Session_End

```
void Session_End(object sender, EventArgs e)
{
  string now = DateTime.Now.ToString("u").Remove((DateTime.Now.ToString().
      Length));
  now = now.Replace('.', ':');
  try
  {
    service.InsertChronology(Session["IDuser"].ToString(), Session["LoginTime"].
        ToString(), now);
    string path = System.Web.Hosting.HostingEnvironment.MapPath("~/") + "/
        App_Data//TabulatoAnalisi" + Session["IDuser"].ToString() + ".xls";
    if (File.Exists(path))
    {
      File.Delete(path);
    }
  }
  catch (Exception)
  {

  }
}
```

Funzioni Visualizzazione Schermate di Amministrazione Le varie funzioni dedicate alla visualizzazione delle schermate dell'amministrazione sono molto simili tra loro, inizialmente resettano la visibilità di tutte le schermate tramite la fuzione *Hidden_administration_div* e poi visualizzano i contenuti della schermata selezionata tramite la funzione *Show_administration_div*, successivamente, tramite una ricerca sul database, inseriscono i valori necessari all'utente per le operazioni, dentro i controlli presenti sulla pagina. Per le funzioni che visualizzano le tabelle viene utilizzata anche la funzione *SourceTable* che indica quali dati caricare nella tabella.

Funzione Hidden_administration_div La funzione *Hidden_administration_div* resetta la visualizzazione e i contenuti delle schermate dell'amministrazione.

Funzione Show_administration_div La funzione *Show_administration_div* rende visibile la schermata di amministrazione in-

dicata come parametro.

Funzione SourceTable La funzione *SourceTable* viene utilizzata per riempire le tabelle con il contenuto di un DataTable passato come parametro.

Eventi DropDown SelectIndexChanged Quando viene selezionato un valore in alcuni oggetti DropDown essi scatenano un evento che, una volta intercettato, permette di modificare in tempo reale i valori nel form in cui sono presenti.

Eventi Oggetti GridView Gli oggetti Asp.NET GridView che permettono di visualizzare una tabella hanno la possibilità di strutturare il contenuto in più pagine, ma per permettere la navigazione del contenuto nelle pagine bisogna implementare il metodo *PageIndexChanging* per ricaricare i contenuti nella tabella. Il metodo *SelectedIndexChanged* viene invece implementato per permettere di modificare il contenuto di un form selezionando un elemento della tabella.

Eventi TextBox TextChanged In alcune schermate dell'amministrazione modificando il contenuto di una TextBox viene resettato il form a cui appartiene per permettere l'inserimento di nuovi dati.

13.5.3 File AnalisiTabella.aspx.cs

Il file *AnalisiTabella.aspx.cs* contiene il codice richiamato dalla pagina *AnalisiTabella.aspx*.

Funzione Page_Load La funzione *Page_Load* viene richiamata al caricamento della pagina e carica da un cookie una query, che viene poi utilizzata per ricercare i dati da inserire nella tabella all'interno della pagina.

Codice 13.7: Funzione Page_Load

```
protected void Page_Load(object sender, EventArgs e)
{
 if (!IsPostBack)
 {
  HttpCookie cookie = Request.Cookies["query"];
  if (cookie != null)
```

```
    {
      string query = Microsoft.JScript.GlobalObject.unescape(cookie.Value);
      cookie = Request.Cookies["type_analysis"];
      string type = Microsoft.JScript.GlobalObject.unescape(cookie.Value);
      DataTable table = service.GetTableAnalysis(query, type);
      GridViewTabellaAnalisi.DataSource = table;
      GridViewTabellaAnalisi.DataBind();
    }
  }
}
```

Funzione Button_SaveTable_Click La funzione *Button_SaveTable_Click* permette di salvare il contenuto della tabella all'interno di un file excel. Tra i veri metodi possibili, il più veloce è tramite una connessione OleDB (Object Linking and Embedding), un sistema sviluppato da Microsoft.

13.6 Script Javascript

Ora passeremo ad analizzare il codice Javascript del sito. Javascript non possiede il concetto di classe come i linguaggi ad oggetti ma le funzioni sono state organizzate in file in base allo scopo per cui sono state sviluppate.

13.6.1 File Global.js

Questo file contiene le definizioni di alcune variabili utilizzate da più script Javascript.

13.6.2 File UtilityFunctionDefautl.js e UtilityFunctionTabella.js

Questi file contengono diverse funzioni ausiliarie utilizzate nelle pagine Default.aspx e AnalisiTabella.aspx.

Funzione init La funzione *init* viene richiamata al caricamento della pagina ed è l'unica funzione presente anche all'interno del file UtilityFunctionTabella.js. Nel file UtilityFunctionDefault.js si occupa di creare una mappa tramite le api di Google, inoltre richiama altre funzioni che si occupano di creare gli oggetti presenti nel menu laterale e ingrandisce la mappa in modo che occupi il maggior spazio possibile all'interno della finestra del browser. Mentre nel file UtilityFunctionTabella.js si occupa di eliminare i cookie utilizzati per il salvataggio delle informazioni.

Funzione changediv La funzione*chagediv* si occupa di modificare la visibilità degli oggetti nel sito in modo da visualizzare gli oggetti della schermata richiesta.

Funzione resizemap La funzione *resizemap* si occupa di modificare le dimensioni della mappa in modo da occupare lo spazio presente nella finestra del browser nel miglior modo possibile.

13.6.3 File MapPageCommand.js

Questo file contiene la definizione di una classe in modo da mantenere uniti tutte le informazioni riguardo un punto, e le funzioni necessarie per lo sfruttamento di questa classe nel javascript.

Classe Point La classe *Point* mantiene unite le informazioni di un punto per facilitare le operazioni di ricerca, elminazione e modifica.

Codice 13.8: Classe Point

```
var Point = function (param_marker_datas, param_info) {
    this.marker_datas = param_marker_datas;
    this.info = param_info;
    this.lines = new google.maps.Polyline(generalPolyOptions);
    this.lines.setMap(map);
    this.addToLines = function () {
        this.lines.getPath().push(this.marker_datas.getPosition());
    }
    this.lengthLines = function () {
        return this.lines.getPath().getLength();
    }
    this.removeFirst = function () {
        return this.lines.removeAt(0);
    }
    this.clear = function () {
        this.deleteOnTable();
        if (this.lines != null) {
            this.lines.setMap(null);
            this.lines.getPath().clear();
            this.lines = null;
        }
        if (this.marker_datas != null) {
            this.marker_datas.setMap(null);
            this.marker_datas = null;
        }
        if (this.info != null) {
            this.info = null;
        }
    }
    this.get = function () {
        return this;
    }
    this.addToTable = function () {
        if ($("#table_realtime").jqGrid('getInd', this.info.ID, false) != false) {
            $("#table_realtime").jqGrid('setRowData', this.info.ID, this.info);
        }
        else {
            $("#table_realtime").jqGrid('addRowData', this.info.ID, this.info);
        }
    }
    this.deleteOnTable = function () {
        if ($("#table_realtime").jqGrid('getInd', this.info.ID, false) != false) {
            $("#table_realtime").jqGrid('delRowData', this.info.ID);
        }
    }
    this.showinfo = function () {
        if (this.info != null) {
            $("#Hidden_realtime_id").val(this.info.ID);
```

```
            $("#Label_realtime_ora").text(this.info.Ora + " " + this.info.Data);
50          var tree = $("#tree").dynatree("getTree");
            $("#Label_realtime_nome").text((tree.getNodeByKey(this.info.ID)).data.title);
52          try {
                $("#Label_realtime_indirizzo").text(this.info.Indirizzo);
54              $("#Label_realtime_citta").text(this.info.Citta);
            }
56          catch (err) {
                $("#Label_realtime_indirizzo").text("Indirizzo non presente");
58              \$("#Label_realtime_citta").text(" ");
            }
60          $("#Label_realtime_latitudine").text(this.info.Latitudine);
            $("#Label_realtime_longitudine").text(this.info.Longitudine);
62          $("#Label_realtime_altitudine").text(this.info.Altitudine);
            $("#Label_realtime_velocita").text(this.info.Velocita);
64          $("#Label_realtime_direzione").text(this.info.Direzione);
            $("#Label_realtime_numsat").text(this.info.NumSat);
66      }
    }
68  }
```

13.6.4 File TreeFunction.js

Questo file contiene le funzioni per la creazione del menu ad albero per la selezione dei dispositivi. Contiene anche le funzioni per creare e utilizzare il menu contestuale che compare premendo il tasto destro su uno dei dispositivi nel menu. Le funzioni sono: *newTree* per la creazione del menu ad albero, *ReloadTree* per ricaricare il manu dopo delle modiche, *bindContextMenu* per collegare il menu contestuale agli elementi del menu ad albero, e le funzioni per i comandi del menu contestuale.

13.6.5 File jGridFunction.js

Questo file contiene le funzioni *createTablePath*, *createTableRealTime*, *clearTablePath* e *clearTableRealTime* utilizzate per inserire ed eliminare i dati dalle tabelle posizionate sotto la mappa, contenenti le informazioni delle posizioni o dei percorsi visualizzati su mappa.

13.6.6 File GestioneCookie.js

Questo file contiene le funzioni per utilizzare i cookie. Le funzioni sono:*testCookie* per testare se sono abilitati i cookie sul browser, *writeCookie* per creare un cookie, *changeCookie* per modificare un cookie, *readCookie* per leggere il contenuto di un cookie, *deleteCookie* per eliminare un cookie.

13.6.7 File RealTimeCommand.js

Questo file contiene le funzioni utilizzate dalla schermata per la visualizzazione una posizione in tempo reale. La funzione *Button_position_onclick* richiama la funzione *TimedFunction* ogni 2 minuti e inoltre se sono abilitati i cookie richiama la funzione *RefreshFunction* ogni 6 ore; la funzione *TimedFunction* si occupa di invocare la richiesta degli ultimi dati gps dei dispositivi selezionati e li mostra tramite una funzione callback; la funzione *GivePos_callBack* si occupa di creare o aggiornare i punti visualizzati sulla mappa inserendo o loro dati nella tabella sottostante la mappa; la funzione *createMarker* crea un oggetto google map Marker indicante la posizione gps sulla mappa; la funzione *RefreshFunction* fa il refresh della pagina intera e mantiene le informazioni come erano visualizzare, questo è necessario per evitare problemi di memory leak causati dai molti spostamenti della mappa creata con le api di google; infine la funzione *CloseRealTime* si occupa di disabilitare l'invocazione continua delle funzioni *TimedFunction* e *RefreshFunction*.
Tra queste funzioni quella di rilievo è la funzione *GivePos_callBack*, il parametro richiesto *pos* è rappresentato da un array di classi contenenti le informazioni più recenti riguardo la posizione dei dispositivi indicati; nel caso in cui la comunicazione con il server sia stata interrotta o la memoria ram assegnata al browser sia stata occupata interamente la classe *pos* risulta nulla.

Codice 13.9: Funzione GivePos_callBack

```
function GivePos_callBack(pos) {
  pl.clear();
  var icon;
    if (pos == null) {
        CloseRealTime();
        var now = new Date();
        var time = now.getFullYear() + "-" + (now.getMonth() + 1) + "-" + now.
        getDate() + " " + now.getHours() + ":" + now.getMinutes() + ":" + now.
        getSeconds();
        alert("Comunicazione col server interrotta il: " + time);
        return;
    }
    else {
        first_loop:
        for (var i = 0; i < pos.length; i++) {
            if (pos[i] != null) {
                if (points.length == 0) {
                  if (($("#table_realtime").jqGrid('getDataIDs')).length <= 0) {
                        createTableRealTime(null);
                      }
                      icon = googleico.replace('color', iconcolor[0]); ;
                    points.push(new Point(createMarker(pos[i], icon), pos[i]));
                    points[0].addToLines();
```

```
                points[0].addToTable();
                continue first_loop;
            }
            second_loop:
            for (var j = 0; j < points.length; j++) {
              if ((i) <= 5) {
                x = i;
              }
              else {
                x = (i) % 5;
              }
              icon = googleico.replace('color', iconcolor[x]); ;
                if (pos[i].ID == points[j].info.ID) {
                    if ((pos[i].Time).getTime() != (points[i].info.Time).getTime()) {
                        if (points[i].marker_datas != null) {
                            points[i].marker_datas.setMap(null);
                        }
                        points[i].marker_datas = createMarker(pos[i], icon);
                        points[i].info = pos[i];
                        points[i].addToTable();
                        points[i].addToLines();
                        if (points[i].lengthLines() > maxlengthlines) {
                            points[i].removeFirst();
                        }
                    }
                    continue first_loop;
                }
            }
            points.push(new Point(createMarker(pos[i], icon), pos[i]));
            points[points.length - 1].addToTable();
            points[points.length - 1].addToLines();
            points.sort(confronta_point);
        }
    }
    if (points != null) {
        if (followPoint != null) {
            var cont = findPoint(followPoint);
            CenterPos = new google.maps.LatLng(points[index].info.Latitudine, points[
index].info.Longitudine);
            map.panTo(CenterPos);
        }
        if (CenterPos == null) {
            var tmp_lat = 0;
            var tmp_long = 0;
            for (var k in points) {
                tmp_lat = tmp_lat + points[k].info.Latitudine;
                tmp_long = tmp_long + points[k].info.Longitudine;
            }
            tmp_lat = tmp_lat / points.length;
            tmp_long = tmp_long / points.length;
            CenterPos = new google.maps.LatLng(tmp_lat, tmp_long);
            map.panTo(CenterPos);
        }
    }
```

```
    }
76 }
```

13.6.8 File PathCommand.js

Questo file contiene le funzioni utilizzate dalla schermata per la visualizzazione di un percorso. La funzione *Button_timepath_onclick* si occupa di invocare la richiesta di tutti i dati compresi tra due date dei dispositivi indicati e li mostra tramite una funzione callback; la funzione *GiveTimePath_callBack* invoca la funzione *CreateRoute* e inserisce i dati dei percorsi nella tabella sottostante la mappa, inoltre se è mostrato solo un percorso abilita i comandi per simulare il movimento sul percorso; la funzione *CreateRoute* crea una spezzata sulla mappa tramite un oggetto google map polyline e se richiesto pone dei segnalini nei punti di sosta indicanti il tempo della sosta; infine la funzione *deletePath* si occupa di eliminare i dati dei percorsi.

13.6.9 File PlayerCommand.js

Questo file contiene le funzioni utilizzate dai comandi per la simulazione del movimento di un veicolo su un percorso. La simulazione avviene spostando un simbolo rappresentante un veicolo lungo il percorso mostrato; per mezzo delle funzioni presenti è possibile far partire il movimento o riavvolgerlo, decidere la velocità del movimento, fermarlo, spostarsi di un passo alla volta, spostarsi direttamente all'inizio o alla fine del percorso e infine aggiornare la posizione quando si agisce sulla barra di scorrimento.

13.6.10 File InterrogationCommand.js

Questo file contiene le funzioni utilizzate dalla schermata per l'analisi delle posizioni. Le funzioni *ActivateDrawing* e *DeactivateDrawing* abilitano e disabilitano i comandi di google maps che permettono di disegnare figure geometriche sulla mappa; la funzione *Button_interrogation_getonmap_onclick* si occupa di invocare la richiesta dei dati all'interno di una zona selezionata compresi tra due date dei dispositivi indicati e li mostra tramite una funzione callback; la funzione *GetInfoOnMap_callBack* mostra i percorsi risultanti dalle richieste della funzione precedente; la funzione *Button_interrogation_getinfo_onclick* si occupa di invocare la richiesta dei dati all'interno di una zona selezionata compresi tra due date dei dispositivi indicati e di mostrarli in una nuova pagina web in forma tabellare, se i dati riguardo la zona richiesta non sono presenti a lato client vengono richiesti i dati riguardo la zone scelta ed è la funzione *GetInfoZone_callBack* che li

mostra in nuova pagina web; le altre funzioni presenti si occupano di salvare i dati di zone selezionate, visualizzare su mappa una zona scelta da un elenco oppure resettare il contenuto della mappa.

13.6.11 File di Amministrazione

Questi file contengono le funzioni per le schermate riservate all'amministrazione del sistema.
Il file *Edit-configCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati delle configurazioni.
Il file *Edit-deviceCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati dei dispositivi.
Il file *Edit-vehicleCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati dei veicoli.
Il file *Edit-groupdeviceCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati dei gruppi di dispositivi.
Il file *Edit-groupuserCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati dei gruppi di utenti.
Il file *Edit-associationCommand.js* contiene le funzioni per creare le query che permettono di interagire con le associazioni tra le varie tabelle del database.
Il file *Edit-userCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati degli utenti.
Il file *Edit-permitsCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati dei permessi.
Il file *Edit-positionCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati delle posizioni.
Il file *ChronologyCommand.js* contiene le funzioni per creare le query che permettono di interagire con i dati della cronologia degli accessi al sito.

13.7 Sviluppo WebService

Dato che le operazioni sul database sono rese possibili solo da codice C# lato server e data la necessità di disporre delle informazioni in esso contenute anche a lato client nel codice javascript si è reso necessario sviluppare il codice per interagire con il database all'interno di un Web Service e l'utilizzo di una classe SoapClient che tramite Ajax fornisca l'accesso alle funzioni in esso contenute.

Ora passeremo ad analizzare le funzioni sviluppare all'interno del Web Service. Esso è sviluppato in C# ed è composto da una classe che implementa la classe *WebService* del C#, mentre tutte le funzionalità sono contenute in funzioni accessibili dall'esterno per mezzo dell'attributo *WebMethod* posizionato sopra la funzione. Alcune delle funzioni presenti hanno come valore di ritorno una classe che viene serializzata grazie all'attributo *Serializable*, in questo modo è possibile disporre facilmente dei dati contenuti nella classe anche all'interno del codice javascript.

Ora verranno studiate le funzioni che si occupano dell'elaborazione delle posizioni e delle loro informazioni. I valori di ritorno di queste funzioni sono composti dalle classi *Punto* e *Percorso*.

Classe Punto La classe *Punto* raggruppa i dati inerenti una posizione.

Codice 13.10: Classe Punto

```
[Serializable()]
public class Punto
{
  private UInt32 _id;
  private string _nome;
  private Double _latitudine;
  private Double _longitudine;
  private Int16 _altezza;
  private UInt16 _velocita;
  private Int16 _direzione;
  private DateTime _time;
  private string _data;
  private string _ora;
  private Byte _fix;
  private Byte _numsat;
  private string _indirizzo;
  private string _citta;
  private string _stoptime;

  public Punto()
```

```
{
}
public Punto(UInt32 id, string nome, Double latit, Double longit, Int16 altitud,
    UInt16 veloc, Int16 direct, string data, string ora, Byte fix, Byte numsatelliti,
    string indirizzo, string citta)
{
  this.ID = id;
  this.Nome = nome;
  this.Latitudine = latit;
  this.Longitudine = longit;
  this.Altitudine = altitud;
  this.Velocita = veloc;
  this.Direzione = direct;
  this.Data = data;
  this.Ora = ora;
  this.Fix = fix;
  this.NumSat = numsatelliti;
  this.Indirizzo = indirizzo;
  this.Citta = citta;
}
public UInt32 ID
{
  get { return _id; }
  set { _id = value; }
}
public string Nome
{
  get { return _nome; }
  set { _nome = value; }
}
public Double Latitudine
{
  get { return _latitudine; }
  set { _latitudine = value; }
}
public Double Longitudine
{
  get { return _longitudine; }
  set { _longitudine = value; }
}
public Int16 Altitudine
{
  get { return _altezza; }
  set { _altezza = value; }
}
public UInt16 Velocita
{
  get { return _velocita; }
  set { _velocita = value; }
}
public Int16 Direzione
{
  get { return _direzione; }
  set { _direzione = value; }
```

```
  }
  public DateTime Time
  {
    get { return _time; }
    set { _time = value; }
  }
  public string Data
  {
    get { return _data; }
    set { _data = value; }
  }
  public string Ora
  {
    get {return _ora;}
    set {_ora = value;}
  }
  public Byte Fix
  {
    get { return _fix; }
    set { _fix = value; }
  }
  public Byte NumSat
  {
    get { return _numsat; }
    set { _numsat = value; }
  }
  public string Indirizzo
  {
    get { return _indirizzo; }
    set { _indirizzo = value; }
  }
  public string Citta
  {
    get { return _citta; }
    set { _citta = value; }
  }
  public string StopTime
  {
    get { return _stoptime; }
    set { _stoptime = value; }
  }
}
```

Classe Percorso La classe *Percorso* raggruppa i dati inerenti un percorso, un percorso è formato da una lista di oggetti della classe *Punto*.

Codice 13.11: Classe Percorso

```
[Serializable()]
public class Percorso : ICollection
{
  private UInt32 _id;
```

```
  private List<Punto> _posizioni;
6
...
8
...
10
  public UInt32 ID
12 {
    get { return _id; }
14  set { _id = value; }
  }
16 public List<Punto> Posizioni
  {
18  get { return _posizioni; }
    set { _posizioni = value; }
20 }
  public void Posizione(Punto posizione)
22 {
    Posizioni.Add(posizione);
24 }
  public Punto this[int index]
26 {
    get { return (Punto)Posizioni[index]; }
28 }
  public void Add(Punto posizione)
30 {
    Posizioni.Add(posizione);
32 }

34 public void CopyTo(Array array, int index)
  {
36  if (object.ReferenceEquals(array, null))
    {
38    throw new ArgumentNullException(
        "Null array reference",
40      "array"
        );
42  }

44  if (index < 0)
    {
46    throw new ArgumentOutOfRangeException(
        "Index is out of range",
48      "index"
        );
50  }

52  if (array.Rank > 1)
    {
54    throw new ArgumentException(
        "Array is multi-dimensional",
56      "array"
        );
58  }
```

```
    foreach (object o in this)
    {
      array.SetValue(o, index);
      index++;
    }
  }

  public int Count
  {
    get { return Posizioni.Count; }
  }

  public bool IsSynchronized
  {
    get { return false; }
  }

  public object SyncRoot
  {
    get { return this; }
  }

  public IEnumerator GetEnumerator()
  {
    return Posizioni.GetEnumerator();
  }
}
```

Funzione GivePos La funzione *GivePos* viene utilizzata per ritornare un array di punti, che rappresentano le posizioni più recenti dei dispositivi indicati in input.

Codice 13.12: Funzione GivePos

```
[WebMethod(Description = "Connette al db e ritorna le posizioni più recenti dei
    dispositivi selezionati")]
public Punto[] GivePos(string[] parametro)
{
  Double latitudine;
  Double longitudine;
  Int16 altezza;
  UInt16 velocita;
  Int16 direzione;
  Byte fix;
  Byte numsat;
  UInt32 id;
  string data;
  string ora;
  string indirizzo;
  string citta;
  string nome;
```

```
18  Array.Sort(parametro);

20  Punto[] pos = new Punto[parametro.Length];

22  for (int j = 0; j < parametro.Length; j++)
    {
24    string querystring = "SELECT Dispositivo.UID, Dispositivo.Nome, Datigps.
        Latitudine, Datigps.Longitudine, Datigps.Altezza, Datigps.Velocita, Datigps.
        Direzione, Datigps.Data, Datigps.Fix, Datigps.NumeroSatelliti, Datigps.Indirizzo
        FROM Dispositivo, Datigps WHERE Dispositivo.UID = (paramet) AND
        Dispositivo.UID = Datigps.Dispositivo_UID AND Datigps.Fix > 1 ORDER BY
        Datigps.Data DESC LIMIT 1";
      querystring = querystring.Replace("paramet", parametro[j]);
26    DataTable table = ExecuteSelectOnDb(querystring);
      try
28    {
        id = (UInt32)table.Rows[0]["UID"];
30      nome = table.Rows[0]["Nome"].ToString();
        latitudine = (Double)table.Rows[0]["Latitudine"];
32      longitudine = (Double)table.Rows[0]["Longitudine"];
        altezza = (Int16)table.Rows[0]["Altezza"];
34      velocita = (UInt16)table.Rows[0]["Velocita"];
        direzione = (Int16)table.Rows[0]["Direzione"];
36      data = ((DateTime)table.Rows[0]["Data"]).ToString("dd/MM/yyyy");
        ora = ((DateTime)table.Rows[0]["Data"]).ToString("HH:mm:ss");
38      fix = (Byte)table.Rows[0]["Fix"];
        numsat = (Byte)table.Rows[0]["NumeroSatelliti"];
40      if (table.Rows[0]["Indirizzo"].ToString() == "Non Elaborato")
        {
42        indirizzo = "";
          citta = "";
44      }
        else
46      {
          try
48        {
            indirizzo = table.Rows[0]["Indirizzo"].ToString().Split('-')[0] + " " + table.
          Rows[0]["Indirizzo"].ToString().Split('-')[1];
50          citta = table.Rows[0]["Indirizzo"].ToString().Split('-')[2];
          }
52        catch (Exception)
          {
54          indirizzo = "";
            citta = "";
56        }
        }
58
        pos[j] = new Punto(id, nome, latitudine, longitudine, altezza, velocita, direzione,
          data, ora, fix, numsat, indirizzo, citta);
60      pos[j].Time = ((DateTime)table.Rows[0]["Data"]);
      }
62    catch
      {
```

```
      pos[j] = null;
    }
  }
  return pos;
}
```

Funzione GiveTimePath La funzione *GiveTimePath* viene utilizzata per ritornare un array di percorsi effettuati dai dispositivi indicati in input e durante il periodo temporale indicato in input. E' possibile anche indicare se si desideran o meno visualizzare le soste effettuate dal veicolo.

Codice 13.13: Funzione GiveTimePath

```
[WebMethod(Description = "Connette al db e ritorna tutte le posizioni all'interno di un
      certo periodo di tutti i dispositivi selezionati")]
public Percorso[] GiveTimePath(string[] parametro, string datainizio, string datafine,
      bool show_stop)
{
  Array.Sort(parametro);
  int numpos = parametro.Length;

  string querystring = "SELECT Dispositivo.UID, Dispositivo.Nome, Datigps.
      Latitudine, Datigps.Longitudine, Datigps.Altezza, Datigps.Velocita, Datigps.
      Direzione, Datigps.Data, Datigps.Fix, Datigps.NumeroSatelliti, Datigps.Indirizzo,
      Datigps.StatoVeicolo AS Stato FROM Dispositivo, Datigps WHERE Datigps.
      Data BETWEEN ('startdate') AND ('enddate') AND Dispositivo.UID IN (
      paramet) AND Dispositivo.UID = Datigps.Dispositivo_UID";
  string device = "";
  for (int i = 0; i < parametro.Length; i++)
  {
    device += parametro[i] + ",";
  }
  device = device.Substring(0, device.Length - 1);
  DateTime inizio = DateTime.Parse(datainizio);
  DateTime fine = DateTime.Parse(datafine + ":59");
  String stringinizio = inizio.Year + "-" + inizio.Month + "-" + inizio.Day + " " +
      inizio.TimeOfDay;
  String stringfine = fine.Year + "-" + fine.Month + "-" + fine.Day + " " + fine.
      TimeOfDay;

  querystring = querystring.Replace("paramet", device);
  querystring = querystring.Replace("startdate", stringinizio);
  querystring = querystring.Replace("enddate", stringfine);
  DataTable table = ExecuteSelectOnDb(querystring);

  return CreatePath(table, numpos, show_stop);
}
```

Funzione CreatePath La funzione *CreatePath* crea un array di oggetti della classe *Percorso* partendo dalle posizioni contenute in una tabella e li

raggruppa in percorsi diversi in base al dispositivo a cui appartengono; se richiesto inoltre calcola il tempo di sosta verificato tra due punti nel caso in cui il veicolo si sia fermato.

Codice 13.14: Funzione CreatePaht

```
[WebMethod(Description = "Analizza i dati presenti nella tabella ed elabora una serie di percorsi")]
private Percorso[] CreatePath(DataTable table, int numpos, bool show_stop)
{
  Byte fix;
  Percorso[] percorso = new Percorso[numpos];
  try
  {
    int numperc = 0;

    percorso[numperc] = new Percorso();
    percorso[numperc].ID = (UInt32)table.Rows[0]["UID"];
    percorso[numperc].Posizioni = new List<Punto>();

    for (int i = 0; i < table.Rows.Count; i++)
    {
      DateTime time_stop;
      DateTime time_restart;
      TimeSpan time_diff;
      Punto pos = new Punto();
      if (percorso[numperc].ID != (UInt32)table.Rows[i]["UID"])
      {
        numperc++;
        percorso[numperc] = new Percorso();
        percorso[numperc].ID = (UInt32)table.Rows[i]["UID"];
        percorso[numperc].Posizioni = new List<Punto>();
      }
      if ((fix = (Byte)table.Rows[i]["Fix"]) > 1)
      {
        try
        {
          if (show_stop)
          {
            pos.StopTime = "";
            if (((Byte)table.Rows[i]["Stato"]) == 0)
            {
              time_stop = (DateTime)table.Rows[i]["Data"];
              int j = i + 1;
              if (j < table.Rows.Count - 1)
              {
                while ((j < table.Rows.Count - 1) && ((Byte)table.Rows[j]["Stato"] == 0) && ((UInt32)table.Rows[i]["UID"] == (UInt32)table.Rows[j]["UID"]))
                {
                  j++;
                }
                time_restart = (DateTime)table.Rows[j]["Data"];
                time_diff = time_restart - time_stop;
                if (time_diff.Days != 0)
```

```
47              {
                  pos.StopTime = time_diff.Days + "giorni " + time_diff.Hours + ":" +
        time_diff.Minutes + ":" + time_diff.Seconds;
49              }
                else
51              {
                  pos.StopTime = time_diff.Hours + ":" + time_diff.Minutes + ":" +
        time_diff.Seconds;
53              }
                }
55            }
            }
57          pos.ID = (UInt32)table.Rows[i]["UID"];
            pos.Nome = table.Rows[i]["Nome"].ToString();
59          pos.Latitudine = (Double)table.Rows[i]["Latitudine"];
            pos.Longitudine = (Double)table.Rows[i]["Longitudine"];
61          pos.Altitudine = (Int16)table.Rows[i]["Altezza"];
            pos.Velocita = (UInt16)table.Rows[i]["Velocita"];
63          pos.Direzione = (Int16)table.Rows[i]["Direzione"];
            pos.Data = ((DateTime)table.Rows[i]["Data"]).ToString("dd/MM/yyyy");
65          pos.Ora = ((DateTime)table.Rows[i]["Data"]).ToString("HH:mm:ss");
            pos.NumSat = (Byte)table.Rows[i]["NumeroSatelliti"];
67          if (table.Rows[i]["Indirizzo"].ToString() == "Non Elaborato")
            {
69            pos.Indirizzo = "";
              pos.Citta = "";
71          }
            else
73          {
              try
75            {
                pos.Indirizzo = table.Rows[i]["Indirizzo"].ToString().Split('-')[0] + " " +
        table.Rows[0]["Indirizzo"].ToString().Split('-')[1];
77              pos.Citta = table.Rows[i]["Indirizzo"].ToString().Split('-')[2];
              }
79            catch (Exception)
              {
81              pos.Indirizzo = "";
                pos.Citta = "";
83            }
            }
85          pos.Fix = fix;

87          percorso[numperc].Add(pos);
        }
89      catch (Exception)
        {
91      }
      }
93  }

95  return percorso;
    }
97  catch (Exception)
```

```
  {
    return null;
  }
}
```

Funzioni Schermata Analisi Le funzioni *SaveZoneInterrogationCircle* e *SaveZoneInterrogationRectangle* permettono di salvare le zone selezionate sulla mappa. Le funzioni *GetZonesName* e *GetZoneFigure* vengono utilizzate per ottenere le informazioni necessarie riguardo le zone. La funzioni *GetInfoOnMap* analizza le informazioni all'interno di una zona ed elabora i dati in modo da poterli visualizzare su mappa. La funzione *GetTableAnalysis* crea la tabella che verrà visualizzata nella pagina *AnalisiTabulato.aspx*.

Codice 13.15: Funzione GetInfoOnMap

```
[WebMethod(Description = "Ritorna le posizioni all'interno di una zona indicata, dei
      dispositivi indicati durante il periodo indicato")]
public Percorso[] GetInfoOnMap(string[] device, string datainizio, string datafine,
      string idzone, string latitfrom, string longitfrom, string latitto, string longitto
      )
{
  try
  {
    string querystring = "";
    if (idzone == "all")
    {
      querystring = "SELECT Dispositivo.UID AS UID, Dispositivo.Nome AS Nome,
        Datigps.Latitudine AS Latitudine, Datigps.Longitudine AS Longitudine,Datigps.
        Altezza AS Altezza, Datigps.Data AS Data, Datigps.Velocita AS Velocita,Datigps.
        Direzione AS Direzione, Datigps.StatoVeicolo AS Stato, Datigps.Fix AS Fix,
        Datigps.NumeroSatelliti AS NumeroSatelliti, Datigps.Indirizzo AS Indirizzo
        FROM Dispositivo, Datigps WHERE Datigps.Data BETWEEN ('datafrom')
        AND ('datato') AND Dispositivo.UID IN (iddevice) AND Dispositivo.UID =
        Datigps.Dispositivo_UID;";
    }
    else
    {
      if (idzone != "null")
      {
        ZoneFigure figure = GetZoneFigure(idzone);
        latitto = figure.LatNordEst.ToString().Replace(',','.');
        longitto = figure.LongNordEst.ToString().Replace(',', '.');
        latitfrom = figure.LatSudOvest.ToString().Replace(',', '.');
        longitfrom = figure.LongSudOvest.ToString().Replace(',', '.');
      }

      querystring = "SELECT Dispositivo.UID AS UID, Dispositivo.Nome AS Nome,
        Datigps.Latitudine AS Latitudine, Datigps.Longitudine AS Longitudine,Datigps.
        Altezza AS Altezza, Datigps.Data AS Data, Datigps.Velocita AS Velocita,
        Datigps.Direzione AS Direzione, Datigps.StatoVeicolo AS Stato, Datigps.Fix AS
```

```
        Fix, Datigps.NumeroSatelliti AS NumeroSatelliti, Datigps.Indirizzo AS Indirizzo
        FROM Dispositivo, Datigps WHERE Datigps.Latitudine BETWEEN ('latitfrom')
         AND ('latitto') AND Datigps.Longitudine BETWEEN ('longfrom') AND ('longto
        ') AND Datigps.Data BETWEEN ('datafrom') AND ('datato') AND Dispositivo.
        UID IN (iddevice) AND Dispositivo.UID = Datigps.Dispositivo_UID;";
      querystring = querystring.Replace("latitfrom", latitfrom);
      querystring = querystring.Replace("latitto", latitto);
      querystring = querystring.Replace("longfrom", longitfrom);
      querystring = querystring.Replace("longto", longitto);
    }

    int numdevice = device.Length;
    string param = "";
    for (int i = 0; i < device.Length; i++)
    {
      param += device[i] + ",";
    }

    param = param.Substring(0, param.Length - 1);
    DateTime inizio = DateTime.Parse(datainizio);
    DateTime fine = DateTime.Parse(datafine + ":59");
    String stringinizio = inizio.Year + "-" + inizio.Month + "-" + inizio.Day + " " +
        inizio.TimeOfDay;
    String stringfine = fine.Year + "-" + fine.Month + "-" + fine.Day + " " + fine.
        TimeOfDay;
    querystring = querystring.Replace("datafrom", stringinizio);
    querystring = querystring.Replace("datato", stringfine);
    querystring = querystring.Replace("iddevice", param);
    DataTable table = ExecuteSelectOnDb(querystring);

    return CreatePath(table, numdevice, true);
  }
  catch
  {
    return null;
  }
}
```

Codice 13.16: Funzione GetTableAnalysis

```
[WebMethod(Description = "Ritorna un datatable contenente le posizioni ricavate dalla
      richiesta di analisi fatta")]
public DataTable GetTableAnalysis(string query, string type_analysis)
{
  DataTable table = ExecuteSelectOnDb(query);
  DataColumn column;
  if (type_analysis.Equals("complete"))
  {
    column = new DataColumn();
    column.ColumnName = "Sosta";
    column.DataType = System.Type.GetType("System.String");
    table.Columns.Add(column);
    string sosta;
```

```
    TimeSpan time_diff;
14  DateTime time_stop;
    DateTime time_restart;
16
    for (int i = 0; i < table.Rows.Count; i++)
18  {
      sosta = "";
20    int j = i;
      if ((Byte)table.Rows[i]["Stato"] == 0)
22    {
        time_stop = (DateTime)table.Rows[i]["Data"];
24      while ((j < table.Rows.Count - 1) && ((Byte)table.Rows[j]["Stato"] == 0) &&
        ((UInt32)table.Rows[i]["ID"] == (UInt32)table.Rows[j]["ID"]))
        {
26        j++;
        }
28
        time_restart = (DateTime)table.Rows[j]["Data"];
30      time_diff = time_restart - time_stop;
        string day_diff = "";
32      string hour_diff = time_diff.Hours.ToString();
        string minute_diff = time_diff.Minutes.ToString();
34      string second_diff = time_diff.Seconds.ToString();

36      if (time_diff.Days != 0)
        {
38        day_diff = time_diff.Days + " giorni";
        }
40      if (time_diff.Hours < 10)
        {
42        hour_diff = "0" + time_diff.Hours;
        }
44      if (time_diff.Minutes < 10)
        {
46        minute_diff = "0" + time_diff.Minutes;
        }
48      if (time_diff.Seconds < 10)
        {
50        second_diff = "0" + time_diff.Seconds;
        }
52      sosta = day_diff + " " + hour_diff + ":" + minute_diff + ":" + second_diff;
      }
54    table.Rows[i]["Sosta"] = sosta;
    }
56
    table.Columns.Remove("Stato");
58  }
    else if (type_analysis.Equals("reduced"))
60  {

62  }
    else
64  {
      return null;
```

```
  }
  return table;
}
```

Funzione ExecuteChangeOnDb La funzione *ExecuteChangeOnDb* sfrutta la stringa query ricevuta per eseguire modifiche sul database come l'inserimento, la modifica o l'eliminazione di elementi e ritorna un booleano indicante se l'operazione è andata a buon fine o meno.

Codice 13.17: Funzione ExecuteChangeOnDb

```
private bool ExecuteChangeOnDb(string queryString)
{
  string connectionString = "Data Source=" + Config.Datasource + "; Database=" +
      Config.Database + ";User Id=" + Config.Userid + ";Password=" + Config.
      Password;
  MySqlConnection connection = new MySqlConnection(connectionString);
  try
  {
    if (connection.State != ConnectionState.Open)
    {
      connection.Open();
    }
  }
  catch
  {
    connection.Close();
    return false;
  }
  try
  {
    MySqlCommand ins = new MySqlCommand(queryString, connection);
    int ret = ins.ExecuteNonQuery();
    connection.Close();
    if (ret < 0)
    {
      return false;
    }
    return true;
  }
  catch
  {
    connection.Close();
    return false;
  }
}
```

Funzione ExecuteSelectOnDb La funzione *ExecuteSelectOnDb* sfrutta la stringa query ricevuta per ottenere i dati contenuti nel database e li fornisce all'interno di una struttura DataTable.

Codice 13.18: Funzione ExecuteSelectOnDb

```
private DataTable ExecuteSelectOnDb(string queryString)
{
  string connectionString = "Data Source=" + Config.Datasource + "; Database=" +
      Config.Database + ";User Id=" + Config.Userid + ";Password=" + Config.
      Password;
  DataSet tableset = new DataSet();
  MySqlConnection connection = new MySqlConnection(connectionString);
  try
  {
    if (connection.State != ConnectionState.Open)
    {
      connection.Open();
    }
  }
  catch
  {
    connection.Close();
    return null;
  }
  try
  {
    MySqlDataAdapter tableadapter = new MySqlDataAdapter(queryString, connection
        );
    tableadapter.Fill(tableset);
    connection.Close();
    return tableset.Tables[0];
  }
  catch
  {
    connection.Close();
    return null;
  }
}
```

Funzione CheckLogin La funzione *CheckLogin* viene utilizzata per controllare se i dati di login come user e password sono corretti e sono presenti sul database; se i dati sono corretti la funzione fornisce l'ID dell'utente con cui poter ricavare ulteriori informazioni necessarie.

Funzione GetInformationUser La funzione *GetInformationUser* permette di ottenere tutti i dati dei permessi associati all'utente con l'ID inserito come parametro, con i quali il sistema identificherà quali schermate rendere visibili all'utente.

Funzione GiveTree La funzione *GiveTree* elabora quali sono i gruppi di dispositivi e i dispositivi a cui ha accesso un utente e li raggruppa in una struttura con cui costruire il menu ad albero.

Funzioni di Hashing Vi sono due diverse funzioni di hashing; la prima viene utilizzata per criptare le password utilizzate nella procedura di login in modo da non memorizzare nel database le password in chiaro; la seconda viene utilizzata per creare un codice identificativo in base ai diversi parametri delle configurazioni, in questo modo due configurazioni con gli stessi parametri vengono identificate con lo stesso codice e quindi non vengono duplicate nel database.

Funzioni di Creazione Query Vi sono delle funzioni il cui scopo è creare le stringhe query con cui poter interagire con il database. Alcune di queste funzioni permettono di recuperare le informazioni nel database e si appoggiano alla funzione *ExecuteSelectOnDb* per eseguire la query creata; altre funzioni invece permettono di aggiungere, modificare ed eliminare dati dal database e si appoggiano alla funzione *ExecuteChangeOnDb* per eseguire la query creata.

Capitolo 14

Conclusioni e Sviluppi Futuri

Durante lo sviluppo di questa tesi si è creato un sistema in grado di gestire la comunicazione con dispositivi dotati di un sistema di rilevazione gps ed in grado di indicare la posizione del veicolo su cui è installato.
Successivamente è possibile memorizzare i dati ricevuti per poi renderli disponibili.
Infine tramite un sito web è possibile visualizzare i dati raccolti ed effettuare analisi sugli stessi, inoltre dal sito web è possibile gestire tutte le varie componenti di questo sistema.

Per quanto riguarda gli sviluppi futuri si potranno aggiungere dei sistemi per gestire altre funzionalità dei dispositivi come un sistema di allarmi nel caso i veicoli entrino od escano da determinate zone, oppure nel caso in cui vengano utilizzati in determinati orari; inoltre verranno aggiunti dei sistemi che permetteranno di gestire e raccogliere altri dati rilevati da sensori presenti sui veicoli.
Ulteriori migliorie sono possibili per rendere più efficiente il sistema di memorizzazione delle informazioni, soprattutto nel caso della ricezione di un gran quantitativo di dati da molte fonti contemporaneamente.

Bibliografia

Testi Consultati:

- Gabriel Svennerberg, Beginning Google Maps Api 3 - Apress
- Beneventano D., Bergamaschi S., Vincini M., Progetto di Basi di Dati Relazionali - Pitagora Editrice

Siti Internet:

- Wikipedia - www.wikipedia.it - www.wikipedia.com
- MSDN - http://msdn.microsoft.com/it-it/library/default.aspx
- Google Maps Javascript Api - https://developers.google.com/maps/documentation/javascript/
- MySQL Reference Manuals - http://dev.mysql.com/doc/
- jQuery - http://jquery.com/
- JavaScript SOAP Client - http://www.guru4.net/articoli/javascript-soap-client/
- HTML.IT - http://www.html.it/