Project Report
On

Voice Based Automated Transport Enquiry System

TABLE OF CONTENTS

INTRODUCTION

1. INTRODUCTION

1.1 Purpose

Now this is the age of speed. Everything happens in the speed of supersonic. The data can be transferred at the speed of light in the digital medium, can travel in the supersonic speed, hence three is a need of information inflow in the same speed. Here is one such need of information fast enough. We have experienced in waiting to a transport terminals for transport controllers to get the information about the transport facility. We encounter so many times there will be no person for providing these information which significantly wastes the time just to know whether there is any facility or not. Here is one solution for such a problem which lessens the human intervention in providing such information in the transport terminals.

Voice Based Automated Transport Enquiry System is the enquiry system which operates based on the voice input given by the user. There is no communication which is understood more appropriately than voice. This system too uses the voice commands and gives the required information in the form of voice.

This system is can be installed in any transport terminal like Bus stands, Railway terminals or airports.

1.2 About the Project

Voice Based Automated Transport Enquiry System is developed for providing the information for the enquiry in transport terminals.

This project is developed using .Net technology using c# Programming language. This uses sql server for storing the information to be provided to the user. This user Microsoft Speech recognition to detect the voice from the user and uses the speech control to deliver the voice output. This also displays the results on the screen for further verification.

1.3 Benefits

- It works in more interactive way in the form of speech.
- It needs less or no human intervention.
- It is automated.
- It needs very less maintenance.

1.4 Disadvantages

- Heavy noise is a very crowded place can disturb the result.

SYSTEM REQUIREMENTS SPECIFICATION

2. SYSTEM REQUIREMENTS SPECIFICATION

Requirement Specification is the activity of translating the information gathered during the analysis into a requirement document.

2.1 Classification

- User Requirements
- System Requirements
    - Hardware Requirements
    - Software Requirements
    - Functional Requirements
    - Non-Functional Requirements
    - Feasibility Study
- Software Design Specification

2.1.1 User Requirements

Voice Commands:

1. Command should be accepted in the form of voice.
2. The system should recognize the voice command
3. The system shall process the voice command
4. The appropriate information shall be retrieved from the database.
5. The retrieved information is read and the output is given in the form of voice.
6. The relevant information also shall be displayed in the screen.
7. User shall be able to move between the Previous and Next result for the same query.

8. User can change the query.
9. User can stop the current query from continuing.
10. User shall be able to add new commands.
11. User shall be able edit the timings of the routes.
12. Shall display all the commands.

2.1.2 System Requirements

A set of system services and constraints in detail, The System requirements are the more detailed specification of the User Requirements it some times serves as a contract between the user and the developer.

Software requirements

1. Microsoft .Net framework 2.0
2. Visual studio 2005
3. C# .Net
4. MS Speech SDK
5. MS SQL

Hardware requirements(minimum)

1. Processor	: Pentium IV
2. Monitor	: SVGA
3. RAM	: 128MB
4. Speed	: 1.5GHz
5. Secondary Device	: 20GB
6. Speaker	
7. Microphone	

2.2 Functional Requirements:

Voice Commands:

The commands are given to the system are in the form of voice commands. The given commands are processed using voice processing.

Speech Recognition:

The given command is to process using speech recognition.

Search the Result:

The system shall search the appropriate result according to the given command.

Display Result:

The system shall display the retrieved result on the screen.

Display Commands:

The system shall display the commands that are present in the system.

Manage Information:

The system shall provide option to add new information like route information and the timings at which the transport facility is available.

Browse through Result:

User shall be allowed to browse through the retrieved result. It shall allow the user to move to previous and next result through the voice commands.

Voice Output:

The retrieved result shall be converted to voice through the speech control of Microsoft.

2.3 Non-Functional Requirements

These are constraints on the services or functions offered by the system. They include constraints on the development process etc.

Non-Functional requirements are requirements which are not directly concerned with the specific functionality.

The nonfunctional requirement for the current system is that the voice commands are to be clear to be recognized and no intermediate noise are allowed in and around the system.

The other classifications are:

- Product requirements.
- Organizational requirements.
- External Requirements.

Product Requirements: The source code should use the tools provided by visual studio and software development kit. The product should provide user friendly interface so that the user can be benefited by utilizing the device.

Organizational Requirements: This is similar to the product requirements they are derived from the user policies and their requirements.

External Requirements: These requirements are derived from factors external to the system and its development process. This includes how the user is going to interact with the system.

Feasibility Study

The feasibility study concerns with the considerations made to verify whether the system is fit to be developed in all terms. Once an idea to develop the system is put forward, the question that arises first will pertain to the feasibility aspects. Feasibility study is a test of a system proposal according to its work ability.

In the feasibility study it focuses on mainly three questions:

- What is the user needs and how does the system meet them?
- What resources are available for given systems? Is the problem worth solving?
- What are the likely impact of the system on the organization and how it fit with the System plans?

In the feasibility study of the project was studied to various feasibility studies performed that is technical feasibility whether existing equipment, software were sufficient for completing the project.

DESIGN

3. DESIGN

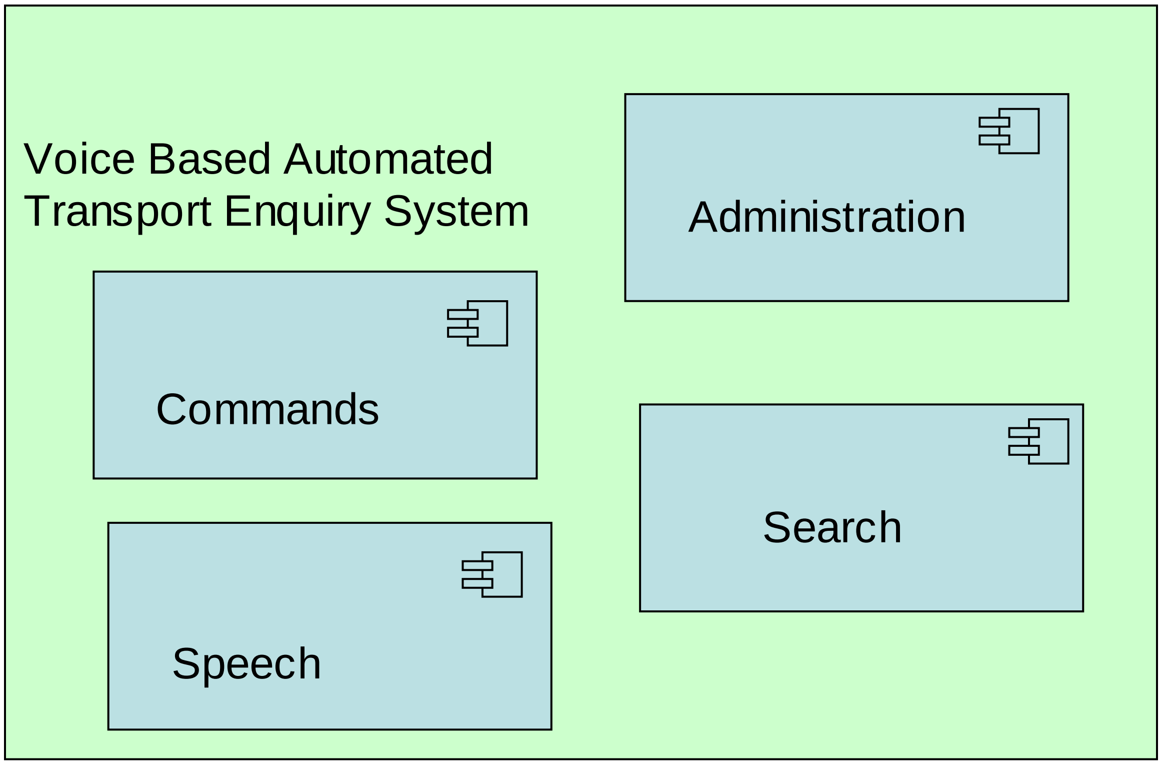
A software design is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between the system components and sometimes the algorithms used. Designers do not arrive at a finished design immediately but develop the design iteratively through a number of different versions. The design process involves adding formality and detail as the design is developed with constant backtracking to correct earlier designs.

3.1 Design Process

The specific design process activities are:

- **Architectural design**: The sub-system making up the system and their relationships are identified and documented.
- **Object oriented design**: In Object oriented design we thought of "things" instead of operations and functions, the executing system in made up of interacting objects that maintain their local state and operations on that state operation.
- **User interface design**: Good user interface design is critical to the success of the system, an interface that is difficult to use will, at best, result in a high level of user errors.

3.2 Architectural Design



The Componets which makes the current system is shown above. It has Four Componets which are listed below.

- Commands
- Speech
- Search
- Administration

Commands:

This is one of the major components of the current system which recognizes the commands given by the user. This component is responsible for recognizing the commands and interpreting the command and sending appropriate request to the Search component.

Search:

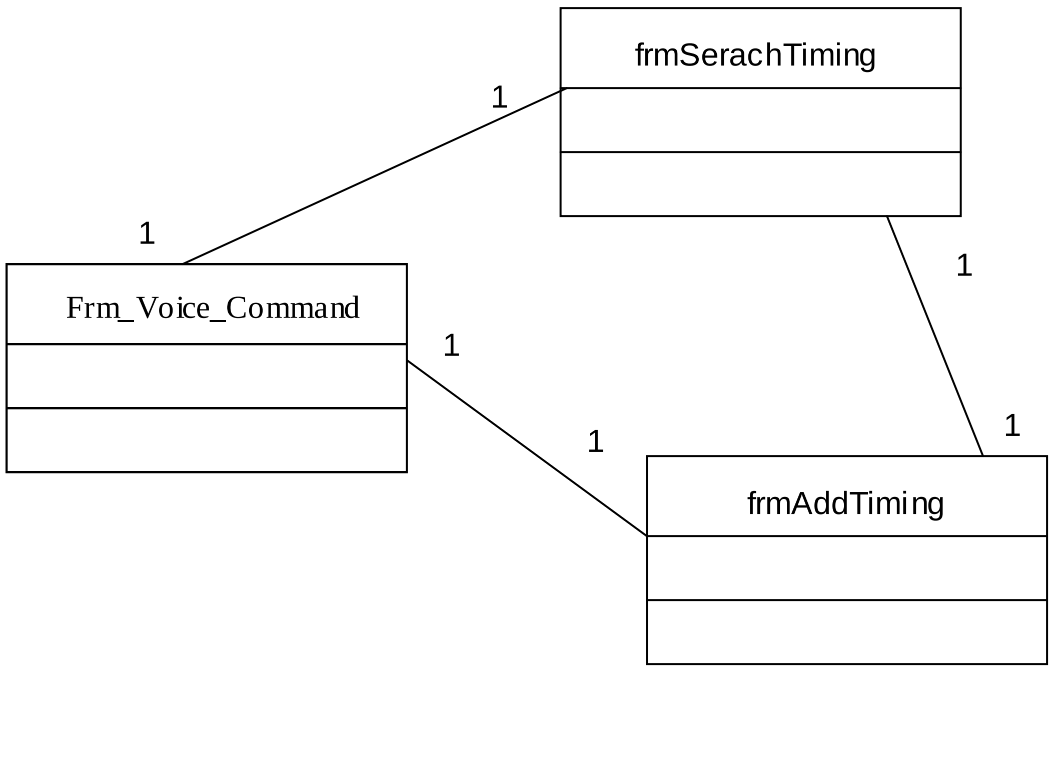
Search components take the input as the request from the Command component and retrieve the appropriate result from the database. it gives back to the display component and the speech component.

Speech:

This component is used to deliver the result in the form of the voice using Microsoft speech control. This takes input form the Search component.

Administration:

Through this component the maintenance personnel can update the information and also the commands to the system.



ASP.NET

4. Introduction to ASP.NET

ASP.NET is more than the next version of Active Server Pages (ASP); it provides a unified Web development model that includes the services necessary for developers to build enterprise-class Web applications. While ASP.NET is largely syntax compatible with ASP, it also provides a new programming model and infrastructure for more scalable and stable applications that help provide greater protection. You can feel free to augment your existing ASP applications by incrementally adding ASP.NET functionality to them.

ASP.NET is a compiled, .NET-based environment; you can author applications in any .NET compatible language, including Visual Basic .NET, C#, and JScript .NET. Additionally, the entire .NET Framework is available to any ASP.NET application. Developers can easily access the benefits of these technologies, which include the managed common language runtime environment, type safety, inheritance, and so on.

ASP.NET has been designed to work seamlessly with WYSIWYG HTML editors and other programming tools, including Microsoft Visual Studio .NET. Not only does this make Web development easier, but it also provides all the benefits that these tools have to offer, including a GUI that developers can use to drop server controls onto a Web page and fully integrated debugging support.

Developers can use Web Forms or XML Web services when creating an ASP.NET application, or combine these in any way they see fit. Each is supported by the same infrastructure that allows you to use authentication schemes, cache frequently used data, or customize your application's configuration, to name only a few possibilities.

Web Forms allow you to build powerful forms-based Web pages. When building these pages, you can use ASP.NET server controls to create common UI elements, and program them for common tasks. These controls allow you to rapidly build a Web Form out of reusable built-in or custom components, simplifying the code of a page. For more information, see Web Forms Pages. For information on how to develop ASP.NET server controls, see Developing ASP.NET Server Controls.

An XML Web service provides the means to access server functionality remotely. Using XML Web services, businesses can expose programmatic interfaces to their data or business logic, which in turn can be obtained and manipulated by client and server applications. XML Web services enable the exchange of data in client-server or server-server scenarios, using standards like HTTP and XML messaging to move data across firewalls. XML Web services are not tied to a particular component technology or object-calling convention. As a result, programs written in any language, using any component model, and running on any operating system can access XML Web services. For more information, see XML Web Services Created Using ASP.NET and XML Web Service Clients.

Each of these models can take full advantage of all ASP.NET features, as well as the power of the .NET Framework and .NET Framework common language runtime. These features and how you can use them are outlined as follows:

If you have ASP development skills, the new ASP.NET programming model will seem very familiar to you. However, the ASP.NET object model has changed significantly from ASP, making it more structured and object-oriented. Unfortunately this means that ASP.NET is not fully backward compatible; almost all existing ASP pages will have to be modified to some extent in order to run under ASP.NET. In addition, major changes to Visual Basic .NET mean that existing ASP pages written with Visual Basic Scripting Edition typically will not port directly to ASP.NET. In most cases, though, the necessary changes will involve only a few lines of code. For more information, see Migrating from ASP to ASP.NET.

Accessing databases from ASP.NET applications is an often-used technique for displaying data to Web site visitors. ASP.NET makes it easier than ever to access databases for this purpose. It also allows you to manage the database from your code. For more information, see Accessing Data with ASP.NET.

ASP.NET provides a simple model that enables Web developers to write logic that runs at the application level. Developers can write this code in the Global.asax text file or in a compiled class deployed as an assembly. This logic can include application-level events, but developers can easily extend this model to suit the needs of their Web application. For more information, see ASP.NET Applications.

ASP.NET provides easy-to-use application and session-state facilities that are familiar to ASP developers and are readily compatible with all other .NET Framework APIs. For more information, see ASP.NET State Management.

For advanced developers who want to use APIs as powerful as the ISAPI programming interfaces that were included with previous versions of ASP, ASP.NET offers the IHttpHandler and IHttpModule interfaces. Implementing the **IHttpHandler** interface gives you a means of interacting with the low-level request and response services of the IIS Web server and provides functionality much like ISAPI extensions, but with a simpler programming model. Implementing the **IHttpModule** interface allows you to include custom events that participate in every request made to your application. For more information, see HTTP Runtime Support.

ASP.NET takes advantage of performance enhancements found in the .NET Framework and common language runtime. Additionally, it has been designed to offer significant performance improvements over ASP and other Web development platforms. All ASP.NET code is compiled, rather than interpreted, which allows early binding, strong typing, and just-in-time (JIT) compilation to native code, to name only a few of its benefits. ASP.NET is also easily factorable, meaning that developers can remove modules (a session module, for instance) that are not relevant to the application they are developing. ASP.NET also provides extensive caching services (both built-in services and caching APIs). ASP.NET also ships with performance counters that developers and system administrators can monitor to test new applications and gather metrics on existing applications. For more information, see ASP.NET Caching Features and ASP.NET Optimization.

Writing custom debug statements to your Web page can help immensely in troubleshooting your application's code. However, they can cause embarrassment if they are not removed. The problem is that removing the debug statements from your pages when your application is ready to be ported to a production server can require significant effort. ASP.NET offers the Trace Context class, which allows you to write custom debug statements to your pages as you develop them. They appear only when you have enabled tracing for a page or entire application. Enabling tracing also appends details about a request to the page, or, if you so specify, to a custom trace viewer that is stored in the root directory of your application. For more information, see ASP.NET Trace.

The .NET Framework and ASP.NET provide default authorization and authentication schemes for Web applications. You can easily remove, add to, or replace these schemes, depending upon the needs of your application. For more information, see Securing ASP.NET Web Applications.

ASP.NET configuration settings are stored in XML-based files, which are human readable and writable. Each of your applications can have a distinct configuration file and you can extend the configuration scheme to suit your requirements. For more information, see ASP.NET Configuration.

Applications are said to be running side by side when they are installed on the same computer but use different versions of the .NET Framework. To learn how to use different versions of ASP.NET for separate applications on your server, see Side-by-Side Support in ASP.NET.

IIS 6.0 uses a new process model called worker process isolation mode, which is different from the process model used in previous versions of IIS. ASP.NET uses this process model by default when running on Windows Server 2003. For information about how to migrate ASP.NET process model settings to worker process isolation mode, see IIS 6.0 Application Isolation Modes.

C#

5. C#

C# (pronounced "see sharp") is a multi-paradigm programming language encompassing imperative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within the .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270). C# is one of the programming languages designed for the Common Language Infrastructure.

Features

By design, C# is the programming language that most directly reflects the underlying Common Language Infrastructure (CLI). Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate Common Intermediate Language (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or FORTRAN.

Some notable distinguishing features of C# are:

- There are no global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.
- Local variables cannot shadow variables of the enclosing block, unlike C and C++. Variable shadowing is often considered confusing by C++ texts.
- C# supports a strict Boolean datatype, bool. Statements that take conditions, such as while and if, require an expression of a type that implements the true operator, such as the boolean type. While C++ also has a boolean type, it can be freely converted to and from integers, and expressions such as if(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach on the grounds that forcing programmers to use expressions that return exactly bool can

prevent certain types of common programming mistakes in C or C++ such as if (a = b) (use of assignment = instead of equality ==).

- In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined null value; it is impossible to obtain a reference to a "dead" object (one which has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System.IntPtr type, but it cannot dereference them.
- Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory which is no longer needed.
- In addition to the try...catch construct to handle exceptions, C# has a try...finally construct to guarantee execution of the code in the finally block.
- Multiple inheritance is not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI.
- C# is more type safe than C++. The only implicit conversions by default are those which are considered safe, such as widening of integers. This is enforced at compile-time, during JIT, and, in some cases, at runtime. There are no implicit conversions between booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ copy constructors and conversion operators, which are both implicit by default.
- Enumeration members are placed in their own scope.
- C# provides properties as syntactic sugar for a common pattern in which a pair of methods, accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class.
- Full type reflection and discovery is available.

Speech Recognition

6. How Speech Recognition Works

Overview

You might have already used speech recognition in products, and maybe even incorporated it into your own application, but you still don't know how it works. This document will give you a technical overview of speech recognition so you can understand how it works, and better understand some of the capabilities and limitations of the technology.

Speech recognition fundamentally functions as a pipeline that converts PCM (Pulse Code Modulation) digital audio from a sound card into recognized speech. The elements of the pipeline are:

1. Transform the PCM digital audio into a better acoustic representation
2. Apply a "grammar" so the speech recognizer knows what phonemes to expect. A grammar could be anything from a context-free grammar to full-blown English.
3. Figure out which phonemes are spoken.
4. Convert the phonemes into words.

Transform the PCM digital audio

The first element of the pipeline converts digital audio coming from the sound card into a format that's more representative of what a person hears. The digital audio is a stream of amplitudes, sampled at about 16,000 times per second. If you visualize the incoming data, it looks just like the output of an oscilloscope. It's a wavy line that periodically repeats while the user is speaking. While in this form, the data isn't useful to speech recognition because it's too difficult to identify any patterns that correlate to what was actually said.

To make pattern recognition easier, the PCM digital audio is transformed into the "frequency domain." Transformations are done using a windowed fast-Fourier transform. The output is similar to what a spectrograph produces. In frequency domain, you can identify the frequency components of a sound. From the frequency components, it's possible to approximate how the human ear perceives the sound.

The fast Fourier transform analyzes every 1/100th of a second and converts the audio data into the frequency domain. Each 1/100th of a second results is a graph of the amplitudes of frequency components, describing the sound heard for that 1/100th of a second. The speech recognizer has a database of several thousand such graphs (called a codebook) that identify different types of sounds the human voice can make. The sound is "identified" by matching it to its closest entry in the codebook, producing a number that describes the sound. This number is called the "feature number." (Actually, there are several feature numbers generated for every 1/100 th of a second but the process is easier to explain assuming only one.)

The input to the speech recognizer began as a stream of 16,000 PCM values per second. By using fast Fourier transforms and the codebook, it is boiled down into essential information, producing 100 feature numbers per second.

This doesn't work because of a number of reasons:

- Every time a user speaks a word it sounds different. Users do not produce exactly the same sound for the same phoneme.
- The background noise from the microphone and user's office sometimes causes the recognizer to hear a different vector than it would have if the user was in a quiet room with a high quality microphone.
- The sound of a phoneme changes depending on what phonemes surround it. The "t" in "talk" sounds different than the "t" in "attack" and "mist".
- The sound produced by a phoneme changes from the beginning to the end of the phoneme, and is not constant. The beginning of a "t" will produce different feature numbers than the end of a "t".

Text-to-Speech

7. How Text-to-Speech Works

Overview

You might have already used text-to-speech in products, and maybe even incorporated it into your own application, but you still don't know how it works. This document will give you a technical overview of text-to-speech so you can understand how it works, and better understand some of the capabilities and limitations of the technology.

Text-to-speech fundamentally functions as a pipeline that converts text into PCM digital audio. The elements of the pipeline are:

1. Text normalization
2. Homograph disambiguation
3. Word pronunciation
4. Prosody
5. Concatenate wave segments

I'll cover each of these steps individually

7.1 Text Normalization

The "text normalization" component of text-to-speech converts any input text into a series of spoken words. Trivially, text normalization converts a string like "John rode home." to a series of words, "john", "rode", "home", along with a marker indicating that a period occurred. However, this gets more complicated when strings like "John rode home at 23.5 mph", where "23.5 mph" is converted to "twenty three point five miles per hour".

7.2 Homograph Disambiguation

The next stage of text-to-speech is called "homograph disambiguation." Often it’s not a stage by itself, but is combined into the text normalization or pronunciation components. we’ve separated homograph disambiguation out since it doesn’t fit cleanly into either.

Text-to-speech engines use a variety of techniques to disambiguate the pronunciations. The most robust is to try to figure out what the text is talking about and decide which meaning is most appropriate given the context. Once the right meaning is know, it’s usually easy to guess the right pronunciation.

7.3 Word Pronunciation

The pronunciation module accepts the text, and outputs a sequence of phonemes, just like you see in a dictionary.

To get the pronunciation of a word, the text-to-speech engine first looks the word up in it’s own pronunciation lexicon. If the word is not in the lexicon then the engine reverts to "letter to sound" rules.

The letter-to-sound rules are "trained" on a lexicon of hand-entered pronunciations. The lexicon stores the word and it’s pronunciation, such as:

hello h eh l oe

An algorithm is used to segment the word and figure out which letter "produces" which sound. You can clearly see that "h" in "hello" produces the "h" phoneme, the "e" produces the "eh" phoneme, the first "l" produces the "l" phoneme, the second "l" nothing, and "o" produces the "oe" phoneme. Of course, in other words the individual letters produce different phonemes. The "e" in "he" will produce the "ee" phoneme.

Once the words are segmented by phoneme, another algorithm determines which letter or sequence of letters is likely to produce which phonemes. The first pass figures out the most likely phoneme generated by each letter. "H" almost always generates the "h" sound, while "o" almost always generates the "ow" sound. A secondary list is generated, showing exceptions to the previous rule given the context of the surrounding letters. Hence, an exception rule might specify that an "o" occurring at the end of the word and preceded by an "l" produces an "oe" sound. The list of exceptions can be extended to include even more surrounding characters.

When the letter-to-sound rules are asked to produce the pronunciation of a word they do the inverse of the training model. To pronounce "hello", the letter-to-sound rules first try to figure out the sound of the "h" phoneme. It looks through the exception table for an "h" beginning the word followed by "e"; Since it can't find one it uses the default sound for "h", which is "h". Next, it looks in the exceptions for how an "e" surrounded by "h" and "l" is pronounced, finding "eh". The rest of the characters are handled in the same way.

7.4 Prosody

Prosody is the pitch, speed, and volume that syllables, words, phrases, and sentences are spoken with. Without prosody text-to-speech sounds very robotic, and with bad prosody text-to-speech sounds like it's drunk.

The technique that engines use to synthesize prosody varies, but there are some general techniques.

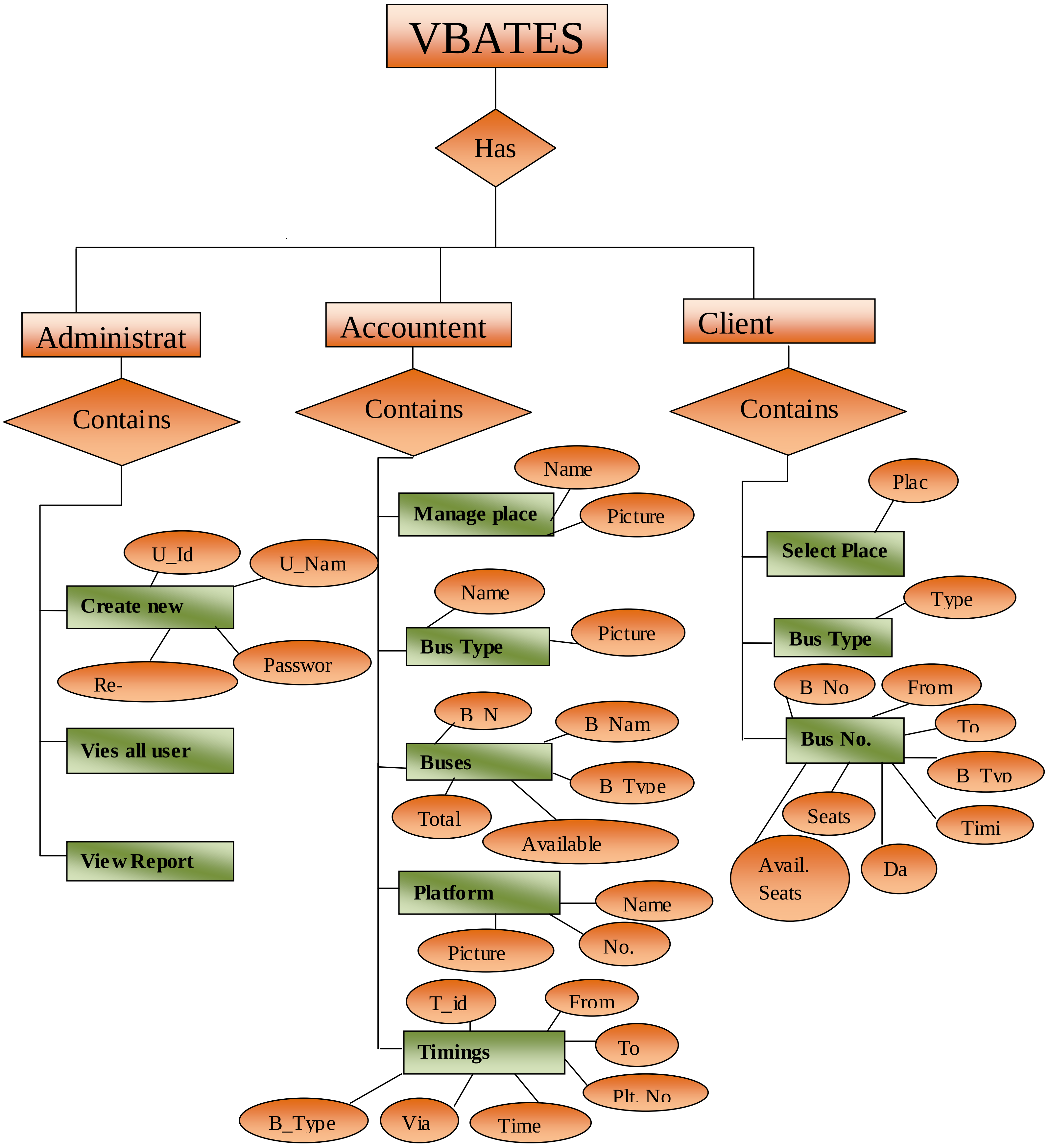
First, the engine identifies the beginning and ending of sentences. In English, the pitch will tend to fall near the end of a statement, and rise for a question. Likewise, volume and speaking speed ramp up when the text-to-speech first starts talking, and fall off on the last word when it stops. Pauses are placed between sentences.

Engines also identify phrase boundaries, such as noun phrases and verb phrases. These will have similar characteristics to sentences, but will be less pronounced. The engine can determine the phrase boundaries by using the part-of-speech information generated during the homograph disambiguation. Pauses are placed between phrases or where commas occur.

Algorithms then try to determine which words in the sentence are important to the meaning, and these are emphasized. Emphasized words are louder, longer, and will have more pitch variation. Words that are unimportant, such as those used to make the sentence grammatically correct, are de-emphasized. In a sentence such as "John and Bill walked to the store," the emphasis pattern might be "JOHN and BILL walked to the STORE." The more the text-to-speech engine "understands" what's being spoken, the better it's emphasis will be.

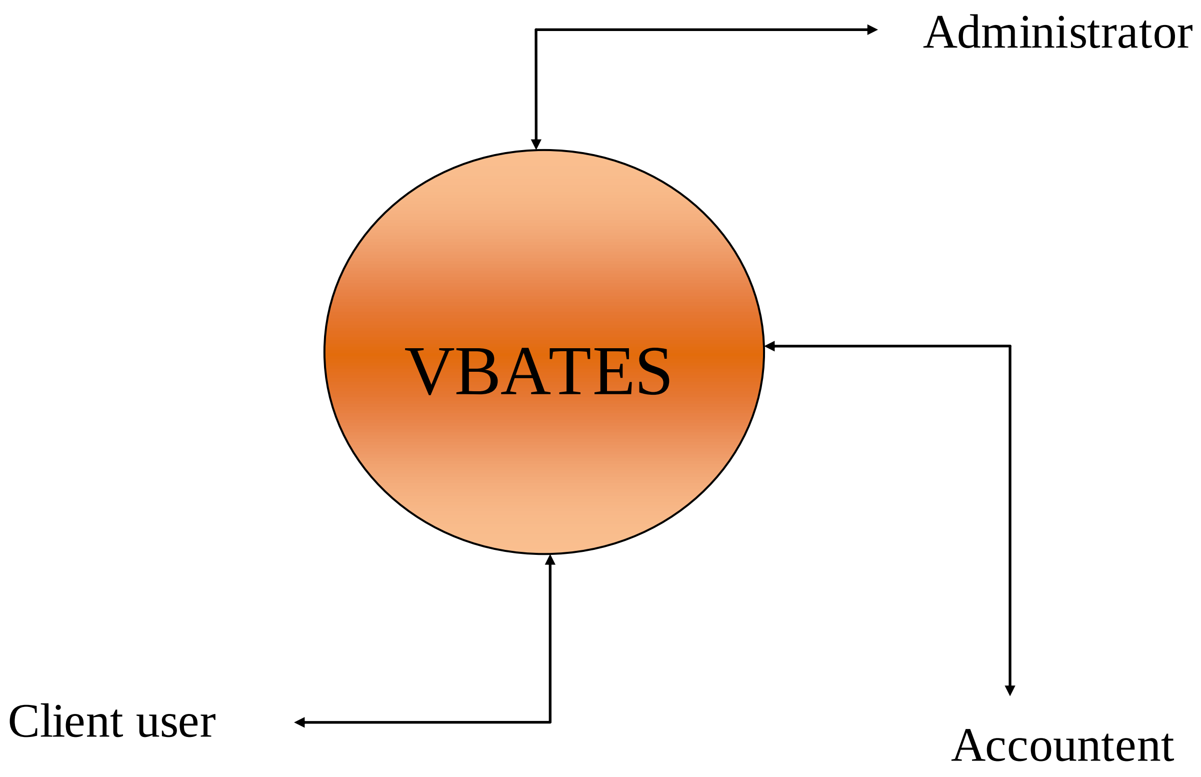
ER-DIAGRAM

8. ER-DIAGRAM


VBATES
Has
Administrat
Accountent
Client
Contains
Contains
Contains
Create new
U_Id
U_Nam
Re-
Passwor
Vies all user
View Report
Manage place
Name
Picture
Bus Type
Name
Picture
Buses
B_N
B_Nam
B_Type
Total
Available
Platform
Name
No.
Picture
Timings
T_id
From
To
Plt. No
B_Type
Via
Time
Select Place
Plac
Bus Type
Type
Bus No.
B_No
From
To
B_Typ
Timi
Seats
Da
Avail. Seats

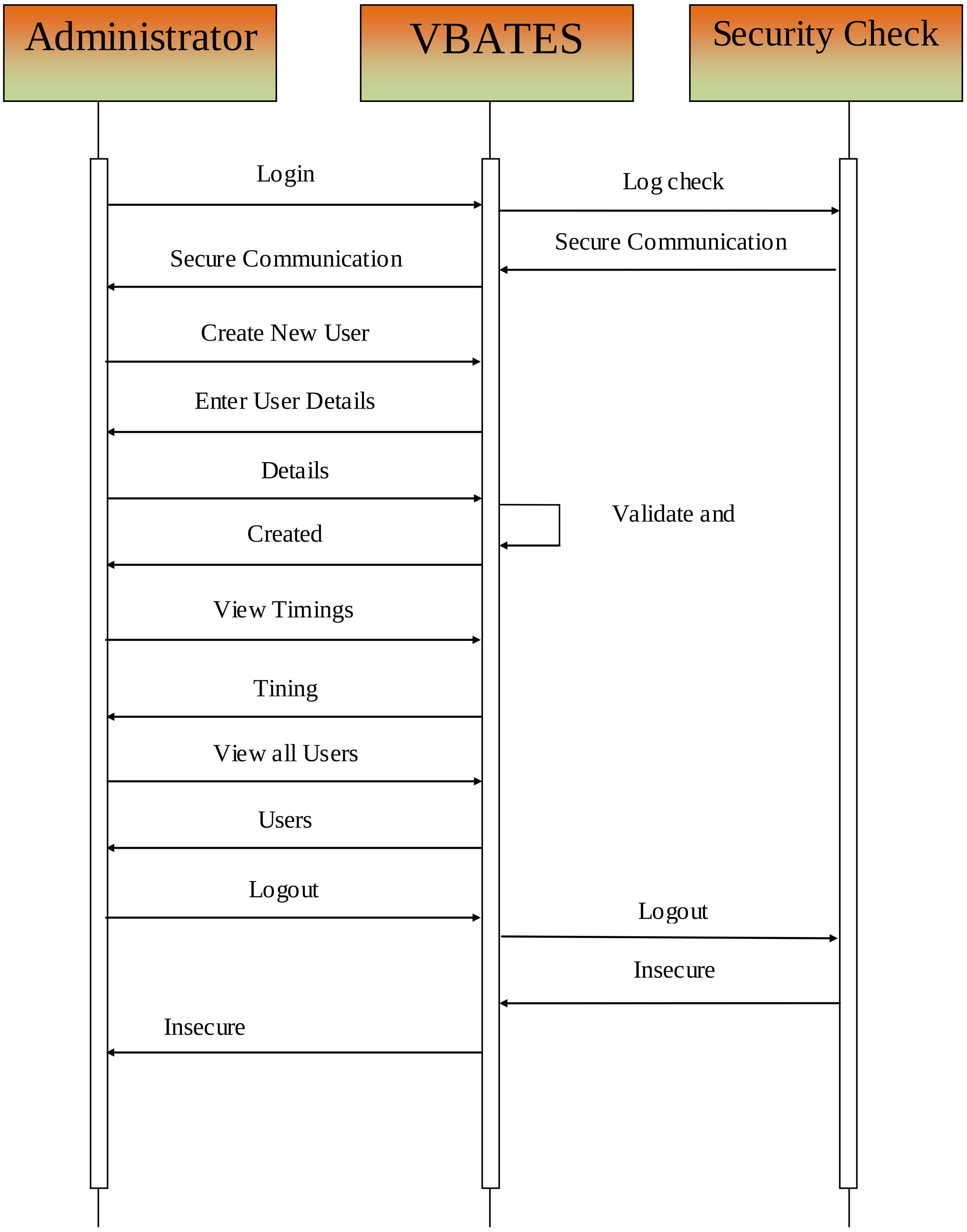
DATA FLOW

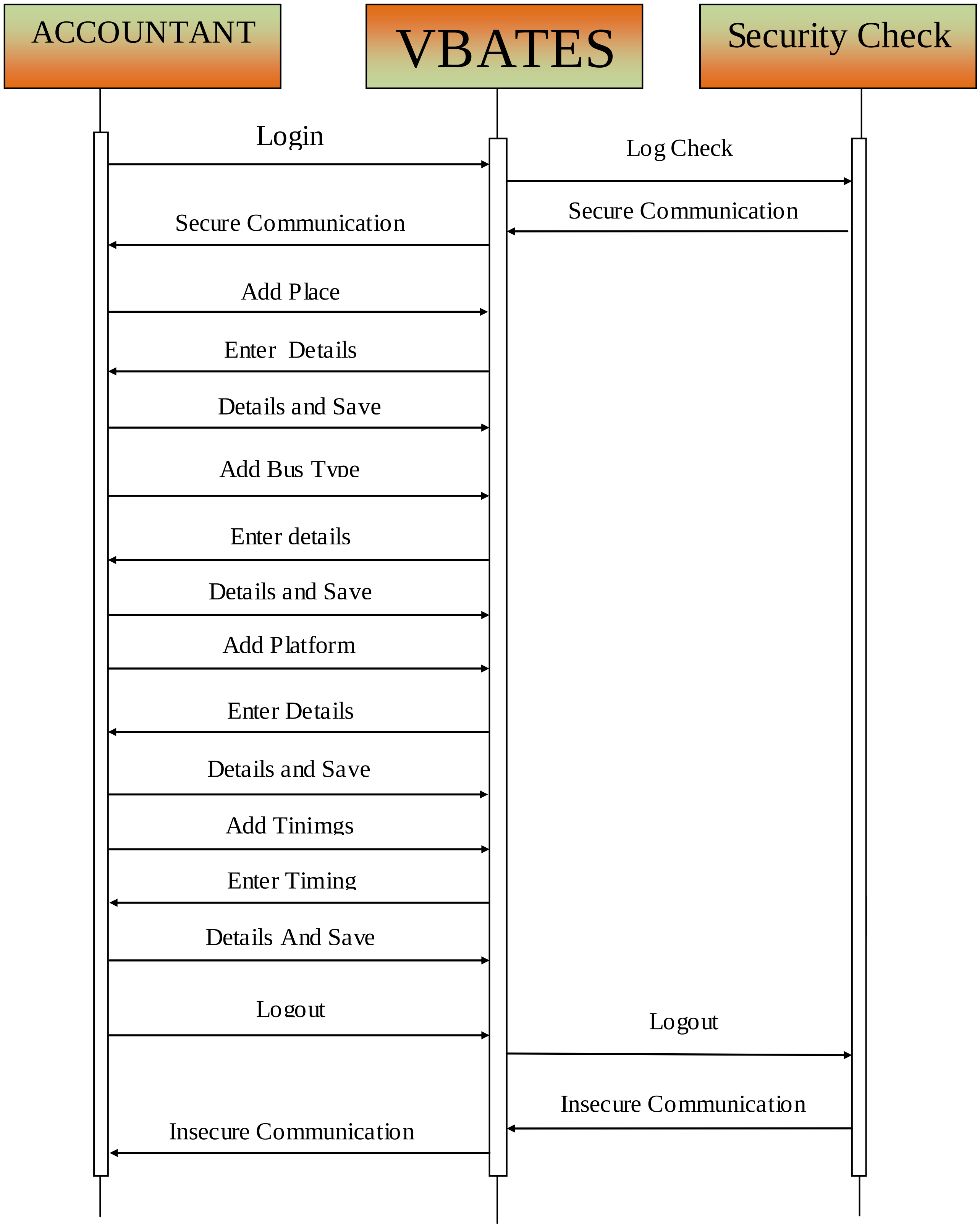
9. DATA FLOW DIAGRAM

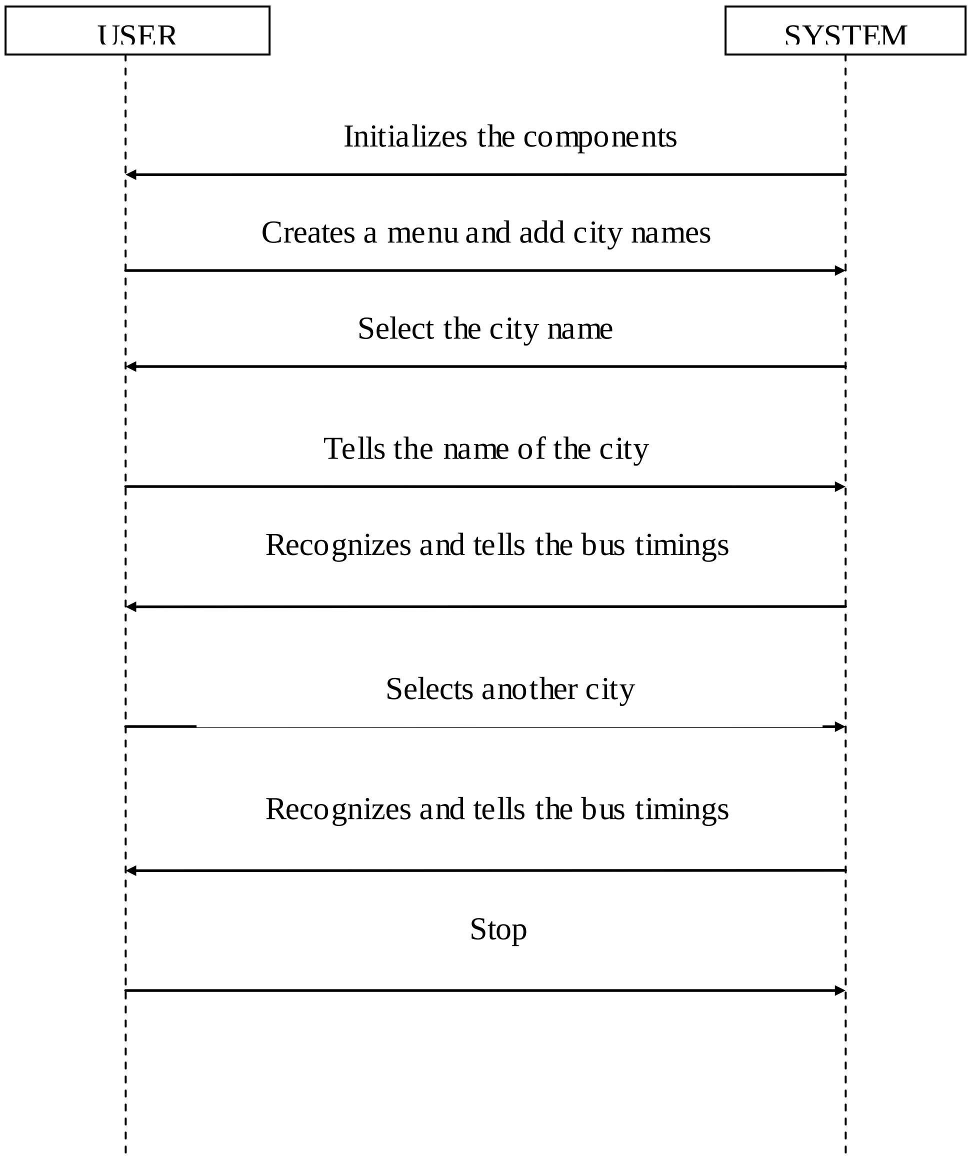


SEQUENCE DIAGRAM

10. Sequence Diagram

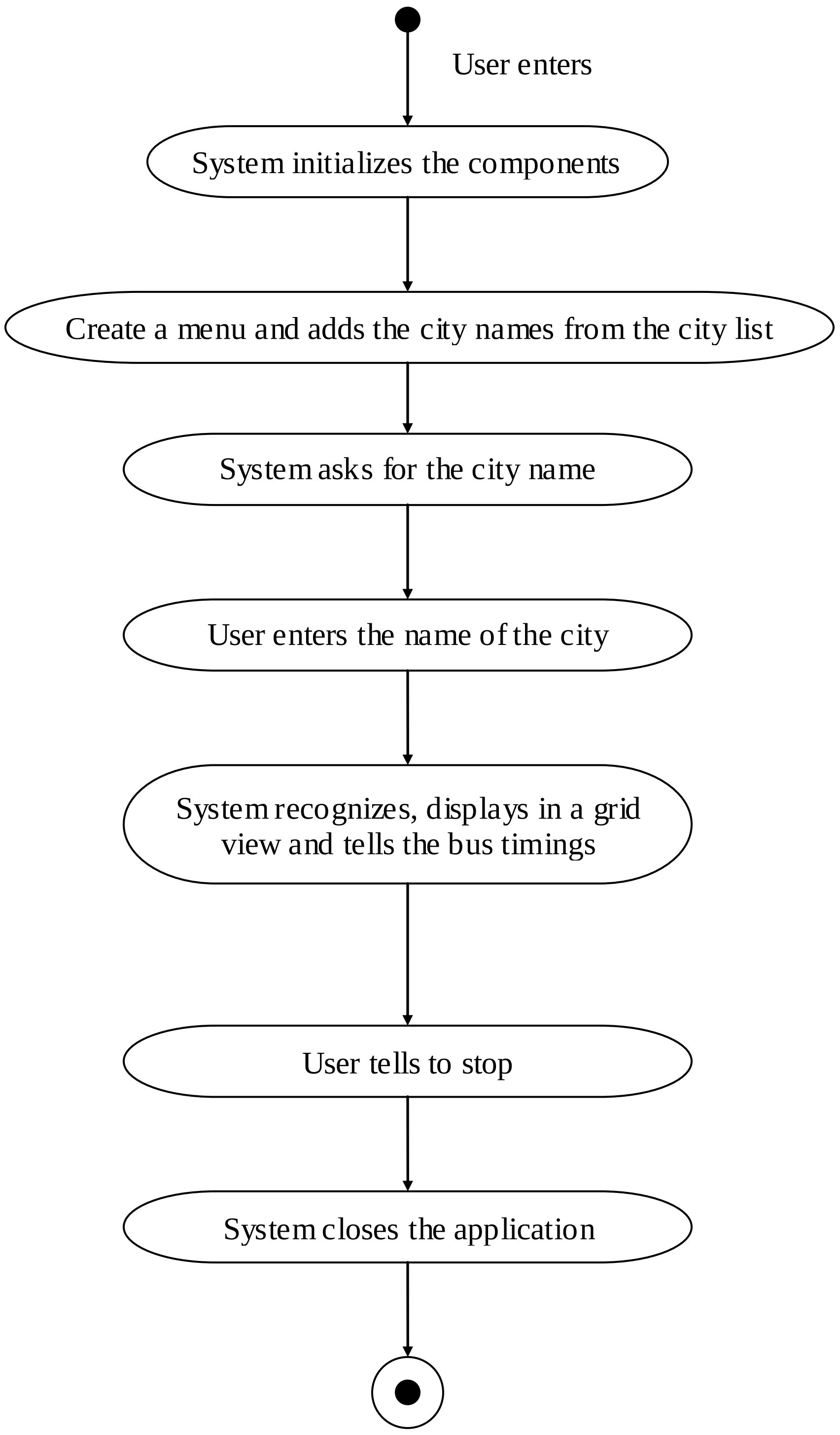



ACCOUNTANT
VBATES
Security Check
Login
Log Check
Secure Communication
Secure Communication
Add Place
Enter Details
Details and Save
Add Bus Type
Enter details
Details and Save
Add Platform
Enter Details
Details and Save
Add Tinimgs
Enter Timing
Details And Save
Logout
Logout
Insecure Communication
Insecure Communication


USER
SYSTEM
Initializes the components
Creates a menu and add city names
Select the city name
Tells the name of the city
Recognizes and tells the bus timings
Selects another city
Recognizes and tells the bus timings
Stop

FLOW DIAGRAM

11. FLOW DIAGRAM


User enters
System initializes the components
Create a menu and adds the city names from the city list
System asks for the city name
User enters the name of the city
System recognizes, displays in a grid view and tells the bus timings
User tells to stop
System closes the application

CODE

12. SAMPLE CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace ATSNew1
{
    public partial class Form1 : Form
    {
        string ConstPath = @"E:\Final ATS C#\WebATES";
        private int pvMenu;
        private int Loop1;
        private int TCount;
        private int pageSHow;
        bool Process = false;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            SpeakWords.Visible = false;
            VoiceCmd.Visible = false;
            this.LoadFrom();
            pageSHow = 0;
            this.ResetUI();
            this.GuideCustomer("Wel come To Automated Transport Enquiry
System.... Select The Option Listed Bellow.... ");

        }



        private void LoadFrom()
        {
            // Initialize the voice control...
            VoiceCmd.Initialized = 1;
            // Create and return a Menu control
            pvMenu = VoiceCmd.get_MenuCreate("ATSEn", "Home", 4);
            // Enable our voice control
            VoiceCmd.CtlEnabled = 1;
            // Suppress any voice errors that may occur
            //VoiceCmd.SuppressExceptions = 1
            // Load our list of commands into the menu.
            lbMenu.Items.Clear();
            TCount = VoiceCmd.get_CountCommands(pvMenu);
            if (TCount > 0)
            {
                for (Loop1 = TCount; Loop1 >= 1; Loop1 += -1)
                {
                    VoiceCmd.Remove(pvMenu, Loop1);
                }

            }

      VoiceCmd.AddCommand(pvMenu, 4, "Next", "Next", "Wel-come list", 0, "");
      VoiceCmd.AddCommand(pvMenu, 5, "Back", "Back", "Wel-come list", 0, "");
      VoiceCmd.AddCommand(pvMenu, 6, "Exit", "Exit", "Wel-come list", 0, "");
      VoiceCmd.AddCommand(pvMenu, 8, "Change Option", "Option", "Wel-come
list", 0, "");
      VoiceCmd.AddCommand(pvMenu, 7, "From To", "From To", "Wel-come list",
0, "");
            lbMenu.Items.Add("From To");
            this.AddCommandToVoiceControl();
            VoiceCmd.Activate(pvMenu);
        }
        DataManager dm = new DataManager();
        private void AddCommandToVoiceControl()
        {
            DataSet dsP = dm.getPlaceList();
            lbPlacelist.DataSource = dsP.Tables[0];
            lbPlacelist.DisplayMember = "Place";
            lbPlacelist.ValueMember = "Place";
            int i = 9;
            foreach (DataRow dr1 in dsP.Tables[0].Rows)
            {
          VoiceCmd.AddCommand(pvMenu, i, dr1["Place"].ToString(), "Place",
"Palce list", 0, "");
                i++;
            }
            DataSet dsBusType = dm.GetBusType();
            lbBusType.DataSource = dsBusType.Tables[0];
            lbBusType.DisplayMember = "Name";
            lbBusType.ValueMember = "Name";
            foreach (DataRow dr2 in dsBusType.Tables[0].Rows)
            {
                // Add Command to Menu1
                VoiceCmd.AddCommand(pvMenu, i, dr2["Name"].ToString(),
"BusType", "BusTypelistenlist", 0, "");
                i++;
            }
            DataSet dsBusList = dm.getBusList();
            foreach (DataRow dr3 in dsBusList.Tables[0].Rows)
            {
                // Add Command to Menu1
                VoiceCmd.AddCommand(pvMenu, i, dr3["Number"].ToString(),
"BusList", "BusListlistenlist", 0, "");
                i++;
            }

        }
        private void GuideCustomer(string prmString)
        {
            SpeakWords.AudioReset();
            SpeakWords.Speak(prmString);
        }


        private void VoiceCmd_CommandRecognize(object sender,
AxHSRLib._VcommandEvents_CommandRecognizeEvent e)
        {
            RecogVoice(e.command.ToString());
        }

        private string pvPreviousCmd;


        private void RecogVoice(string prmCommand)
        {
            //  Process = true;
            if (pvPreviousCmd == prmCommand)
            {
                // return;
            }
            pvPreviousCmd = prmCommand;

            this.Text = prmCommand;
            if (prmCommand != "")
            {

                switch (prmCommand)
                {
                    case "Next":
                        lblNext.ForeColor = Color.Maroon;
                        lblBack.ForeColor = Color.Blue;
                        lblChangeOption.ForeColor = Color.Blue;
                        NextCommand();
                        break;
                    case "Back":
                        lblNext.ForeColor = Color.Blue;
                        lblChangeOption.ForeColor = Color.Blue;
                        lblBack.ForeColor = Color.Maroon;
                        BackCommand();
                        break;
                    case "Change Option":
                        lblNext.ForeColor = Color.Blue;
                        lblBack.ForeColor = Color.Blue;
                        lblChangeOption.ForeColor = Color.Maroon;
                        ChangeOptionCommand();
                        break;
                    case "Exit":
                        ExitCommand();
                        break;
                    case "From To":
                        lblNext.ForeColor = Color.Blue;
                        lblBack.ForeColor = Color.Blue;
                        lblChangeOption.ForeColor = Color.Blue;
                        FromToCommand();
                        break;
                    default:
                        lblNext.ForeColor = Color.Blue;
                        lblBack.ForeColor = Color.Blue;
                        lblChangeOption.ForeColor = Color.Blue;
                        if (panelPlacelist.Visible == true)
                        {
                            if (PlaceSelected == false)
                            {
                                this.ChoosePlace(prmCommand);
                            }
                            else
                            {
                                this.GuideCustomer("You have already Selected
place... To Change Option say, Change option....");
                         lblChangeOption.Text = "Already  Selected place...";

                            }
                        }
                        if (panelBusType.Visible == true)
                        {
                            if (BusTypeSelected == false)
                            {
                                this.ChooseBusType(prmCommand);
                            }
                            else
                            {
                                this.GuideCustomer("You have already Selected
Bus Type... To Change Option say, Change option....");
                      lblChangeOption.Text = "Already Selected Bus Type...";

                            }
                        }
                        if (panelBusList.Visible == true)
                        {

                            if (BusSelected == false)
                            {

                                        this.ChooseBus(prmCommand);


                                System.Threading.Thread.Sleep(500);

                            }
                            else
                            {
                                this.GuideCustomer("You have already Selected
Bus... To Change Option say, Change option....");
                                lblChangeOption.Text = "Already Selected
Bus...";

                            }
                        }
                        break;




        }
        bool PlaceSelected = false;
        private void ChoosePlace(string Place)
        {

            this.GuideCustomer("You Have Choosen " + Place + ", To Continue
Say Next...");
            lblChangeOption.Text = Place;
            PlaceSelected = true;
            lbPlacelist.SelectedValue = Place;
            dm.ToPlace = Place;

            string s1 = ATES.DisplayPlaceImage(Place);
            pbPlacePic.ImageLocation = ConstPath + s1;
            if ((!string.IsNullOrEmpty(dm.ToPlace)) &&
(!string.IsNullOrEmpty(dm.BusType)))
            {
               //bus this.getBusResult();
                this.GetResultBus();
            }
        }
        bool BusTypeSelected = false;
        DataSet dsBuses;
        private void ChooseBusType(string PrmBusType)
        {
            if (dsBuses != null)
            {
                dsBuses.Clear();
            }
            this.GuideCustomer("You Have Choosen " + PrmBusType + "To
Continue Say Next");
            BusTypeSelected = true;
            lblChangeOption.Text = PrmBusType;

            lbBusType.SelectedValue = PrmBusType;
            dm.BusType = PrmBusType;
            string s2 = ATES.DisplayBusTypeImage(PrmBusType);

            pbBusTypePic.ImageLocation = ConstPath + s2;
            if ((!string.IsNullOrEmpty(dm.ToPlace)) &&
(!string.IsNullOrEmpty(dm.BusType)))
            {
               // bus this.getBusResult();
                this.GetResultBus();

            }
            //this.getBusResult();
        }

        private void getBusResult()
        {
            dsBuses = dm.GetResult();

            gvBusList.DataSource = dsBuses.Tables[0];
        }
   localhost.ATESWebServiceManager ATES = new
ATSNew1.localhost.ATESWebServiceManager();
        private void GetResultBus()
        {
            string dt = DateTime.Now.ToString("HH.mm");
            string day1 = DateTime.Now.DayOfWeek.ToString();
            double dt1 = Convert.ToDouble(dt);

                   try
                   {
                        dsBuses  = ATES.GetResult("Bangalore", dm.ToPlace,
day1, dt1.ToString(),dm.BusType);
                        if (dsBuses.Tables.Count > 0)
                            gvBusList.DataSource = dsBuses.Tables[0];

                   }
                   catch (Exception ex)
                   {
                       MessageBox.Show(ex.ToString());
                   }


        }
        private void ChangeOptionCommand()
        {
            if (panelPlacelist.Visible == true)
            {
                if (PlaceSelected == true)
                {
                    PlaceSelected = false;
                 this.GuideCustomer("Choose Place From The List Bellow...");
                    dm.ToPlace = null;
                    lblChangeOption.Text = "Choose Option";
                    pbPlacePic.ImageLocation = "";
                    dm.BusSelected = null;
                    this.BusSelected = false;
                    BusTypeSelected = false;
                    //this.ChoosePlace(prmCommand);
                }

            }
            if (panelBusType.Visible == true)
            {

                if (BusTypeSelected == true)
                {
                    BusTypeSelected = false;
              this.GuideCustomer("Choose Bus Type From The List Bellow...");
                    lblChangeOption.Text = "Choose Option";
                    pbBusTypePic.ImageLocation = "";
                    dm.BusType = null;
                    dm.BusSelected = null;
                    // change BusTypeSelected = false;
                    this.BusSelected = false;

                    //this.ChooseBusType(prmCommand);
                }
                else
                {
                    this.GuideCustomer("You have Not Selected Option..");

                }
            }
            if (panelBusList.Visible == true)
            {
                if (BusSelected == true)
                {
                    BusSelected = false;
            this.GuideCustomer("Choose Bus Number From The List Bellow...");
                    lblChangeOption.Text = "Choose Option";

                    dm.BusSelected = null;
                    //change BusTypeSelected = false;

                    //this.ChooseBus(prmCommand);
                }
                else
                {
                    this.GuideCustomer("You have Not Selected Option..");

                }
            }

        }
        bool BusSelected = false;
        private void ChooseBus(string PrmBus)
        {
    this.GuideCustomer("You Have Choosen " + PrmBus + "To Continue Say Next");
            lblChangeOption.Text = PrmBus;

            BusSelected = true;
                        dm.BusSelected = PrmBus;
        }


        private void GetBusDetails()
        {



            DataRow dr = dm.GetBusDetails(dsBuses);
            if (dr != null)
            {
lblBusNo.Text = dr["BusNumber"].ToString();
lblBusType.Text = dr["BusType"].ToString();
lblFrom.Text = dr["FromPlace"].ToString();
lblTo.Text = dr["ToPlace"].ToString();
lblTime.Text = dr["Time"].ToString();
lblAvailableseats.Text = dr["AvailableSeats"].ToString();
lblTotalseats.Text = dr["TotalSeats"].ToString();
lblPlatformName.Text = dr["PlatformName"].ToString();
lblPlatformNo.Text = dr["PlatformNumber"].ToString();
pictureBoxBusType.ImageLocation = ConstPath + dr["BusTypeImage"].ToString();
pictureBox1.ImageLocation = ConstPath + dr["RootMap"].ToString();
pictureBoxPlatformImage.ImageLocation =ConstPath +
dr["PlatformImage"].ToString();
                lblVia.Text = dr["Via"].ToString();
                this.GuideCustomer("Avalibility Of Seats for The Bus Number,"
+ lblBusNo.Text +   "From ," + lblFrom.Text +  "To ,"  + lblTo.Text  + "Is "
+ lblAvailableseats.Text + "Thank You Happy Journey");
            }


        }
        private void ResetUI()
        {
            switch (pageSHow)
            {
                case 0:
                    panelFromTo.Visible = true;
                    panelPlacelist.Visible = false;
                    panelBusType.Visible = false;
                    panelBusList.Visible = false;
                    panelBusdetails.Visible = false;
                    break;
                case 1:
                    if (PlaceSelected == false)
                    {
                 this.GuideCustomer("Select Place From The List Bellow....");
                    }

                    panelFromTo.Visible = false;
                    panelPlacelist.Visible = true;
                    panelBusType.Visible = false;
                    panelBusList.Visible = false;
                    panelBusdetails.Visible = false;
                    break;
                case 2:
                    if (BusTypeSelected == false)
                    {
      this.GuideCustomer("Select Bus Type From The List Bellow....");
                    }
                    panelFromTo.Visible = false;
                    panelPlacelist.Visible = false;
                    panelBusType.Visible = true;
                    panelBusList.Visible = false;
                    panelBusdetails.Visible = false;
                    break;
                case 3:
                    if (BusSelected == false)
                    {
   this.GuideCustomer("Select Bus Number From The List Bellow....");
                    }
                    panelFromTo.Visible = false;
                    panelPlacelist.Visible = false;
                    panelBusType.Visible = false;
                    panelBusList.Visible = true;
                    panelBusdetails.Visible = false;
                    break;
                case 4:
                    panelFromTo.Visible = false;
                    panelPlacelist.Visible = false;
                    panelBusType.Visible = false;
                    panelBusList.Visible = false;
                    panelBusdetails.Visible = true;
                    break;
                default:
                    break;
            }
        }
        private void BackCommand()
        {
            if ((pageSHow < 5) && (pageSHow > 0))
            {
                pageSHow -= 1;
                ResetUI();
            }
        }
        private void NextCommand()
        {
            if (this.CheckConditions() == true)
            {
                 if (pageSHow < 4)
                {
                     pageSHow += 1;
                     ResetUI();
                }
                 else
                {
```

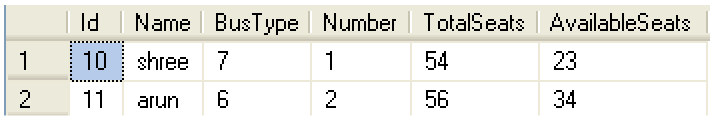
```
            //pageSHow += 1;
           }
       }
   }


   private bool CheckConditions()
   {
       bool Assigned = false;
       switch (pageSHow)
       {
           case 0 :
               Assigned = true;
               break;
           case 1:
               if (!string.IsNullOrEmpty(dm.ToPlace))
               {
                   Assigned = true;
               }
               else
               {
                   this.GuideCustomer("Select Place First....");
               }
               break;
           case 2:
               if (!string.IsNullOrEmpty(dm.BusType))
               {
                   Assigned = true;
               }
               else
               {
         this.GuideCustomer("Select Bus Type, Then Continue ....");
               }
               break;
           case 3:
               if (!string.IsNullOrEmpty(dm.BusSelected))
               {
                   Assigned = true;
               }
               else
               {
             this.GuideCustomer("Select Bus Number, Then Continue....");
               }
               break;
           case 4:
               Assigned = true;

               break;
         }
         return Assigned;
   }
   private void ExitCommand()
   {
       Application.Exit();
```

```
        }

        private void FromToCommand()
        {
            if (panelFromTo.Visible == true)
            {
                this.GuideCustomer("You Have Selected Form To Option.... To
Continue Say Next");

            }
        }
        private void VoiceCmd_CommandOther(object sender,
AxHSRLib._VcommandEvents_CommandOtherEvent e)
        {
            if (Process == true)
            {
                return;
            }
            RecogVoice(e.command.ToString());

        }

        private void panelBusdetails_VisibleChanged(object sender, EventArgs
e)
        {
            if (panelBusdetails.Visible == true)
            {
                if ((BusSelected == true) && (dsBuses != null) &&
(dsBuses.Tables[0].Rows.Count > 0))
                {
                   // VoiceCmd.Deactivate(pvMenu);
                    GetBusDetails();





                }
                else
                {
                    this.GuideCustomer("NO Result Found... Try Again");

                }
            }
        }
    }
}
```

DATA BASE

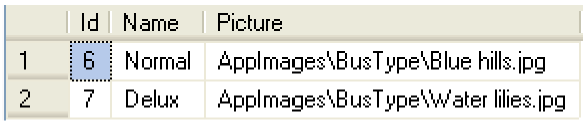
13. DATABASE

BUS



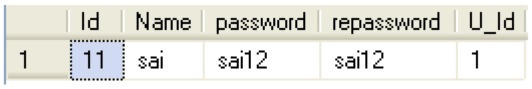
	Id	Name	BusType	Number	TotalSeats	AvailableSeats
1	10	shree	7	1	54	23
2	11	arun	6	2	56	34

BUS TYPE



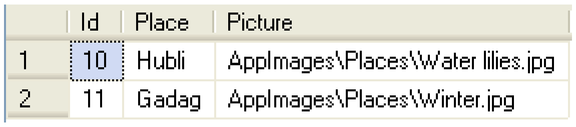
	Id	Name	Picture
1	6	Normal	AppImages\BusType\Blue hills.jpg
2	7	Delux	AppImages\BusType\Water lilies.jpg

CREATENEWUSER



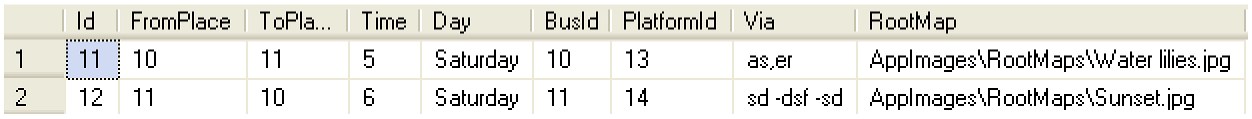
	Id	Name	password	repassword	U_Id
1	11	sai	sai12	sai12	1

PLACES



	Id	Place	Picture
1	10	Hubli	AppImages\Places\Water lilies.jpg
2	11	Gadag	AppImages\Places\Winter.jpg

TIMING



	Id	FromPlace	ToPla...	Time	Day	BusId	PlatformId	Via	RootMap
1	11	10	11	5	Saturday	10	13	as,er	AppImages\RootMaps\Water lilies.jpg
2	12	11	10	6	Saturday	11	14	sd -dsf -sd	AppImages\RootMaps\Sunset.jpg

Application

1) Bus Stand
2) Railway Station
3) Air Ports.

Extended Feature

1) Cost effective
2) Does not require any extra hardware component rather than micro phone(Mice).
3) This same technology can be used in other fields like hotels etc.

Advantages

1) Anybody can use with just voice knowledge .
2) 24 hrs useable with efficiency and better performance.
3) Easy interaction and can get as many information as needed.
4) Physically challenged people can also use unlike touch screen.

Disadvantages

1) Not work properly if disturbance is there.
2) Require prior voice training.

BABILLOGRAPHY

14. BABILLOGRAPHY

WEB SITE:-

http://www.w3schools.com/aspnet/aspnet_intro.asp
http://www.speech-topics-help.com/self-introduction-speech.html
http://www.ecma-international.org/activities/Languages/Introduction%20to%20Csharp.pdf
http://www.csharp-station.com/Tutorial.aspx

BOOKS
Murach's ASP.NET 2.0 Web Programming with VB 2005
Pro ASP.NET 2.0 in C# 2005