Bachelor Thesis

Anthony Kamau

Web based weather data analysis and presentation of a local weather station

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Anthony Kamau

Web based weather data analysis and presentation of a local weather station

Bachelor Thesis based on the examination and study regulations for
the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner : Prof. Dr.rer.nat Hans-Jürgen Hotop
Second examiner : Prof. Dr. Müller Wichards

Day of delivery June 7th 2007

Anthony Kamau

Title of the Bachelor Thesis

Web based weather data analysis and presentation of a local weather station

Keywords

Web server, Database, Weather parameter, Dynamic web page; HTML,PHP script, HTML form, HTML request , PHP Session ,Three tier architecture.

Abstract

In this report a web based application and presentation of weather data from a local weather station is described. The design using PHP and HTML technologies is extensively described. Also described is how the SQL query can be embedded in PHP to read data from the database.

The realization of various dynamic web pages is examined where the output presented to the web user will depend on user controlled input and data update in the database due to weather data variation.

The weather station is located at the roof of a building in Hamburg. The local weather station measures the weather data. This data is then processed and then stored in a database. Its from this database that the connection with PHP will be established. The data once available in PHP will be presented on the front web page using familiar instrument images to reflect the current weather conditions. Next the data will be processed, analyzed and presented to the user interactively. The real data and the calculated means and range for different time dependent parts will be presented graphically.

Anthony Kamau

Thema der Bachelorarbeit

Internetbasierte Wetterdatenanalyse und Darstellung einer lokalen Wetterstation .

Stichworte

Webserver, Datenbank, Wetterparameter, Dynamische Webseite; HTML, PHP Schrift, HTML-Form, HTML-Bitte, PHP Sitzung, Drei Schichten Architektur

Kurzzusammenfassung

In dieser Arbeit wird eine Internet basierte Darstellung von Wetterdaten einer lokalen Wetterstation vorgestellt. Die Anwendung wurde mittels PHP und HTML Technologien entwickelt, dabei werden die Daten aus einer SQL Datenbank gelesen und die Anbindung der Datenbank in PHP vorgestellt.

Verschiedene dynamische Webseiten wurden entwickelt um dem Internetnutzer die verschiedenen Wetterdaten darzustellen. Dabei hat der Nutzer die Möglichkeit zwischen verschiedenen Darstellungsmöglichkeiten zu wählen. Die Wetterstation ist auf einem Hochhausdach in Hamburg installiert. Sie mißt die lokalen Wetterdaten und stellt die bearbeiteten Wetterdaten in einer Datenbank zur Verfügung. Mittels eines PHP Programms werden die Daten aus der Datenbank gelesen und zunächst die aktu-

elle Wettersituation auf der Eingangsseite der Internet Präsentation in Form von bekannten Messinstrumenten dargestellt. Zusätzlich werden die Daten bearbeitet und analysiert, um eine interaktive Präsentation für den Nutzer zur Verfügung zu stellen. Die realen Daten und der jeweilige Mittelwert sowie die Standardabweichung für unterschiedliche zeitliche Ausschnitte werden grafisch dargestellt.

Contents

List of Figures

List of tables

1. Introduction

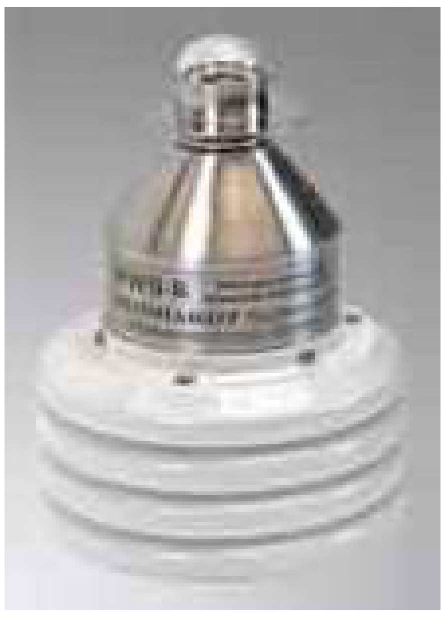
Weather data is of great importance to people from all walks of life. It is particularly very useful to the professionals working in the weather forecasting field, Airline industry, farmers, disaster response teams just to name a few. Past weather trends are particularly very useful since they serve as the basis for planning and prediction. It's therefore very important for the weather data to be recorded accurately, stored safely and presented to an interested group in a manner that's easy to interpret. Components of a modern weather forecasting system include: Data collection, Data assimilation, Numerical weather prediction, Model output post-processing and forecast presentation to end-user. Collection and accurate past weather data is very essential for accurate weather forecasting. If the data is not accurately recorded then this leads to wrong weather prediction. It's also important to state that weather forecasters require real time data or data that's near real time. This is well described in this thesis where the latest weather data presentation is described. Data that's as old as one minute can be presented to the weather forecasters, as it will be seen from this thesis.

This thesis seeks to discuss how past and near present weather data that's accurately recorded and stored in a database is presented on a website in an interactive way to the user. In this case a user could be any of the above-mentioned professionals or anybody who would like to know the past weather trends in a graphical and tabular format.

The weather station is based on the roof of the building Berliner Tor 5 at the Hamburg University of applied sciences. The weather station captures and records six weather parameters since March 30, 2006, 5:07 pm within time intervals of one second and a 60 seconds cycle mean is computed to give a clearer trend of the data. The 60 seconds cycle mean data is less noisy than the one second data. The 60 seconds mean for each parameter is then stored in a Sybase database.

This thesis describes a web application development that seeks to fetch this data from the database, analyzes and presents it in a friendly manner that's easy to interpret is discussed and outlined. In particular this data is presented in form of graphs or both graphs and tables.

2 Weather Station



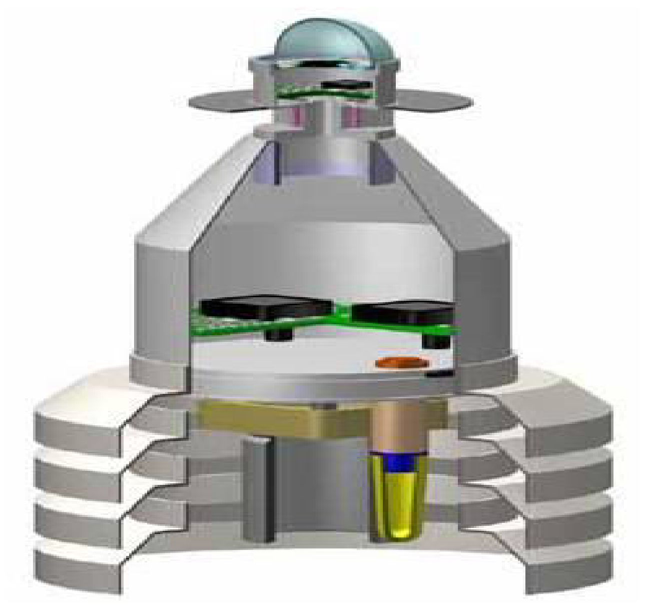


Figure 1: MWS 6 Weather station **Figure 2: Internal view of the weather station**

2.1 Description of the weather station MWS 6

The weather station is from the firm Reinhardt – Testsysteme. Their external and internal views are shown in figure 1 and figure 2 above. It measures six different weather parameters in SI units as shown in the table 1 below

Table 1: Weather parameters and the general measurement instruments

Parameter	Unit	Instrument	Abbreviation
Temperature	°Celsius	Thermometer	TE
Atmospheric Pressure	hectoPascal	Barometer	DR
Wind Direction	°	Electronic sensor	WR
Solar Radiation	Watt/m^2	Pyranometer	SO
Humidity	%	Hygrometer	FE
Wind Speed	km/h	Electronic sensor	WG

The weather station also calculates four more weather parameters that are very useful in weather data analysis. The calculated parameters and the units are described on table 2 below

Table 2: Weather parameters calculated by the weather station

Parameter	Unit	Instrument	Abbreviation
Wind-chill Temperature	°Celsius	Calculated	WC
Wind Maximum Peak speed	km/h	Calculated	WS
Prevalent Wind Direction	°	Calculated	WV
Wind Average Peak speed	km/h	Calculated	WD

2.2 Description of the weather parameters measured

Temperature is a physical property of a system that describes how hot or cold something that is. The temperature of a system is defined as simply the average energy of microscopic motions of a single particle in the system. Temperature is measured using a thermometer which is calibrated using various units. The scientific and SI unit for temperature is Celsius.

Atmospheric pressure is described as the pressure at any point on the earth's atmosphere. It's approximated by the pressure caused by the air above the measurement point. Low pressure areas have less atmospheric mass above their location, whereas high pressure areas have more atmospheric mass above their location. The SI unit for measuring the atmospheric pressure is Pascal. In the weather station it's measured in hectoPascals.

Wind direction is the direction from which the wind is blowing. It is usually reported in cardinal directions or in degrees. Traditionally wind direction is measured using wind sock or wind vane. Today wind direction is measured using Electronic Anemometers which can accurately measure the wind direction. The SI unit for the wind direction is ° (degrees).The degrees measured are interpreted to cardinal direction as shown in figure 3 and table 2 below:

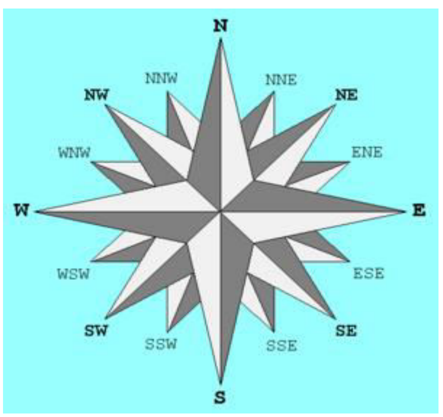


Figure 3: Compass showing cardinal wind direction

Table 3: Degree to wind direction conversion

Degrees	Cardinal Direction
0°	North
90°	East
180°	South
270°	West

Solar radiation is radiant energy emitted by the sun, particularly electromagnetic energy. It's measured in Watt/m^2. In general solar radiation is measured by a Pyranometer.

Humidity is the amount of water vapor in the air. It is measured in three ways: absolute humidity, relative humidity, and specific humidity. Relative humidity is the most frequently encountered measurement of humidity because it is regularly used in weather forecasts. It is an important part of weather forecasts because it indicates the likelihood of precipitation, dew, or fog. Relative humidity is, a ratio of how much energy has been used to free water from liquid to vapor form to how much energy is left. Relative humidity is expressed as a percentage. Humidity is commonly measured using a Hygrometer.

Wind speed is the speed of movement of air relative to a fixed point on the Earth. Wind speed is commonly measured using an Anemometer. In the MWS 6 wind speed and wind direction are measured by an electronic sensor shown in figure 4 below

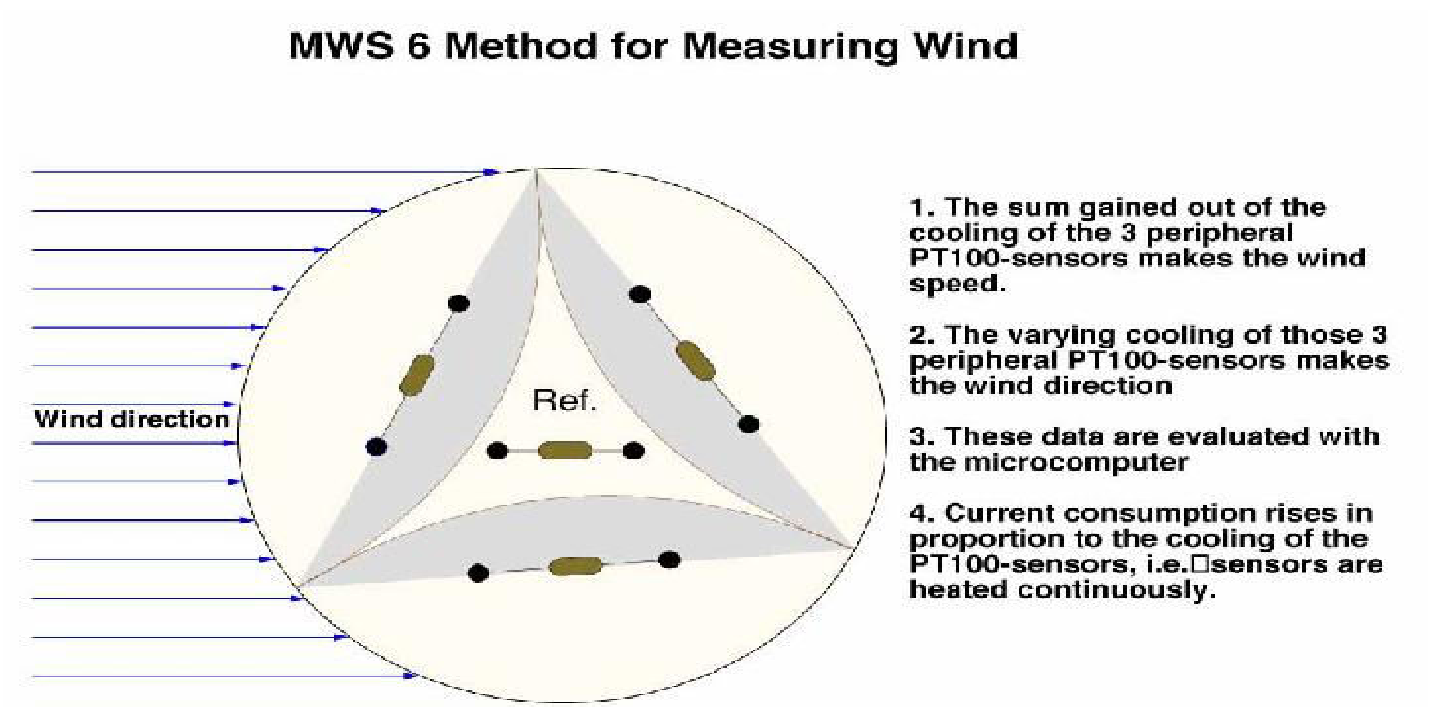

MWS 6 Method for Measuring Wind
Wind direction
Ref.
1. The sum gained out of the cooling of the 3 peripheral PT100-sensors makes the wind speed.
2. The varying cooling of those 3 peripheral PT100-sensors makes the wind direction
3. These data are evaluated with the microcomputer
4. Current consumption rises in proportion to the cooling of the PT100-sensors, i.e.□sensors are heated continuously.

Figure 4: Wind measurement principle

Wind chill temperature or simply wind chill factor measures the effect of the combination of temperature and wind speed on human comfort. Both temperature and wind speed do not have the same effect on inanimate objects, or even on other animals or on plants. Nor do humans who are sheltered from the wind feel this effect. Wind chill factors can be expressed as an equivalent temperature on either the Celsius or Fahrenheit scale. In the MWS 6 weather station it's expressed in Celsius.

Wind chill factor is published in tables. There are many such tables that publish the wind chill factors and most of them do not agree on all the equivalents. There are some discussions going on in an attempt to agree on a common table. Presence of various tables has led to various formulas used to calculate the wind chill factor. The formula the National Weather Service uses to compute wind chill factor is:

$T(wc) = 0.045(5.27V*0.5 + 10.45 - 0.28V)\ (T - 33) + 33$
Where V is the speed of wind in km/h and T is the temperature in ° Celsius.

Its worth noting that the above wind chill calculation formula is not valid at all for wind speeds greater than 90 km/h.

Maximum Peak wind speed is the maximum instantaneous wind speed measured since the last routine observation. It's primarily determined with the help of speed recorders. **Average Peak wind Speed** is the mean of all the maximum peak wind speeds observed over given fixed intervals for a given time.

Prevailing wind direction is the direction from which wind blows most frequently across a certain region within a given time period. Different regions on Earth have different prevailing wind directions, which are dependent upon the nature of the general circulation of the atmosphere and the latitudinal wind zones. The periods most frequently used are the observational day, month, season, and year. Methods for determination vary from a simple count of periodic observations to the computation of a wind rose. Compare resultant wind.

2.3 Description of the weather database

Database represents the internal layer in the three-tier[1] architecture. It's composed of the database management system that manages the database containing the data that users create, delete, modify or query.

In this case the weather data is managed using a Sybase DBMS. It's stored in a relational database called **Weather**. In a relational database, all data are held in **tables**, which are made up of **rows** and **columns**. Each table has one or more columns, and each column is assigned a specific data type, such as an integer, a sequence of characters (for text), or a date. Each row in the table has a single value for each column.

A database can have one to many tables as shown in figure 5.

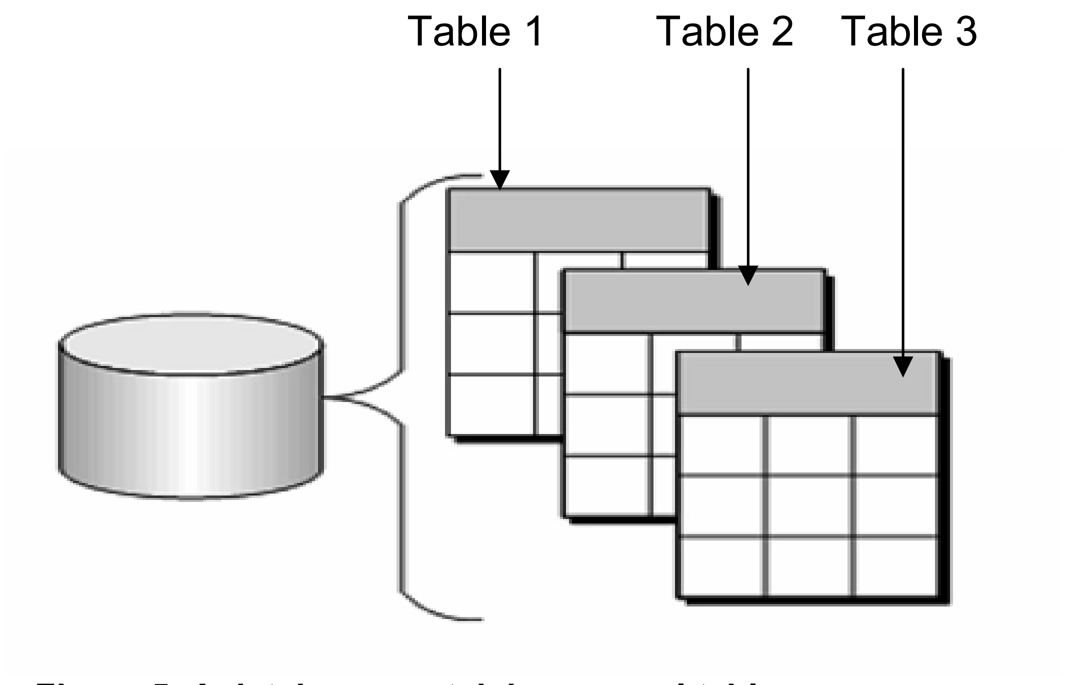


Figure 5: A database containing several tables

Characteristics of relational database table

The table in a relational database has some important characteristics. In a relational database table there is no significance to the order of the columns or rows. Each row contains one and only one value for each column, or contains NULL, which indicates that there is no value for that column. All values for a given column have the same data types. Each row can be identified uniquely with the help of the **primary key**. A primary key is a column or the minimum combination of columns that uniquely identify a given row.

A primary key has some special characteristics that are important for the definition of tables. A primary key should be stable. That means that it should not change over time. It should be minimal. That means it should have the minimum attributes possible. It should also be fact less. That means it should not have any hidden information except uniquely identifying the rows. It should be definitive. That means it should always exist and have a value. It should not be null. It should be accessible. That means it should be available when the data is created. Finally a primary key should be unique. That means it should have no duplicate values.

The Weather database has one table named **dba.tblRawdata**. This table has 11 fields that are defined that represent the 11 data types of the 10 weather parameters and 1 for the time when the data was recorded as shown in the table 3. A common characteristic with all the defined field types is that each field name begins with the letters fld… to show that it's a field name for easier identification. In this case the primary key is fldRecordedAt. It serves only the purpose of keeping the record of the time when the data was recorded. In recording the time the date format used is:

Year-Month-Day Hour: Minute: Second. Millisecond

Table 4: dba.tblRawData table data definition

Column Name	Data Type
fldRecordedAt	Timestamp
fldTemperature	Decimal(5,2)
fldsolarRadiation	Decimal(7,2)
fldWindSpeed	Decimal(5,2)
fldWindDirection	Decimal(6,2)
fldBarometer	Decimal(7,2)
fldHygrometer	Decimal(6,2)
fldWindChillFactor	Decimal(5,2)
fldWindMaximumPeak	Decimal(5,2)
fldWindAveragePeak	Decimal(5,2)
fldPrevWindDirection	Decimal(6,2)

The original weather measurement system sends data each second to the data processing chip. Most of this data like the wind direction and wind speed are very noisy. That means the data varies in magnitude so much that it's would be impossible to make sense out of it. Once the data is measured by the weather station MWS 6 the mean of all the data over 60 cycles is calculated and then stored in the database described above. By calculating the mean for every 60 seconds cycle, the noise is reduced dramatically making the recorded data meaningful. When the 60 seconds mean is evaluated, the mean computation completion time is recorded in the field fldRecordedAt. By 23rd March 2007 about half million rows of weather data have been recorded.

3 Used Software

As it has been discussed in the last chapter where the data is accurately recorded into the database, the tools that are used for presenting the data to the user are described. Several tools were used in order to present the data on the web in a manner that's easy to understand and interpret. These are the Database Management System, the Web Server, Server Side Scripting language and other supporting modules.

3.1 Database Management System

A database management system (DBMS) is computer software designed for the purpose of managing databases. Typical examples of DBMSs include Oracle, DB2, Microsoft Access, Microsoft SQL Server, Sybase and SQLAnywhere. It has been mentioned in the last chapter that the weather data is stored in a SQLAnywhere database. In order to be able to read the data from the database it was necessary to install SQLAnywhere on the local machine. By so doing it becomes possible to query the database from the local machine. This was very important since it enables one to use the server side script supporting modules supporting modules to connect to the database. Furthermore installing it on the local machine provided a means to compare the query results from a script and the results obtained from querying the database directly using the SQLAnywhere user interface. The query results from the two procedures are supposed to give the same results.

SQLAnywhere

SQL Anywhere is a Relational Database management System product from **iAnywhere** Solutions. iAnywhere is a subsidiary of **Sybase**. SQL Anywhere is a comprehensive package that provides technologies for data management and enterprise data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.

SQLAnywhere has some important features that make it very important in terms of data management. One of the most important features of SQLAnywhere is that its database files are operating-system-independent. This means that they can be copied between supported platforms. SQLAnywhere can be run on Windows, Windows CE, Novell NetWare, and various UNIX platforms, including Linux. It has several standard interfaces (ODBC, JDBC, ADO.NET) and some special interfaces (e.g. PHP). These interfaces are very important since they provide a means of interacting with the database. It supports powerful encryption of both database files and client-server communication. This makes it very useful because of its security. Version 10 supports materialized views, database mirroring, server clustering, and **snapshot isolation**. Snapshot isolation is a guarantee that all reads made in a transaction will see a consistent snapshot of the database, and the transaction itself will successfully commit only if no updates it has made conflict with any concurrent updates made since that snapshot.

SQL Anywhere is widely used as an embedded database, as an application data store; for example, it's used in network management products, backup products, and others. Its ability to be used with minimal administration is a distinguishing feature in this role.

SQL Anywhere is also widely used for mobile computing. It includes scalable data synchronization technology that provides change-based replication across many **mobile databases**. A Mobile database is a database than can be connected to by a mobile computing device over a mobile network. The client and server have wireless connections. A cache is maintained to hold frequent data and transactions so that they are not lost due to connection failure. Sybase has a 60% market share in mobile databases followed by Oracle (20%) and IBM (15%). There are over 10 million deployed seats of SQL

3.2 Web Server

The most important tool that's required is a web server. A web server just like the name implies is a computer or a computer program that is responsible for accepting HTTP requests1 from clients, which are known as Web browsers, and serving them HTTP responses along with optional data contents. The contents usually are Web pages such as HTML documents and linked objects. Usually Web servers have also the capability of logging some detailed information, about client requests and server responses, to log files; this allows the Webmaster to collect statistics by running log analyzers on log files. A log file is a file that's automatically created in the server that maintains a history of page requests. Some information about the request, such as client IP address, request date/time, page requested, HTTP code, user agent, and referring file are typically added.

Web servers are definitely essential in optional authorization request (request of user name and password) before allowing access to some or all kind of resources. Handling of both static and dynamic web contents is another purpose for the web server. Static web content means content which always comprises the same information in response to all download requests from all users. Usually it comes from an existing file. Dynamic contents are web contents that can change in response to different conditions or contexts such as user input. Dynamic content is dynamically generated by some other program or script or API called by the Web server. In order for the web server to handle dynamic content then it needs to support a related interface such as SSI (Server Side Includes), CGI (Common Gateway Interface), SCGI (Simple Common Gateway Interface), JSP (Java Server Pages), PHP (Hypertext Preprocessor), ASP (Active Server Pages). Web servers also help to compress the content to reduce the size of response and thus lowering the bandwidth. They also serve to limit the speed of responses in order to not saturate the network and to be able to serve more clients.

One of the most common HTTP serving programs is the Apache HTTP server from the Apache Software Foundation. Others are Internet Information services (IIS) from Microsoft and Sun Java System Web Server from Sun Microsystems just to name a few. In the web based weather data analysis and presentation project Apache HTTP server was used.

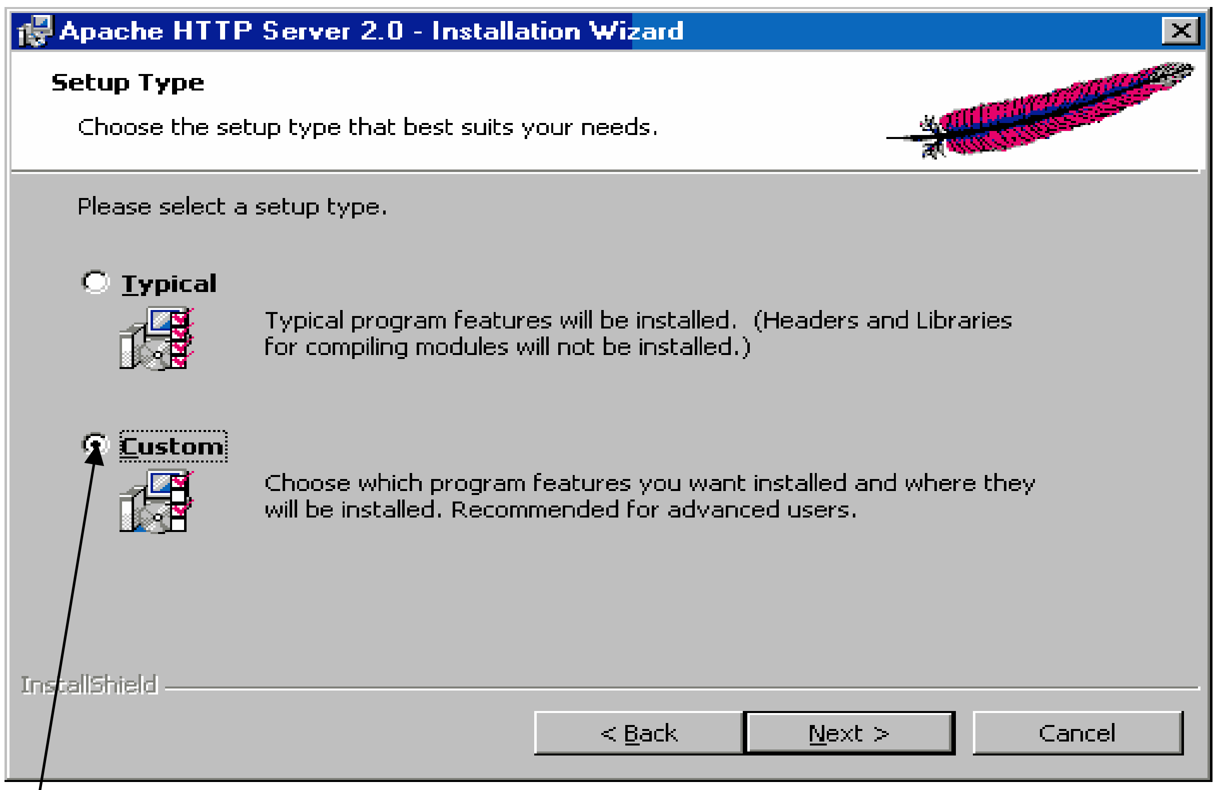
Apache HTTP Server is commonly referred to as Apache, is a web server that has strongly contributed to the growth of the World Wide Web. Apache was the first alternative to the Sun Java System Web Server. Since April 1996 Apache has been the most popular HTTP server on the World Wide Web; as of March 2007 Apache served 58% of all websites. Apache is developed and maintained by an open community of developers under the name of the Apache Software Foundation. The application is available for a wide variety of operating systems including Microsoft Windows, Novell NetWare and Unix-like operating systems such as Linux and Mac OS X. Apache is free and open source software.

Installation of Apache

Before Apache was downloaded a directory structure was set up and a folder where it's going to reside was created on the local computer. Apache can be downloaded from *http://httpd.apache.org/download.cgi#apache20*

Apache version 2.0.59 Win32 Binary was downloaded and saved into the directory created above.

In order to install the installer file stored in the created folder above is double clicked. After reading the Information and the agreements and clicking next to move to the next screen. The network domain name, Server name and the Administrator's email were added and "next" was clicked. On clicking next the next screen was as follows .The customer installation was selected as shown in figure 6 below:



Apache HTTP Server 2.0 - Installation Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

( ) Typical

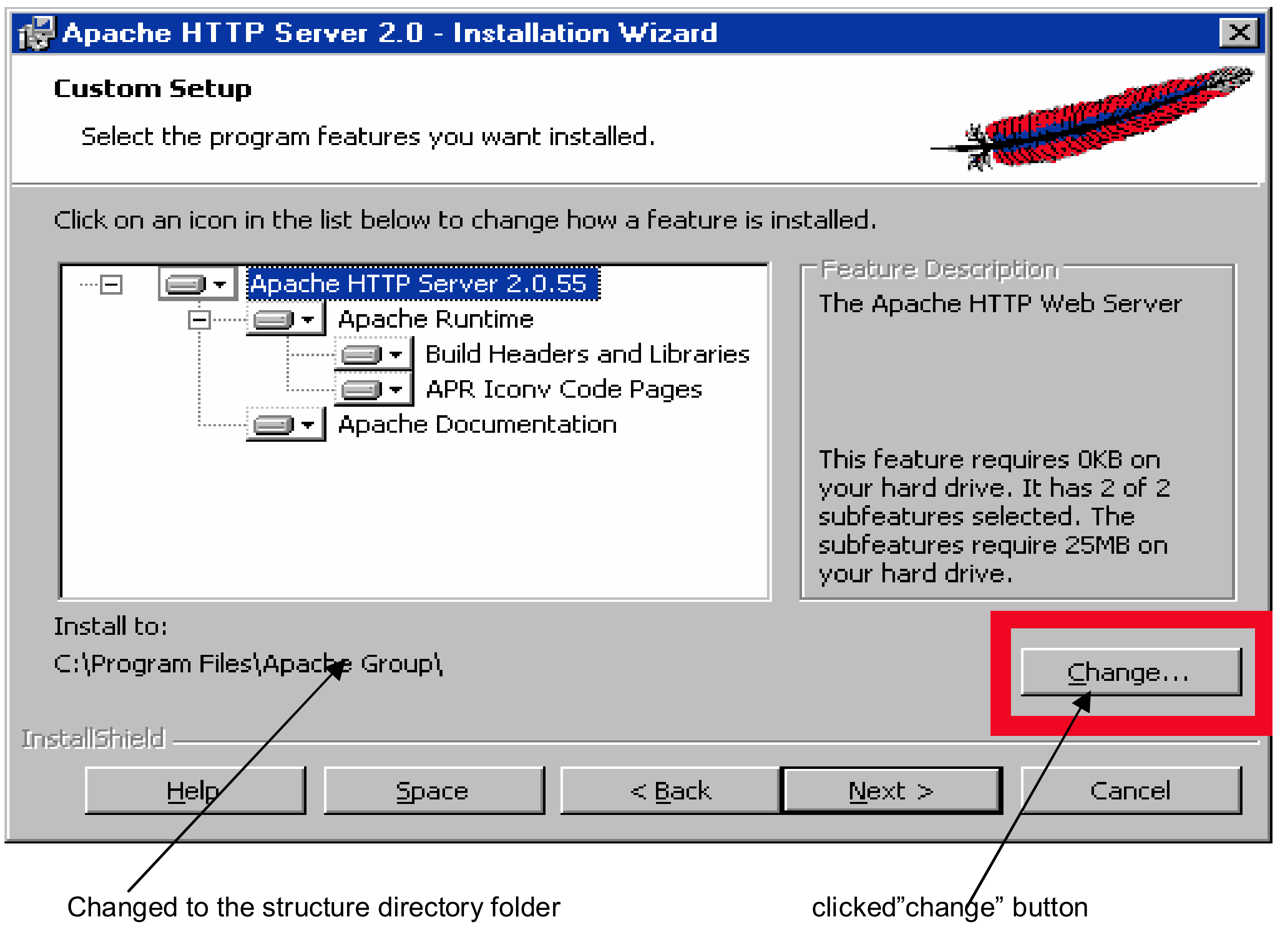
Typical program features will be installed. (Headers and Libraries for compiling modules will not be installed.)

(•) Custom

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

Figure 6: Apache installation wizard

The correct directory where the Apache is to be installed was changed to the folder with the directory structure created above. This was done by first clicking"Change" button as it can be seen from the figure 7 below:



Apache HTTP Server 2.0 - Installation Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

- Apache HTTP Server 2.0.55
  - Apache Runtime
    - Build Headers and Libraries
    - APR Iconv Code Pages
  - Apache Documentation

Feature Description

The Apache HTTP Web Server

This feature requires 0KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 25MB on your hard drive.

Install to:
C:\Program Files\Apache Group\

Figure 7: Apache installation step one

On the next screen”Install” was selected and after installation was complete then”finish was selected”.
In order to start the server from the start menu to the”All programs” menu and the following selection in figure 8 was made.

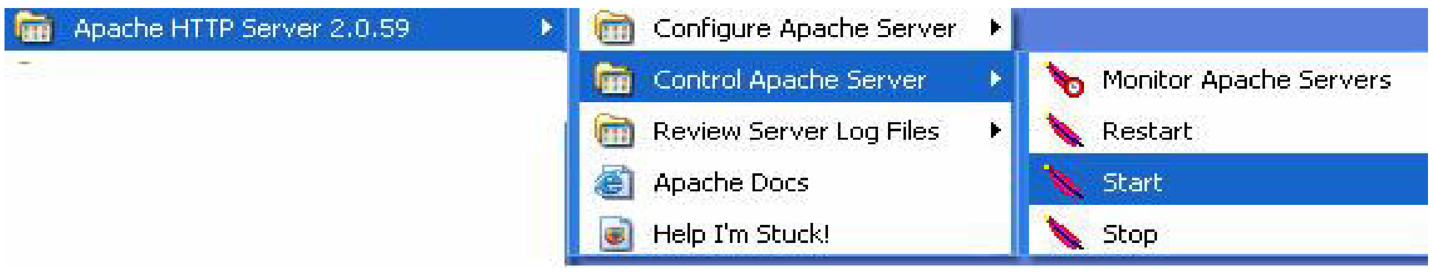


Figure 8: Apache installation step two

Apache configuration file is located in the folder Apache2/config and is called *httpd.conf*. This file was opened with a text editor and the most important configuration performed to the Apache is to set the directory of PHP.

3.3 Server Side Scripting

Server-side scripting is a web server technology in which a user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. HTML, short for *Hypertext Markup Language*, is the predominant markup language for the creation of web pages. It provides a means to describe the structure of text-based information in a document

Server side scripting is different from the client side scripting because the scripts are run in the web server where as a client side script is run by the viewing web browser. Client side scripting and server side scripting will be closely compared in the next chapter in more details.

Server-side scripting is preferred over client side scripting because of its ability to highly customize the response based on the user's requirements, access rights, or queries into data stores. In this case it’s highly effective in the generation of the dynamic web content described in the last subchapter.

ASP and ASP.NET from Microsoft are some of the server side scripting technologies used. ASP abbreviates Active Server Pages. ESP or Escapade enjoys wide use in Europe but has limited acceptance in US.

JSP, which abbreviates Java Server Pages and PHP abbreviating Hypertext Preprocessor are the two most commonly used server side scripting technologies. Others in use in the market are CodFusion, Lasso, Server side JavaScript, SMX etc. PHP was the technology used in the weather project. The reasons for its preference have been discussed in a later subchapter.

PHP

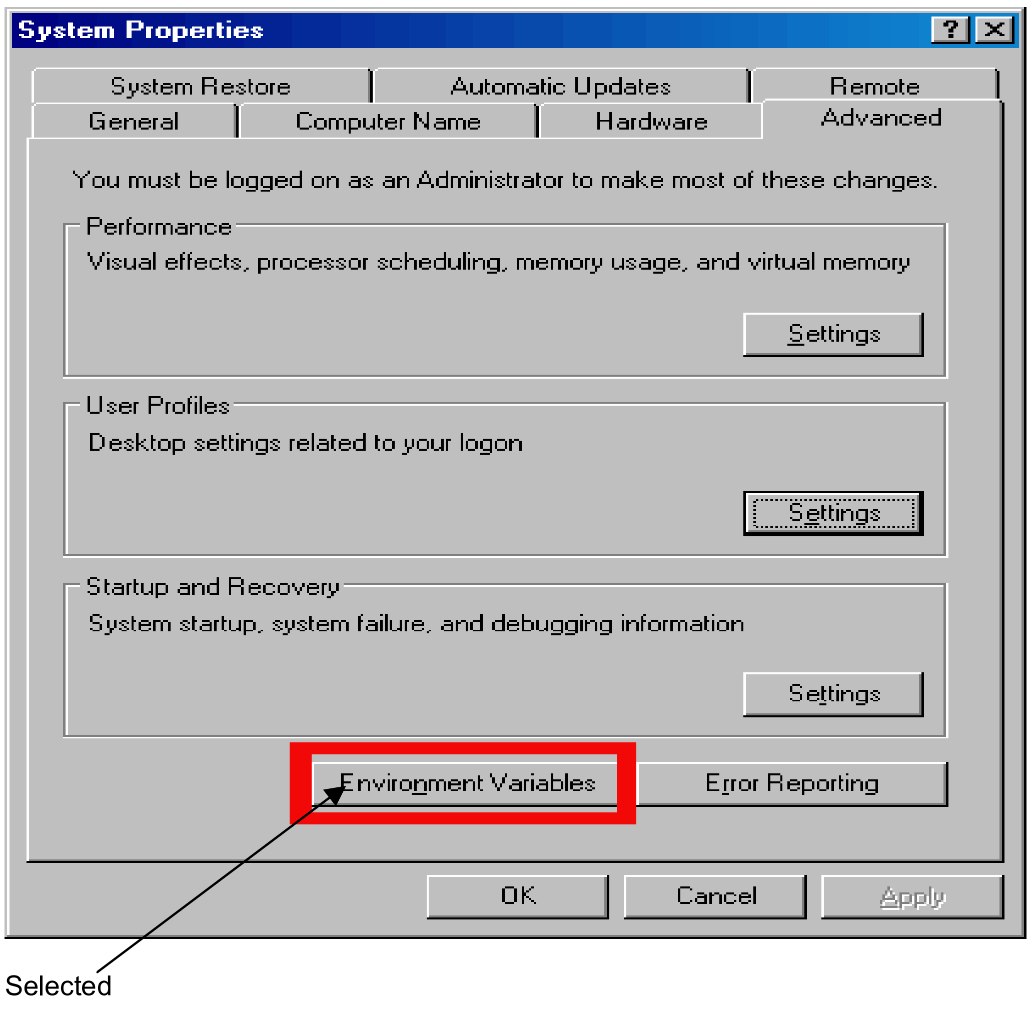
PHP is an abbreviation for Hypertext Preprocessor. It's a reflexive programming language designed for producing dynamic web pages. PHP was originally designed for producing dynamic web pages. It's mainly used for server side scripting but can also be used for command line interface (CLI). A CLI is used whenever a large vocabulary of commands or queries, coupled with a wide range of options, can be entered more rapidly as text, than with a pure GUI. PHP is a free software and it's produced by the"The PHP group" and released under PHP licence. The PHP group is a worldwide group of PHP developers who are continuously developing source code for various applications. The PHP group then avails the code on the internet for anyone to download and use in their application and also share their development with the others on the web.

PHP runs on a web server taking PHP code as an input and creating web pages as an output. PHP can be deployed on most web servers and on almost every OS platform free of charge. The PHP Group also provides the complete source code for users to build, customize and extend for their own use. This way users of PHP are able to spend time developing new source code instead of reinventing the wheel (creating what has already been created before).

Installation of PHP

PHP can be downloaded from http://www.php.net/downloads.php download page. For the purpose of realizing the weather data online presentation project PHP version 5.1.4 zip package under the "windows binary" was downloaded. This meant that manual installation was preferred than using the installer. The reason for opting for manual installation was because it's easier to configure it manually than let the installer do the default configuration. The Zip file was then placed in the folder that was created in the above directory structure where the Apache web server was placed. The file was then unzipped and the original zip file was deleted.

In order for PHP to work with Apache, the PHP folder was added to the *PATH* environment variable. In order to do this the icon "My Computer" was clicked then "Properties". In the window that popped up "Environment variables" button was selected as it can be seen in figure 9 below:



You must be logged on as an Administrator to make most of these changes.

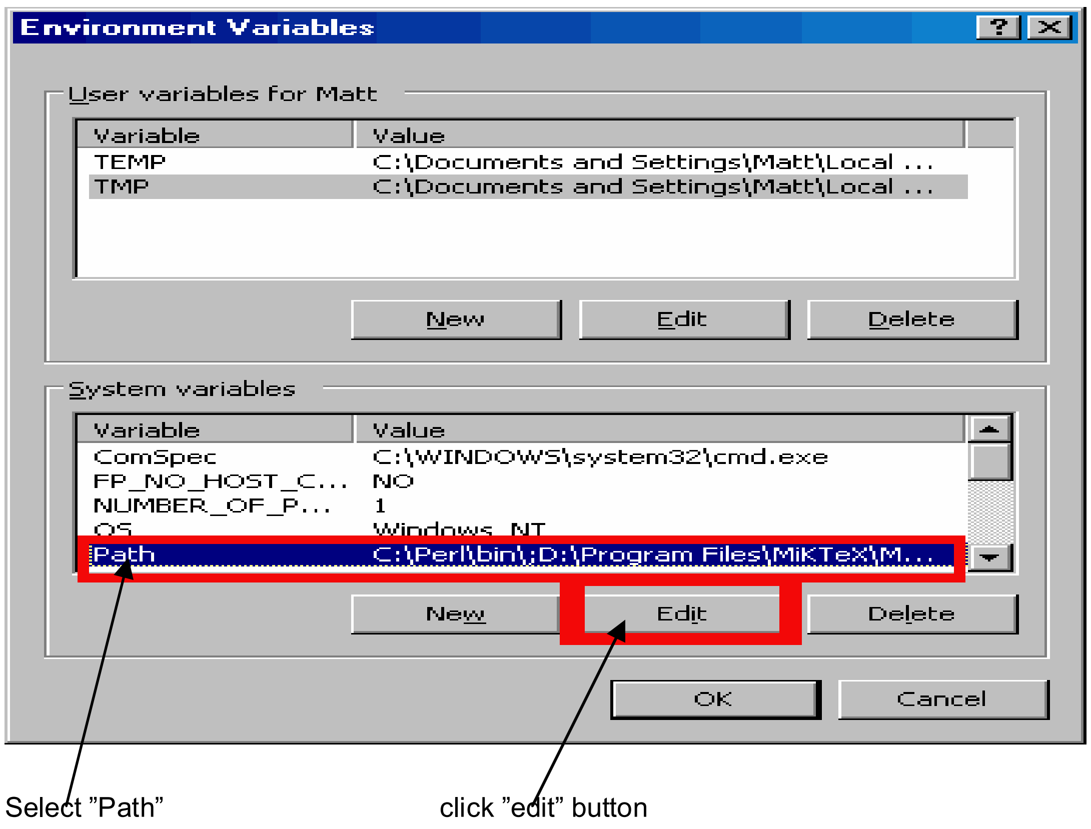
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

Figure 9: Setting PHP path variable

In the window that appeared the "Path" variable line from the "System Variables" menu was selected and ”edit” button was clicked as it can be seen in the figure 10 below:



User variables for Matt

Variable	Value
TEMP	C:\Documents and Settings\Matt\Local ...
TMP	C:\Documents and Settings\Matt\Local ...

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\Perl\bin\;D:\Program Files\MiKTeX\M...

Figure 10: Editing PHP path variable

At the end of the "Variable Value" field, a semicolon (;) was typed and the full path to the PHP folder was then typed as shown in figure 11 below :

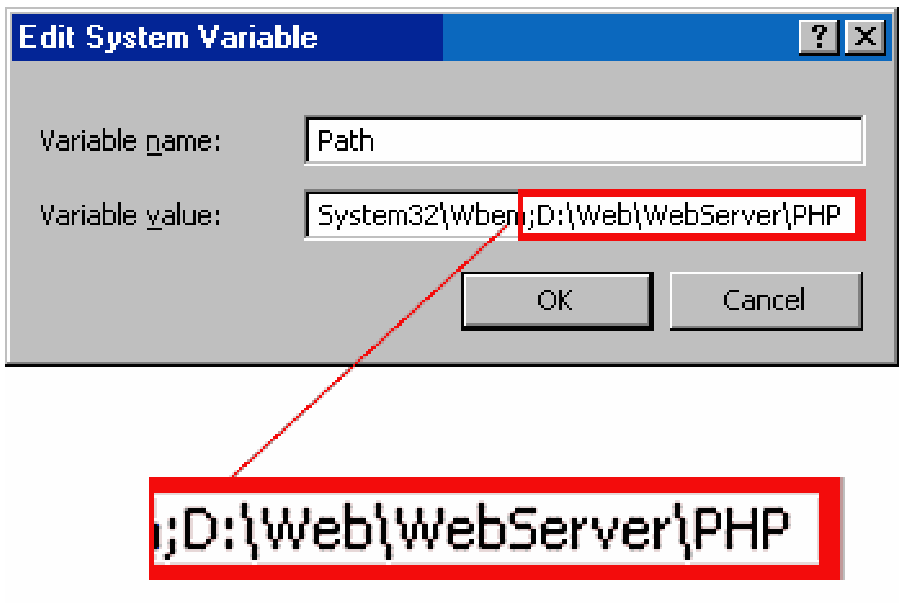


Figure 11: Adding full PHP path variable

Then”ok” was selected on each window until they were all closed. It was then tested in PHP path was correctly set by selecting ”Run” in the start menu ”cmd” was typed then ”OK” pressed to bring up a command window. In the command window the command”php –v” was typed then”enter” was pressed.

The output shown below indicated that PHP path was set correctly.

```
PHP 5.1.4 (cli) (built: May 4 2006 10:35:22)
Copyright (c) 1997-2006 The PHP Group
Zend Engine v2.1.0, Copyright (c) 1998-2006 Zend Technologies
```

Configuration of PHP was performed in a file called *php.ini-recommended* located in the PHP folder. At first the name of this file was changed to *php.ini*. This file was then opened with a text editor, in this case the Note Pad. The main purpose of configuring PHP is to enable its functionalities to work with the web server and to enable it to use the PHP **extensions**. Extensions are reusable PHP components. Some are downloaded together with PHP installation files and others can be downloaded separately.

It’s important to note that only the standard installation of PHP has been discussed. Two more PHP libraries were downloaded and added to the present installation like **JPGraph** and **SQL Anywhere PHP Module**. JpGraph is an Object-Oriented Graph creating library for PHP newer than version 4.3.1. The library is completely written in PHP and ready to be used in any PHP scripts. All the functions that were to create the weather data graph are defined in this library. SQL Anywhere PHP Module is a

PHP library that can be used to retrieve data from an Adaptive Server Anywhere database. The Weather database is managed by an Adaptive Server Anywhere database. So all the functions that were used to establish connection to the database and to query the database are actually defined in this library.

SQL Anywhere PHP Module

It has been stated in the introductory part of this chapter that by installing SQLAnywhere on the local machine, one is able to use supporting modules to connect to the database. PHP provides the ability to retrieve information from many popular databases, such as SQL Anywhere. Included with SQL Anywhere is module that provides access to SQL Anywhere databases from PHP. This module and the PHP language can be used to write stand-alone scripts and create dynamic web pages that rely on information stored in SQL Anywhere databases. This module is referred to as SQLAnywhere PHP Module. This module can be downloaded from *http://www.php.net/* .

In order to be able to use this module it was required that SQLAnywhere client software be installed on the same machine as the PHP and the web server. The dynamic link library *dblib10.dll*, which provides the functionality of PHP to access the database, is present in SQL Anywhere client software. To install the SQL Anywhere PHP module on Windows, the DLL from the SQL Anywhere installation directory is copied to the PHP installation. Optionally, an entry to the PHP initialization file is added so as to load the module automatically, so there is no need to load it manually in each script.

JPGRAPH

JPGraph is an Object Oriented graph library, which can be used to draw graphs. The real advantage in using JPGraph is its simplicity. Creating a graph at runtime may sound complex. But JPGraph class has been designed in such a way that they hide the complex part from the user. Few lines of coding can create amazing graphs, which can never be thought of. The first step would be to download the JPGraph include files. This can be found at (http://www.aditus.nu/jpgraph/). Extract all the contents to a folder and put the folder in the same root directory as PHP. These libraries can be referred to whenever needed.

First step would be to ensure that the **GD library** is enabled. GD is an open source code library for the dynamic creation of images by programmers. GD is written in C, and "wrappers" are available for Perl, PHP and other languages. GD creates PNG, JPEG and GIF images, among other formats. GD is commonly used to generate charts, graphics, thumbnails, and most anything else, on the fly. While not restricted to use on the web, the most common applications of GD involve web site development. JPGraph uses the GD library features to draw and create images. One can use the phpinfo() to ensure that the GD library is enabled. By calling this function all the settings and modules present together with their configurations are outputted.

JPGRAPH has very many features such as flexible scales, supports text-lin, text-log, lin-lin, lin-log, log-lin and log-log and integer scales. It supports PNG, GIF and JPG graphic formats. Supports caching of generated graphs to lessen burden of a HTTP server. In the particular project the caching feature was disabled due to the limited storage. It has intelligent auto scaling which uses common scaling values, i.e. multiples of 2:s and 5:s. It supports color and brightness adjustments of images directly in PHP. More so it supports, line-plots, filled line-plots, accumulated line-plots, bar plots, accumulated bar plots, grouped bar plots, error plots, line error plots, scatter plots, radar plots, 2D and 3D pie charts. Other features of JPGRAPH include the ability to support an unlimited number of plots in each graph, makes it easy to compose complex graph which consists of several plot types, automatic legend generation, rotation of linear graphs and more than 400 named colors.

3.4 Other possibilities of realizing web based data presentation

There are 2 types of web pages .These are *dynamic web pages* and *static web pages.* Dynamic web pages are web pages whose contents can change in response to different conditions. This kind of interactivity can be realized in using client side scripting and Server side scripting.

Client side scripting

Using client side scripting is where one can change the behaviour of a specific web page by keyboard or mouse event or specific timing e.g. auto refresh after a given time duration. This type dynamic web page can be realized using a client side scripting languages like JavaScript or Action script which is used for Dynamic HTML, found in animations, sound and changing text. One of the ways where interactivity could be realized in the web based weather data presentation would have been to store the data of limited time duration in a file from where it’s retrieved by the web server and sent to the user’s computer. The user's web browser executes the script, and then displays the document, including any visible output from the script. A URL is a short text that identifies a web resource and provides a means of locating the resource by describing its primary access mechanism (e.g., its network ‘location’.

Using Server Side Scripting to realize a Dynamic Web Page

Using server side scripting to change the supplied page source between pages, adjusting the sequence or reload of the web pages or web content supplied to the browser. Data posted on a HTML form may cause the server to respond in a specific way. Server response can also be influenced by parameters in the Uniform Resource Locator (URL), the type of browser being used, the passage of time, or a database or server configuration state.

The server side scripting works in a manner that’s more complicated than the client side scripting to generate a dynamic page. The script or the program to be executed receives input from a query string or a standard user input from a submitted web form. The browser sends a HTTP request to the web server. The web server retrieves the requested script which could as well be a program. The server executes the script which results in a HTML web page.

In the web based weather data presentation server side scripting could be implemented by executing scripts using a web server which could typically be Apache web server, Tomcat just to name a few. The user selects the preferred dates and time to view the weather history on a HTML form. On submitting the form the script is executed by the web server and the output is sent back to the user user’s browser in form of a HTML output. The user cannot view the source code unless the author publishes it separately. Documents produced by the client side scripting may also contain the client side script.

Comparison of client side scripting and server side scripting

Client-side scripts have greater access to the information and functions available on the user's computer. The primary advantage of client side scripting is that they do not require installation of addition software in the server. This makes them popular with lots of authors who lack administrative rights on the server. One major disadvantage of using this kind of dynamic web pages in weather data presentation project is that the user would be limited to the choices of the weather data stored in the file. Therefore there is lack of complete interactivity and flexibility. The other disadvantage of client side scripting is that some browsers do not support certain client side scripting languages. So the behaviour of a web page developed using a given client side scripting language can behave differently from one browser to another.

In contrast to the client side scripts, the server side scripts when executed produce output that’s understandable to the browser. Usually this is HTML. Therefore they produce the same output regardless of the browser or the system settings of the user’s computer. The server side scripting is more effective in generating dynamic content. One major drawback with it is that its used can strain low end, high traffic machines. Moreover it could be exploited to gain access to the machine if not properly secured.

In each of the above case dynamic web page is the result. The two methods of generating dynamic web pages can be used together. In the web based weather data analysis and presentation project the server side scripting was heavily used because it turned out to be more effective especially in the data analysis part of the project and the graphical out put presentation. This could only be realized with the help of server side scripting.

3.5 Different technologies of realizing the server side script

Server side scripting can be realized by many technologies. As mentioned earlier some of the most popular technologies in use in the market are ASP, ASP.NET, PHP, Perl, JSP, whose abbreviation has been explained earlier in chapter 3. JSP and PHP are the most popular in the generation of dynamic web content. The implementation of the web based data presentation using PHP and JSP is discussed here and in the end of this subchapter a comparison of the two is done.

PHP usage in web data analysis and presentation

There are three types of dynamic information that are there on the web namely dynamic data, dynamic web pages and dynamic content. In dynamic data only the variables within a web page are generated. In the dynamic web page the whole web page is generated where as in dynamic content only a portion of the web page are generated. In the context of the weather data presentation the dynamism on the weather home page could be viewed as dynamic data. This is because at each refresh only the readings on the instruments changed but the rest of the web page remains unchanged. As for the HTML page returned to the user who is interested in weather history the whole web page is generated. So this is a case of a dynamic web page.

All the dynamic content have one thing in common: they all come from a data source outside the originating page. Table 4 below shows a list of possible data sources and the PHP functions that handle them.

Table 4: Possible data sources and PHP functions

Data source	**PHP function**	**Comments**
User	`$_POST` `$_GET`	These functions handle data that's directly entered by the user on a web form
Database(local or remote)	sqlanywhere_connect()	This function is only specific to SQLAnywhere database. It establishes a connection to the database from a PHP script

In the web based weather data analysis and presentation PHP was one of the possible server side scripting language that could be used. Being a robust programming language its usefulness can not be underestimated. The user input could be processed and all the calculations done with PHP. Using its rich graphical library JPGRAPH the weather data could easily be presented in a clear and an analytical manner through the charts. The JPGRAPH which is used in PHP there are a lot of inbuilt functions which are used to create and format charts. Such functions are very useful in the weather data presentation using a line graph. Using the PHP_sqlanywhere module one can establish a connection to a SQLAnywhere database and query the data from the database from a PHP script. By so doing it's possible to query the latest data from the database and use it to update the weather readings on the weather homepage.

PHP has a lot of advantages over many other server side scripting languages. It can easily be embedded into HTML and its use is very widespread. It can also include a lot of server functionality that takes the user input, manipulate and display it on the web. PHP version 5 is fully object oriented and platform independent. Its speed makes it useful in building large and complex web applications. In fact PHP is reputed to be the fastest scripting language. PHP is an Open Source Code .This means that the actual code that is PHP is available to the public free of charge. So PHP is very cheap. Because PHP is open source there is a large community of developers that help each other with the code. Therefore they can write reusable pieces of codes called functions and classes instead of reinventing the wheel. As a result the production time for a PHP script is dramatically reduced. The syntax for PHP is quite easy. It's similar to C language.

On the other hand PHP has disadvantages as well. PHP has very poor error handling capabilities. However this can be over come using a feasible advantage solution.

Java Graphing using JGRAPH

JGraph is another possible way in which charts in Java can be realized. It should not be confused with JPGraph. As stated earlier JPGraph is a graphing library used in PHP where as JGraph is a graphing library in Java. JGraph is a free, mature, and robust Java Graphing framework that fully complies with Swing design principles. It contains all the graph visualization and interaction functionality that's expected in a graph library, including multiple views, layering, zoom, drag and drop, undo, automatic expanding and collapsing, routing, and layouts. One can create surprising good workflow editors, call graphs, CAD tools, network diagrams, database visualization tools, and more. It can also be deployed on the server-side with a large range of image exporting functionality. It is fully documented and commercially supported. The main drawback of using JGraph is that it's slow and uses a lot of resources:

JSP

JSP is an abbreviation for Java Server Pages. JSP is a Java technology that allows the developers to dynamically create HTML or XML document in response to a user request.

In order to run a JSP script one need to have a JSP capable web server that's up and running. There are many servers that are available most of which are free for evaluation purposes. Some of the servers available freely are Blazix from Desinderata Software, Tomcat from Apache Software Foundation, Weblogic from BEA Systems just to name a few.

JSP simply puts Java inside HTML pages. For example a HTML file can be changed to a JSP file by simply changing the extension from **.html** to **.jsp**. Once each of the two files are loaded they produce the same output but the JSP file will take longer to load for the first time. The reason for this is that the HTML file is being converted to a Java file, compiled then loaded. This happens only once and so after the first load the file doest take long to load anymore.

The HTML file with a JSP script embedded in it looks close to one with PHP embedded in it except that the JSP script is enclosed with the tags ”<>” and ”%”.

Example

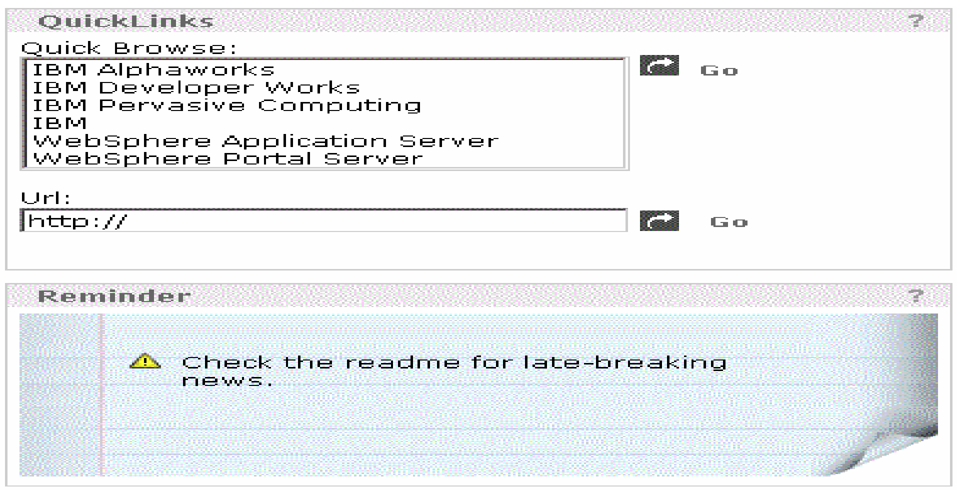
```
<HTML>
<BODY>
Hello!  The time is now <%= new java.util.Date() %>
</BODY>
</HTML>
```

Each time the page is reloaded in the browser, it comes up with the current time. The character sequences <%= and %> enclose Java expressions, which are evaluated at run time. This is how JSP generates dynamic HTML pages.

Creating dynamic charts using JSP

In order to realize a time series chart in JSP one needs to download the JFreeChart library. This is an open source graphing and charting library. It can be freely downloaded from the link:
http://www.jfree.org/jfreechart/download.html

The library supports the standard array of pie, bar, line, area, and other charts. In order to use the library one needs to create a Portlet[16]. Portlets are pluggable user interface components that are managed and displayed in a web portal. Web Portals provide a secure, single point of interaction with diverse information, business processes, and people, personalized to a user's needs and responsibilities. A portlet is a Java servlet that operates inside a portal. Portlets can be set up using an IDE such as Sun Java Studio Creator or even Eclipse. Below in figure 12 is an example of a Portal with two Portlets. In this example, one portlet allows the user to go to a URL; the other portlet provides some reminder text to the user.



QuickLinks ?

Quick Browse:

IBM Alphaworks
IBM Developer Works
IBM Pervasive Computing
IBM
WebSphere Application Server
WebSphere Portal Server

Go

Url:

http://

Go

Reminder ?

Check the readme for late-breaking news.

Figure 12: Example of a portal with two portlets

ChartPortlet resides on a portal server, such as Sun Java System Portal Server. When the portal server renders the complete view, it calls the portlet's *doView()* method. In that method, the following is performed: connection to the data source, in this case the Weather database to find out if some new data is available. If there is some new data it’s fetched throw a sql query. The API of the JfreeChart library is called to generate the time series chart. Using the Java 2D API from the *javax.imageio* package the image is saved in the web server’s tempfiles. The URL for the image as a portlet request parameter named CHART_PORTLET_IMAGE_URL is generated and saved. The above process has been illustrated diagrammatically using a sequence diagram on the figure 13 below

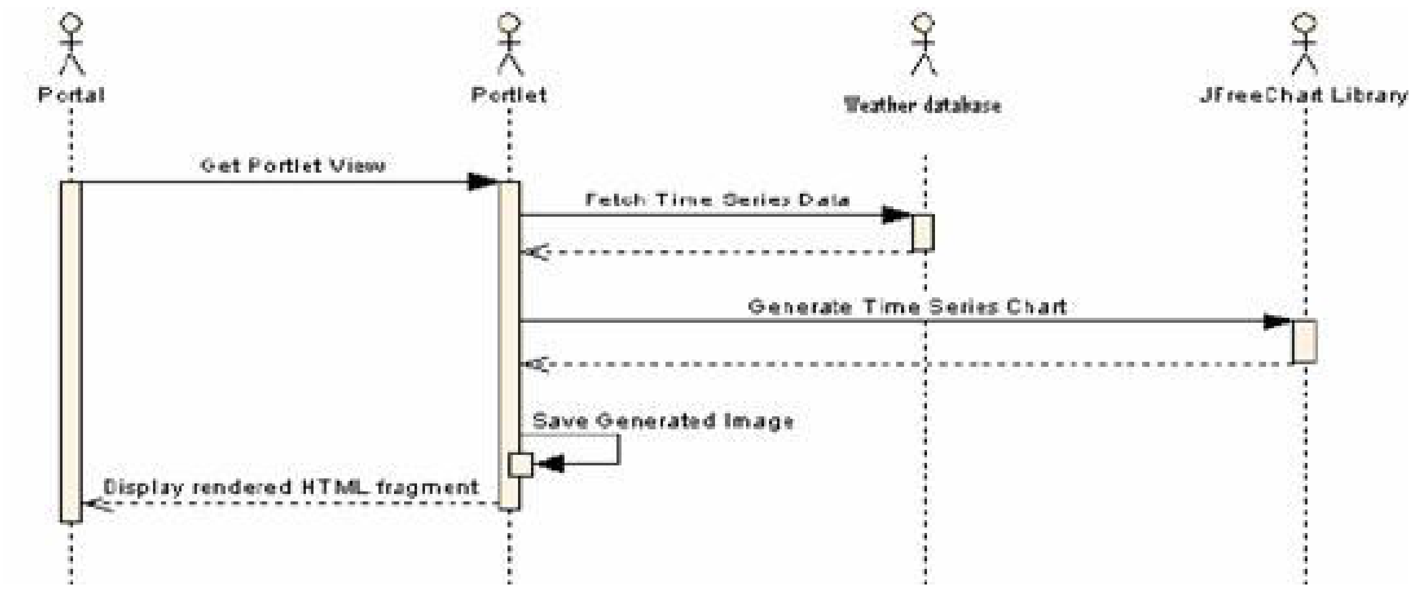


Figure 13: Sequences diagram describing generation of dynamic HTML page

JSP has many advantages and it’s widely used. Most JSP pages are primarily HTML. So any Java programmer who is familiar with HTML should be able to learn JSP techniques within a matter of hours. Debugging JSP pages is also quite simple, as many popular Java IDE tools and Web page design tools now support debugging or syntax-highlighting of JSP pages. It’s more efficient where heavy loads are concerned. It’s also an open source technology.

On the other hand although JSP is good at producing HTML content, JSP pages must include logic to produce dynamic content. This means that presentation and logic code coexist in JSP pages, and are tightly coupled. This makes development and maintenance more costly. The greatest disadvantage for JSP is that JSP pages often run slower than similar pages such as PHP due to being more robust and intensive.

From the above discussion of JSP and PHP it’s quite clear that both methods of realizing the solution would work perfectly in the web based weather data analysis and presentation. It calls for very careful needs analysis before the web developer decides on which one to settle for between the two. Essentially, this choice depends on whether the project in question is viewed as a web site or as an application -- which just happens to have a web site as its user interface. Another way of putting it is that developer needs to pose this question: Are the most challenging tasks that will be faced about validating form input, or about problem domain things that have nothing to do with HTML? If the most challenging tasks fall in the second category, then the developer would want to look seriously at Java and JSP as the better option to go for. If the most challenging tasks faller in the first category, PHP could be close to ideal. Of course, real applications fit somewhere in the middle of these two categories. In the web based data analysis and presentation project, PHP was preferred. This is due to the huge amount of data involved in the results returned by the queries from the weather database; PHP technology was preferred due to its speed and ease of use.

4 Requirements and Requirement Analysis

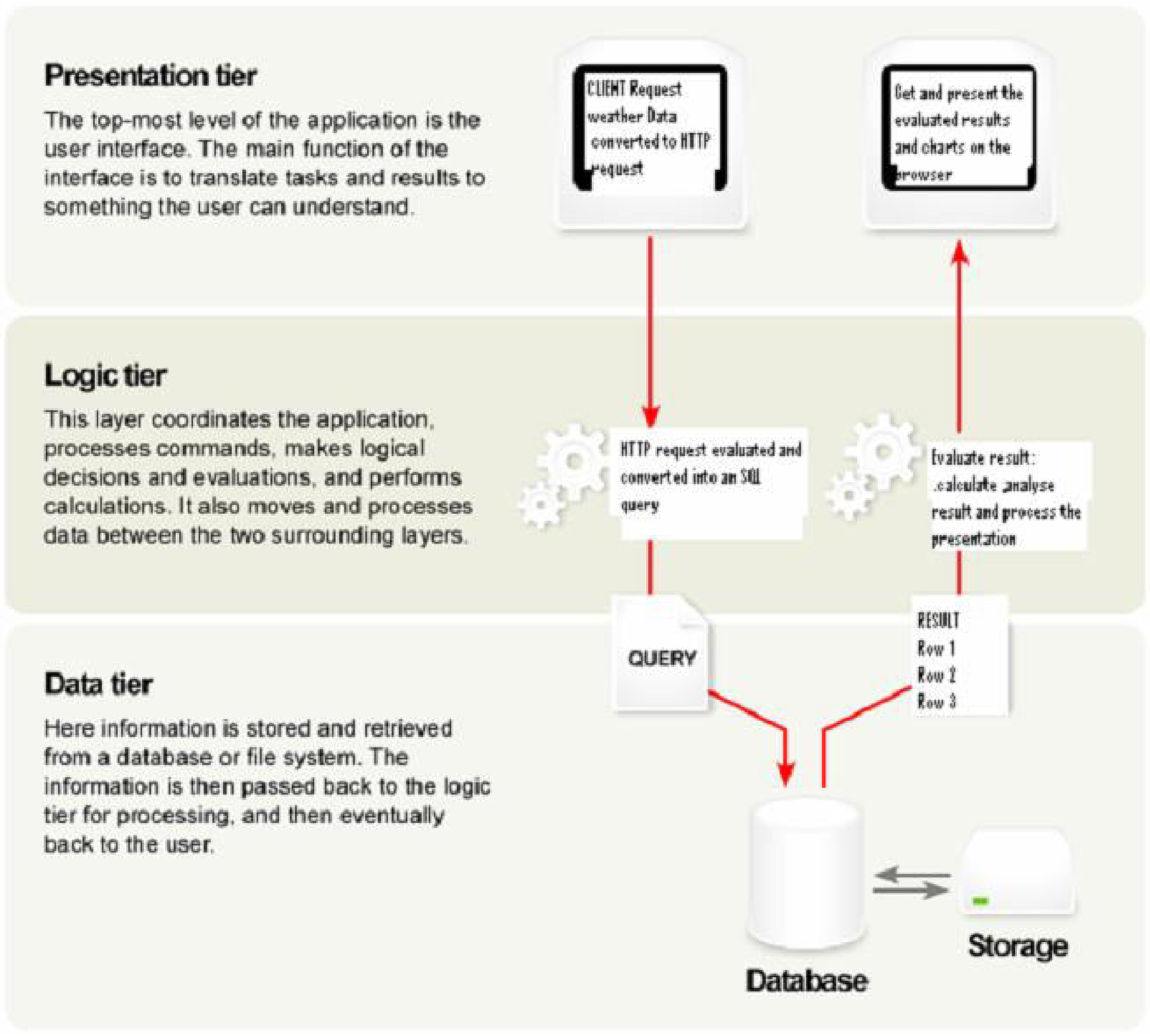
The weather station project requirement are widely discussed and analysed here. The requirements for the web based weather data presentation and analysis can be widely viewed in the context of three-tier architecture. Before the discussion of the project requirement commences its important to discuss the three-tier architecture.

4.1 Description of Three-Tier Architecture

'***Three-tier'* is** a client-server architecture in which the **user interface**, **functional process logic**, **data storage** and data access are developed and maintained as independent modules, most often on separate platforms. In the client-server architecture the client computer request fro the services from the server computer and the server computer processes some data and gives the results to the client computer.

Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface, functional process logic may consist of one or more separate modules running on a workstation or application server, and an RDBMS(Relational Database Management System) on a database server or mainframe contains the data storage logic. The middle tier may be multi-tiered itself (in which case the overall architecture is called an "n-tier architecture").In the Web development field, three-tier is often used to refer to Websites. A front-end Web server serving static content, a middle dynamic content processing and generation level Application server and a back end Database comprising both data sets and the Database or RDBMS software that manages and provides access to the data.

The three tier architecture in the context of the web based data analysis and presentation weather station is explained further using the figure 14 below



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

CLIENT Request weather Data converted to HTTP request
Get and present the evaluated results and charts on the browser
HTTP request evaluated and converted into an SQL query
Evaluate result: calculate analyse result and process the presentation
QUERY
RESULT
Row 1
Row 2
Row 3
Database
Storage

Figure 14: Overview of a three-tier application

4.2 Requirements in the context of the three tier architecture

Having discussed the three tier architecture it's important to focus the attention on the task at hand. The weather station data is recorded in the Weather database. This data is updated after every one minute where a new row of data is added to the table in the weather database.

Weather Homepage Requirements

The weather station data stored in the database data needed to be availed to the users on the internet and in this case availed in a friendly format. It should also be easy to interpret the data as it's presented. The current weather conditions are very important to the user in a manner that is easy to have a quick view of the present weather parameter reading during the time of view. It's therefore very important that the user views it in form of graphical images of the weather parameter instrument that is familiar to him. Example: an image of thermometer on the weather homepage indicating the present temperature condition would be quite essential. In order to fulfill this need the graphical images of the weather instruments were required. It was therefore required that these graphical images of the weather measuring instruments be available on the home page reflecting the present weather condition in all their readings. It was further required that the readings on each of the instrument be updatable as often as the weather conditions can change.

The link to the HAW Hamburg home page was also required on the home page to help the users to easier navigate and find out more about HAW Hamburg .The home page was also required to provide user with a means to access the past weather data or weather history. Further to this the means for the user to obtain the weather history for a given hour, day, month and year was required. The user should also be provided with means to view weather history of user custom time period. Therefore the buttons under which lay the links to each of the user preferred views of the weather history were needed.

Weather History Requirements

In order for the user to view the weather history in form of data stored in the Weather database the user needs to request the data to be presented to him and in a certain format. By clicking any of the buttons leading to the user defined history the user should be able to select the preferred time, date or year through a user interface. Therefore the user interface was very much required. On the user input form it was a requirement that the user should be able to input the start date for the preferred weather history duration. Specifically for the hourly weather history the user selection form was required to allow the user to select the year, month, date and the hour. For the daily weather history it should be possible for the user to select the year, month and date. Same applied to the monthly history where it should have been possible to select the year and the month of choice. In the annual history there should be a provision to select the year the user wishes to view the history.

Due to the nature of the flexibility that a user was allowed to have when viewing the custom weather history special handling of the data fetched was required. It’s important to note that the user was allowed to select time interval within which to view the weather history of a maximum of one year. The reason for this was that the amount of data involved in a time period of more than one year would take a very long time to calculate and analyse.

The user request is converted into a HTTP request by the client, the browser in this case. Once the HTTP request is accepted by the web server it needs to be processed, evaluated and converted into a sql query so as to request the data from the database. The DBMS only understand request in form of a sql query. The DBMS answers the query in form of a row of results representing the columns requested. These results are returned to the application server(middle tier).From here the results are evaluated, analyzed and converted in the format requested by the client(user).The middle tier was therefore definitely needed in order to fulfill the whole task. All the components that needed to be developed in order to realize the expected result are diagrammatically explained in the figure 15 below.

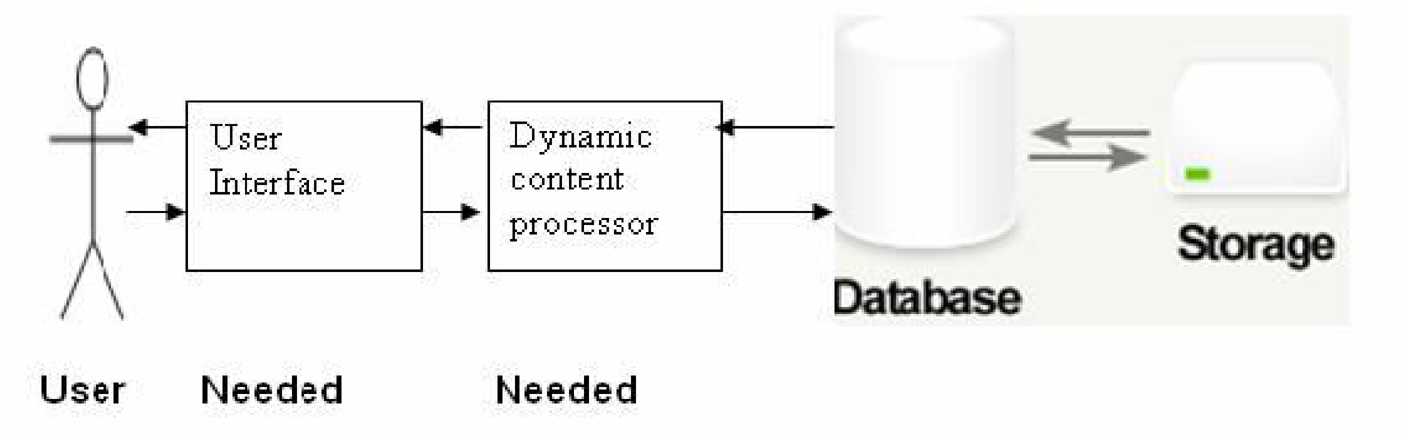


Figure 15: Required/Present 3 tier components in the Weather Data presentation project

4.3 Output Requirements

Once the user is able to feed the system with the correct weather history inputs it's upon the system to deliver the correct output to the user. In terms of output it was required that the system should be able to present the data for the past weather conditions as requested by the user in a timely and an easily understandable and interpretable manner. The minimum requirement was the presentation of the requested past data in figures. However due to the extra requirement that the data has to be presented in an easily understandable and interpretable manner it was required that it be presented in form of line graphs and tables. The main reason for requiring the line graph is that it's easy to see the trend with changing time.

For the hourly and the daily weather histories, it was required to present the weather data for all the weather parameters on the same page. That means that in the user input selection form it was not a requirement to select the weather parameter to output since they would all be presented. Furthermore the hourly and daily weather data was not expected to be large. It was therefore necessary to present the data in form of graphs. For the monthly, Annual and user custom data it was required that the user be able to select on the parameter of choice on top of selecting the dates. It was therefore required that the output of this data be both graphical and tabular for easier interpretation.

It has been stated earlier that the 60 seconds mean was calculated and recorded in the database for every weather parameter. It was still observed that this data when plotted as it is was still very noisy and therefore required to be smoothed. So plotting the smoothed the data was an important requirement to be fulfilled. Data smoothing techniques are used to eliminate "noise" and extract real trends and patterns.

Some of the available smoothing methods available are the *Random method*. This method is best when each period's data has no relationship to the pattern in the previous data. Under this condition, the best prediction for the next value in a series is simply the average of all previous data points. *Moving Average* is another possible data smoothing technique that can be used. This method works well if the data contains no trend or cyclic pattern. Moving averages is calculated using the formula below:

$$y'_k = \frac{1}{n} \sum_{j=k-n}^{k-1} y_j$$

n is a user-supplied constant greater than zero defining the number of consecutive points to average. ***y'k*** is the kth smoothed element. Higher values cause greater smoothing.

For all the cases in the **mean**, **standard deviation** and **1 SD** band were required to be plotted in all the charts. The arithmetical mean is defined as the sum of all the members of the list divided by the number of items in the list. For a data set the mean is the sum of the observations divided by the number of observations. The general formula for calculating the arithmetic mean is as shown below:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + \cdots + x_n).$$

x_i is the ith element from the list of elements whose mean is to be calculated and n is the total number of elements.

In order to be able to explain the Standard deviation it's important to define the variance first. **Variance** is a measure of its statistical dispersion, indicating how its possible values are spread around the expected value .The expected value shows the location of the distribution, the variance indicates the scale of the values. Usually mean which is described above is taken to be the expected value. In general, the population variance of a finite population of size *N* is given by:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2$$

Where xi is the ith element of the population and is the population mean. Although variance was not explicitly required for the data presentation it's important to calculate it since Standard Deviation is calculated from it. The variance is not a very interpretable measure of dispersion. A more interpretable measure is the square root of the variance, called the standard deviation abbreviated as SD. It gives in a standard form an indication of the possible deviations from the mean. It is usually denoted with the letter σ (lower case sigma). It is defined as the square root of the variance. In other words, the standard deviation is the root mean square (RMS) deviation of values from their arithmetic mean.

It's calculated using the formula below:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

Where x_i is the ith element of the population and $\overline{X}$ is the population mean. Standard deviation is a statistic that shows how tightly all the various observations are clustered around the mean in a set of data. When the observations are pretty tightly bunched together and the bell-shaped curve is steep, the standard deviation is small. When the observations are spread apart and the bell curve is relatively flat, that implies that there is a relatively large standard deviation. One standard deviation on either side of the mean gives **one standard deviation band range**. It accounts for around 68 percent of the total observations as it can be seen from figure 16 below:

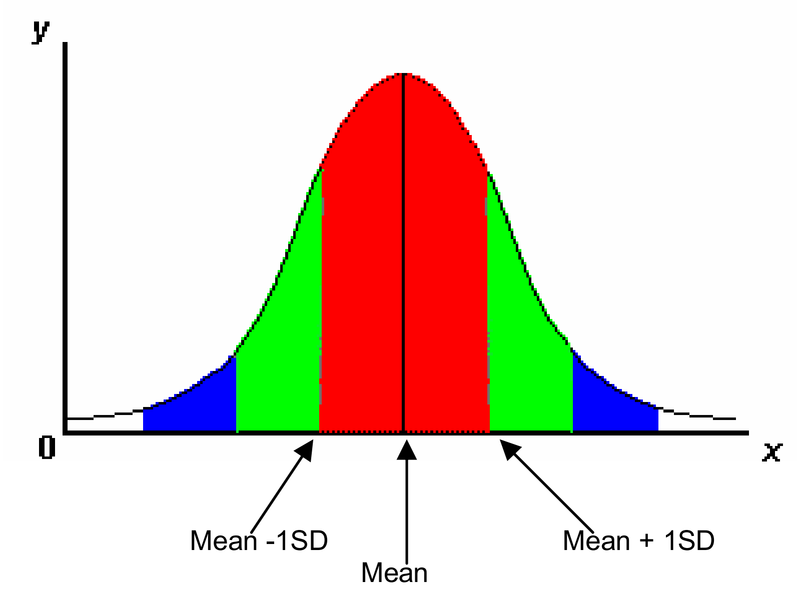


Figure 16: Normally distributed data showing 1 standard deviation band range

If this curve above were flatter and more spread out, the standard deviation would have to be larger in order to account for those 68 percent or so of the observations. That's how the standard deviation can be an indicator of how the observations in a set are spread out from the mean.

The requirement was to plot the 1 SD sigma band range. However there exists other sigma band ranges as well such as 2 SD sigma, 3 SD sigma band and 4 SD sigma bands that can be used depending of the analytical needs. Statistically for a normally distributed population, there is a 68% probability that the population will fall within 1 standard deviation, a 95% probability that the population will fall within 2 standard deviations. There is 99.7% probability that the population lies within 3 standard deviations and 99.9% probability that the population lies within 4 standard deviations. For Gausian normal distribution with mean = 0 and SD =1, N(0,1) the actual values of probability can be obtained from the standard normal distribution tables.

5 Design

5.1 Weather data Presentation Homepage

The figure below shows a block diagram describing how the weather homepage was designed to look like figure 17 below:

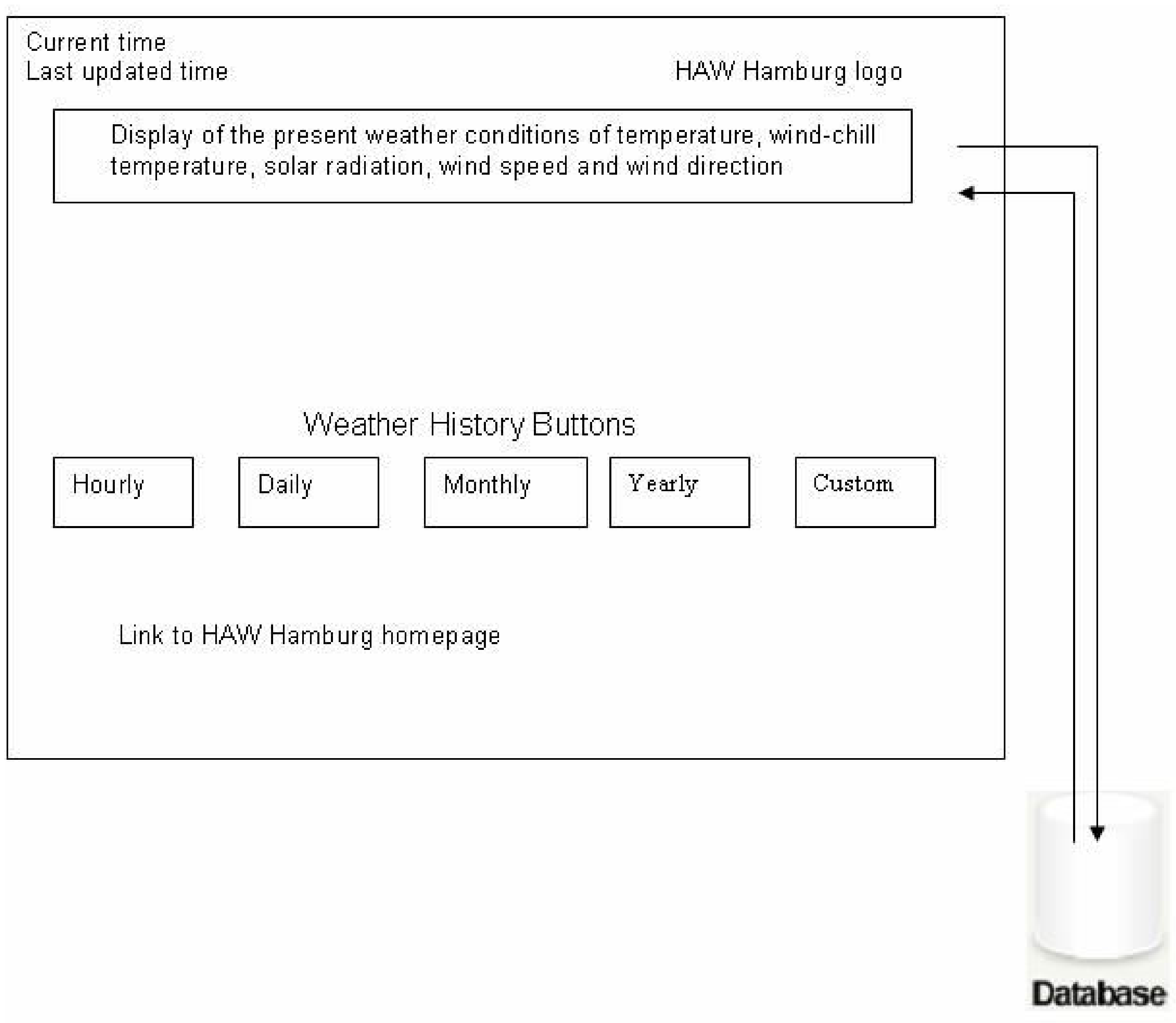

Current time
Last updated time
HAW Hamburg logo
Display of the present weather conditions of temperature, wind-chill temperature, solar radiation, wind speed and wind direction
Weather History Buttons
Hourly
Daily
Monthly
Yearly
Custom
Link to HAW Hamburg homepage
Database

Figure 17: Weather homepage design

As mentioned in the previous chapter, the first requirement was the development of a web page which should be able to display the current weather conditions. In order to fulfill this requirement five different drawings of the weather measurement instruments were drawn using PHP functions.

Presentation of the current weather situation

In order to present the latest weather data from the database the connection was established from the PHP script, used to draw the weather instrument to the database. This was achieved with the help of the PHP function:

connection = sqlanywhere_connect("UID=;PWD=;eng=;dbn=;links=") . 'UID' means the user name used to access the database and 'PWD' means the user passord.'eng' is the name of the data server. 'links' is assigned the name of the networking protocol[17] used, e.g. 'tcpip' .

An SQL query was then formulated which read the top most data from the database which was only one minute old as shown below. Example of reading atmospheric pressure to be displayed by the barometer:

query string="select top 1 fldRecordedAt,fldBarometer from dba.tblRawData order by fldRecordedAt DESC";

The field names have been explained in chapter 3

The connection $con together with the query $q were used to read the data from the database and returned a result set comprising of one row of weather record as required. This was done with the help of the method : *resultresult = sqlanywhere_query(connection, query).* The column number one was assigned to the local variable

A local variable is assigned the fetched column i from the result set which goes under the name *column[i]* . The variable name column[i] is an array element defined and it actually refers to the ith column of each row of the result set. It refers to the various columns of the returned value of the result set fetched after execution of an SQL query.

After the results are obtained and assigned to a local variable it is necessary to free the database resources associated with the results returned. It is also necessary to close the connection. This is achieved with the method sqlanywhere*_disconnect($conn).* The reason for disconnecting from the database is because the number of users connected to a given database at one given time is limited. Usually when this number is reached then the other users would have to wait for connection by queuing which means long delays. By disconnecting from the database after fetching the data then there are more connections available and that means the performance is good.

The local variable $pressure was used to calculate the height of the mercury fill that filled the barometer to indicate the pressure. Once the height was calculated a loop was used to fill up the barometer tube drawing up to the level calculated. The tube was well marked and the height of the tube was drawn to scale. So the actual reading on the tube was exactly the same as the reading obtained from the database. The height is calculated and the fill level is calculated. Similar procedure was used to display the latest weather data for the solar radiation, wind direction, wind speed, temperature and wind chill temperature. It’s important to note that the drawing and displaying of the weather parameter in each case differed a little although the same principle was used to draw the images of the instruments and display the weather parameters. Notable difference was in the solar radiation where the actual reading was presented with figures as they were read from the database and not graphically.

Weather History buttons

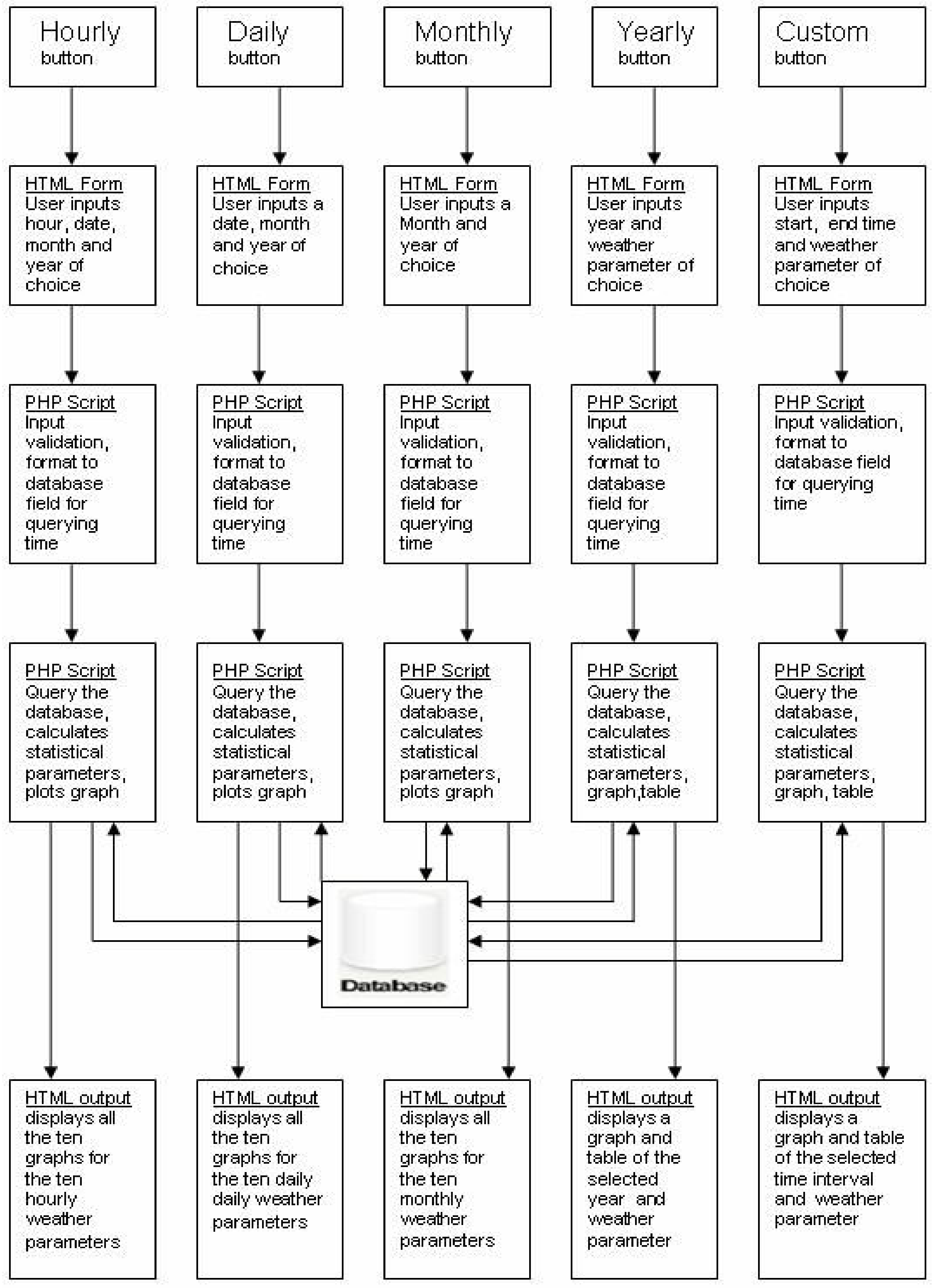
Another important requirement was the ability of the user to view the weather history. The home page is designed in such a way that the user could choose the duration of the time interval of interest. Four fixed time intervals are added namely hourly, daily, monthly and yearly. In addition a user custom time interval provided where the user could flexibly choose the time interval of interest. For all these possibilities buttons on the homepage are provided that link the user to the HTML form. On this HTML form the user can input the times one interest. The details of what actually happens behind the buttons are discussed in the next subchapter.

5.2 The weather history presentation

On clicking any of the weather history buttons on the homepage, the user is directed to an interactive webpage where user inputs can be inputted. This is known as the HTML form.

The HTML form basically provides a medium through which the user can send a message to the application. In three-tier architecture it acts like a bridge between the client layer and middle layer. In order to make it easier for the user, the option selector was used instead of the user having to type the input. This was very important because the problems of misspelling month names by the user and invalid dates are completely eliminated. In the case of hourly weather history the user needs to select the hour, date, month and year of choice. For the daily weather history then user needs to select the date, month and the year. In the monthly weather history the user needs to select the month and the year. In the year weather history the user needs to select the year and the weather parameter of choice. Due to the large amount of data involved in the year weather history, it wasn’t possible to present all the weather parameters together. So they are presented separately depending on the wishes of the user. So at this point the user is given the chance to select the parameter of wish.

Once the user has selected the inputs and submits the HTML form by clicking ”submit” the input is availed to the PHP script for validation using the $_GET variable and the HTTP GET method. The $_GET variable is an array of variable names and values sent by the HTTP GET method. The $_GET variable is used to collect values from a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and it has limits on the amount of information to send (max. 100 characters).The figure 18 below gives a block diagram of how the presentation of weather history was designed.



Hourly button
Daily button
Monthly button
Yearly button
Custom button
HTML Form User inputs hour, date, month and year of choice
HTML Form User inputs a date, month and year of choice
HTML Form User inputs a Month and year of choice
HTML Form User inputs year and weather parameter of choice
HTML Form User inputs start, end time and weather parameter of choice
PHP Script Input validation, format to database field for querying time
PHP Script Input validation, format to database field for querying time
PHP Script Input validation, format to database field for querying time
PHP Script Input validation, format to database field for querying time
PHP Script Input validation, format to database field for querying time
PHP Script Query the database, calculates statistical parameters, plots graph
PHP Script Query the database, calculates statistical parameters, plots graph
PHP Script Query the database, calculates statistical parameters, plots graph
PHP Script Query the database, calculates statistical parameters, graph,table
PHP Script Query the database, calculates statistical parameters, graph, table
Database
HTML output displays all the ten graphs for the ten hourly weather parameters
HTML output displays all the ten graphs for the ten daily daily weather parameters
HTML output displays all the ten graphs for the ten monthly weather parameters
HTML output displays a graph and table of the selected year and weather parameter
HTML output displays a graph and table of the selected time interval and weather parameter

Figure 18: Design of presentation and analysis of the past weather situation

User input validation and input error handling

The $_GET variable in the PHP script receives the variables sent by HTTP GET method.

In the last subchapter on HTML form it was stated that the option selector is used instead of the user having to type the inputs. Thus the unrecognised inputs problem is eliminated. However another problem arises at this point that the data is only available from 31.3.2006 at 17:01 hrs. A user can possibly select a date before this date and time. A user can as well select to view weather history of a future date or time. No data is available for future date. As a result a *JPGRAPH error* is issued and the user cannot proceed from there. This was solved by asking the user at the top of the HTML page to only select date after the date when the data was first available (March 30th 2006 5:07 pm). In case the user still went ahead and ignored the warning then the same message is repeated and redirected back to the same HTML.

Time formatting

After having checked the user input and found to be valid the next thing is format the time input to a format that marches the *fldRecordedAt* time column in the *tbl.dba.RawData* table of the weather database. The column fldRecordedAt is in the time format:

"year-month-date hour:minute:second.millisecond".
A part from converting the user inputted time. It's necessary to take into account the weather history that the user would like to view. Then an end time marking the end of the hour, day, month or year that the user would like to view was calculated as shown below

end time =(start time + 1 hour) : for hourly weather history view

end time =(start time + 1 day) : for daily weather history view

end time =(start time + 1 month) : for month weather history view

end time=(start time + 1 year) : for yearly weather history view

These above calculations do not apply to user custom weather history since the user selected both start and end times of the time of interest.

Armed with these two local variables namely $end and $start its all ready to use the variables to query the database. A separate PHP script was written so that it could handle the task of querying the database and other tasks concerned with the manipulation of the queried data. In order to pass this data from the present PHP script to the other PHP script

SESSIONS were used. Sessions and their usage have been discussed in the next sub chapter.

5.3 Sessions

A normal HTML website will not pass data from one page to another. In other words, all information is forgotten when a new page is loaded. This makes it quite a problem for tasks like the PHP embedded in the HTML page described above to be remembered from one page to the other. A PHP session solves this problem by allowing one to store information on the server for later use (start time and end time). However, this session information is temporary and is usually deleted as quickly as the user has left the website that uses sessions. It's important to state that sessions are not recommended for sites that require high security like log in web sites.

Before beginning to store the information in the PHP session, one must first start the session. When a session is started, it must be at the very beginning of the code, before any HTML or text is sent. A session can be started like the script shown below shows it:

```
<?php
session_start(); // start up a PHP session!
?>
```

In order to store a variable in a session a session associative array is used. In an associative array a key is associated with a value. This is where session data is both stored and retrieved. This can be done as shown here below:

```
<?php
session_start();
$_SESSION['views'] = 1; // store session data
echo "Pageviews = ". $_SESSION['views']; //retrieve data
?>
```

The above PHP script would yield the output '1'.

In order to pass the variables from the first PHP script after it is formatted; to the next PHP script such a session is used. Once the data has been stored in the session the name of the session can be called from any script in the same server and assigned to a local variable of the script. In the second PHP script the session name is simply called and its contents assigned to local variables

The above variables were then used in the second script in order to do further processing of the data and presentation

5.4 Connecting to the database and data fetching

In this section the connection to the database and the fetching of the data from the database is going to be discussed. Also to be discussed is the assignment of the fetched data to the local variables for use in the statistical analysis and presentation.

As mentioned earlier the DBMS used in the weather database is SQLAnywhere. As has been explained earlier in subchapter 5.1.2 the PHP function is used to connect to the database as shown below:

connection = sqlanywhere_connect("UID=;PWD=;eng=;dbn=;links=") .

Where UID is username,PWD = password,eng= data server name,

dbn = database name and links = networking protocol.

The above function establishes a connection to an Adaptive Server Anywhere database. It returns a positive Adaptive Server Anywhere link identifier on success (connection), or an error and 0 on failure.

A query is a form of questioning in a line of inquiry. A database query is the standard way information is extracted from the database .In order to extract the data from the weather database a query is formulated as shown below:

Query string = "select fldRecordedAt, fldBarometer, (SELECT MINUTE (fldRecordedAt)) from dba.tblRawData where fldRecordedAt BETWEEN start time and 'end time"

This is an extract of the query as used to fetch the atmospheric pressure data.

Select is an SQL keyword used to specify the columns that should be included in the result set*. fldRecordedAt* and *fldBarometer* are column names in the dba.tbl.RawData table where the weather data is recorded.

MINUTE (d) is an SQL function that extracts minutes from a date d.
Example: MINUTE ('2006-03-30 17:07:55.953') returns 7.
Similar SQL functions exist that extract different components of a date as shown below:
MONTH(d) that returns month , *DAY(d)* returns a date , *HOUR(d)* returns hour and *YEAR(d)* returns year from a date.

From is an SQL keyword that precedes the name the table where the data is recorded. In the case of the weather database all the data is recorded in the table called *dba.tbl.RawData*.

Where is an SQL keyword that is used to restrict the returned result set to only the rows that meet the specifications stated after the keyword *where* .In the query that was defined in order to extract the weather data the result set was restricted to the rows of records that were recorded between the dates specified. It’s important to recall that the dates specified in the query are the local variables assigned to the session variables stored from the previous PHP script. The PHP script had in turn obtained these dates from the user selection in the HTML form.

The result of the query is assigned to a local variable for easier handling and use in the PHP script to obtain the data from the database. With the query and the connection to the database established, the query is executed using the PHP function:

result = sqlanywhere_query (connection, query string)

The above function prepares and executes the SQL query on the connection. It returns a positive value (result) representing the result ID on success, or 0 and an error message on failure.

After executing the query the data is fetched row by row using a *while loop* with the function as shown below:

while ((row = sqlanywhere_fetch_row (result set)))

This function fetches one row from the result set at a time. This row is returned as an array that can be indexed by the column indexes only. In general this function returns an array that represents a row from the result set or false when no rows are available. Each of the columns of the rows in the result set is assigned to a local variable. This is done to every row one by one where each row is accessed using a while loop until now more rows are remaining as shown below

```
while ((row = sqlanywhere_fetch_row (result set))) {
          curr_row++
            local variable= row[i]
}
```

curr_row is an internal counter that’s incremented each time a new row of the result set is fetched.

Once all the required result has been fetched, the database resources associated with the result identifier $result is freed using the function:

sqlanywhere_free_result (result)

In addition the connection that had been established using the *sqlanywhere_connect()* is closed using the function

sqlanywhere_disconnect(connection), whose purpose has been explained in the last subchapter.

In order to calculate the mean the data fetched is separated by time intervals of one minute. So each minute is associated with a given row of the data. In order to represent this data in various graphs the data mean is calculated for each unit time interval in the required weather history.

The parameter data for each minute is added to a locally defined variable which was initially zero. At the same time the minute counter is incremented by one and the minute counter is checked if it’s equal to sixty which is the minutes that make up one hour. If so the parameter total is divided by the number of minutes so far to get the hourly average. The hourly data total and the minutes counter are reset to begin calculating for a new hour. It’s important to note that due to the problem of the missing data it is not automatic that the number of data recorded within one hour is sixty. For the hourly weather design history, the data is plotted as it is obtained. The problem of the missing data will be discussed in details in another chapter a head.

For the daily, monthly and annual histories it is now possible to plot all the data for every minute. It is therefore designed in such a way that the mean of the next smaller date element average is calculated and used. For example for the daily weather history the hourly means for that particular day are calculated. So for one day there would be 24 data points each representing the hourly mean.

Similar approach was used when populating the array for the monthly and annual weather histories. The parameter mean for each day was calculated in order to populate the array holding the data for one month. Each element of this array represents the daily parameter average. However special treatment was needed here because of the missing data. For the monthly weather history the array element for a day when the data was completely missing, the array’s element was populated with an element ”” , to signify that no data is available for that time. Missing data problem will be discussed at another chapter.

In the monthly, annual and user custom weather history the data in this array was used in displaying the data in tabular form. In the case of monthly data where the data for a complete whole day was missing it was used to plot the graph. In all other cases it was smoothed before using it to plot the graph. Smoothing will be discussed in a later subchapter.

Handling of the user custom data

The above description for the weather history implementation applies to the weather history for the hourly, daily, monthly and annual weather history. Due to the fact that the user can select start and end date, handling of the inputted data is designed in a slightly different manner.

In order to calculate the data to be used to plot the graph for the user custom history, the whole data within the time interval selected is divided into 50 equal portions. This is achieved by dividing the total length of the weather data array by 50 to obtain the *interval*. The mean for each portion is then calculated and inserted into an array of length 50 used to store these mean data. In order to calculate the mean for each portion, elements of each portion are summed up index-wise while checking if the present index is divisible by the *interval*. This is done by checking if the (current array index mod *interval*) =0. Example if the current array index is 57, then 57 mod 50 = 7. If current array index is 2000, then 2000 mod 50 =0. If the modulus is 0 then the sum of the elements is divided by *interval* and the 50 elements of this array represent the data points to be used in plotting the graph.

Smoothing the parameter data

In this project the moving average technique is used to smooth the data. In this case a smoothing factor of six was used. In this case the first six elements of the array original data array are summed up and then divided by six to obtain the mean, which was assigned to 0^{th} element of the smoothed data array. Then first element to $(1+6)^{th}$ element are summed up as well and divided by six to obtain the first element of the smoothed data array. The same procedure is used to obtain the next elements of the smoothed array up to (number of elements in the original array- smoothing factor)th element. At this point the remaining elements are averaged and the average is assigned to the (number of elements in the original array - smoothing factor)th element of the smoothed array. Each of the remaining elements of smoothed array is obtained by summing successive elements each time beginning from the (present element+1). This procedure is used to smooth data for all the weather histories. However for the annual history the smoothing factor of 2 is used due to the number of data points involved. At most there can be 12 data points in the annual history making smoothing factor of 6 unreasonable.

Calculating Mean, Variance, Standard Deviation and 1 SD Band Range

The parameter mean for the required time interval is calculated by summing all the elements of the parameter data array then dividing by the number of non-null elements. In this case an array is defined whose elements were the mean of the array. If the user is interested on monthly mean and all the days in that month have data then the data is summed up and then divided by the number of days in that month. This leads to a constant valued element array.

Variance in the web data analysis was calculated by first calculating the sum of squares of each element of the population. That is each element the data array is subtracted from the mean and the difference is squared to obtain the sum of squares. Before each element is considered for addition into the sum of squares it’s first checked if its non null. If so then it’s used in the calculation of the sum of squares. If it’s a null value then it’s excluded from the calculation of the sum of squares. Finally the variance is calculated by dividing the sum of squares by the number of non null elements.

Standard deviation is calculated by simply calculating the square root array from the variance array. The whole array of standard deviation is therefore made up of elements made up of equal values which are square root of the variance.

The calculation of the band range is done by defining two arrays. The first array holds equal elements with value (standard deviation + mean). This represents the upper band line. The second array is populated with equal values obtained from the equation (standard deviation - mean). This represents the lower band line

Formatting and displaying the graph

In order to use the data to plot the graph the data is first assigned to a self descriptive array so as to make it easier to recognize what data the array represents. Only the data to be plotted is assigned. In PHP its allowed to define an array and specify its name that's shows the type of elements to be stored like *data['Actual']= elements of actual data*

In order to draw the graph there are two JPGRAPH library files were required and are therefore included. These are:

include ("C:/Documents and Settings/Kamanu_A/My Documents/Web/webserver/jpgraph-2.1.4/src/jpgraph.php");

include ("C:/Documents and Settings/Kamanu_A/My Documents/Web/webserver/jpgraph-2.1.4/src/jpgraph_line.php");

These two files are usually included in the PHP code before they are used, preferably at the beginning. In the project the two lines were included at the beginning of the script. By including the two modules above then it becomes possible to use all the JPGRAPH functions in PHP required to draw line graph.

A graph is drawn, its scaling defined and title added. In order to achieve this, an object of the type GRAPH is created as shown below:

Object name = new Graph (width, height,"cache storage");
The third parameter cache storage is only used if the images are stored in a cache for future retrieval in case they are required. In this particular project it was designed in such a way that only real time images are generated and outputted the images were not stored.

The graph scale was set and a title is added using PHP functions found in JPGRAPH.

The type of scale that should be used for the graph where by x axis is set using PHP functions. There are various types of scales that can be defined such as"textlin", "intint". The first one means that x axis is going to be text defined scale. This means it's going to be scaled according to the data available for the axis as it appears. The y axis is going to be scaled automatically in a linear manner. Different texts with different colors are added to the graph in order to describe which color graph represented what data. Their position in the graph is also defined by specifying the coordinates on the graph area.

A line graph can have as many line plots as one wish. In order to plot one line graph an object of the linePlot is created from an array of data to be plotted. Each data point in the line graph was marked with a red colored filled circle of width four. The color of this particular line which is used for plotting the mean is blue. The above description was implemented as shown below:

linePlot object = new LinePlot($data['smoothened']);//line graph using array //$data['smoothened']

linePlot object->mark->SetType(MARK_FILLEDCIRCLE);// data points marked with // a filed circle
linePlot object->mark->SetFillColor("red");//marks set to red
linePlot object ->mark->SetWidth(4); //marks have width 4

linePlot object ->SetColor('blue'); //line graph color set

The above description just describes how to plot the line for the actual data. Similar technique was used to plot the lines for the mean, upperband and lowerband data. After creating the lineplot object for each of the three line plots, all the three lines they are added to the graph using the Add () function as shown below:

// Add the plot to the graph

$graph->Add(linePlot object1);
$graph->Add(linePlot object2);
$graph->Add(linePlot object3);

Finally the graph is displayed with all the added lines after calling the method stroke() as shown below:

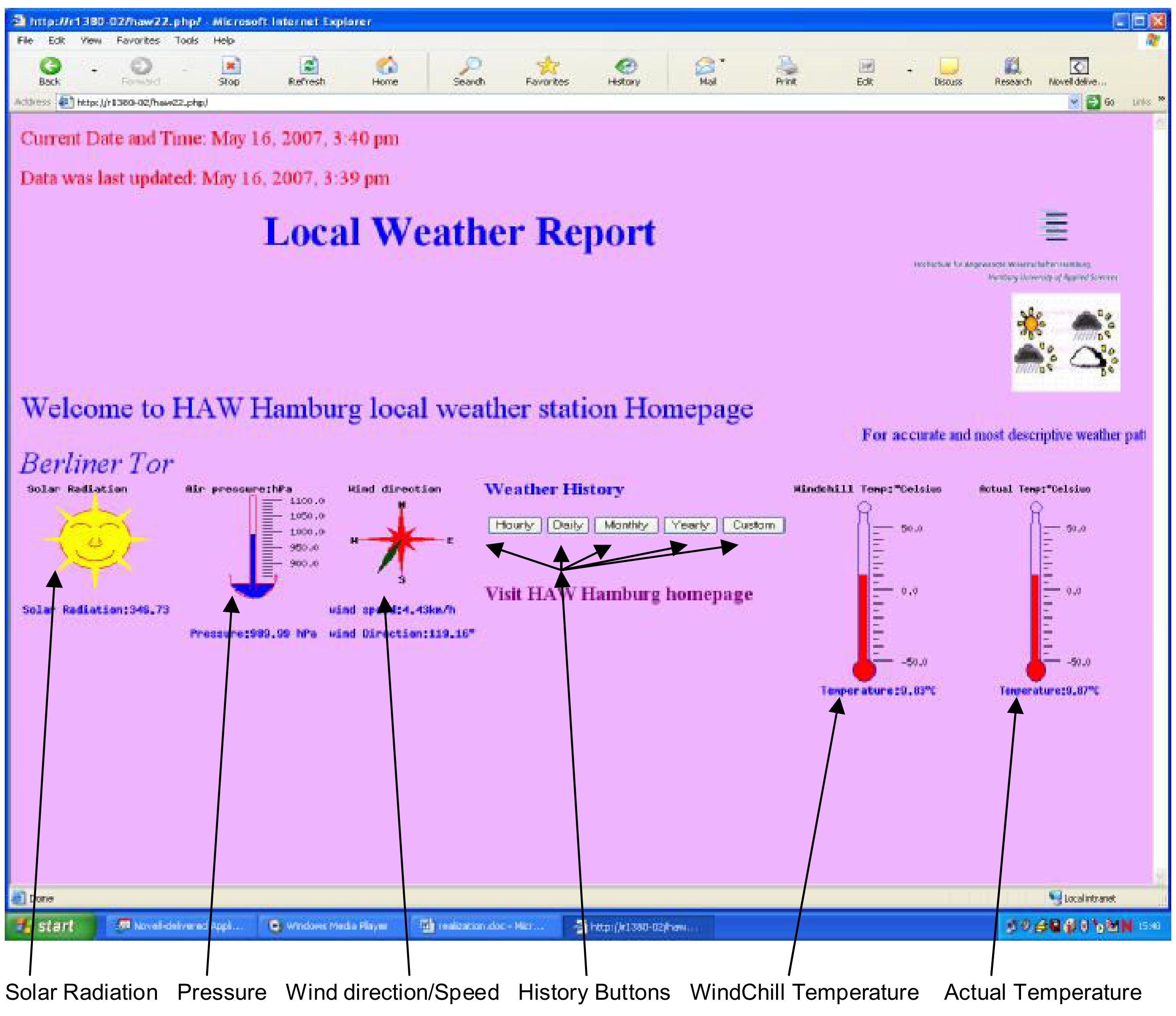
$graph->Stroke();

.

6 Realization

6.1 Weather homepage and present weather situation presentation

As discussed in the last chapter and on chapter two, the first and most important requirement was presentation of the present weather condition. This was realised in the weather homepage as shown in figure 19 below:



Current Date and Time: May 16, 2007, 3:40 pm

Data was last updated: May 16, 2007, 3:39 pm

Local Weather Report

Welcome to HAW Hamburg local weather station Homepage

For accurate and most descriptive weather pat

Berliner Tor

Solar Radiation:346.73

Pressure:989.99 hPa

Wind speed:4.43km/h

Wind Direction:119.16°

Weather History

Visit HAW Hamburg homepage

Temperature:9.83°C

Temperature:9.87°C

Figure 19: Weather home page realized

On typing : *http://r1380-02.haw22.php/* on the browser the above shown weather page opened. Six weather parameters showing the present weather conditions were presented using actual diagrams of the weather instruments used in measuring them and the figures. Solar radiation was displayed with a diagram of the sun and the actual solar radiation was displayed in figures. A diagram of barometer used to measure atmospheric pressure was also realized on the homepage. It reflected the actual atmospheric pressure at Berliner Tor at the moment of view. Apart from graphical image the actual atmospheric pressure in figures was displayed below the image of barometer. Wind direction reading was realized using a compass with a pointer pointing to the direction of the wind. Wind speed is also displayed below the wind direction indicator (compass).Actual temperature and the wind chill temperature present condition presentation was realized with images of two thermometers. Mercury fill, which is red in color in the thermometer images, reflect the actual reading. Weather history buttons are also present on the homepage. With one of this buttons the user can navigate from the homepage to the weather history.

Drawing Instrument Images using PHP functions

Each of the familiar weather instruments can be drawn with the help of PHP functions. For example, such functions were used to draw a filled tube of a barometer with a blue color to represent the mercury fill. The function ImageCreate (int, int) was used to draw the background area on which the image lay. ImageArc($im, int, int, int, int, int, int, $color) was used to draw the curved parts of the image where as: imageline ($im,int,int,int,int,$color) was used to draw the linear part of the weather instrument drawing. In this case $im is the name of the image, and the integer parameters that follow represent the x and y start and end coordinates of the line. In a case where all the four integer arguments of the imageline function are the same, then that represents one single point of the image. The fifth argument is used to represent the color of the image and was predefined using another function *$red = ImageColorAllocate ($image,255, 0, 0).*This could be extended using a loop to fill up a large area in the image like shown in the code snippet below

```
$i=130;
    for($j=0;$j<15;$j++){
        if($i<135)
           imageline ($im,76,$i,(97-$j*0.65),$i++,$blue);
        else
           imageline ($im,76,$i,(97-$j),$i++,$blue);
    }
```

The output from the above code snippet yields the image of a calibrated barometer reflecting the present atmospheric pressure with the mercury fill. This can be seen in the figure 20 below:

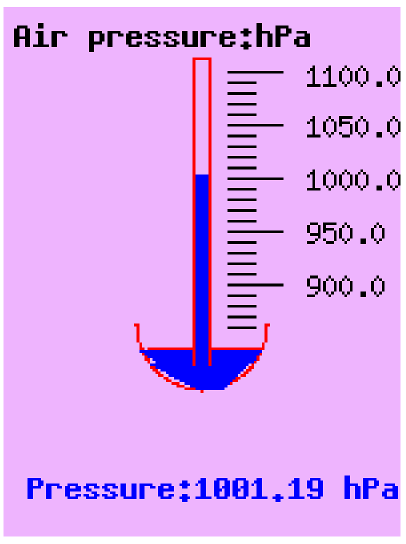
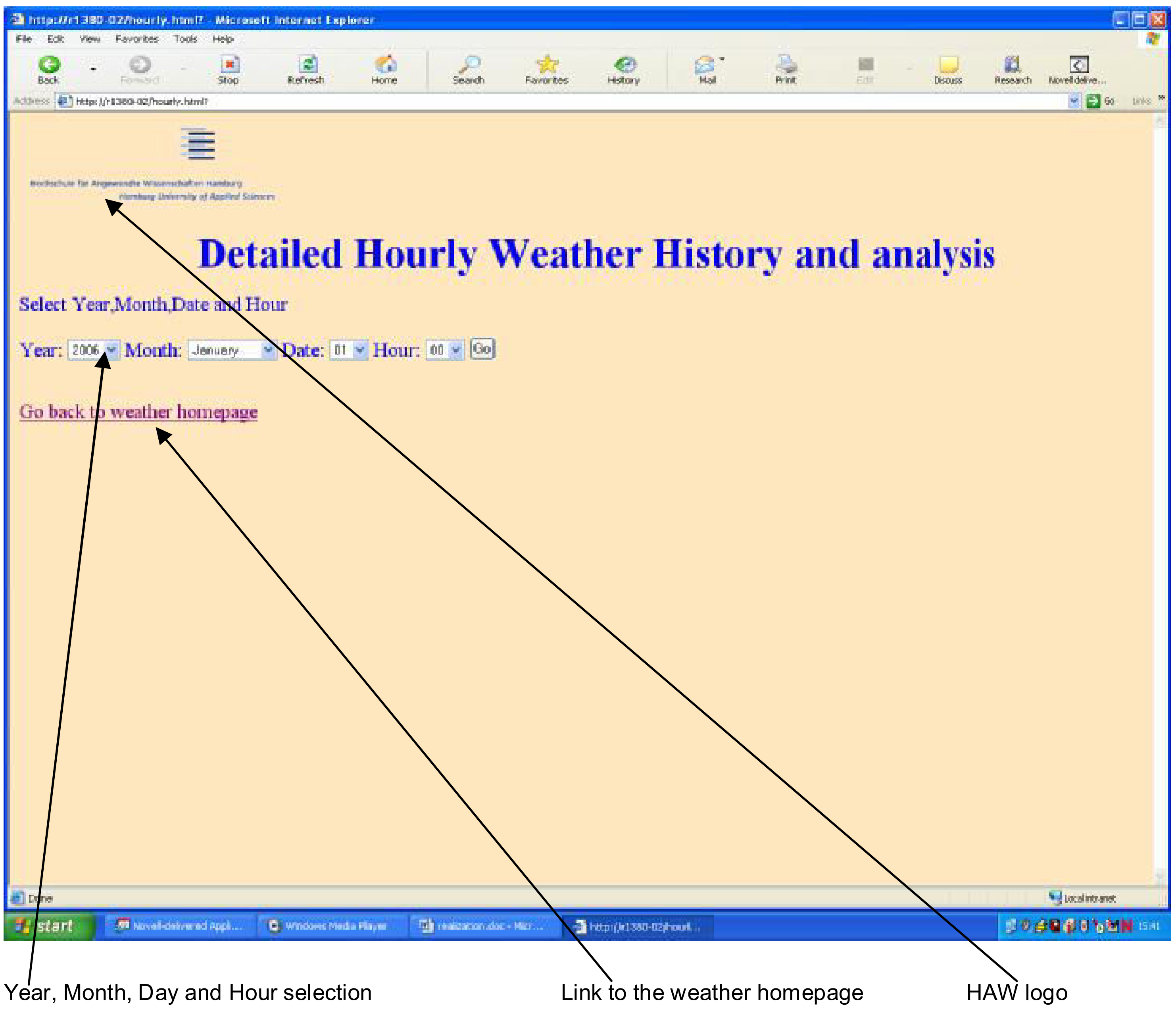


Figure 20: PHP program drawn Barometer

6.2 Weather History Presentation

On selecting and clicking the "hourly" weather history button a HTML form shown below was loaded by execution of the following line of HTML code: *<form action = "http://r1380-02/hourly.html" > <input type ="submit" value = "Hourly"></form></td>*
This means that the file named hourly.html is loaded. This file is actually the HTML form in the figure 21 below.
From this form loaded, the user can select the preferred year, month, date and hour of weather history of interest then click the "Go" button to submit the user selection. Also included in the HTML form is the Hamburg University of Applied Sciences (HAW) logo and the link through which the user could navigate the way to the weather homepage.


Year, Month, Day and Hour selection
Link to the weather homepage
HAW logo

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Detailed Hourly Weather History and analysis

Select Year,Month,Date and Hour

Year: 2006 Month: January Date: 01 Hour: 00 Go

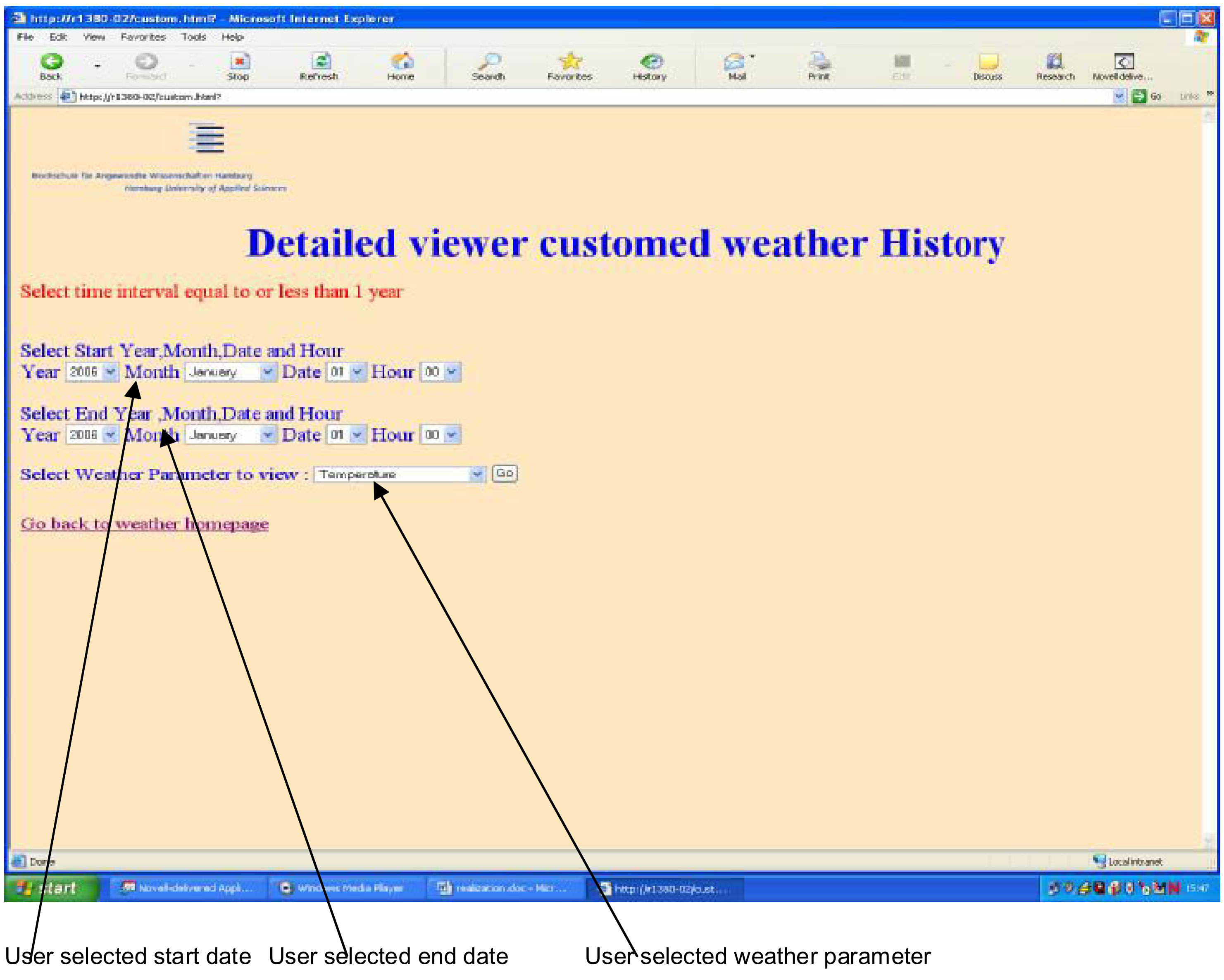
Go back to weather homepage

Figure 21: Hourly weather history selection form

On selecting and clicking the "Daily" weather history button a HTML form shown below was loaded. The HTML was the similar to the one for the hourly weather history except that the user only needs to select the year, month and date of interest. From it the user can then click the "Go" button to submit the user selection.

The monthly, custom and the annual weather history HTML forms are similar to each other but slightly different from the others described before. In this case the user is required to select the weather parameter of interest in addition to the time interval of interest. This is due to the huge amount of data involved and therefore is not possible to the huge amount of data involved. It's not possible to view all the weather parameters at the same time. Outputting of the data for the annual weather history can take almost 45 seconds. On the annual HTML form, the user is alerted about the expected delay in the presentation of graph and table for the data with the message "Due to the huge amount of data involved, processing and presentation may take about 45 seconds. Thank you for your patience".

The user custom HTML form requires additional input of the end time for the time interval the user wishes to view the weather history. This can be seen on the figure 22 below:


http://r1380-02/custom.html? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss Research Novell delive...
Address http://r1380-02/custom.html?
Go
Links
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences
Done
Local intranet
start

Detailed viewer customed weather History

Select time interval equal to or less than 1 year

Select Start Year,Month,Date and Hour
Year 2006 Month January Date 01 Hour 00

Select End Year ,Month,Date and Hour
Year 2006 Month January Date 01 Hour 00

Select Weather Parameter to view : Temperature Go

Go back to weather homepage

User selected start date
User selected end date
User selected weather parameter

Figure 22: User custom weather history selection form

HTML output

On clicking the submit button in the hourly history HTML form the weather data for the ten different weather parameters is displayed in form of line graphs. Behind the scenes the following code line was executed:

<form action = "http://r1380-02/hourly.php" method = "GET">

This applied to the hourly weather history. The HTTP function GET is used to send user input to a PHP script.

```
  $year= $_GET["Year"];
  $month= $_GET["Month"];
  $date= $_GET["Date"];
```

The variables received are assigned to local variables $year, $month and $date. The PHP script *hourly.php* receives the user inputs and validates them to guard against users selecting dates when data was not available like a future date not reached yet.

In order to implement the solution to the problem described above a user input validation code was implemented. All the local variables received from the user input referring to time are appended as string format into one continuous string as shown below.

```
$minsec = ':00:00.000';
    $space =' ';
    $mystime =
($date.$space.$month.$space.$year.$space.$hour.$minsec);
```

The above variable *$mystime* is a string and it represents a time format that can be used by the PHP function *strtotime()* to convert the time inputted in the above string format to unixtime[18]. This PHP function receives a string argument of time in the format : “date month year hour:minute:second.millisecond”.

The date and time when the data was first available on the database is on “March 30, 2006, 5:07 pm”. This is then converted using the above method to UNIX time. The two UNIX times (user inputted time and the first date when the data was available in UNIX format) are compared. If the user input time is less than first available data date then the user is asked to input dates after “March 30, 2006, 5:07 pm” and is redirected back to the user input HTML form. The same principle is used to check if the dates selected by the user are a future date meaning that no data is available. In this case the user selected date is compared with the present date which can be obtained in UNIX time format using the function as shown below:

```
 $unixtime = time();
```

The time formatted to UNIX time format above was converted to this format using the PHP function:

```
$start = date("F d, Y, G:i :s",$stime );
```

Where $stime is the time in UNIX time to be converted to the required format. This format is the same time format as the time format used in the weather database table. This is the column *namefldRecordedAt.*

Once the user input time was converted to the required format it was then passed to the next PHP script using a session as shown below in the code snippet:

```
session_start();
$_SESSION['hstart']= $start;
```

```
$_SESSION['hend']= $end;
$_SESSION['displaydatetime']= $displaydatetime;
```
It was then retrieved by the second PHP script and stored locally as a local variable as shown below:
```
session_start();
 $start= $_SESSION['hstart'];
 $end= $_SESSION['hend'];
```
These two local variables were used to formulate an SQL query $q that was used to fetch the data from the database as shown below:
```
$q = "select fldRecordedAt,fldBarometer,(SELECT MINUTE( fldRecordedAt))
from dba.tblRawData where fldRecordedAt BETWEEN '$start' and '$end'";
```

A connection was then established to the database as shown in the function below

```
$conn = sqlanywhere_connect( "UID=;PWD=;eng=;dbn=;links=")
```

The arguments and the results returned have been explained in the design chapter. Once a query string and connection were established, the query was executed and the result was a set of rows and columns called result set.

```
$result = sqlanywhere_query( $conn, $q )
```

After query execution, the result was then fetched as

```
while( ($row = sqlanywhere_fetch_row( $result )))
```
The obtained result set from the above function was then assigned to locally defined array variables as shown in the code snippet below:
```
while( (row = sqlanywhere_fetch_row( result set ))) {
            $ curr_row++
              $hdata[$i]= $row[o];
              $xaxis[$i]=$row[1];
 }
```
In this case the array $hdata contained number of elements that is equal to the time unit describing the required weather history. For example for daily weather history $hdata contained the hourly average. So there were 24 elements representing the average for each hour.

The interval mean for the respective selected weather history way calculated. For the hourly weather history it was calculated as follows using the code snippet below:

```
$htotal += $row[1]; //adds each parameter data

if(($minute[$i])==0){ //checks if current minute counter is 60
$hdata[$j]=($htotal/(count($minute))); //calculates hourly mean and //mean
assigned to jth element of the local array
              $j++; // and j incremented
              $htotal = 0; // total is reset
              $minute = array(); //minute array is reset to zero count

              }
```

Each weather parameter has its own graph. In all cases for the weather history and the weather parameters line plots on the graph showing the smoothed data was plotted with marked data points. In addition to this, the actual data point plot, mean and the 1SD band range are also plotted in the same graph. The mean was calculated by summing all the elements of the parameter data array then dividing by the number of non null elements .In this case an array was defined whose elements were the mean of the array. This was a constant valued element array.

Variance in the web data analysis was calculated by first calculating the sum of squares of each element of the population. That is each element the array with the original data $ydata is subtracted from the mean and the difference is squared as shown in the code snippet below.

```
    $sumOfSquares =0;
     for($i = 0; $i < count($ydata); $i++ ){
    if($ydata[$i]!="x") //check if the element is null:if not proceed
    $sumOfSquares += pow(($ydata[$i]-$mean[$i]),2)
}
```

Finally the variance was obtained by dividing the sum of squares by the number of elements in the original array. For example the hourly weather history the number of elements was 60 if no data was missing. From the variance the standard deviation was simply computed by getting the square root of the variance thereby giving a constant valued array.

The calculation of the band range was done by defining two arrays namely $upperband and $lowerband. The upperband array was populated by adding one standard deviation to the mean. The lowerband was populated by subtracting one standard deviation from the mean. Below is a code snippet showing how this was achieved:

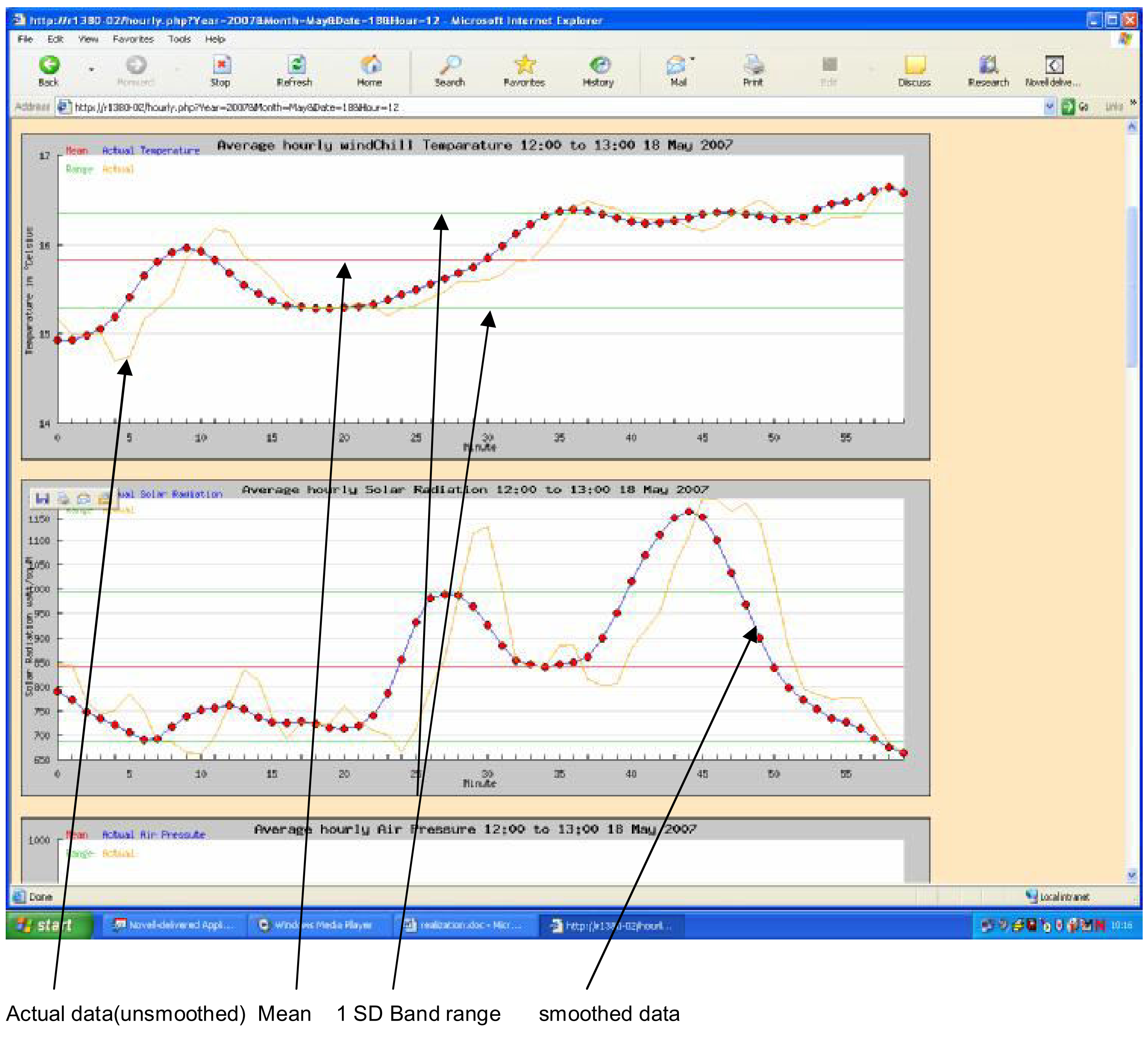
```
//computing the range

for($i=0;$i<count($ydata); $i++){
$upperband[$i]=$mean[$i]+$sd[$i];
$lowerband[$i]=$mean[$i]-$sd[$i];

}
```

In order to realize smoothed data in this project, the moving average technique was used to smooth the data. In this case a smoothing factor of six was used. In this case the first six elements of the original data array were summed up and then divided by six to obtain the mean which was assigned to 0^{th} element of the smoothed data array. Then first element to $(1+6)^{th}$ element are summed up as well and divided by six to obtain the first element of the smoothed data array. The same procedure is used to obtain the next elements of the smoothed array up to
(number of items in original data array called $ydata - smoothing factor)th element. At this point the remaining elements are averaged and the average is assigned to the (number of items in $ydata - $factor)th element of the smoothed array. Each of the remaining elements of smoothed array is obtained by summing successive elements, each time beginning from the (present element+1) then dividing by the number of elements summed up.

For the hourly weather history each data mark represents one minute. That means each of the graphs has a display range of 60 minutes. In the daily weather history each data mark represents one hour. Therefore there were a total of 24 data marks representing the number of hours in a day. At the top of each graph there is a heading indicating the weather parameter under which the graph is plotted from and the time interval of the graph. The hourly and the daily outputs are realized as shown in figure 23 and figure 24 respectively as shown below. In order to realize the hourly chart below in the user selected the year to be 2007, month to May, date to 18th and hour to 12. The output was 10 graphs representing each of the weather parameter as it can be seen from the figure 23 below. In each graph there was the actual data plotted in orange, the smoothed data plotted in blue with red marks each representing a data point. The mean was also plotted in red which was actually a straight line. The 1 SD sigma bands were also plotted in green.



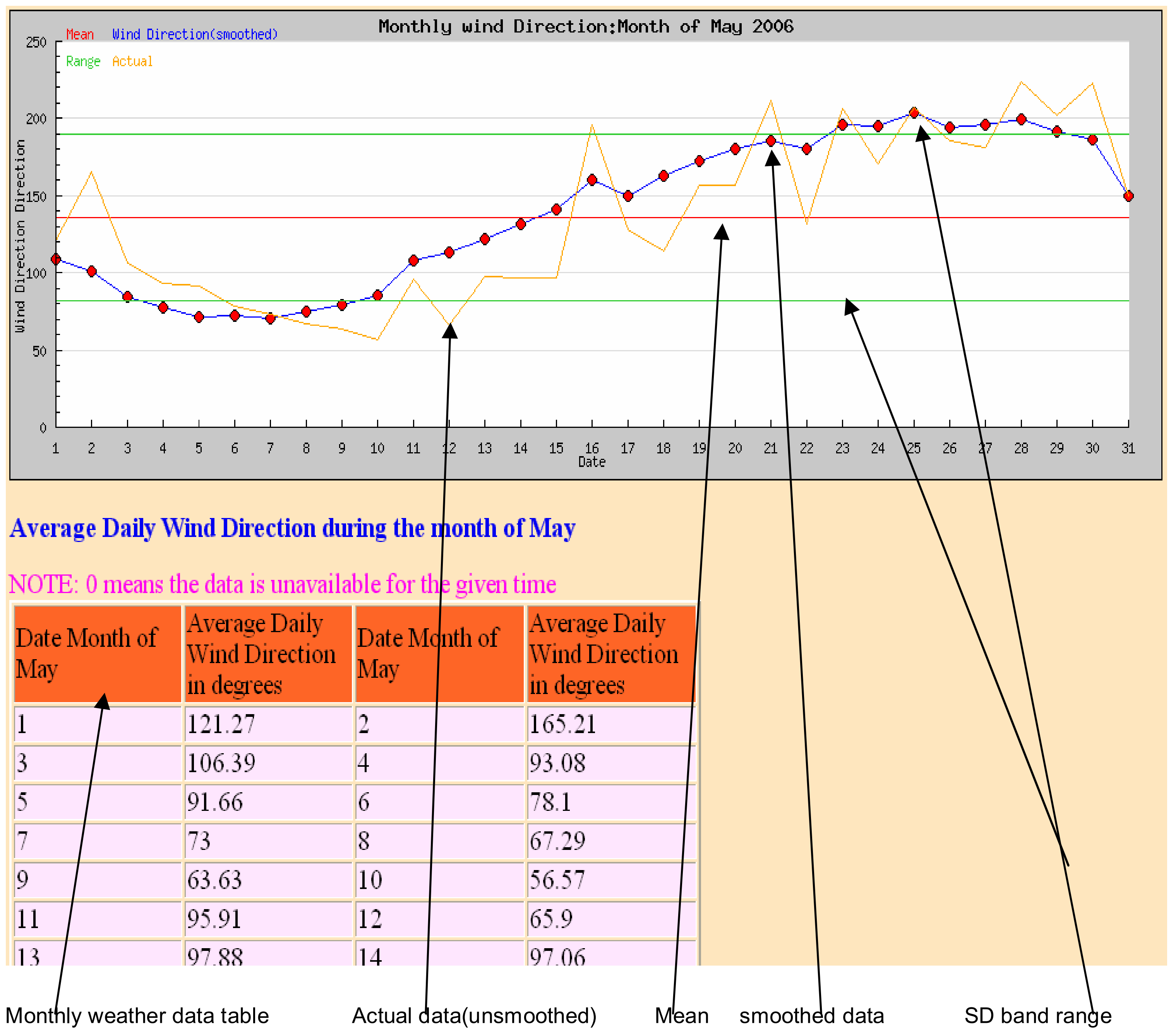
http://r1380-02/hourly.php?Year=2007&Month=May&Date=18&Hour=12 - Microsoft Internet Explorer
Average hourly windChill Temparature 12:00 to 13:00 18 May 2007
Average hourly Solar Radiation 12:00 to 13:00 18 May 2007
Average hourly Air Pressure 12:00 to 13:00 18 May 2007
Minute
Actual data(unsmoothed)
Mean
1 SD Band range
smoothed data

Figure 23: Average hourly history for the hour 18th May 2007 12:00 to 13:00 pm

The output of the monthly and the annual weather history is similar. As observed earlier in the HTML for the monthly and annual weather histories the user is required to select the weather parameter of interest. So in the output there is only one graph showing the smoothed data of the weather parameter that is selected in the HTML form. Just like in the daily graph, there is the actual data plot, mean and one standard deviation band range.

The look of the annual and monthly graph is similar to the daily and hourly graphs discussed earlier. The only difference is that in the monthly graph each data point represents the daily data mean for that particular parameter. So the number of data points plotted depended on the number of days in the selected month. For the annual graph each of the data point represents the mean monthly data for the selected parameter. For example in the year 2006 the data is displayed from April when the data was first available to December. In addition to the graph there was a table showing the actual data for each day or month for monthly and annual data respectively.

The output for the monthly weather history is shown in the figure 24 below. For the monthly history, the weather parameter selected is “Wind direction”, month of “May” and “Year” 2006 are selected yielding the figure 24 below: The annual weather history output looks similar to monthly output except that the data points are months based. That means on each table and graph there is data for every month for that year. In case of 2006.The output will be tabular and graphical presentation for monthly data beginning from April to December. Due to the similarity of the annual and monthly outputs only the monthly output is shown here on figure 24 below:



Average Daily Wind Direction during the month of May

NOTE: 0 means the data is unavailable for the given time

Date Month of May	Average Daily Wind Direction in degrees	Date Month of May	Average Daily Wind Direction in degrees
1	121.27	2	165.21
3	106.39	4	93.08
5	91.66	6	78.1
7	73	8	67.29
9	63.63	10	56.57
11	95.91	12	65.9
13	97.88	14	97.06

Figure 24: Average monthly wind direction for the month May 2006

6.3 Display of the missing data

One of the serious challenges encountered in the process of trying to design the website data presentation was that of the missing data. This is a problem that occurs whenever the data server where the weather data is recorded goes off for some reason. This could be for maintenance purposes, upgrade etc. Whenever this happens, the data captured by the weather station is not stored in the database, as long as the data server is still disconnected from the weather station. This means it's just lost. Once the data server is turned on then the data for the entire period when the server was off is not there. It is a problem in two ways. If a user wishes to view the present weather conditions then the only the latest available data is presented meaning it could be as old as the time when the server went off. It becomes even more of a problem when the user wishes to view weather history of a time interval within which there is a missing data interval.

If the data is to be presented in form of a graph then more work needed to be done in order to realize the graph output described in the previous chapter concerning the missing data. Such data was presented using a graph with a gap on the line where the data was found to be missing as shown in figure 26**.** In designing the presentation such a problem had to be taken into consideration.

In anticipation of such a case the data was first fetched from the database in two columns namely fldRecordedAt and the weather parameter in question. The column name fldRecoredAt refers to the time in

year-month-date-hour-minute-milliseconds that the data was recorded. On fetching each record of data, the time interval that the user needs to view the weather history in is extracted. For example if the user needs to view the monthly weather history then individual time measurement components of the fldRecordedAt are extracted and stored in an array in the same position as the weather parameter corresponding to that date. Example:

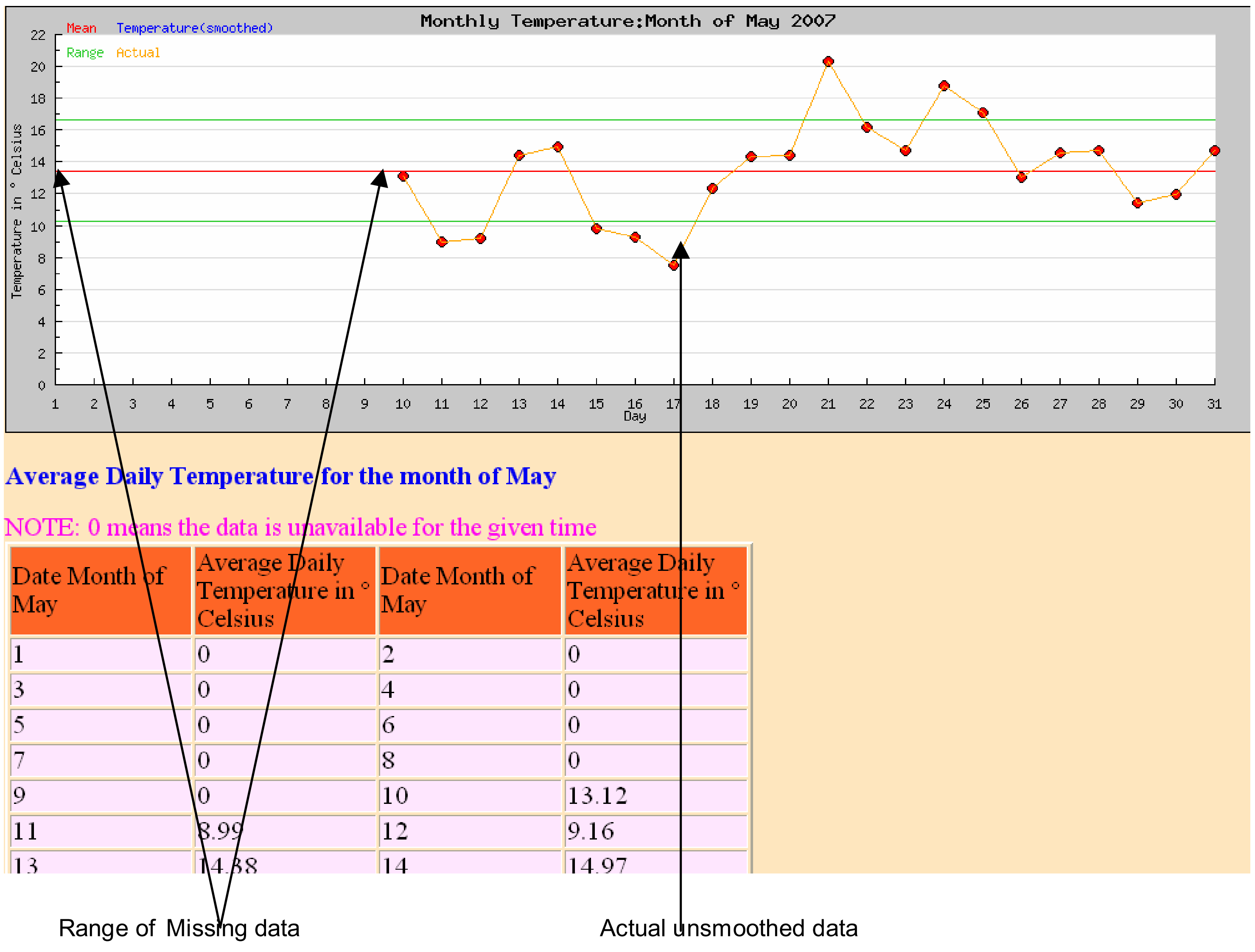
$q = "select fldRecordedAt,fldSolarRadiation,(SELECT HOUR(fldRecordedAt)),(SELECT MINUTE(fldRecordedAt)),(SELECT DAY(fldRecordedAt)),(SELECT MONTH(fldRecordedAt)) from dba.tblRawData where fldRecordedAt BETWEEN '$start' and '$end'";

$temp[$i]=$row[4]; //assign date component to a local array

$tempe[$i]=$row[1]; / assign parameter(solar radiation) to a local array

In the above case it means that array $temp will hold the date values of all the time data fetched. The weather parameter solar radiation was assigned to a local array $tempe. Both arrays have equal lengths and each element in one corresponds to the element in the same position in the other. Every element of the date array is compared with the previous element. If the two elements are the same then the next element of the weather parameter array like solar radiation is added to a local variable called $htotal. If the two consecutive elements of the date array differs by 1 then it is recognised that a the corresponding data belongs to a new day and so the present $htotal is averaged(divided by the number of non null elements for that day) and stored in an array called $hdata. If the two consecutive array elements differ by more than one then missing dates are recognised. First the number of missing dates are determined and next elements representing missing days are assigned the value “”. In the date array the next element becomes (the present date element +1). The same procedure is used to test every element until the end of the array is reached. At this point the values of $hdata array are presented graphically or in tabular format. The special handling of the missing data is described with the help of a flow chart diagram as it can be seen on figure 26.

There were several cases where the data was found to be missing for a given interval .In such a case the part of the time interval with missing data in the graph is represented by a gap. A good example is the month of May 2007, where there was no data that was recorded in the database from 1st to 9th day of May. On the corresponding table, missing data is represented as “0”. Mean and 1 Standard Deviation band range are computed from the existing non zero data. Below is the output derived by selecting to view the monthly temperature for the month of May 2007 on figure 25.



Average Daily Temperature for the month of May

NOTE: 0 means the data is unavailable for the given time

Date Month of May	Average Daily Temperature in ° Celsius	Date Month of May	Average Daily Temperature in ° Celsius
1	0	2	0
3	0	4	0
5	0	6	0
7	0	8	0
9	0	10	13.12
11	8.99	12	9.16
13	14.38	14	14.97

Figure 25: May 2007 Output with null values

The whole process of how the missing or the null values were handled can best be described using a flowchart as shown in figure 26 on the next page. This basically described how a monthly data with some days with missing values was handled. The handling for the other intervals was dealt with in the same way .

tempe[] :stores the weather data
temp[]:stores the day element of the date
hdata[]: stores the mean data points to be plotted

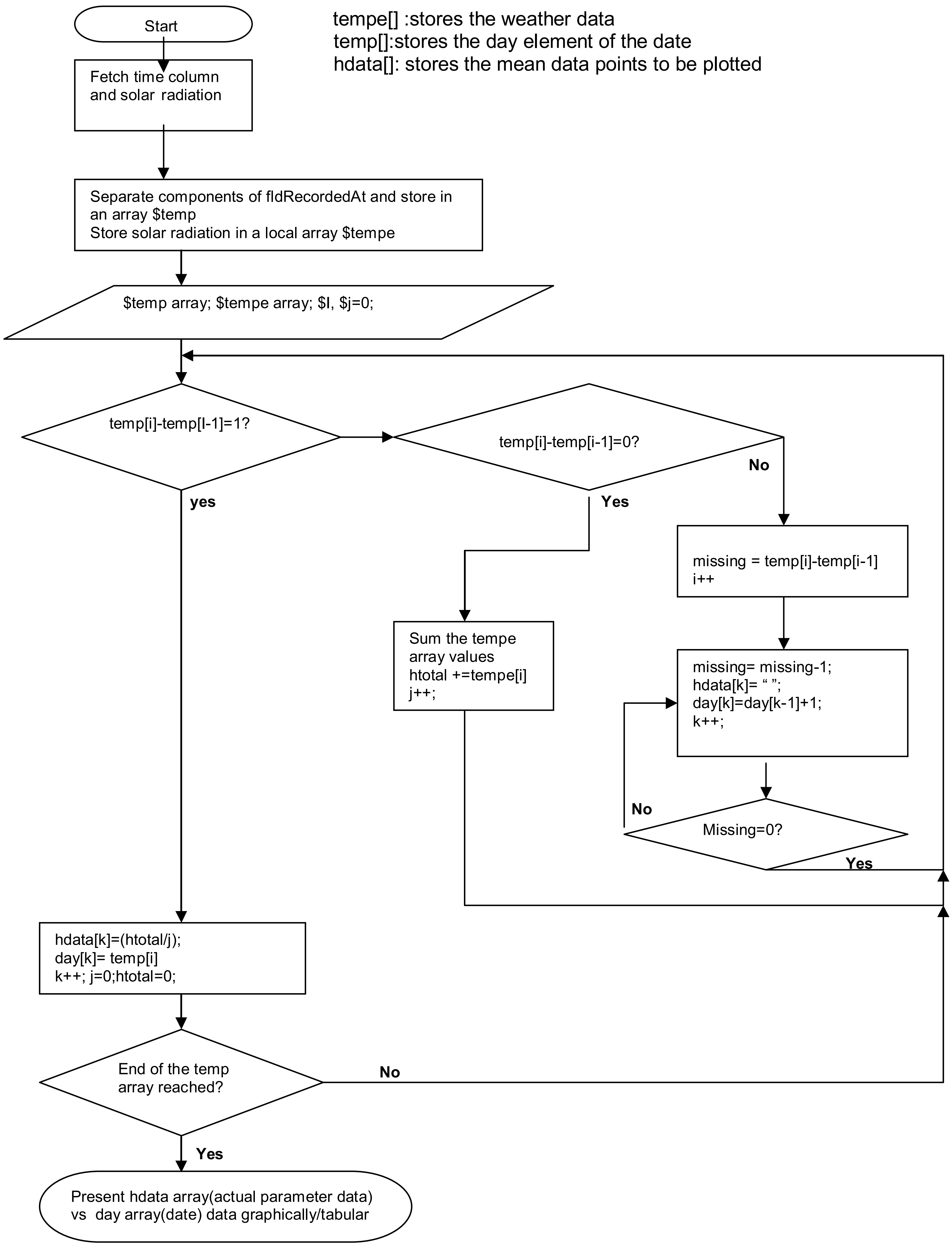
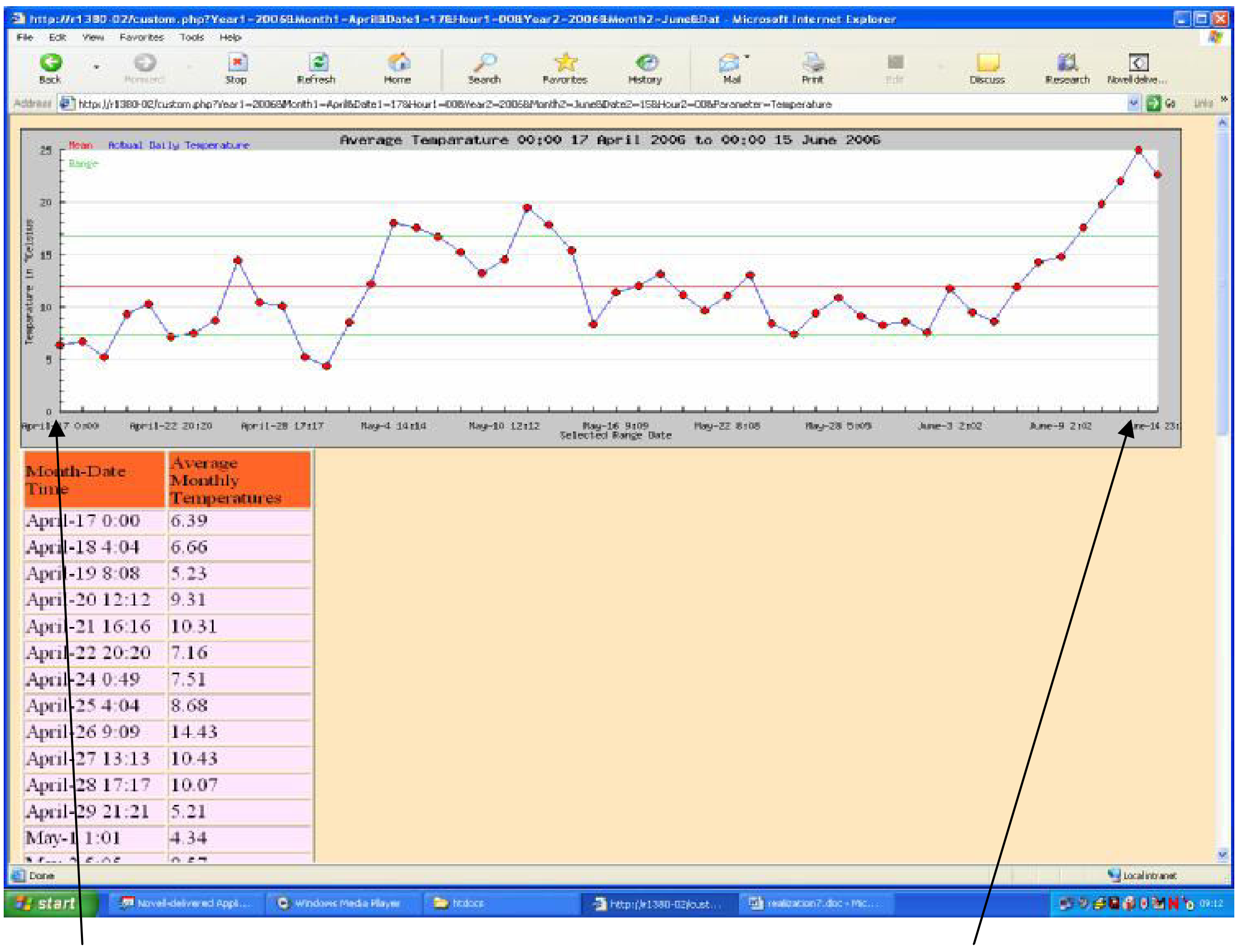

Start
Fetch time column and solar radiation
Separate components of fldRecordedAt and store in an array $temp
Store solar radiation in a local array $tempe
$temp array; $tempe array; $I, $j=0;
temp[i]-temp[I-1]=1?
temp[i]-temp[i-1]=0?
No
yes
Yes
missing = temp[i]-temp[i-1]
i++
Sum the tempe array values
htotal +=tempe[i]
j++;
missing= missing-1;
hdata[k]= " ";
day[k]=day[k-1]+1;
k++;
No
Missing=0?
Yes
hdata[k]=(htotal/j);
day[k]= temp[i]
k++; j=0;htotal=0;
End of the temp array reached?
No
Yes
Present hdata array(actual parameter data) vs day array(date) data graphically/tabular

Figure 26: Handling of null values in monthly weather history

Custom weather history output is realized with a 50 data point graph and a table consisting of 50 rows. The 50 data points were evenly spaced in terms of time separating them. However the labelling of the x-axis is done after every five data points. This helps to make the time axis readable. Only data from one weather parameter as selected by the user is plotted and tabulated. The code snippet below describes how calculation of the 50 data points was done:

```
$interval=(count($tempe)/50);// intervals which to calculate a mean
for($k=0;$k<count($tempe);$k++){
        if(($k % $interval)==0){//mark the end of each interval
$axis[$i]=$month[$k].'-'.$timeDay[$k].' '. $timeHr[$k].':'.$timeMin[$k]; // time
label to appear on x axis
            $hdata[$i]= ($htotal/$interval);//calculate mean for each interval
            $htotal = 0;//reset the total
            $i++;}
    else
        $htotal += $tempe[$k]; // add data
```

On plotting this data against the time axis the following graph on figure 27 is realized. In the user custom graph below the start date selected is 17-04-2006 00:00 and end date selected is 15-06-2006 00:00


http://r1380-02/custom.php?Year1=2006&Month1=April&Date1=17&Hour1=00&Year2=2006&Month2=June&Dat - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss Research Novell delive...
Address http://r1380-02/custom.php?Year1=2006&Month1=April&Date1=17&Hour1=00&Year2=2006&Month2=June&Date2=15&Hour2=00&Parameter=Temperature
Average Temparature 00:00 17 April 2006 to 00:00 15 June 2006
Mean
Actual Daily Temperature
Range
Temperature in °Celsius
25
20
15
10
5
0
April-17 0:00
April-22 20:20
April-28 17:17
May-4 14:14
May-10 12:12
May-16 9:09
Selected Range Date
May-22 8:08
May-28 5:05
June-3 2:02
June-9 2:02
June-14 23:
Done
start
User selected start date
User selected end date

Month-Date Time	Average Monthly Temperatures
April-17 0:00	6.39
April-18 4:04	6.66
April-19 8:08	5.23
April-20 12:12	9.31
April-21 16:16	10.31
April-22 20:20	7.16
April-24 0:49	7.51
April-25 4:04	8.68
April-26 9:09	14.43
April-27 13:13	10.43
April-28 17:17	10.07
April-29 21:21	5.21
May-1 1:01	4.34

Figure 27: Custom history output between 17-04-2006 00:00 to 15-06-2006 00:00

7 Testing and major challenges encountered

In the web based data analysis and presentation project a few difficulties were encountered in design and realization. Some were fully overcome and others were not.

7.1 Testing

Testing of the entire website was actually done by three independent people at various times. Each of the tester checked a different aspect of the website. In each of the test the tester assumed the role of a user who intended to view a given aspect of the weather from the website. In the first case the tester found that the pictorial images of the weather instruments did not reflect the then current weather conditions. On further consultation about this problem it turned out that the connection from the weather station to the database is based had been turned off. As a result of this the latest data that was available was the last data that was recorded in the database, which were several days old. This was corrected and in this regard the presentation of the present weather readings on the homepage works fine.

Testing of all the links on the homepage was also carried out and was found to be working fine. The homepage contained four buttons that linked the user to the weather history user input form all these were found to be properly working. Other link to HAW Hamburg found to be working fine.

The outputs given back to the user after inputting the preferred time of view was also tested. This test was carried out from other computers on the network and the output turned out to be what was expected. One of the testers selected a time interval with starting date that was before the data was first available. The result of this was a JPGRAPH error, which was not user friendly at all and could not be easily understood by an ordinary user: Therefore a better error handling measure was recommended. An error handling measure was implemented .In this case if the user selected an illegal date and clicks “Submit” then an error message is issued telling the user to select dates “to select dates between 30th March, 2006 and now”. In addition the wrong inputs were not submitted but user referred to the same user input page. The problem of invalid date input was also detected in this test where a user can enter an invalid date such as 31st February. This was solved by having an option select form instead of having a user typed input form. Several recommendations on how to improve the output were proposed. The third test was meant to ensure that everything worked fine. From this test the general performance such as speed of presentation of the past weather situation was assessed. It turned out that for the annual weather history the presentation took longer than acceptable. At the moment it takes around 115 seconds to realize the whole annual output. This will be discussed in the next subchapter. The situation was the same for the user custom weather history. This was however solved by telling the user that the annual weather history may take up to1 minute and 45 seconds. For the User custom weather history the user was restricted to a maximum time period of 6 months.

7.2 Challenges encountered

Delayed output for the Annual and Custom weather history

By the time of designing this project there were more than 460,000 records of all the weather parameter data. Therefore if the user was to request to view the weather history of the entire duration from 1st April 2006, then the number of records to be accessed and the amount of data to be processed is enormous. Assuming that for every one minute a new row of data is added to the table in the database then in one hour we have 60 new records. In one day we have:

60 *24 = 1440 records

1440*365=525,600 records in one year

That means when this data is fetched from the database it takes time to read all this data and avail it to the PHP script for processing. Moreover this data has to be stored in locally defined arrays so as to be processed and means for various intervals calculated. If two data columns such as recorded time and temperature are fetched by a PHP script and stored in local arrays, this will dramatically reduce the amount of memory available for use in further calculations involving this data.

This brings a lot of overhead onto the processor and the processing computers memory. Further more the mean data is then plotted on a line graph. All these factors combined lead to a delayed output of the annual weather history and the user custom weather history where the time interval in more than 6 months. In the worst case it takes about 45 seconds after submitting the user input in HTML form, before the output is seen. In monthly, annual and user custom weather histories it was therefore impossible to view all the parameters at the same time due to the resources required to do this.

One of the measures that are taken in order to minimise the delay effect of this problem was to display the weather history of each weather parameter separately. The user is expected to select which weather parameter is of interest at that particular moment. By so doing the output is presented with out so much delay. In the annual weather history the user is still forewarned that the output would take up to1 minute and 45 seconds and therefore asked to be patient. At the moment it takes 1 minute and 40 seconds to realize the full annual output(table and graph).

In the user custom weather history the user is asked to select a time interval in which to view the weather history of 6 months or less. In case the user still goes ahead and selects a longer time interval then selection is not submitted and a message asking the user to select a time interval less than 6 months is repeated. By so doing it's ensured that the amount of data being calculated and outputted will not cause long unacceptable delays.

Incompatibility between versions of PHP and SQLAnywhere module

As mentioned earlier PHPSQLAnywhere module is a PHP library that has important functions that can be used to connect to a SQLAnywhere database. In all cases every version PHPSQLAnywhere module worked only when installed in a specific version of PHP. It's sometimes a problem because matching versions are not readily available since they are downloaded from different sources. This is especially so for the latest version of PHP (version 2.2).This version of PHP has a lot of useful features but for the purpose of connection to SQLAnywhere database there is no matching PHPSQLAnywhere module that's compatible with it. Furthermore compatibility of PHP and Apache was also version dependent. This was a problem during the setting up of tools needed for the project discussed. This was overcome by downgrading the installations to slightly older versions of each that are compatible in each case.

Recommendations for further improvement of the website

One of the recommendations that would be suggested concerning this problem is distributing the work of calculating the weather data across several processors. At the moment the computer where the web application is based is a 3 GHZ Pentium processor. Distributing this work across two such processors will reduce the delay times considerably. Usage of more memory can also go along way in minimising the problem due to the need to store large arrays. However the most important thing is to strike a balance between the additional hardware requirement and the acceptability of the present delays. Additional hardware also leads to higher cost of implementation and maintenance.

8 Conclusion

Web based weather data presentation is very important task. By presenting the current weather condition of a given place on the web this provides a near to real time update of the weather pattern to the user. Once the data is available on the web in any form that an interested party would want to view it then it opens up different angles of interpretation. Traditionally presentation of the weather data to the external users was mainly done in written form where the weather data is recorded on paper and published in form of printed weather reports. The number of interested users who would be able to access this recorded data is definitely limited. Moreover there was limited way one could interpret the data since it was not available in many different views such as yearly, monthly. With emergence of the World Wide Web and dynamic web pages technologies like the one discussed though out this thesis this has become possible. The user can now view weather data that is as old as one minute ago. The web based weather data analysis provides groundwork for weather forecasting. The observed data is one of the factors used in the weather forecasting.

In the web based weather data presentation and analysis the realization includes a homepage which presents the actual weather situation. In addition it contains the links to other web pages that present the past weather situation in form of y-x graphs and tables in some cases. The data that's presented in both cases originates from the weather station and is transmitted through a serial interface after processing to a database in the data server where it's stored. From the database the data is fetched and used in the programs that realize the weather website. The program reads the data from the database compresses the data for easier presentation and viewing. In addition in the graphs the mean and standard deviation is calculated and shown and the data are shown in form of tables.

9 References

Hugh E. Williams and David Lane. Web Database applications with PHP and MySQL, O'REILLY, Köln 2002

Flanagan, D., Java in a Nutshell, A Desktop Quick Reference, O'REILLY, ISBN 0-596-00283-1, 2002, 4th edition.

Arnold, K., Gosling, J., Holmes, D., The Java Programming Language Third Edition, Addisob- Wesley ISBN 02017044331, 2001.

Yalcin, C., Zukunft, O.,Raasch, J., Software Engineering Konzepte in PHP: eine Untersuchung, Hamburg, 2006.

http://en.wikipedia.org/wiki/Three-tier_(computing)
(Three tier Architecture) 25.04.2007

http://www.reinhardt-testsystem.de/PRAESW13.HTM
(MWS 6 weather station) 24.04.2007

http://groups.google.com/group/sybase.public.sqlanywhere.general/browse_thread/thread/d729939ecbdcd9d5/20cbc5ee65fcfd89?lnk=raot
(SQLAnywhere- PHP connection) 21.02.2007

http://www.carbonblock.net/docs/jpgraph/html/manual_jpgraph.html
(JPGraph) 14.04.2007

http://en.wikipedia.org/wiki/Web_server
(About webserver) 26.04.2007

http://www.ii.uam.es/~saiz/webnet98-paper.html
(dynamic web pages) 26.04.2007

http://www.mediavue.net/programming/programLanguage/jsp_programming.html
(About Java Server pages) 27:04:2007

http://www.articlesbase.com/programming-articles/advantages-of-php-development-71741.html
(Advantages of PHP) 29.04.2007

http://en.wikipedia.org/wiki/JavaServer_Pages
(About Java Server pages) 30.04.2007

http://builder.com.com/5100-6371-1044979.html
(Dynamic web pages) 30.04.2007

http://developers.sun.com/portalserver/reference/techart/jfreechart.html
(Plotting charts using Java) 30.04.2007

http://www.tizag.com/phpT/phpsessions.php
(PHP sessions) 03.05.2007

http://www.ianywhere.com/developer/product_manuals/sqlanywhere/0902/en/html/dbpgen9/00000614.htm
(Establishing a connection to an SQLAnywhere database) 07.05.2007

http://sqlzoo.net/fun_day
(SQL date functions) 07.05.2007

http://www.ianywhere.com/developer/product_manuals/sqlanywhere/9.0/php/html/query.html
(Querying SQLAnywhere database from PHP) 07.05.2007

http://www.vanguardsw.com/DpHelp4/dph00109.htm
(Moving Average) 07.05.2007

http://en.wikipedia.org/wiki/Arithmetic_mean
(Calculating arithmetic mean) 07.05.2007

http://en.wikipedia.org/wiki/Apache_HTTP_Server
(Apache Server) 10.05.2007

http://www.expertsrt.com/tutorials/Matt/install-apache.html#dirsetup
(Apache installation) 29.05.2007
http://mathcentral.uregina.ca/qq/database/QQ.09.98/fama1.html
(Standard deviation) 29.05.2007

http://en.wikipedia.org/wiki/Database_management_system
(Database management Systems) 30.05.2007

http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1000/en/html/dbpgen10/pg-php-s-4387280.html
(Installation of SQLAnywhere PHP modules) 30.06.2007

http://ourworld.compuserve.com/homepages/Gene_Nygaard/windchil.htm
(Calculation of wind Chill factor) 01.06.2007

http://www.nwas.org/committees/avnwinterwx/winter_teach2.htm
(Wind parameters explanation) 01.06.2007

http://www.velocityreviews.com/forums/t138563-jsp-and-php.html
(Advantages of JSP) 04.06.2007

10 Appendix

This Bachelor Thesis contains an appendix of program listings, configuration files for PHP and Apache, on a CD. The appendix also contains a text file named *A_Description_of _code_files.txt* .This file basically explains the purpose for which each of the source code and configuration files, and how they were used.
This Appendix is deposited with Prof. Dr.rer.nat Hans-Jürgen Hotop.

Declaration

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, 7th June 2007, Anthony Kamau