Touch Screen Kiosk

at

JOOLA Table Tennis Club.

High Level Architecture.

Author: Sergey Satskiy

Table of Contents

1. Introduction

The document describes high level architecture.

2. Overall Architecture

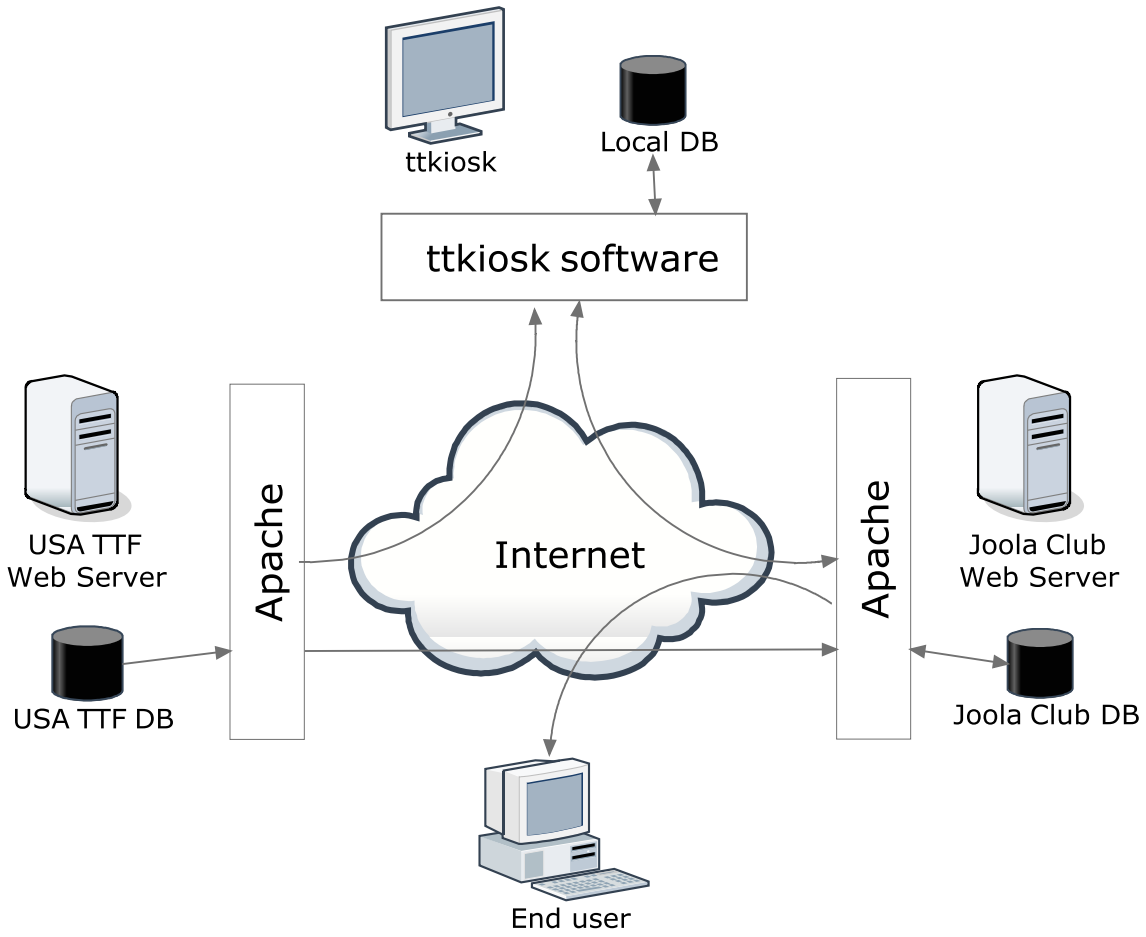

Figure 1 Overall Architecture

There are four main parts involved:

- USA TTF web site where a database of the USA TTF members is located. This database holds the players ratings, history of played events, matches and games as well as some personal information. The USA TTF database is always a source of data and is never updated by the ttkiosk software.
- Joola club web site where the ttkiosk main database is located. This database holds the joola club players who are not mandatory members of USA TTF. The database also stores history of played events at the club, matches and games as well as some players personal players

information. The database may also keep a copy of some USA TTF database for faster access and reducing the load on the USA TTF web site. The data can be updated from three sources. The ttkiosk can update data when a game result is entered, when a new player is registered, when a new event is created or when something is corrected. The end users can update their personal information. The USA TTF database can be updated and the cached data can be updated.

- The end user i.e. a Joola club member or a site visitor may look through the history of games of all the registered players. The Joola club members can also update their personal information.
- The ttkiosk part is serving events at the club. New players can be created, new events can be created and the ttkiosk provides ability to enter and look through the data. All new data are stored on the Joola club web site database. The ttkiosk can have some portion of data from the Joola club web site database cached at a local database. It increases interactivity of the software.

3. ttkiosk Architecture

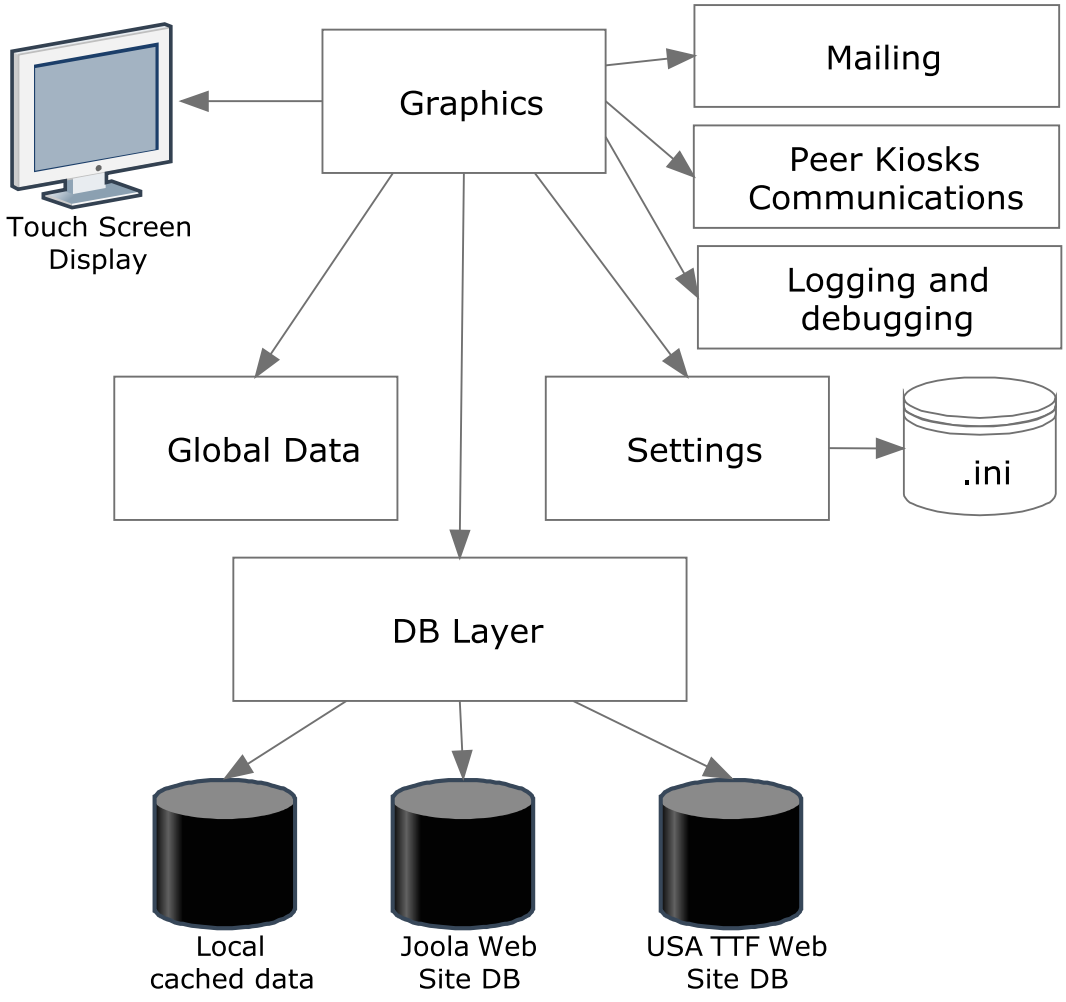

Figure 2 ttkiosk Architecture

The ttkiosk software is going to be developed in Python programming language and to run under Linux OS. The X Server on the ttkiosk host is not going to have any window manager. All the application forms will not be resizable by the kiosk users.

The graphics part is going to use PyQT library which is essentially wrappers around the corresponding classes and functions of the multiplatform QT library. So, pretty much everything available in the QT library will be available for the graphics part. An important feature to mention is QT style sheets for forms control elements.

Global data is a singleton which is available at any ttkiosk module. It holds important application wide data like a reference to the QT application object, a list of startup forms, the current mode of the kiosk (administrative or user) etc. The global data members are available for updating and for reading.

Settings are also implemented as a singleton. The settings are read from an ini file and available for reading only. The singleton is available at any ttkiosk module. The settings are read from the ini file at the moment of the object creation.

Mailing is implemented as a set of functions which rely on a standard Python module. To send an e-mail a google account is used.

The peer kiosks communications are intended to work over LAN so UDP protocol is used. The QT support of network communications is used for this because it is tightly integrated with a graphics application main loop

The support of debugging and logging is implemented as a set of functions and a screen form. The functions can save a message in a log file and the screen form can display a message on the screen.

3.1 Graphics Part Architecture

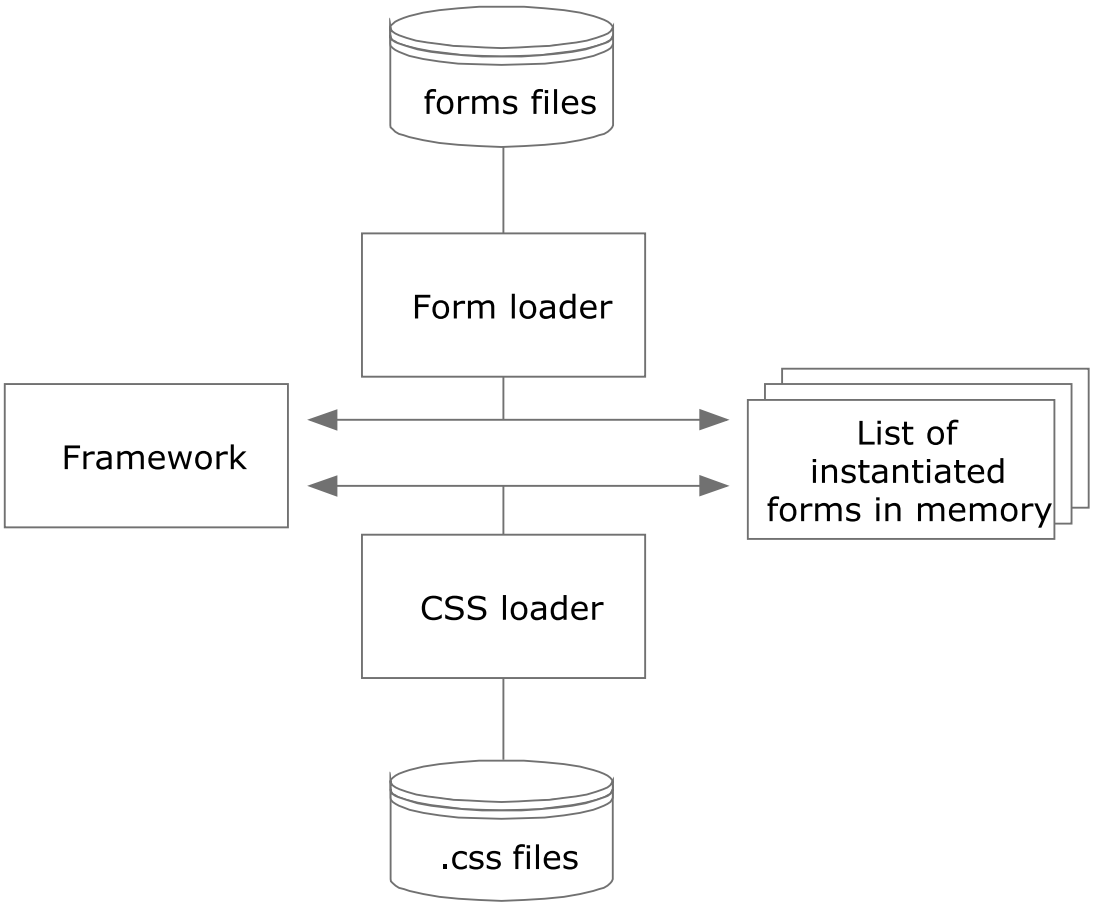

Figure 3 Graphics Part Architecture

The graphics part relies on some agreements. The most important of them are as follows:

- Each screen form has a unique text identifier.
- The form identifier matches the corresponding widget class.
- Each form widget class resides in a single file and the file name follows the rule <FormIdentifier>.py.
- Each form widget class file must hold a single widget class definition.
- It is supported that each form has its own CSS file specific for a skin. A CSS file name must follow the rule <FormIdentifier>.css.

So, the general idea of the graphics framework is to instantiate all the forms in memory, apply CSS to forms and provide ability to show or hide them by referencing a certain form by name.

The framework starts from looking for the form files in the directory which name depends on a screen resolution. This allows supporting many different screen resolutions. The framework searches for all the .py files and instantiates widget classes from them. Subdirectories are supported. After this stage a map between form identifiers and instantiated classes is available.

Then the framework is looking for all the CSS files in the directory which name depends on a current skin name. Subdirectories are supported as well. The syntax of the CSS files matches the QT CSS syntax with a few additions: remarks are supported and the INCLUDE statement is supported. A special file application.css is searched first of all. If it is found then the CSS is applied on the application level. Then the rest of found CSS is applied to the corresponding forms.

The last step the framework does at the beginning is that it searches for the layout.ini file in the forms classes directory. The layout.ini file holds a set of GEOMETRY statements which define the size and location of each screen form. The file must also have at least one STARTUP statement which define forms comprising the start screen. Having this information the geometry is applied to forms and startup forms are shown.

The framework provides as well a few functions to switch on and off forms. It is supported to change forms location and size. Those functions are intended to be used by other forms when it is necessary to change the set of forms on the screen. It is also supported to save a state of the screen in a stack and then get back to the previous state.

3.2 DB Layer Architecture

TBD

3.3 Global Data

At the time of writing the global data singleton stores the data shown in the table below.

Field	Description
isAdmin	True if it is currently an administrative mode
screenWidth	Screen width in pixels
screenHeight	Screen height in pixels
startupForms	List of form names which comprise the start screen
application	A reference to the QT application

The list of fields in the singleton can be extended later.

3.4 Settings

The settings are coming from the ttkiosk.ini file. They are not modifiable for the ttkiosk software and if the corresponding values are changed in the file, the ttkiosk application must be restarted to pick up the new values. The format of the file is an industry standard ini files format with support of groups and values.

The table below shows what settings are supported at the time of writing.

Group	Value	Description
[mail]		
	smtpUser	User name to connect to SMTP server
	smtpPassword	Password to connect to SMTP server
	smtpServer	SMTP server address
	smtpPort	SMTP server port
	from	Address which appears in the <from> field
	errorRecipient	List of e-mail addresses where an error report is delivered
[path]		
	logs	Path to the directory where log files are stored
	videos	Path to the directory where videos are searched for to show them when ttkiosk is in idle state
	slides	Path to the directory where pictures are searched for to show them when ttkiosk is in idle state
	templates	Path to the directory where e-mail templates are searched when a certain e-mail notification is going to be sent.
	skins	Path to the directory where skins subdirectories are located
[general]		
	skin	The name of the current skin
[timeout]		

	idle	Inactivity timeout in seconds which triggers a transition to the idle state

The list of settings can be extended later.

3.5 Mailing

Mailing to certain recipients is used in a few major cases:

- When a player has to be notified about a certain event similar to challenging in certain type of events.
- When a player wants to receive a copy of an event results in his mail box.
- When an unhandled exception has happened an e-mail will be sent to the developers.

Mailing is implemented as a set of functions and relies on a standard Python library. The code connects to a google mail box which is used as a relay to forward an e-mail to the recipient.

3.6 Peer Kiosks Communications

When some information is entered on one kiosk all the other kiosks has to update their displayed information correspondingly. The Joola club LAN is not going to be heavy loaded so the UDP broadcast messages are used for notifications. Each UDP notification packet payload includes the information about what was updated. It may also include a new value.

The information about what exactly was updated is transferred in a form of hierarchy identifier. For example as soon each form has a unique identifier and each form field can be uniquely identified as well an identifier may look as follows: <formIdentifier>.<fieldIdentifier>.

A non mandatory part of a notification package may include new values as strings in a form of semicolon separated list of pairs: <identifier>=<value>.

3.7 Logging and Debugging

Logging is supported by a single function which decorates the given message with a timestamp and saves in a log file. The location of log file is given to the system logratate mechanism which provides automatic archiving and optionally e-mailing of logs.

Debugging is supported by a single function and a special form. If the form 'DebugBar' is shown on the screen then the last 300 debug messages are displayed there. The debug mode must also be switched on by a command line option key or by clicking a certain button on an administrative panel.

4. Joola Club Web Site Architecture