**CIS 470 Design Specification**

|  |  |
| --- | --- |
| **Team:** | **Team F** |
| **Team Members:** | **Norman Brandon, Khanh Nguyen, Nathan Strawhand, Robert Descrescentis** |
| **Date:** | **1/21/2018** |
| **Project Title** | **TPS eCommerce Web Site** |

# Scope

Taylor’s Professional Services, or TPS for short, is looking to provide a web site that will allow their clients to complete a staffing request over the internet, allow staff members to update their credentials, and allow managers to have ease of access in managing clients, staff, and approving staff information. The goal is to have a working model up and running by the end of February, 2018 for review.

## System Description

Taylor’s Professional Services is a technical and engineering staffing service. When a TPS client company determines that it will need a temporary professional or scientific employee, it issues a staffing request against the contract it previously negotiated with TPS.

TPS wants to provide a web site so that their clients can complete a staffing request over the internet. In addition, TPS wants to provide their clients with a list of potential candidates based on experience, education, salary, and location. A client will be able to select up to three potential staff members along with the location of work, type of work, and salary and submit the request to the contract manager. Once a client issues a staffing request, the system shall provide an automated response stating that the contract manager will validate their request within 24 hours of receipt.

Once a staffing request has been issued, the client will be able to log into the site and search for a staff request by number. The staff request query will result in a page that contains all staff request information along with a field that states whether the staff request is valid, invalid, unable to fill, or filled.

In addition, TPS staff members should be able update their resumes and picture through the web site.

The two areas of the website will be partitioned so that only clients with valid contract numbers and password are able to enter the client area and staff members with only valid employee numbers and password will be able to enter the staff area. Only the contract manager has full access to both locations.

## Major Software Functions

The application will provide the following functionality:

* Creation, modification, storage, and retrieval of staffing request information
* Creation, modification, storage, and retrieval of contract information.
* Creation, modification, storage, and retrieval of staff information.
* Creation, modification, storage, and retrieval of user access information.
* Allows a client to enter a staffing request into the database.
* Allows a client to retrieve staffing request information.
* Allows the contract manager to retrieve a staffing request from the database
* Allows the contract manager to retrieve contract information
* Allows the contract manager to validate the staffing request
* Allows the contract manager to close out the staffing request.
* Allows a staff member to update their personal information, resume, availability, and picture.

## Database Description

### Design Constraints and Limitations

1. Small business network, with no more than 20 nodes
2. Standard Windows 2000/XP workstations.
3. MS Office 2000/XP, with MS Access
4. MS Outlook 2000/XP

# Design Description

## Data Description

Data is handled through user interaction with the site, in which the user’s login credentials is checked against a MS Access file for the table titled login. Session variables are used to carry forth successful logins to access other areas of the site according to the user security type. Other site sections allow for access to requests and staff information that is also stored in the MS Access database file. Other data is handled through directory mapping and upload of files, such as resumes, pictures, and contracts.

## Data Flow

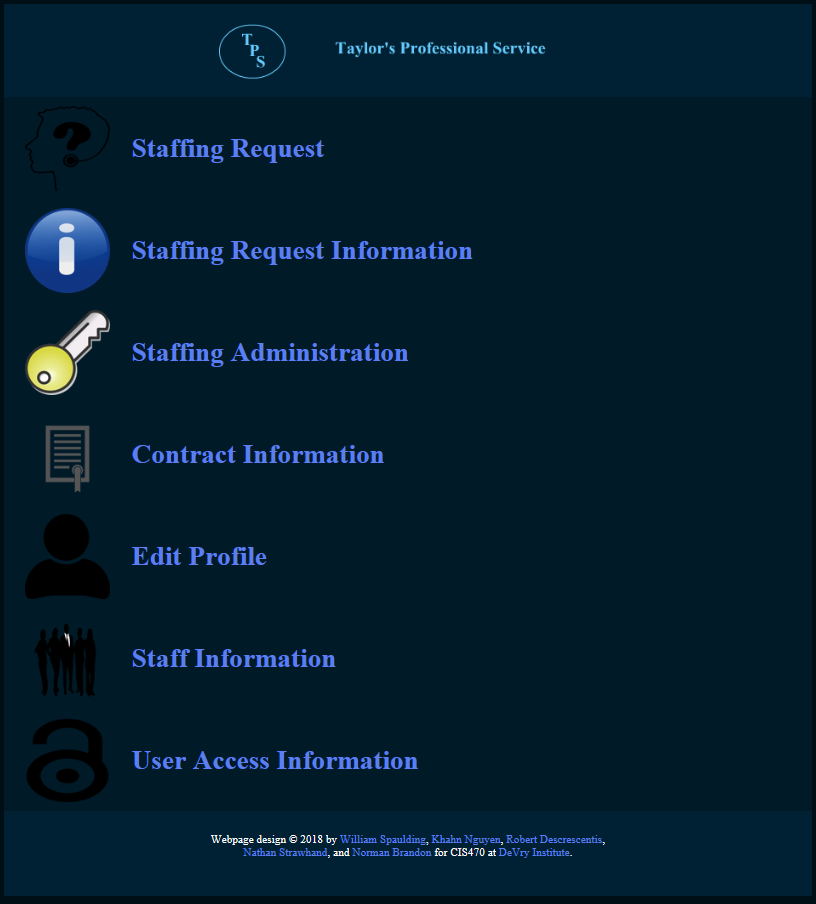


## Program Architecture



## Component Interfaces

The login.aspx page handles access to the site, checking against a TPSDataHandler.cs file for database access. The image provided in this section goes, index.aspx page, shows navigation to the rest of the components for the site (staff-request.aspx, staff-request-info.aspx, staff-admin.aspx, contract-info.aspx, profile.aspx, staff-info.aspx, and user-access-info.aspx). Each component has an associated interface with how the controls will be interacted with for user control.



# Detailed Design (one section for each component)

## Processing Description

The processing description section will define the procedures (functions, methods, routines, or subroutines) and which processes take input, generate output, and manipulate data. This will be broken down based on the .cs page files for the project. Each .cs file has its own process which allows other elements within the website to function.

**login.aspx.cs**

The login.aspx.cs file is the code that sets the overall security base for what is required to access any of the site. This is where all session variables are set that will dictate whether a certain user would have access to any other part of the site. The login page checks credentials against the MS Access database with the use of the TPSDataHandling csharp module. If valid login credentials are used, a valid security session is generated to move forward and access the site accordingly.

**index.aspx.cs**

The index.aspx.cs file is the code that is an indicator to what a user will be able to access on the site with the given security credentials. Certain components of the site will not be visible to users unless they have the appropriate credentials. If a user does not have any security clearance established, they are sent back to the login page.

**staff-request.aspx.cs**

The staff-request.aspx.cs file is the code that allows a client to generate a staff request to fulfill a job. This will include a gridview that lists all the potential staff to choose from, which is populated through use of the TPSDataHandling cs file. This page allows the client to sort, add requested staff, set worktype, set salary offered, and specify location for a staff request. When the client is satisfied with the request field, they will be able to submit their request for review by the manager.

**staff-request-info.aspx.cs**

The staff-request-info.aspx.cs file is the code that allows a client to search for a specific requestid associated with a given staff request. When found, and if the request belongs to the user, it will allow for complete editing of the staff request as well as show the current status of the request. This page allows the client to add staff through clicking a gridview, delete staff by clicking a remove button next to the associated staff name, change worktype through a dropdown list, change salary offered by a text box, and change location specified by a textbox for a given staff-request if needed to be changed for fulfillment. A gridview is populated with the current staff listing and is set to visible when the staff request is found, to allow for changes in client requested staff. A button is then used when the update is ready to be made.

**contract-info.aspx.cs**

As with all pages displayed the user authenticated through session login credentials. If the authentication fails, the user is defaulted back to the index page. A contract is created by the TPS company with a client’s contract to hire a staff member or for services rendered. After contract info is agreed upon, it is uploaded to the server in a doc file to the Contract folder. The contract-info.aspx page scans that directory, returns the title of every doc in it and populates a gridview with those titles on the page. The gridview can be sorted for easier location of the files displayed. If someone clicks on it, then they can click open or delete. When “open” is selected it downloads it to the desired location on their local machine for the individual to view and edit. There is an option to upload new or edited contracts to the server.

**staff-admin.aspx.cs**

As with all pages displayed the user authenticated through session login credentials. If the authentication fails, the user is defaulted back to the index page. This file contains the gridview and populates all the information from the request table within the Staf-Admin.aspx. The drop-down list provides options that allows the user to validate, invalidate, or unable to fill, the staffing request on the staff-admin.aspx page through drop-down controls. Once the drop-down selection is made the user then can update the request using btnUpdate\_Click. This will update selected request’s status in the request table. The user can also choose to delete a specific request using btnDelete\_Click.

**staff-info.aspx.cs**

The staff-info.aspx.cs file is the code behind for the staff-info.aspx page. It allows staff to view user information for viewing and selecting. The code behind allows the information from the GridView to transfer to labels on the page, once selected. The profile picture and resume will appear if the user has uploaded those items from the profile.aspx page.

There are several functions within in the page\_load method to note. Once a user logs in, they have full access to view the staff-info.aspx file. The security level is set to “S” (staff). The page\_load method also makes the initial calling of the TPSDataHandling.cs class file. After the TPSDataHandling class file is called, the system looks for the grabDataSet method (which has the information to select the information within the database so that the user can place information in it). After this, the system uses the PostBack method to find any data from previous forms or pages and transfers it to the gridview. This is done through the DataSource and Dictionary methods. After the data sourcing is done, the information is bound to the gridview.

The RowCreated method allows the user to select individual rows. The code to perform onclick, onmouseover, and a tool tip which explains, “Click to select row”, assists the user to select the rows.

The SortDataSet method allows for the gridview data to be sorted if the user wants sorting performed. The SortDataSet method first calls the DataView class and initializes it to sort the gridview. Then an if statement is performed which allows the user to sort either descending or ascending. After a row heading is clicked, the user’s data is sorted and bound to the gridview. From that point, the DataView session is called to show current selected staff and calls the updateSelected method.

The updateSelected method handles which row is highlighted. When a user clicks a row, the row highlights and the information is placed in a label below the gridview. If the user uploaded their resume and profile picture, it will display. This is performed through the Directory method. The directory method pulls the file from the folder on the server and displays it in the designated control on the page. A color translator is used to designate the color each row is highlighted once clicked.

The staff\_grid\_SelectedIndexChanged method handles selecting a line item on the gridview. Once a user selects staff, the updateSelected method is called to handle which row is highlighted.

**profile.aspx.cs**

This file contains textbox, labels, file upload, buttons and drop down list controls to submit information to the grid view after populated by the staff. After the staff enters the appropriate information and uploads the picture and resume, it transfers to the staff-info.aspx page. The information is stored in the database and transfers to the grid view so users can access data.

There are several functions within in the page\_load method to note. Once a user logs in, they have full access to view the staff-info.aspx file. The security level is set to “S” (staff). The page\_load method also makes the initial calling of the TPSDataHandling.cs class file. After the TPSDataHandling.cs class is called, a PostBack method is used to identify the dictionary through the UserID. If the dropdown list items contain a value then it is saved to the dictionary once the user clicks the Update Profile button. Other form data, resume and pictures are also saved to the dictionary once the user clicks the Update Profile button.

The updateProfile method occurs when the update button is clicked. After the user fills in the appropriate information, they have to click the Update Profile button to transfer the information to the profile.aspx page through the database. Once the database has updated, the user will be notified that the information has been updated successfully. If there was an error, the user will be notified to update the information and resubmit. This method also contains the picture upload and resume upload code instructions. If the resume and profile picture exceed the file size requirements, then a message would appear explaining their files are too large and to re-upload. If not, success messages will appear.

The lbResume\_Click method handles viewing the resume within the link button. This method accesses the Directory method so that the Resume can be saved to the appropriate file directory. This method gets the file name and extension for upload. It also verifies if the file meets the file type criteria for upload (i.e. text/HTML, text/plain, image/GIF, application/PDF, application/msword). If it does, then the file is written to the directory and the method ends.

The valueChange method allows the user to change or update the information placed in the textbox and dropdown list controls. This allows for easy updating for later access.

**user-access-info.aspx.cs**

UserAccessInfo.aspx.cs  will ensure proper credentials are present to access the page for security purpose. This cs uses allows the authorized user to view information of all users of the system through the gridview; which is linked to the userId database. The btn\_AddUser\_Click method uses the addUser method in TPSDataHandling.cs to pass the information into the GridView/database. The btnDeleteUser\_Click method ensures that the user exists and erases all user information from staff and user databases. Similarly, the btn\_Update\_Click method will ensure users exists and update user information in both user and staff databases. The gridview is dynamic allowing authorized personnel to select a user and modify information through the texboxes, drop down list, and buttons.

## Interface Description

The login page contains the security access to the site. The user must input their designated credentials in the username and password textfields provided. From this point the user is required to press the submit button in order to validate the credentials provided. If valid credentials are given, access is allowed.

The index page contains visual aids towards what the user is capable of accessing according to security level. The user will click on the associated image or label to access the displayed component. From there, the user will proceed to the selected link.

The staff-request page gives the user the ability to create staff requests through interaction of a gridview, textfields, and a submit button. The user will enter the appropriate information in the given fields to fulfill the information about the role and then hit submit when done. It will then be uploaded to the database and be ready for review by a manager.

The staff-request-info page gives the user the ability to update staff requests by searching for them according to the assigned requestid. A user enters a requestid, and then edits the appropriate fields according to what needs to be updated. The controls offered are the same as that of the staff-request page for editing. When the user is done editing, they will hit an update button to update the new information for the database.

The contract-info page contains the data from the contracts folder. The main interface control on the contract-info page is a gridview control. It pulls any documents contained in the contract folder and populates the gridview with the documents found inside the folder. The user then has the option to view, update, and delete each desired document.

The staff-admin page main interface control on the page is a GridView control, an update button, and delete button. The control gets the information from the request table in the database, and the populates it inside the GridView. The user then selects the desire request to view it. When the request is retrieved, a drop-down list is used to allow the user to change the requests’ status to valid, invalid, or cannot validate (due to various reasons such as missing information).

The profile.aspx and the staff-info.aspx files communicate with each other. The profile page contains the data from the database which is transferred to the staff-info page. The main interface control on the profile page is a web form. It gathers information the user places so that it can be transferred to the staff-info.aspx page. The user can also upload a resume and profile pic so that it can be saved to the respective folders (Pictures and Resume) for later download.

UserAccessInfo.aspx will allow the user to add new users. The user will also be able to be able to view all users and modify or delete any of the users in the database. This form will be only accessible to users that have Manager security levels. To choose the security level of the user, a dropdown list control will be used to allow the user to select from M, C, or S. Also included in UserAccessInfo.aspx is a gridView to display all users’ names, password and security level and give the option to delete or edit information with the Delete button and Update Button. Finally, an Add User button is included to allow the user to add new users.

## Pseudocode

**login.aspx.cs**

Class login:

TPSDataHandling useDatabase

Page\_Load:

useDatabase = new TPSDataHandling();

Session["UserID"] = nothing

Session["SecurityLevel"] = nothing

End Method Page\_Load

submit\_Click:

SecurityLevel = useDatabase.validateUser(username textbox entry, password textbox entry)

If SecurityLevel != nothing:

Session["UserID"] = username textbox entry

Session["SecurityLevel"] = SecurityLevel

SendUserTo index.aspx

End If

End Method submit\_Click

End Class login

**index.aspx.cs**

Class index:

Page\_Load:

If Session["SecurityLevel"].ToString() == "S":

Set component aspx file link visibility accordingly

Else If Session["SecurityLevel"].ToString() == "M":

Set component aspx file link visibility accordingly Else If Session["SecurityLevel"].ToString() == "C":

Set component aspx file link visibility accordingly Else:

SendUserTo login.aspx

End If

End Method Page\_Load

End Class index

**staff-request.aspx.cs**

Class staff\_request:

TPSDataHandling tpsData

DataSet ds

array current\_staff

void Page\_Load:

if (Session["SecurityLevel"].ToString() == "S"):

Server.Transfer("index.aspx", true)

else if (Session["SecurityLevel"].ToString() == "M" || Session["SecurityLevel"].ToString() == "C"):

// Do nothing

else:

SendUserTo login.aspx

tpsData = new TPSDataHandling()

ds = tpsData.grabDataSet("SELECT [userid] AS [ID], [full\_name] AS [Name], [experience] AS [Experience (Years)], [degree] AS [Has Degree?], [salary] AS [Salary], [city] AS [City], [state] AS [State] FROM staff")

staff\_grid.DataSource = ds

staff\_grid.DataBind()

End Method Page\_Load

void addStaff:

current\_staff append GetSelectedRow(staff\_grid)

End Method addStaff

void removeStaff(object):

current\_staff remove object

End Method removeStaff

void submitRequest:

tpsData.addRequest(userid, staff, location, worktype, salary)

End Method submitRequest

End Class staff\_request

**staff-request-info.aspx.cs**

Class staff\_request\_info:

TPSDataHandling tpsData

void Page\_Load:

if (Session["SecurityLevel"].ToString() == "S"):

Server.Transfer("index.aspx", true)

else if (Session["SecurityLevel"].ToString() == "M" || Session["SecurityLevel"].ToString() == "C"):

// Do nothing

else:

SendUserTo login.aspx

End Method Page\_Load

void addStaff:

current\_staff append GetSelectedRow(staff\_grid)

End Method addStaff

void removeStaff(object):

current\_staff remove object

End Method removeStaff

void updateRequest:

tpsData.updateRequest(requestid, staff, location, worktype, salary, status)

End Method submitRequest

void getStaffRequest():

myDict = tpsData.getRequest(requestid, userid)

SET staff to myDict["staff"]

SET location to myDict["location"]

SET worktype to myDict["worktype"]

SET salary to myDict["salary"]

tpsData = new TPSDataHandling()

ds = tpsData.grabDataSet("SELECT [userid] AS [ID], [full\_name] AS [Name], [experience] AS [Experience (Years)], [degree] AS [Has Degree?], [salary] AS [Salary], [city] AS [City], [state] AS [State] FROM staff")

staff\_grid.DataSource = ds

staff\_grid.DataBind()

staff\_grid.visible = true

End Method getStaffRequest

EndClass staff\_request\_info

**contract-info.aspx.cs**

Class Contract\_info

Method PageLoad

string selectedValue;

string[] dirs;

DataSet ds;

If Session is equal to "M"

Do nothing

Else if Session is equal to "S" or Session is equal to "C"

Call method Server.Transfer with "index.aspx", true

Else

Call method Server.Transfer with "login.aspx", true

EndIf

Create new DataTable

Call method myTable.Columns.Add with "Contracts"

Set dirs to Directory.GetFiles with Server.MapPath "Contracts", +"\\"

Foreach string in dirs

Initialise fn to Path.GetFileName with dir

Call method myTable.Rows.Add with fn

EndForeach

Create new DataSet

method ds.Tables.Add with myTable

If not IsPostBack

Set staff\_grid.DataSource to ds

Call method staff\_grid.DataBind

EndIf

Method RowCreated

If e.Row.RowType is equal to DataControlRowType.DataRow

Set "onclick" of e.Row.Attributes to Page.ClientScript.GetPostBackClientHyperlink with staff\_grid, "Select$" plus e.Row.RowIndex

Set "onmouseover" of e.Row.Attributes to "this.style.cursor='pointer';"

Set e.Row.ToolTip to "Click to select row"

EndIf

Method SortDataView

Dataview allows for sorting given data

Change according to session variable column header, according to last known direction

Set dv.Sort to e.SortExpression.ToString plus " DESC"

Set e.SortExpression.ToString of Session to "DESC"

Else

Set dv.Sort to e.SortExpression.ToString plus " ASC"

Set e.SortExpression.ToString of Session to "ASC"

EndIf

Bind new data and set postback data

Call method staff\_grid.DataBind

Call to show current selected staff

Method staff\_grid\_SelectedIndexChanged

Set selectedValue to position in staff\_grid.Rows

Set btnOpen.Visible to true

Set btnDelete.Visible to true

Call method updateSelected

Method UpdateSelected

Set row backcolor

If row.Cells is equal to selectedValue

Set row.BackColor to System.Drawing.ColorTranslator.FromHtml with "#4286f4"

Else

Set row.BackColor to System.Drawing.Color.Transparent

EndIf

EndForeach

Method btnDelete\_Click

Initialise dirs to Directory.GetFiles with Server.MapPath "Contracts", +"\\"

Foreach string in dirs

If dir.Contains with position in staff\_grid.Rows

Call method File.Delete with dir

Set "Title" of Session to "<h1>Contract Deleted!</h1>"

Set "Message" of Session to "The specified contract file has been deleted from the system."

Call method Server.Transfer with "success.aspx", true

Return

EndIf

EndForeach

Method btnOpen\_Click

Initialise dirs to Directory.GetFiles with Server.MapPath "Contracts", +"\\"

Foreach string in dirs

If dir.Contains with position in staff\_grid.Rows

Create new System.IO.FileInfo

Initialise name to Dfile.FullName

Initialise ext to Dfile.Extension

Initialise type to ""

set known types based on file extension

Switch on ext.ToLower

Case ".htm"

Case ".html"

Set type to "text/HTML"

End of Case

Case ".txt"

Set type to "text/plain"

End of Case

Case ".GIF"

Set type to "image/GIF"

End of Case

Case ".pdf"

Set type to "Application/pdf"

End of Case

Case ".doc"

Case ".rtf"

Set type to "Application/msword"

End of Case

EndSwitch

EndIf

Call method Response.AppendHeader with "content-disposition", "attachment; filename=" plus name

If type is not equal to ""

Set Response.ContentType to type

EndIf

Call method Response.WriteFile with dir

Call method Response.End

Return

EndIf

EndForeach

Method btnUpload\_Click

Resume Upload

File must be less than 1 MB

Try

Initialise fn to Path.GetFileName with file.FileName

Initialise SaveLocation to Server.MapPath with "Contracts", +"\\" plus fn

Initialise dirs to Directory.GetFiles with Server.MapPath "Contracts", +"\\"

Foreach string in dirs

If dir.Contains with fn

Call method File.Delete with dir

break;

EndIf

EndForeach

Call method file.SaveAs with SaveLocation

Set "Title" of Session to "<h1>Contract Uploaded!</h1>"

Set "Message" of Session to "The contract file has been uploaded to the system. You may now reference it from the contract-info page at any time."

Call method Server.Transfer with "success.aspx", true

Return

Catch Exception ex

Call method System.Diagnostics.Debug.WriteLine with ex.ToString

EndTry

EndIf

End Class Contract\_Info

End Contract\_Info.aspx.cs

**staffing-admin.aspx.cs**

TPSDataHandling tpsData;

DataSet ds;

string selectedValue;

Method Page\_Load

If Session is equal to "M"

Do nothing

Else if Session is equal to "S" or Session is equal to "C"

Call method Server.Transfer with "index.aspx", true

Else

Call method Server.Transfer with "login.aspx", true

EndIf

Create new TPSDataHandling

Set ds to tpsData.grabDataSet with "SELECT \* FROM requests"

If not IsPostBack

Call method staff\_grid.DataBind

Set selectedValue to ""

EndIf

Method rowCreated

If e.Row.RowType is equal to DataControlRowType.DataRow

Set "onclick" of e.Row.Attributes to Page.ClientScript.GetPostBackClientHyperlink with staff\_grid, "Select$" plus e.Row.RowIndex

Set "onmouseover" of e.Row.Attributes to "this.style.cursor='pointer';"

Set e.Row.ToolTip to "Click to select row"

EndIf

Method sortDataSet

Dataview allows for sorting given data

Change according to session variable column header, according to last known direction

Set dv.Sort to e.SortExpression.ToString plus " DESC"

Set e.SortExpression.ToString of Session to "DESC"

Else

Set dv.Sort to e.SortExpression.ToString plus " ASC"

Set e.SortExpression.ToString of Session to "ASC"

EndIf

Bind new data and set postback data

Call method staff\_grid.DataBind

Call to show current selected staff

Method staff\_grid\_SelectedIndexChange

Set selectedValue to position in staff\_grid.Rows

Set pnlControl.Visible to true

Call method updateSelected

Method updateSelected

Set row backcolor

If row.Cells is equal to selectedValue

Set row.BackColor to System.Drawing.ColorTranslator.FromHtml with "#4286f4"

Else

Set row.BackColor to System.Drawing.Color.Transparent

EndIf

EndForeach

Method btnDelete\_Click

Call method tpsData.deleteRequest with position in staff\_grid.Rows

Set "Title" of Session to "<h1>Request Deleted</h1>"

Set "Message" of Session to "The request has been deleted. Please contact TPS for any and all questions. Thank you and have a great day!"

Call method Server.Transfer with "success.aspx", true

Method btnUpdate\_Click

Call method tpsData.updateRequest with position in staff\_grid.Rows, position in staff\_grid.Rows, position in staff\_grid.Rows...

Set "Title" of Session to "<h1>Request Updated</h1>"

Set "Message" of Session to "The request has been updated. Please contact TPS for any and all questions. Thank you and have a great day!"

Call method Server.Transfer with "success.aspx", true

End Class staffing\_admin

End staffing\_admin.aspx.cs

**profile.aspx.cs**

Public partial class profile

Declare TPSDataHandling data class

Declare Dictionary<string, string> class

Protected void Page\_Load

If the session “SecurityLevel”

Then transfer to index.aspx

Else if session “SecurityLevel”

Do nothing

Else

Transfer to index.aspx

Call TPSDataHanding method

If postback

Dictionary equals TPSDataHandling getStaff with session “UserID” to string

Textbox name equals dictionary “full\_name”

If dropdown list Degree contains list item from dictionary “degree”

Then select value

Textbox experience equals dictionary “experience”

Textbox salary equals dictionary “salary”

Textbox city equals dictionary “city”

Textbox zipcode equals dictionary “zipcode”

String array dirs. Equals directory get files from folder “Pictures” on server

Foreach string directory in string array dirs.

If directory contains “UserID”

Then imgProfile image url equals Pictures folder in Session “UserID” get directory extension path as substring

imgProfile control is visible

string array dirs equals Directory GetFiles in Resume folder on server

Foreach string directory in string array dirs.

If directory contains “UserID”

Then linkbutton resume is visible

Protected void updateProfile

String message equals “Your information has been updated accordingly!”

If textbox name equals no text or dropdown list Degree has no selected value or textbox experience equals no text or Double TryParse textbox experience is out number or textbox salary equals no text or textbox street equals no text or textbox city equals no text or textbox state equals no text or textbox zipcode equals no text

Then create message stating, “However: There was an error found in your entry fields, resulting in a failure to store field information. Make sure that all fields are filled and that the Experience field is a numerical value”.

Else

TPSDataHandling updateStaff with session “UserID”

HttpPostedFile equals PictureUpload posted file

If file is not equal to null is IsImage file

Then Bitmap equals new Bitmap file InputStream

If Bitmap width is less than 200 is true or false bitmap height is less than 200

Try

String fn equals Path GetFileName

String FileExtension equals “.” Plus Path GetExtension file name with substring 1

Save file in location “Pictures” and session UserID and FileExtension

File SaveAs and save in folder location

String array dirs. Equals directory get files from folder “Pictures” on server

Foreach string directory in string array dirs.

If directory contains session UserID true or false directory equals location saved

Then delete file

Catch exception

If Create message, “Your information has been updated accordingly!”

Then message is equal to “However:”

Message is equal to, “The image selected was unable to upload. “

Else

If message equals, “Your information has been updated accordingly!”

Then message is equal to “However:”

Then message, “The image failed to upload since it was not less than or equal to 200x200. “

Else

If imgProfile is visible

If message equals, “Your information has been updated accordingly!”

Then message is equal to “However:”

Then message, “There was either an error uploading an image, or you have yet to upload one.”

File equals HttpPostedFile FileUpload Resume

If file is equal to null true or false file content length is greater than 0

Then try

String fn equals Path GetFileName

String FileExtension equals “.” Plus Path GetExtension file name with substring 1

Save file in location “Pictures” and session UserID and FileExtension

File SaveAs and save in folder location

String array dirs. Equals directory get files from folder “Pictures” on server

Foreach string directory in string array dirs.

If directory contains session UserID true or false directory equals location saved

Then delete file

Catch exception

System diagnostics debug write line exception to string

If Create message, “Your information has been updated accordingly!”

Then message is equal to “However:”

Message is equal to, “The resume selected was unable to upload. “

If linkbutton resume is visible

If message equals, “Your information has been updated accordingly!”

Then message is equal to “However:”

Then message, “There was either an error uploading a resume, or you have yet to upload one.”

Session “Title” equals to, “Information Updated”

Session “Message” equals message

Redirect to success.aspx

Private bool IsImage

Return file is equal to null check for true or false regular expressions regex is match, check file content type and length

Protected void lbResume\_Click

String array dirs. Equals directory get files from folder “Resume” on server

Foreach string directory in string array dirs.

If directory contains session “UserID”

Then file info equals new file info directory

String name equals file name

String extension equals file extension

String type equals to “”

If extension is equal to null

Then switch extension to lower case

Case .htm

Case .html

Type equals text/HTML

break

Case .txt

Type equals text/plain

break

Case .GIF

Type equals image/GIF

break

Case .pdf

Type equals application/pdf

Break

Case .doc

Case .rtf

Type equals Application/msword

Break

Response append header content-disposition, attachment, filename

If type is equal to “”

Then response content type equals type

Write file to directory

End response

Protected void valueChange

If sender is textbox

Then session textbox sender ID equals to textbox sender

Else if sender is dropdownlist

Then session dropdownlist sender ID equals to dropdownlist sender

**staff-info.aspx.cs**

Public partial class staff\_info

Declare TPSDataHandling data class

Declare Dictionary<string, string> class

Declare DataSet

String selectedStaff

Protected void Page\_Load

If the session “SecurityLevel”

Then transfer to index.aspx

Else if session “SecurityLevel”

Do nothing

Else

Transfer to index.aspx

Call TPSDataHanding method

Call TPSDataHandling grabDataSet method

If page is postback

Then session “DataView” equals new DataView data set tables set to 0

Gridview datasource equals data set

Session “Staff” equals “”

Else

String selectedStaff equals session “Staff”

Gridview datasource equals DataView session “DataView”

Mydict equals Dictionary method Session “REQUEST”

Then

Gridview databind

Protected void RowCreated

If row type is equal to DataControlRowType in DataRow

Then Row Attributes “onclick” equals Page ClientScript GetPostBackClientHyperLink gridview select RowIndex

Row Attributes “onmouseover” equals “this.style.cursor=’pointer;”

Row tooltip equals “Click to select row”

Protected void SortDataSet

DataView equals new DataView DataSet Tables 0

If session SortExpression to string is equal to null true or false Session SortExpression to String is equal to “ASC”

Then DataView sort equals to SortExpression to string “DESC”

Session SortExpression to string “DESC”

Else

Then DataView sort equals to SortExpression to string “ASC”

Session SortExpression to string “ASC”

GridView DataSource equals DataView

GridView DataBind

Session “DataView” equals DataView

Call updateSelected method

Protected void staff\_grid\_SelectedIndexChanged

String selectedStaff equals GridView Rows SelectedIndex Cells 0 Text

Session “Staff” equals selectedStaff

Call updateSelected method

Protected void updateSelected

Label name equals GridView rows SelectedIndex Cells 1 text

Profile image is not visible

Resume linkbutton is not visible

String array dirs. Equals directory get files from folder “Pictures” on server

Foreach string directory in string array dirs.

If directory contains selectedStaff

Then profile image url equals to “Pictures” plus selectedStaff

Profile image is visible

String array dirs. Equals directory get files from folder “Resume” on server

Foreach string directory in string array dirs.

If directory contains selectedStaff

Then profile image url equals to “Resume” plus selectedStaff

Linkbutton Resume is visible

Then panel information is true

Foreach GridViewRow in gridview

If row cells is equal to selectedStaff

Then row back color equals System Drawing ColorTranslator from HTML “#4286f4”

**user-access-info.aspx.cs**

//For security

if (Session["SecurityLevel"] == "S" || Session ["SecurityLevel"] == "C")

{

Server.Transfer("index.aspx", true);

}

else

{

Server.Transfer("login.aspx", true);

// Connect to database and dataset

ds = userDatabase.grabDataSet("SELECT \* FROM login");

user\_grid.DataSource = ds;

user\_grid.DataBind();

Get user name

Get User Password

Get user Security Level through DropDownList

Click add user

protected void btn\_AddUsers\_Click

if (new TPSDataHandling().IsUserExists(userIdTxt)) // existing user{

Print: user already exist;

}//endif

If (new user && password added){

Print = "User(" + userIdTxt + ") was successfully added!";

ds = userDatabase.grabDataSet("SELECT \* FROM [login]");

}//endif

else{

Print: "The user could not be added!";

}//endelse

else{

Print: "Please enter valid userName and password!";

}//endelse

// Allows user to select rows on the grid to manipulate

protected void user\_grid\_RowDataBound(object sender, GridViewRowEventArgs e)

{

if (e.Row.RowType == DataControlRowType.DataRow)

{

e.Row.Attributes["onclick"] = Page.ClientScript.GetPostBackClientHyperlink(user\_grid, "Select$" + e.Row.RowIndex);

e.Row.ToolTip = "Please click to select this row.";

}

}

protected void btnDeleteUser\_Click(object sender, EventArgs e)

{

//deletes user from user database and staff database

if (!userid.Text.Trim().Equals(string.Empty))

{

var userIdTxt = userid.Text.Trim();

if (tps.deleteUser(userIdTxt))

{

if (DropDownList\_Level.SelectedValue.Equals("S"))

{

tps.deleteStaff(userIdTxt);

}

//Delete from staff database

Print: "UserId(" + userIdTxt + ") was successfully deleted.";

ds = userDatabase.grabDataSet("SELECT \* FROM [login]");

user\_grid.DataSource = ds;

user\_grid.DataBind();

}

}

else

{

Print = "Invalid UserId.";

}

userid.Text = "";

password.Text = "";

}

//Clicking update button

protected void btn\_Update\_Click(object sender, EventArgs e)

{

if(user does not exist){

Print: "Update :: UserId(" + userIdTxt + ") not found.;

}

if (user exists){

Print: successfully updated.";

}

else

{

Print: "Failed to update UserId(" + userIdTxt + ").";

}

}

else

{

Print: "Invalid UserId.";

}

}

**TPSDataHandling.cs**

using System.Collections.Generic

using System.Data

using System.Data.OleDb

namespace TPS\_Senior\_Project

{

public class TPSDataHandling

{

string \_ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=TPSDatabases\\database.accdb;Mode=ReadWrite;";

OleDbConnection myConnection

OleDbCommand myCommand

OleDbDataReader myReader

OleDbDataAdapter myAdapter

// Initialization of the class call

public TPSDataHandling()

{

try

{

// Initialize the database

New connection

Open connection

New command

My Connection;

Command Text = "CREATE TABLE login([userid] text, [password] text, [security] text)"

Execute Non Query

Command Text = "INSERT INTO login([userid], [password], [security]) VALUES ('root', 'password', 'M')"

Execute Non Query

Command Text = "CREATE TABLE requests([requestid] AUTOINCREMENT PRIMARY KEY, [userid] text, [staff] text, [location] text, [worktype] text, [salary] text, [status] text)"

Execute Non Query

Command Text = "CREATE TABLE staff([userid] text, [full\_name] text, [degree] text, [experience] text, [salary] text, [street] text, [city] text, [state] text, [zipcode] text)"

Execute Non Query

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString());

}

finally

{

Close connection

}

}

// Handles creation of new login accounts

public void addUser(string userid, string password, string security)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "INSERT INTO login([userid], [password], [security]) VALUES ('" + userid + "', '" + password + "', '" + security + "')"

Execute Non Query

Close Connection

Return True

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

Return false

}

// Method check for if the user exists

public bool IsUserExists(string userid)

{

bool Result = false;

try

{

New connection

Open connection

New Comand

My command

Command Text = "SELECT \* FROM login WHERE [userid] = '" + userid + "'"

using (reader)

{

while (read myReader)

{

Result = true;

}

}

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString());

}

finally

{

Close connection

}

return Result;

}

// Handles grabbing userinfo and sends back the security code if password info is correct

public string validateUser(string userid, string password)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "SELECT ALL FROM login WHERE [userid] = '" + userid + "'"

using (myReader = myCommand.ExecuteReader())

{

while (myReader.Read())

{

if (myReader["password"].ToString() == password)

{

string returnVal = myReader["security"].ToString()

myCommand.Connection.Close()

return returnVal;

}

}

}

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

Close connection

return ""

}

// Handles updating user login accounts

public void updateUser(string userid, string password, string security)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "UPDATE login SET [password] = '" + password + "', [security] = '" + security + "' WHERE [userid] = '" + userid + "'"

Execute Non Query

Close Connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

Return false

}

// Handles deleting user login accounts

public void deleteUser(string userid)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "DELETE FROM login WHERE [userid] = '" + userid + "'"

Execute Non Query

Close connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

Return false

}

// Handles creation of new staff requests

public void addRequest(string userid, string staff, string location, string worktype, string salary)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "INSERT INTO requests([userid], [staff], [location], [worktype], [salary], [status]) VALUES ('" + userid + "', '" + staff + "', '" + location + "', '" + worktype + "', '" + salary + "', 'Not yet reviewed')"

Execute query

Close Connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

Return false

}

// Handles grabbing staff requests by requestid and sends back the info in a dictionary

public Dictionary<string, string> getRequest(string requestid, string userid)

{

Dictionary<string, string> myDict = new Dictionary<string, string>

try

{

New connection

Open connection

New command

My connection

Command Text = "SELECT ALL FROM requests WHERE [requestid] = '" + requestid + "'"

using (myReader = myCommand.ExecuteReader())

{

while (myReader.Read())

{

if (myReader["userid"].ToString() == userid)

{

myDict.Add("staff", myReader["staff"].ToString())

myDict.Add("location", myReader["location"].ToString())

myDict.Add("worktype", myReader["worktype"].ToString())

myDict.Add("salary", myReader["salary"].ToString())

myDict.Add("status", myReader["status"].ToString())

Close Connection

Return myDict

}

}

}

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString()

}

Return null

}

// Handles updating staff requests

public void updateRequest(string requestid, string staff, string location, string worktype, string salary, string status)

{

try

{

New connection

Open connection

New command

My connection

myCommand.CommandText = "UPDATE requests SET [staff] = '" + staff + "', [location] = '" + location + "', [worktype] = '" + worktype + "', [salary] = '" + salary + "', [status] = '" + status + "' WHERE requestid = '" + requestid + "'"

Execute query

Close connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

Return false

}

// Handles deleting user login accounts

public void deleteRequest(string requestid)

{

try

{

New connection

Open connection

New command

My connection

myCommand.CommandText = "DELETE FROM requests WHERE [requestid] = '" + requestid + "'";

Execute query

Close connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString());

}

finally

{

Close connection

}

Return false

}

// Handles creation of staff info

public void addStaff(string userid, string full\_name, string degree, string experience, string salary, string street, string city, string state, string zipcode)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "INSERT INTO staff([userid], [full\_name], [degree], [experience], [salary], [street], [city], [state], [zipcode]) VALUES ('" + userid + "', '" + full\_name + "', '" + degree + "', '" + experience + "', '" + salary + "', '" + street + "', '" + city + "', '" + state + "', '" + zipcode + "')"

Execute Query

Close connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString());

}

finally

{

Close connection

}

Return false

}

// Handles grabbing staff info and sends back the info in a dictionary

public Dictionary<string, string> getStaff(string userid)

{

Dictionary<string, string> myDict = new Dictionary<string, string>

try

{

New connection

Open connection

New command

My connection

Command Text = "SELECT ALL FROM staff WHERE [userid] = '" + userid + "'"

using (myReader = myCommand.ExecuteReader())

{

while (myReader.Read())

{

myDict.Add("full\_name", myReader["full\_name"].ToString())

myDict.Add("degree", myReader["degree"].ToString())

myDict.Add("experience", myReader["experience"].ToString())

myDict.Add("salary", myReader["salary"].ToString())

myDict.Add("street", myReader["street"].ToString())

myDict.Add("city", myReader["city"].ToString())

myDict.Add("state", myReader["state"].ToString())

myDict.Add("zipcode", myReader["zipcode"].ToString())

}

}

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

Close connection

return myDict

}

// Handles updating staff info

public void updateStaff(string userid, string full\_name, string degree, string experience, string salary, string street, string city, string state, string zipcode)

{

try

{

New connection

Open connection

New command

My connection

myCommand.CommandText = "UPDATE staff SET [full\_name] = '" + full\_name +"', [degree] = '" + degree + "', [experience] = '" + experience + "', [salary] = '" + salary + "', [street] = '" + street + "', [city] = '" + city + "', [state] = '" + state + "', [zipcode] = '" + zipcode + "' WHERE userid = '" + userid + "'";

Execute Non Query

Close connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

Return false

}

// Handles deleting staff info

public void deleteStaff(string userid)

{

try

{

New connection

Open connection

New command

My connection

Command Text = "DELETE FROM staff WHERE [userid] = '" + userid + "'";

Execute Non Query()

Close connection

Return true

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString());

}

finally

{

Close connection

}

Return false

}

// Grabs the specific sql data and returns a dataset with the found information

public DataSet grabDataSet(string sqlstring)

{

DataSet myDataSet = new DataSet();

try

{

New connection

Open connection

New command

My connection

Command Text = sqlstring;

myAdapter = new OleDbDataAdapter(myCommand)

myAdapter.Fill(myDataSet);

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString())

}

finally

{

Close connection

}

return myDataSet;

}

// Grabs the specific sql data and returns a dataset with the found information

public string grabRequestNumber(string userid)

{

try

{

New Connection

Open Connection

New command

My Connection

Command Text = "SELECT max(requestid) AS [last] FROM requests WHERE userid = '" + userid + "'";

using (myReader = myCommand.ExecuteReader())

{

while (Reader)

{

return myReader["last"].ToString();

}

}

}

catch (OleDbException e)

{

System.Diagnostics.Debug.WriteLine(e.ToString());

}

finally

{

Close connection

}

return "-1";

}

}

}

## Modules Uses

Software programs have many components. Each component has various modules that have several routines. The program is not complete until the modules are linked. This section will describe how the aspx pages are linked. Both communicate and work off each other and it is important to understand their uses.

The login page uses the TPSDataHandling.cs module to validate user login information and set the session variables appropriately. The session security variable set then allows access to the appropriate aspx pages.

The staff-request page uses the TPSDataHandling.cs module to both add new staff requests and receive current staff listing information.

The staff-request-info page uses the TPSDataHandling.cs module to search for and update staff requests, as well as receive current staff listing information.

The contract-info.aspx and the staff-admin.aspx pages use TPSDataHandling.cs to view contract and staffing request information. The GridView is used to store the data from the database and transfer to the staff-admin page. This is performed through the manager selecting the request data from the drop-down controls. From that point, users can view the staff information and make edits to the resume and profile pictures.

As stated in Section 3.1 and 3.2, the profile.aspx and staff-info.aspx pages use each other to view profile information for staff. The GridView is used to store the data from the database which is gathered from the profile.aspx page. This is performed through the user entering data in the textbox controls and clicking on the add user button in the profile.aspx page. From that point, users can view the staff information and make edits to the resume and profile pictures.

The User Access Information page main purpose is to ensure that authorized users to add new users. The user will also to be able to view all users and either add, modify, or delete any of the users in the user and staff databases for added security to the system and operates exclusively with the TPSDataHandling.cs and staff/user databases. This component of the software will make sure all personnel has the correct authorization for system use as well as ability to manipulate the security level in the event of changes.