**2021 Gr11 PAT PHASE 1 – ENCRYPTION/DECRYPTION**

**1. Scenario & Scope:**

A company hired me to create a Human Resources management application as their department are getting bottle-necked by all of the paperwork that needs to be done. They need a simple solution to store information about their employees and have the employees report concerns and general information via a separate form. This information is sensitive and needs the required security(encryption) and proper user account management.

**Scope:** Use a database to store employee data, and securely store the login passphrase by using one-way encryption(hashing). GUI must be easy and intuitive to use

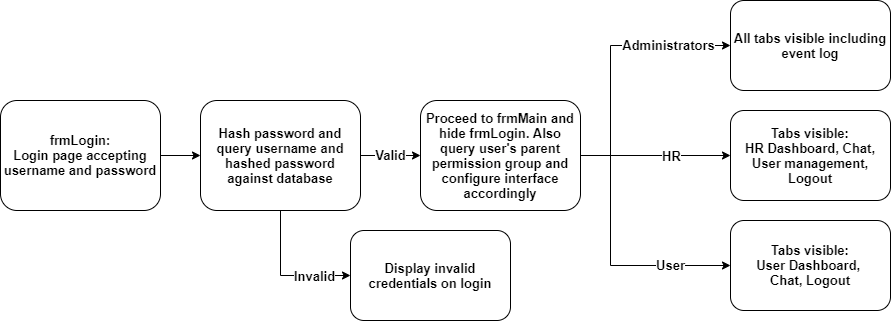
**Solution:**

I will be building one application that can be hosted on an employee’s kiosk for the employees to log into their account and report their concerns and activities such as sick leave or vacations. The HR department can also then log into their account to view this information and the application will automatically generate reports for each business day. All user personal data and credentials will be stored in a database and reports can be exported to text files. I will be implementing the Executable and Linking Format (ELF) hashing algorithm by PJ. Weinberger to securely store the passphrase of the user accounts in the database.

**2. User Requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Users** | **Role (Group)** | **Activities** | **Limitations** |
| Admin | Administrators | Manage Database, view events | None |
| Members of the HR department | HR | View reports add/remove users | Manage everything except Administrators and viewing  event logs |
| General users | user | Report activities and concerns | Add/Remove users  View sensitive info |

**3. Navigation of login (Flow chart):**

****

**4. Data Structures:**

- Database: Will be used to store user credentials and personal data:

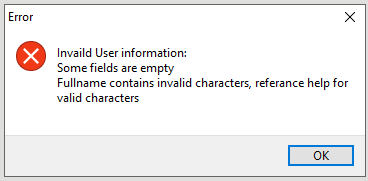
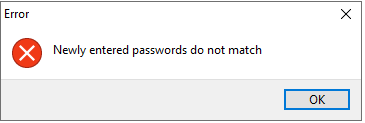
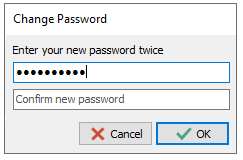
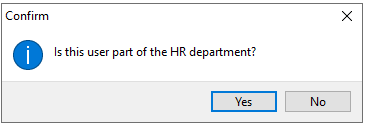
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Username  (primary key) | HashedPASS | Privilege  (Usergroup) | Fullname | Surname | Gender | Email | Enabled |
| Datatype(length)  Database Datatype | String(30)  Text | String(80)  Text | String(5)  Text | String(40)  Text | String(40)  Text | String(1)  Text | String(40)  Text | Boolean  Yes/No |
| E.g. | jdoe2583 | 8ah341maf925k15g2md | user | John | Doe | M | jdoe@pm.me | Yes |

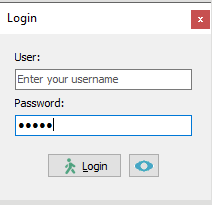
- Text file(s): Will be used to store the event log and to export application data and reports. Specifically all events created will be written to the text file and retrieved when the “event log” tab in the GUI is selected.

- Enum(s): Will be used to define the state of the application and some of the dynamic forms, makes code easier to read and understand.

- Array(s): Will be used for the password hashing, encryption and decryption functions and general data manipulation to generate summaries and reports. Float(real), string and integer. Constant arrays will also be used to store special characters for algorithms requiring it to perform e.g. input validation. (A string can be seen as an array of characters as its physically implemented like that)

- Record(s): For temporally storing the currently authenticated user’s information to limit constant database querying.

**5. GUI design**:

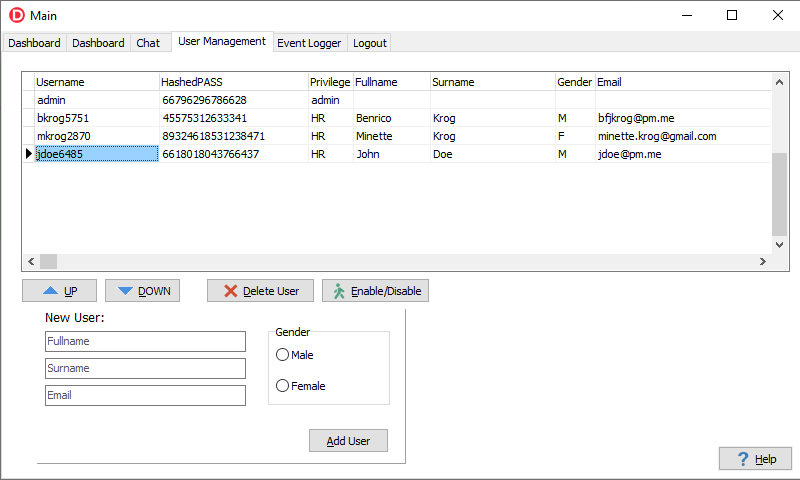
 **Login form**:

* Clear labelling
* Appropriate error messages for invalid inputs
* Button to view clear text password and hides it again after mouse click release
* Use of appropriate icons
* Clear flow of the login process with guiding hints available

when hovering over an object such as a button

**User management tab(add user section)**:

* Simple GUI and clear flow to the user
* Appropriate error messages for invalid inputs
* The use of messageDlg to ask questions
* Simple form to add a password
* Allow the user to cancel the addition process of the new user at the last step

****

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Task:** | **Input** | | | | | | **Processing** | | **Output** | | |
| **General** | | | **Validation** | | |
| **Source(GUI Component)** | **Data Type** | **Format** | **What** | **Method** | **Feedback** | **What** | **How** | **What** | **Format** | **Object** |
| **User Registration** | **Keyboard**  **(edtFullname)** | **String** | **Sentence Case** | **Numbers& Special Characters**  **&Null** | **Loop through and check each char against set of invalid characters** | **Your Fullname contains invalid characters, reference help for valid characters** | **Generate username**  **(primary key)** | **LowerCase(edtFullname.Text[1]**  **+util.noSpace(edtSurname.Text))**  **+IntToStr(RandomRange(1000, 10000));** | **Ask user permission group** | **Text (Question)** | **MessageDlg** |
| **Confirmation message** | **Text** |
| **Keyboard**  **(edtSurname)** | **String** | **Sentence Case** | **Numbers& Special Characters &Null** | **Loop through and check each char against set of invalid characters** | **Your surname contains invalid characters, reference help for valid characters** | **Store data in database (tblUsers)** | **tblUsers.Open;**  **tblUsers.Last;**  **tblUsers.Insert;**  **tblUsers['<Field>'] := <value>; (x7)**  **tblUsers.Post;** | **Data** | **DB records** | **DBgrid** |
| **Keyboard**  **(edtEmail)** | **String** | **Sentence Case** | **None** | **N/a** | **N/a** | **Log event in event logger** | **Text**  **(event log)** | **Textfile**  **(tFile)** |
| **Mouse**  **(rpgGender)** | **Char** | **Char (M/F)** | **Value selected?** | **Check if item index = -1** | **Please Select a gender** |
| **Login** | **Keyboard**  **(edtUser)** | **String** | **Single phrase** | **Check if username exist in database** | **Use Locate function to find username in tblUsers** | **Username does not exist** | **N/a** | **N/a** | **Form swap, Login form closes and Main application form opens** | **N/a** | **Form:**  **frmMain**  **frmLogin** |
| **Keyboard**  **(edtPass)** | **String** | **Passphrase** | **Check hashed version of the entered password against the database** | **Compare newly entered hashed password to hashed version in the database** | **MessageDlg(error)**  **Invalid password** | **Hash entered password** | **Use Util.EOFhash function:**  **Eof hashing algorithm (PJ.**  **Weinberger) followed by a character swapping (Util.charswap) algorithm** |
| **Hashing(ELF+charswap)** | **Util.ELFHash:**  **Function parameter**  **(const sKey : string)** | **String** | **Passphrase** | **None** | **N/a** | **N/a** | **Hash sKey with EOF algorithm** | **// Very large 64 bit prime number**  **iTableSize := 9952135015176462643;**  **for i := 1 to length(sKey) do begin**  **iHash := (iHash shl 4) + ord(sKey[i]);**  **iG := iHash and $F0000000;**  **if (iG<>0) then**  **iHash := iHash xor (iG shr 24) xor iG;**  **end;**  **result := iHash mod iTableSize;** | **Function return result** | **64bit Integer**  **(int64)** | **N/a** |
| **Util.swapChar: Function parameter**  **(const sKey : string)** | **String** | **Single phrase** | **None** | **N/a** | **N/a** | **Swap characters** | **iLen := s.Length;**  **c := c mod iLen;**  **if c = 0 then begin result := s; exit; end;**  **f := Round(b / iLen);**  **r := b mod iLen;**  **sP1 := rotateLeft(s.Substring(0, c), ((iLen mod c) \* f) mod c);**  **sP2 := rotateLeft(s.Substring(c), ((c \* f) mod (iLen -c)));**  **sTemp := sP1 + sP2;**  **for i := 1 to r do begin**  **cTemp := sTemp[i];**  **sTemp[i] := sTemp[(i+c) mod iLen];**  **sTemp[(i + c) mod iLen] := cTemp;**  **end;**  **result := sTemp;** | **Function return result** | **Text**  **(integer characters)** | **N/a** |