**Project Title:**

* Civil registry

**Project Description:**

The project is based on helping people retrieve details of their civil information with all its criteria as it helps people fill out registrations and completing any civil registration with just a click on a computer without going to any place.

**Requirements:**

let's break down the features for each user role:

For the Admin:

1- Addition and Removal of Users/Admins/Employees:

* The admin has the privilege to add new users, admins, or employees to the system.
* Removal of users/admins/employees from the system is possible using only their national ID for identification.
* Complete data entry is facilitated, ensuring that user data is immediately available for extraction (e.g., ID, passport, driver's license issuance).

For the Employee:

1-Data Change Requests Handling:

* Employees can view a list of requests for data changes.
* They have the authority to accept or decline requests based on the validity of the provided data.
* A refresh button is available in case the request list is empty, fetching another set of requests from the database to continue processing.

For the User:

1-Data Update Request:

* Users can enter their data for the first time or request changes to specific data.
* Changes requested by the user await approval from an employee to appear in the system.

2-Data Extraction:

* Users with completed and approved data can extract or obtain a copy of their national ID, passport, or driver's license from the system.

3-Appointment Scheduling:

* Users can schedule appointments for critical data changes that require physical appearance or for any other necessary reason.

These features cater to the specific needs and responsibilities of each user role, ensuring efficient data management and user interaction within the system.

**Class Design:**

**In the project there are three main classes : user, Admin and Employee**

***First user class:***

* the main objectives of this class:

**1.User Management:** The class provides methods for inserting new users into the system (InsertUser), searching for users (SearchNationalIDInInfo, SearchUser), and editing user information (EditUserName, EditUserPhoneNumber, EditUserAddress, EditUserPassword, EditUserBirth, EditUserBloodType).

**2.Data Retrieval:** It allows retrieving user information such as name, address, phone number, gender, birth date, blood type, and issue dates of identification documents (GetName, GetAddress, GetNumber, GetUserGender, GetUserBirth, GetUserBloodType, GetIssueDateId, GetIssueDateDl, GetIssueDateP).

**3.Appointment Management**: The class handles appointment-related functionalities, including checking if an appointment exists for a user (IsAppointmentTaken), inserting a new appointment (InsertAppointment), and retrieving the appointment date (GetAppointmentDate).

**4.Security and Authentication:** It includes methods for checking user credentials (SearchUser, IsAdmin), ensuring only authorized users can perform certain operations.

**Methods Used**

1. **InsertUser:**

* It constructs query string to insert a new record into the "Info" table of the database. The query includes placeholders (**@NationalID**, **@Name**, **@Password**, etc.) for parameterized values.
* It sets parameter values based on the method parameters (**nationalID**, **password**, **gender,etc…**) and some default values (“**0”** for name, phone number, and date of birth, **"user"** for position, **"1753-01-01"** for issue dates, **"0"** for blood type).
* It executes the SQL command using **command.ExecuteNonQuery()**, which returns the number of rows affected by the query.
* Depending on the result of the execution (**rowsAffected**), it displays a success or failure message using **MessageBox.Show**., if row affected then the data will be saved in database.

1. **SearchNationalIDInfo:**

The method is designed to check whether a given national ID exists in the "Info" table of the civil registry system's database. Here's an explanation of its functionality:

* It constructs an SQL query string to count the number of records in the "Info" table where the "National\_ID" column matches the provided national ID.
* it sets the parameter value for the national ID using **command.Parameters.AddWithValue**.
* t executes the SQL command using **command.ExecuteScalar()**, which returns the count of records matching the query condition.
* It converts the count value to an integer using **Convert.ToInt32** and checks if the count is greater than zero. If so, it sets the **found** variable to **true**, indicating that the national ID exists in the database.

1. **SearchUser**

This method is responsible for validating the existence of a user in the civil registry system's database based on the provided national ID and password. Let's break down its functionality:

* It constructs an SQL query string to select all columns from the "Info" table where the "National\_ID" matches the provided national ID and the "Password" matches the provided password.
* it sets the parameter values for the national ID and password using
* It executes the SQL command using **command.ExecuteScalar()**, which returns the first column of the first row in the result set. If a record matching the provided national ID and password exists, the result will be the number of records found; otherwise, it will be null.

1. **EditUserName:**

* Purpose: This method is responsible for editing the name of a user in the civil registry system.
* Parameters: It takes two parameters: nationalID, which uniquely identifies the user, and newName, which represents the updated name.
* Functionality:

It first checks if the user exists in the database using the SearchNationalIDInRequisites method.

If the user doesn't exist, it calls the InsertEmployee method to insert the user with the updated name, assuming other details remain unchanged.

If the user exists, it executes an SQL UPDATE query to modify the user's name in the "Requisites" table.

* Error Handling: It displays a message box with an error message if any exception occurs during the process.

1. **EditUserPhoneNumber:**

* Purpose: This method is similar to EditUserName, but it's used for updating the phone number of a user.
* Parameters: It takes two parameters: nationalID and newPhoneNumber.
* Functionality:

It first checks if the user exists in the database.

If the user doesn't exist, it inserts a new record with the updated phone number.

If the user exists, it updates the phone number in the "Requisites" table.

* Error Handling: It displays an error message if any exception occurs.

1. **EditUserAddress:**

* Purpose: This method updates the address of a user.
* Parameters: It takes two parameters: nationalID and newAddress.
* Functionality:

Similar to the previous methods, it checks if the user exists.

If the user doesn't exist, it inserts a new record with the updated address.

If the user exists, it updates the address in the "Requisites" table.

* Error Handling: It displays an error message if any exception occurs.

1. **EditUserPassword:**

* Purpose: This method updates the password of a user.
* Parameters: It takes two parameters: nationalID and newPassword.
* Functionality: It directly updates the password in the "Info" table using an SQL UPDATE query.
* Error Handling: It displays an error message if any exception occurs.

1. **EditUserBirth:**

* Purpose: This method updates the date of birth of a user.
* Parameters: It takes two parameters: nationalID and newbirth.
* Functionality: It updates the date of birth in the "Info" table using an SQL UPDATE query.
* Error Handling: It displays an error message if any exception occurs.

1. **EditUserBloodType:**

* Purpose: This method updates the blood type of a user.
* Parameters: It takes two parameters: nationalID and newbloodtype.
* Functionality: It updates the blood type in the "Info" table using an SQL UPDATE query.
* Error Handling: It displays an error message if any exception occurs.

1. **SearchNationalIDInRequisites:**

* Purpose: Checks if a given national ID exists in the "Requisites" table.
* Parameter: nationalID (the national ID to search for).
* Functionality: Constructs an SQL query to count the occurrences of the given national ID in the "Requisites" table.

Executes the SQL query using a parameterized SqlCommand object.

Retrieves the count of occurrences using ExecuteScalar.

If the count is greater than 0, it means the national ID exists in the table, and the method sets exists to true.

* Return value: Returns true if the national ID exists in the "Requisites" table; otherwise, returns false.

1. **GetName:**

* Purpose: Retrieves the name associated with a given national ID from the "Info" table.
* Parameter `nationalID` (the national ID for which to retrieve the name).
* Functionality

- Constructs an SQL query to select the name from the "Info" table based on the provided national ID

- Executes the SQL query using a parameterized SqlCommand object.

- Retrieves the result of the query using `ExecuteScalar`.

- If the result is not null, it converts the result to a string and returns the name.

- If the result is null, it displays a message indicating that the user was not found.

- Handles any exceptions that may occur during the process by displaying an error message.

-Returns the name associated with the given national ID if found; otherwise, returns null.

1. **GetUserPassword**

* Purpose: Retrieves the password associated with a given national ID from the "Info" table.
* Parameter: `nationalID` (the national ID for which to retrieve the password).
* Functionality: (similar to `GetName`)
* Returns the password associated with the given national ID if found; otherwise, returns null.

1. **GetAddress**

* Purpose: Retrieves the address associated with a given national ID from the "Info" table.
* Parameter: `nationalID` (the national ID for which to retrieve the address).
* Functionality: (Similar to `GetName` and `GetUserPassword`)
* Returns the address associated with the given national ID if found; otherwise, returns null.

1. **GetUserGender**

* Purpose: Retrieves the gender associated with a given national ID from the "Info" table.
* Parameter: `nationalID` (the national ID for which to retrieve the gender).
* Functionality: (Similar to previous methods)
* Returns the gender (as a string) associated with the given national ID if found; otherwise, returns null.

1. **GetNumber**

* Purpose Retrieves the phone number associated with a given national ID from the "Info" table.
* Parameter: `nationalID` (the national ID for which to retrieve the phone number).
* Functionality: (Similar to previous methods)
* Returns the phone number associated with the given national ID if found; otherwise, returns 0.

1. **GetIssueDateDl , GetIssueDateP and GetIssueDateId:**

* Purpose: Retrieve the issue date of the ID, driver's license and passport, associated with a given national ID from the "Info" table.
* Parameters: `nationalID` (the national ID for which to retrieve the issue dates).
* Functionality: (Similar to previous methods)
* Returns the issue date associated with the given national ID if found; otherwise, returns DateTime.MinValue.

1. **GetUserBirth and GetUserBloodType:**

* Purpose: Retrieve the birth date and blood type, respectively, associated with a given national ID from the "Info" table.
* Parameters: `nationalID` (the national ID for which to retrieve the birth date or blood type).
* Functionality: (Similar to previous methods)
* Returns the birth date or blood type associated with the given national ID if found; otherwise, returns null.

1. **insertEmployee:**

* Purpose: Inserts user information into the "Requisites" table, including both existing and new user details to be sent to the employee having a choice to accept or decline.
* Parameters: Name, phoneNumber, address, nationalID, newname, newPhoneNumber, newAddress.
* Functionality:
* Constructs an SQL query to insert employee details into the "Requisites" table.
* Uses a parameterized SqlCommand object to execute the query, with parameters for each user detail.
* Executes the query using ExecuteNonQuery.
* Displays a message if the insertion is successful.
* Handles any exceptions that may occur during the process by displaying an error message.

1. **IsAdmin:**

* Purpose: Checks if a user position with the given national ID and password by querying the "Info" table.
* Parameters: nationalID, password.

Return value: Returns the position of the user

* Functionality:
* Constructs an SQL query to select the position from the "Info" table based on the provided national ID and password.
* Uses a parameterized SqlCommand object to execute the query, with parameters for the national ID and password.
* Executes the query using ExecuteScalar.
* If the result is not null, it retrieves the position and returns it.
* If the result is null, it implies that the user does not exist or the password is incorrect.

1. **IsAppointmentTaken:**

* Purpose: Checks if an appointment exists for a given national ID.
* Parameters: nationalID.
* Return Value: Returns a boolean indicating whether an appointment exists.
* Functionality:
* Constructs an SQL query to select the appointment from the "Appointments" table based on the provided national ID.
* Uses a parameterized SqlCommand object to execute the query.
* Executes the query using ExecuteScalar.
* Returns true if an appointment exists, false otherwise.
* Catches and handles any exceptions that occur during the process.

1. **InsertAppointment:**

* Purpose: Inserts an appointment into the "Appointments" table.
* Parameters: nationalID, appointmentDate.
* Return Value: Returns a boolean indicating whether the insertion was successful.
* Functionality:
* Constructs an SQL query to insert the appointment details into the "Appointments" table.
* Uses a parameterized SqlCommand object to execute the query.
* Executes the query using ExecuteNonQuery.
* Returns true if rows are affected , false otherwise.
* Catches and handles any exceptions that occur during the process.

1. **GetAppointmentDate**

* Purpose:Get the appointment date for a given national ID.
* Parameters: nationalID
* Return Value: Returns the appointment date as a DateTime object.
* Functionality:
* Constructs an SQL query to select the appointment date from the "Appointments" table based on the provided national ID.
* Uses a parameterized SqlCommand object to execute the query.
* Executes the query using ExecuteScalar.
* Converts the result to a DateTime object and returns it.
* Error appears if the appointment date is not found.
* Catches and handles any exceptions that occur during the process.

1. **SearchAppointment**

* Purpose: Checks if an appointment exists for a given national ID.
* Parameters: nationalID
* Return Value: Returns a boolean indicating whether an appointment exists.
* Functionality:
* Constructs an SQL query to count the number of appointments for the provided national ID.
* Uses a parameterized SqlCommand object to execute the query.
* Executes the query using ExecuteScalar.
* Returns true if the count is greater than 0 , false otherwise.
* Catches and handles any exceptions that occur during the process.

1. **AppointmentRandomize()**

* Purpose: Generates a random appointment date and time for the user.
* Return Value: Returns the randomly generated appointment date and time as a DateTime object.
* Functionality:
* Initializes lists of available days and times for appointments.
* Generates a random day and time from the available options.
* Calculates the next occurrence of the selected day of the week.
* Combines the selected date and time to create a DateTime object representing the appointment.
* Returns the randomly generated appointment date and time.

1. **SaveIssueDateId, SaveIssueDateDl, SaveIssueDateP**

* Purpose: These methods are responsible for updating the issue dates (such as ID issue date, driver's license issue date, passport issue date) of a user in the database.
* Parameters: nationalID, issueDate.
* Functionality:
* Each method constructs an SQL query to update the corresponding issue date field (Issue\_Date\_ID, Issue\_Date\_DL, Issue\_Date\_P) in the Info table for the specified nationalID.
* They use parameterized SqlCommand objects to execute the queries.
* Parameters for the issue date and national ID are set.
* The query is executed using ExecuteNonQuery.
* Any exceptions that occur during the process are caught and handled, displaying an error message via MessageBox.

***Second: Employee class***

The main objective of the class:

1. Data Access:

class establishes connections with the database (Civil-Registry) through a SQL Server connection string.

It provides methods to retrieve and manipulate user data stored in the database.

1. User Data Manipulation:

Methods like AccepetUserEdit handle the acceptance of user edits by updating their

information in the database.

The class retrieves user information, such as name, address, and phone number, from the Requisites table in the database.

1. Database Operations:

The class performs SQL operations like SELECT, INSERT, and DELETE on the Requisites table.

It executes parameterized SQL commands to ensure data integrity and security.

Error Handling:

Exception handling is implemented to catch and handle errors that may occur during database operations.

Error messages are displayed using MessageBox to inform users about any issues encountered.

1. Relation with User Class:

It extends the User class, indicating a relationship where an employee inherits functionality from the User class, potentially overriding some of its methods to adapt behavior specific to employees.

**Methods Used**

1. **AccepetUserEdit**

* Purpose: Accepts user edits by updating their information in the database.
* Parameters: nationalID:
* Functionality:
* Retrieves the new name, phone number, and address of the user from the Requisites table in the database by using method.
* If any of the fields are not provided , it retrieves the corresponding data from the User class.
* Constructs an SQL query to update the user's information in the Info table with the new data.
* Executes the SQL command to update the user's information in the database.
* Deletes the corresponding record from the Requisites table once the user's information is successfully updated in the Info table.

1. **GetName**

* Purpose: Retrieves the new name of the user from the Requisites table.
* Parameters: nationalID:
* Functionality:
* Constructs an SQL query to select the new name of the user from the Requisites table based on the provided nationalID.
* Executes the SQL query to retrieve the new name from the database.
* Returns the retrieved new name, or null if not found.

1. **GetAddress**

* Purpose: Retrieves the new address of the user from the Requisites table.
* Parameters: nationalID.
* Functionality:
* Constructs an SQL query to select the new address of the user from the Requisites table based on the provided nationalID.
* Executes the SQL query to retrieve the new address from the database.
* Returns the retrieved new address, or null if not found.

1. **GetNumber**

* Purpose: Retrieves the new phone number of the user from the Requisites table.
* Parameters: nationalID.
* Functionality:
* Constructs an SQL query to select the new phone number of the user from the Requisites table based on the provided nationalID.
* Executes the SQL query to retrieve the new phone number from the database.
* Parses the retrieved phone number to an integer and returns it, or 0 if not found.

1. **DeleteRequisetByNationalID**

* Purpose: Deletes the record corresponding to the given nationalID from the Requisites table.
* Parameters: nationalID.
* Functionality:
* Constructs an SQL query to delete the record with the provided nationalID from the Requisites table.
* Executes the SQL command to delete the record from the database.

***Third: Admin class***

**Main objective of the class:**

provide functionality related to administrative tasks within the Civil Registry system. This includes managing admin data, such as adding new admin records to the database and deleting existing ones. Additionally, it may encompass other administrative functionalities as needed, such as updating admin information or performing other administrative actions within the system.

**Used Methods**

1. **InsertAdminData:**

* Purpose: This method inserts a new administrative user's data into the database.
* Parameters: nationalID, password, gender, name, phoneNumber, address, position and dateOfBirth.
* Functionality: It constructs a SQL query to insert the provided admin data into the Info table of the database. The parameters are added to the query, and then it's executed. If successful, a message is displayed indicating that the user was added; otherwise, an error message is shown.

1. **AdminDelete:**

* Purpose: This method deletes an admin record from the database along with any associated appointments.
* Parameters: nationalID.
* Functionality: It constructs a SQL query to delete the admin record from the Info table and any associated appointments from the Appointments table using a SQL DELETE statement. The query is executed, and if successful, a message indicating successful deletion is displayed; otherwise, an error message is shown.

**Implementation Details:**

**Inheritance**

The User class inherits from a class named DataAccsess And DataAccsess inherits

from a class named form while employee and admin classes inherits from the User class

**Exception Handling:**

Exception handling using try-catch blocks is used throughout thecode to handle potential exceptions that might occur during database operations.

**Event Handling:**

Event handlers are used to respond to user actions such as clicking buttons (button1\_Click, button2\_Click), changing text (txtUsername\_TextChanged, comboBox1\_SelectedIndexChanged), and pressing keys (TxtComPassword\_KeyDown, National\_ID2\_KeyDown). These events are handled to trigger specific actions in response to user interactions

**Error Handling:**

Error handling is implemented to validate user input and provide feedback to the user. ErrorProvider is used to display error messages when input validation fails (errorProvider2.SetError). Input validation is performed to ensure that required fields are not empty and that passwords match (if statements checking for empty fields and password matching).

**Polymorphism & Abstract**

Polymorphism is exhibited through method overriding. Both the User and Admin classes override abstract methods defined in their base class, DataAccess. This allows different implementations of methods with the same signature, depending on the specific subclass. For example, the GetName, GetNumber, and GetAddress methods are overridden in both the User and Admin classes to provide custom functionality.

**Encapsulation in User and Admin Classes:**

Both the User and Admin classes encapsulate data and methods related to user information and administrative functions. Attributes such as nationalID, name, phoneNumber, address, position, gender, and methods like InsertAdminData, AdminDelete, GetName, GetNumber, GetAddress, etc., are encapsulated within these classes. Access to these attributes and methods is controlled through their access modifiers (e.g., private, protected, public), enforcing encapsulation principles by hiding implementation details from external code.

**Encapsulation in DataAccess Class:**

The DataAccess class encapsulates data access operations, providing an abstract interface for retrieving user information. Methods like GetName, GetNumber, GetAddress are declared as abstract, enforcing encapsulation by defining a contract that subclasses must implement. The connectionString field is encapsulated within the class, limiting its visibility to other classes and promoting data hiding.

**Encapsulation in Database Operations:**

Database operations in methods like InsertAdminData, AdminDelete, GetName, GetNumber, GetAddress, etc., encapsulate SQL queries and connection management logic within the methods. This encapsulation shields the calling code from direct interaction with database-specific details, promoting modularity and code maintainability.

**Encapsulation in Exception**

Handling: Exception handling blocks encapsulate error-handling logic within try-catch blocks, isolating error handling from the main program flow. This encapsulation ensures that error handling is separated from normal program execution, promoting clarity and maintainability of the codebase.

**Factory Method Pattern:**

It starts with the signup and login forms then goes in 3 different patterns (user ,admin employee)

In the admindashboard class, the creation of different forms (such as AdminAdd and admindelete) follows a pattern similar to the Factory Method. Each form is instantiated within the Show method, encapsulating the creation logic and allowing for easy extension with addition forms in the future.

In the userdashboard you can go to update data or extract data

**Testing Strategy:**

* **Unit Testing:**
* Test the initialization of the classes form with the correct data from the database. Test the functionality of the buttons such as button1\_Click, button2\_Click, button3\_Click, and button5\_Click to ensure they perform the intended actions. Test the REGISTER\_Click method to verify that it updates user information correctly in the database.
* **Integration Testing:**
* Test the interaction between the form UI elements and the database operations to ensure seamless data retrieval and manipulation. Test the overall functionality of the form by simulating user interactions and verifying that the application behaves as expected.
* **Error and Exception Handling:**
* Implement try-catch blocks in critical sections of the code to catch and handle exceptions gracefully. Display meaningful error messages to the user in case of database connection failures, query execution errors, or unexpected exceptions. Log errors and exceptions to a log file for later analysis and troubleshooting. interactions and verifying that the application behaves as expected. Error and Exception Handling: Implement try-catch blocks in critical sections of the code to catch and handle exceptions gracefully. Display meaningful error messages to the user in case of database connection failures, query execution errors, or unexpected exceptions. Log errors and exceptions to a log file for later analysis and troubleshooting. User and Admin Classes: Unit Testing: Test each method in the User and Admin classes to ensure they perform the intended database operations accurately. Integration Testing: Test the interaction between the methods in these classes and the database to verify that data is retrieved, inserted, and updated correctly. Error and Exception Handling: Implement try-catch blocks in critical sections of the code to catch and handle exceptions gracefully. Display meaningful error messages to the user in case of database connection failures, query execution errors, or unexpected exceptions. Log errors and exceptions to a log file for later analysis and troubleshooting. updatedata Class: Unit Testing: Test the initialization of the form and ensure that the correct user data is displayed. Test the REGISTER\_Click method to verify that it updates user information correctly in the database. Integration Testing: Test the interaction between the form UI elements and the database operations to ensure seamless data retrieval and manipulation. Test the overall functionality of the form by simulating user interactions and verifying that the application behaves as expected. Error and Exception Handling: Implement try-catch blocks in critical sections of the code to catch and handle exceptions gracefully. Display meaningful error messages to the user in case of database connection failures, query execution errors, or unexpected exceptions. Log errors and exceptions to a log file for later analysis and troubleshooting.

**Screenshots/Visualizations :**

A person in a hoodie with a login page

Description automatically generated

A person in a hoodie with a login page

Description automatically generated

A person in a hoodie with a screen on the side

Description automatically generated

A person in a hoodie with a computer screen

Description automatically generatedA person in a hoodie with a computer login screen

Description automatically generated

A person in a hoodie with a computer screen

Description automatically generatedA person wearing a hoodie with a identification card

Description automatically generatedA computer screen shot of a person's face

Description automatically generatedA computer screen shot of a person's face

Description automatically generatedA person in a hoodie with a computer login

Description automatically generated

A person's face with a computer screen

Description automatically generatedA person in a hoodie with a login screen

Description automatically generated

**Known Issues and Limitations:**

* The national ID section of the database currently only accepts a maximum of 10 digits, rather than the accurate 14-digit national ID number. This limitation may lead to inaccuracies. Future updates will address this issue to ensure data integrity.

**Future Enhancements:**

* **Addition of a print button within the extract form to facilitate easy printing of data directly from the application, eliminating the need to visit an offline civil registry.**
* **Implementation of a new feature allowing users to upload their photos. This enhancement aims to improve user recognition and enhance security measures within the system.**

**Conclusion:**

* User Management: The project involves managing information about users, including their personal details such as name, national ID, address, contact number, gender, date of birth, and blood type. This information is stored in a database, likely SQL Server based on the connection string provided.
* Authentication and Authorization: There is functionality for authentication and authorization, as evidenced by the use of passwords and the distinction between admin, employee, and regular user roles. Admins have additional privileges compared to regular users and employees.
* Appointment Scheduling: The system supports the scheduling of appointments, as indicated by methods such as IsAppointmentTaken, InsertAppointment, GetAppointmentDate, and SearchAppointment.
* Editing User Data: Users can request edits to their personal information, and employees can accept these edit requests through methods like AcceptUserEdit.
* Data Integrity: The project seems to prioritize data integrity, as it includes methods for handling errors and exceptions when interacting with the database.
* Database Design: The database appears to be designed with multiple tables such as Info, Requisites, and Appointments, each serving a specific purpose related to user management, edit requests, and appointments.

**Appendix (Optional):**

* Include any additional information relevant to the project, such as code snippets, detailed class diagrams, or external references.