תכנות באינטרנט

ADO.Net

Connected Methods

גלעד מרקמן

קריית החינוך פארק המדע,
נס ציונה

Microsoft

ASP.NET Core

השיטה המקושרת

ExecuteNonQuery

שלבי העבודה – השיטה המקושרת

- **התחברות למסד הנתונים**
- SqlConnection con = new SqlConnection(connectionString);

- **בניית פקודת SQL**
- SqlCommand cmd = new SqlCommand("SQL String", con);

- **ביצוע הפקודה (שאילתה) מול בסיס הנתונים**
- con.Open();
- int n = cmd.ExecuteNonQuery(); // מחזיר את מספר הרשומות שעודכנו
- con.Close();

הפעולה ExecuteNonQuery()

- הפעולה command.ExecuteNonQuery() מקבלת:
  - פקודת SQL במחרוזת.
  - אובייקט connection עם פרטי שרת בסיס הנתונים.

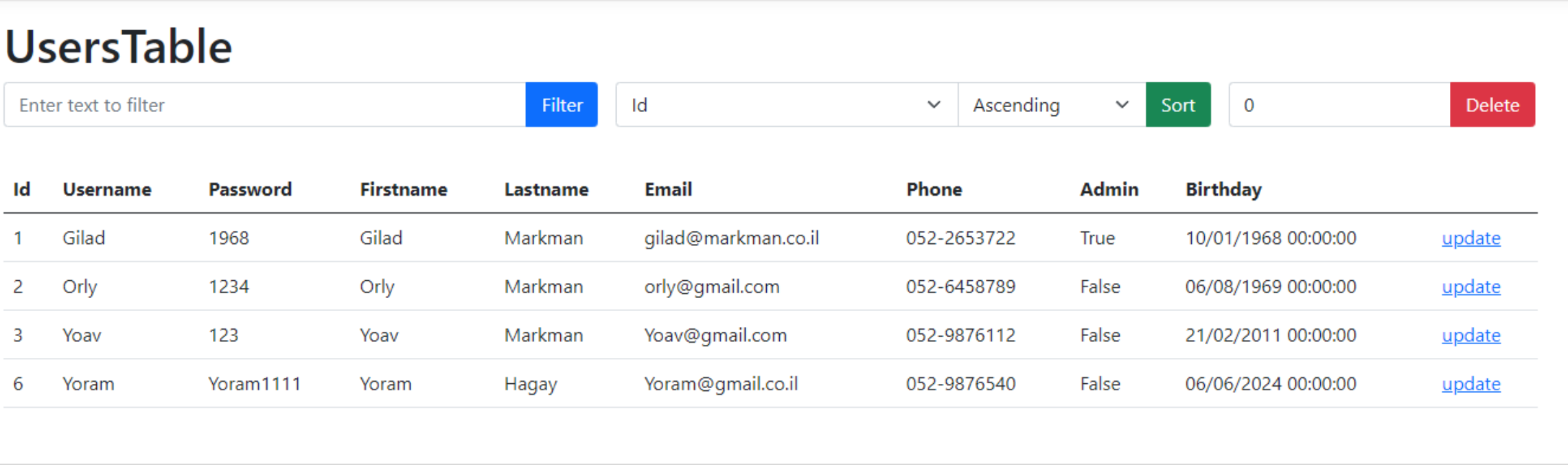
- הפעולה נועדה לבצע פקודות שאינן מחזירות מידע. כלומר, לא נועדה לבצע פקודת SELECT, אלא כל הפקודות האחרות, כגון:
  - DELETE
  - UPDATE
  - INSERT

עדכון רשומה
Update

השיטה המקושרת

עדכון רשומה

- נדגים כיצד אנחנו מטפלים באפשרות לעדכן רשומה מתוך הטבלה.
- העדכון נעשה על ידי לחיצה על הקישור Update.ניתן לעצב את הקישור ככפתור באמצעות Bootstrap.
- לחיצה על הקישור תפנה אותנו לדף Update שבו יבוצע העדכון.



הוספת קישור לטבלה ב- View

- נוסיף בטבלה עמודה לקישור update.
- אנחנו מעוניינים להעביר לטופס update את ה –id של המשתמש אותו בחרנו לעדכן.
- לכן, נוסיף לתגית <a>: asp-route-param=@row[“id”].
- הפקודה ב HTML מצורפת.
- הטופס ישלח לדף Update בקריאה OnGet שזו הקריאה של תגית <a>.

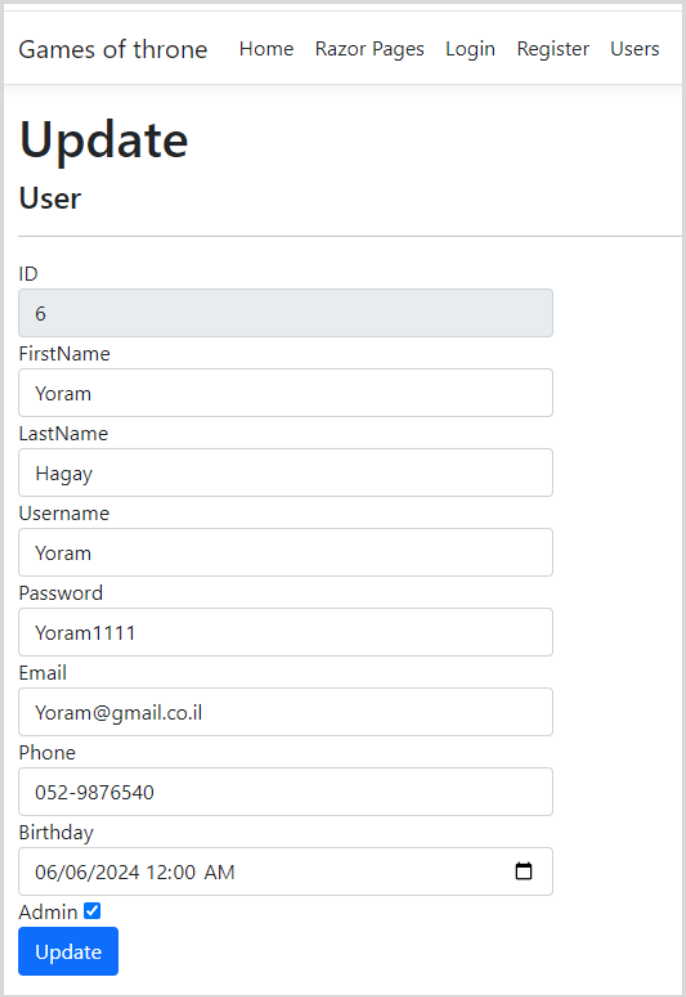
```
<table class='table'>
    <thead>
        <tr>
            @foreach (DataColumn column in Model.dt.Columns)
            {
                <th>@column.ColumnName</th>
            }
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (DataRow row in Model.dt.Rows)
        {
            <tr>
                @foreach (DataColumn column in Model.dt.Columns)
                {
                    <td>@row[column]</td>
                }
                <td><a asp-page="/Users/Update" asp-route-param="@row["Id"]">update</a></td>
            </tr>
        }
    </tbody>
</table>
```

```
<td>
  <a href="/Users/Update?param=6">update</a>
</td>
```

דף Update

- בעת OnGet (כניסה ראשונה לטופס) אנחנו נקבל את ה id של המשתמש, נשלוף את פרטיו מבסיס הנתונים ונציג בטופס.
- בעת OnPost לאחר שהמשתמש עדכן את הפרטים בטופס, ולחץ על לחצן "עדכן", אנחנו נעדכן אותם בבסיס הנתונים.
- אם העדכון לא הצליח אנחנו נדפיס בטופס הודעה עם השגיאה, באמצעות try, catch.
- אם העדכון הצליח נחזור לדף Index ונציג את רשימת המשתמשים עם הפרטים המעודכנים.
- לא לשכוח – בדיקת נתונים באמצעות JS.

View update



- דף update זהה לדף register למעט הוספת תיבת בחירה לעדכון ההרשאה.
- תיבת id הוגדרה כ readonly כיוון שהמשתמש אינו רשאי לעדכן אותה.

```
<div class="form-group">
    <label asp-for="NewUser.ID" class="control-label"></label>
    <input asp-for="NewUser.ID" class="form-control" readonly />
</div>
```

OnGet - Update

- הפעולה OnGet מקבלת פרמטר הכולל את מספר המשתמש לעדכון.
- הפעולה ממירה את מספר המשתמש ל int.
- שולפת את פרטי המשתמש מבסיס הנתונים.
- מעדכנת את המאפיין NewUser בפרטי המשתמש. מאפיין זה קשור (Bind) לטופס ולכן ערכיו יופיעו בטופס.
- לבסוף הפעולה מחזירה את הטופס למשתמש.

```
<td>
  <a href="/Users/Update?param=6">update</a>
</td>
```

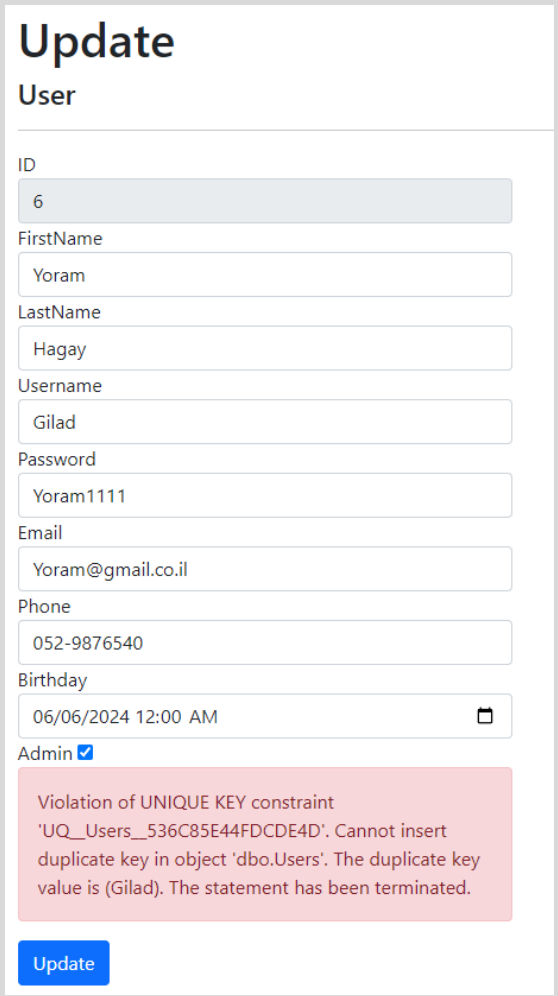
```
public IActionResult OnGet(string param)
{
    int id = int.Parse(param);
    Helper helper = new Helper();
    string SQL = $"SELECT * FROM Users WHERE Id = {id}";
    DataTable dt = helper.RetrieveTable(SQL, "Users");
    DataRow dr = dt.Rows[0];
    //User user = new User();
    NewUser.ID = id;
    NewUser.Username = dr["Username"].ToString();
    NewUser.Password = dr["Password"].ToString();
    NewUser.FirstName = dr["Firstname"].ToString();
    NewUser.LastName = dr["Lastname"].ToString();
    NewUser.Email = dr["Email"].ToString();
    NewUser.Phone = dr["Phone"].ToString();
    NewUser.Admin = (bool)dr["Admin"];
    NewUser.Birthday = DateTime.Parse(dr["Birthday"].ToString());

    return Page();

}
```

OnPost - Update

- הפעולה OnPost מזמנת את הפעולה Update של ה Helper.
- טיפול בחריגות Exception:
  - הפקודות try, catch נועדו לטפל בשגיאות ריצה.
  - אם הפעולה בתוך ה try לא מחזירה שגיאה מדלגים על ה catch. ומחזירים את המשתמש לטבלת המשתמשים.
  - אם הפעולה מחזירה שגיאה. השגיאה נתפסת במשתנה ex. הודעת השגיאה נשמרת ב msg המוצג בדף. הפעולה מחזירה את הדף הנוכחי עם הודעת השגיאה.

```
public IActionResult OnPost()
{
    Helper helper = new Helper();
    try
    {
        int n = helper.Update(NewUser, "Users");
    }
    catch (Exception ex)
    {
        msg = ex.Message;
        return Page();
    }
    return RedirectToPage("Index");
}
```

דף עם הודעת השגיאה



- ניסינו לעדכן עם שם משתמש קיים.
- בעת העדכון בבסיס הנתונים השרת החזיר הודעת שגיאה Violation of UNIQUE KEY constraint.
- הודעת השגיאה מוצגת בטופס והפרטים לא עודכנו בשרת.
- בדרך זו אנו "נתפוס" כל הודעת שגיאה שתקרא בעת ביצוע הפעולה Update.
- ניתן גם לטפל בדרך שונה בסוגים שונים של הודעת שגיאה (חורג ממסגרת הקורס).

הפעולה Update

- הפעולה Update במחלקה Helper מבצעת את עדכון בסיס הנתונים.
- הפעולה יוצרת פקודת SQL על פי הנתונים המעודכנים של ה User שהתקבל מהטופס.
- שימו לב לעניין התאריך שחייב שינוי התבנית של המחרוזת לתבניות המתאימה לבסיס הנתונים (תאריך אמריקאי).

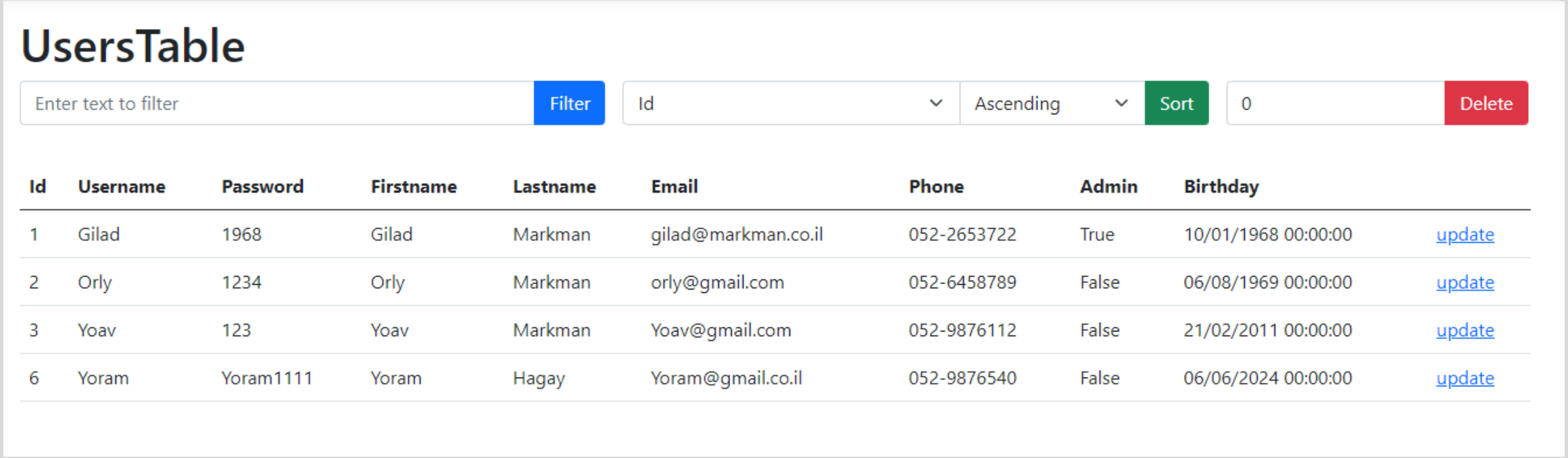
```
public int Update(User user, string table)
{
    string SQL = $"UPDATE {table} " +
        $"SET Username='{user.Username}', Password = '{user.Password}', " +
        $"FirstName  = '{user.FirstName}', LastName = '{user.LastName}', " +
        $"Email = '{user.Email}', Phone = '{user.Phone}',  Admin = '{user.Admin}', " +
        $"Birthday = '{user.Birthday:MM-dd-yyyy HH:mm:ss}' " +
        $"WHERE Id = {user.ID}";
    int n = ExecuteNonQuery(SQL);
    return n;
}
```

השיטה המקושרת

הפעולה DELETE

מחיקת משתמש

- מימשנו מחיקת משתמש באמצעות תיבה בה מנהל המערכת מזין את Id של המשתמש ולוחץ על לחצן Delete.



טופס View

- בטופס המציג את טבלת המשתמשים הוספנו את השדות הנדרשים:

```
<form method="post" asp-page-handler="Delete" style="width:20%">
    <div class="input-group mb-3">
        <input asp-for="Id" class="form-control" placeholder="Id to delete">
        <button class="btn btn-danger" type="Submit" id="Delete_btn">Delete</button>
    </div>
</form>
```

- השתמשנו ב asp-page-handler="Delete" כדי ליצור פעולה OnPostDelete בצד השרת.
- בנוסף הגדרנו בצד השרת מאפיין Id אשר מקושר לשדה למחיקה.

```
[BindProperty]
public int Id { get; set; }
```

צד השרת

- בצד השרת הוספנו פקודה OnPostDelete:

```
public IActionResult OnPostDelete()
{
    Helper helper = new Helper();
    helper.Delete(Id, "Users");
    string SQL = "SELECT * FROM Users";
    dt = helper.RetrieveTable(SQL, "Users");
    return Page();
}
```

- הפקודה מוחקת את המשתמש באמצעות helper.Delete.
- לאחר מכן, שולפת את הטבלה המעודכנת מבסיס הנתונים ושומרת במאפיין dt אשר מציג אותה במסך.

הפעולה Delete

- הפעולה Helper.Delete מקבלת מספר משתמש למחיקה ושם הטבלה.
- הפעולה בוצע פקודת SQL מתאימה ומבצעת אותה באמצעות ExecuteNonQuery.
- הפעולה תחזיר 1 (נחקה שורה אחת) במקרה של הצלחה. אם ה ID לא נמצא תחזיר הפעולה 1-.

```
public int Delete(int id, string table)
{
    if (id == 0)
    {
        return -1;
    }
    string SQL = $"DELETE FROM {table} WHERE ID = {id}";
    int n= ExecuteNonQuery(SQL);
    return n;
}
```

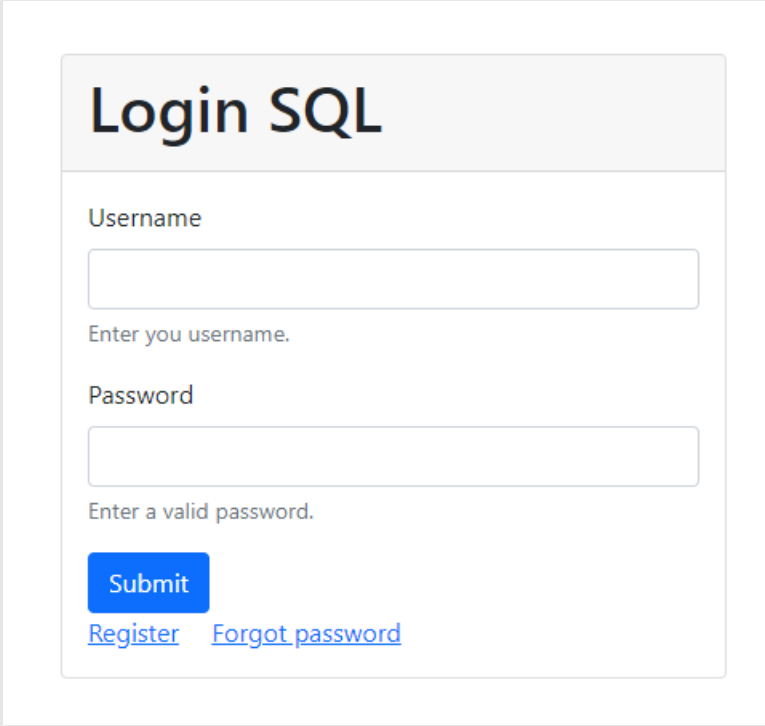
ExecuteScalar

טעינת ערך בודד

- אם אנחנו מעוניינים לטעון ערך בודד Scalar מבסיס הנתונים נוכל להשתמש בפקודה Command.ExecuteScalar().
- הפעולה מבצעת שאילתת SQL המעודכנת ב command.
- מחזירה ערך של שדה אחד הנמצא בעמודה הראשונה של השורה הראשונה.
- הערך המוחזר הוא Object שיכול להכיל את כל סוגי הערכים בטבלה.
- כדי לעשות שימוש בערך המוחזר יש לבצע המרה Casting

כניסה באמצעות ExecuteScalar

- נדגים טופס מימוש טופס כניסה באמצעות הפעולה ExecuteScalar.
- הטופס זהה לטופס הכניסה שהשתמשנו בו קודם לכן:



קוד צד השרת

- בקוד צד השרת נשתמש בפעולה Helper.GetScalar.
- נשים לב שבפעולת SELECT אנחנו מחזירים רק שדה אחד – Admin.
- אם האובייקט שמוחזר הוא null אז אין משתמש כזה.
- אם האובייקט שמחוזר אינו ריק – אזי הוא כולל True/False, ואותו אנחנו נעדכן ב Session.

```
public IActionResult OnPost()
{
    string SQLStr = $"SELECT Admin FROM Users WHERE Username LIKE '{Username}' AND Password LIKE '{password}'";
    Helper helper = new Helper();
    object admin = helper.GetScalar(SQLStr);

    if (admin != null)
    {
        HttpContext.Session.SetString("Login", Username);
        HttpContext.Session.SetString("Admin", admin.ToString());
        return RedirectToPage("/Index");
    }
    msg = "Wrong username or password.";
    return Page();
}
```

Helper.GetScalar בפעולה

- הפעולה GetScalar מבצעת שאילתת SELECT ומחזירה לנו ערך יחיד.
- אם השאילתה מחזירה רשימה של רשומות – הערך המוחזר הוא השדה הראשון ברשומה הראשונה.
- פעולה זו משמשת גם בשאילתות
- חישוביות כמו Count, Sum
- המחזירות לנו ערך בודד.

```
public object GetScalar(string SQL)
{
    // התחברות למסד הנתונים
    SqlConnection con = new SqlConnection(conString);

    // בניית פקודת SQL
    SqlCommand cmd = new SqlCommand(SQL, con);

    // ביצוע השאילתא
    con.Open();
    object scalar = cmd.ExecuteScalar();
    con.Close();

    return scalar;
}
```

DataReader

* רשות

קריאת נתונים מטבלה בשיטה המקושרת

- בשיטה המקושרת אנחנו קוראים נתונים מטבלה באמצעות iterator העובר על כל שורה בטבלה, כשהחיבור לבסיס הנתונים פתוח.
- הפעולה ExecuteReader מחזירה לנו iterator כזה.
- ה iterator "צועד" רק קדימה.
- הפעולות שניתן לבצע הן:

reader.HasRows	אמת אם הוחזרו שורות
reader.Read()	קרא את השורה הבאה מבסיס הנתונים. גם לפני קריאה ראשונה
Reader.getValue(0)	מחזיר אובייקט של העמודה הראשונה – מחייב המרה
reader.GetString(1)	מחזיר מחרוזת - יש לוודא שסוג הנתונים תואם
reader.GetBoolean(5)	מחזיר ערך בוליאני
Reader.GetName(2)	מחזיר שם העמודה

- בסיום יש לסגור את ה iterator.

הפעולה Helper.GetDataREader

- הפעולה מקבלת מחרוזת עם פקודה SQL – פקודת SELECT.
- הפעולה מחזירה אובייקט iterator אשר מאפשר לשלוף שורה שורה מהטבלה שהתקבלה על ידי פקודת SQL.
- הפרמטר CommandBehavior.CloseConnection מבטיח שסגירת ה reader יסגור את החיבור.

```
public SqlDataReader GetDataReader(string SQL)
{
    // התחברות למסד הנתונים
    SqlConnection con = new SqlConnection(conString);

    // בניית פקודת SQL
    SqlCommand cmd = new SqlCommand(SQL, con);

    con.Open();
    // Command behavior insure closing the reader will close the connection
    SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
    return reader;
}
```

הקוד בצד השרת

- נגדיר במודל של צד השרת מאפיין Reader.

- בפעולה OnGet ניצור Iterator לטבלה ונשמור אותו במאפיין.

```
public SqlDataReader Reader { get; set; }

public void OnGet()
{
    Helper helper = new Helper();
    string SQL = "SELECT * FROM Users";
    Reader = helper.GetDataReader(SQL);
}
```

יצירת הטבלה

```
<table class='table'>
    <thead>
        <tr>
            @if (Model.Reader.HasRows)
            {
                for (int i = 0; i < Model.Reader.FieldCount; i++)
                {
                    <th>@Model.Reader.GetName(i)</th>
                }
            }
        </tr>
    </thead>
    <tbody>
        @while (Model.Reader.Read())
        {
            <tr>
                @for (int i = 0; i < Model.Reader.FieldCount; i++)
                {
                    <td>@Model.Reader.GetValue(i)</td>
                }
            </tr>
        }
    </tbody>

</table>
@{
    Model.Reader.Close();
}
```

- ניצור את הטבלה באמצעות לולאות דומות, אך במקום להשתמש ב DataTable נשתמש ב Iterator.

- נשים לב, כי בסיום יצירת הטבלה יש לסגור את ה reader.

כתיבת command באמצעות פרמטרים

```
1 reference
public int ParamExecuteNonQuery()
{
    SqlConnection con = new SqlConnection(_connectionString);

    String SQL = "UPDATE Users " +
        "Set LastName = @lastName " +
        "WHERE LastName LIKE @filter";

    SqlCommand cmd = new SqlCommand(SQL, con);

    SqlParameter param = new SqlParameter ();
    param.ParameterName = "lastName";
    param.Value = "Rom";
    param.SqlDbType = SqlDbType.NVarChar;
    cmd.Parameters.Add(param);

    param = new SqlParameter ();
    param.ParameterName = "filter";
    param.Value = "Yaakov";
    param.SqlDbType = SqlDbType.Char;
    cmd.Parameters.Add(param);

    con.Open();
    int n = cmd.ExecuteNonQuery();
    return n;
}
```

- ניתן לכתוב את פקודת ה SQL המשמשת אותנו לבניית האובייקט SqlCommand באמצעות פרמטרים.
- דרך זו מפחיתה את שגיאות ההקלדה.
- כמו כן, שיטה זו מונעת פרצת אבטחה המכונה SQL Injection.

סיכום

- השלבים בעבודה בשיטת עבודה מקושרת.
- הפקודות בשיטה המקושרת:
  - Command.ExecuteScalar
  - Command.ExecuteNonQuery
  - Command.ExecuteReader
- שימוש באובייקט DataReader
  - reader.HasRows;
  - reader.Read();
  - Reader.getValue(0);
  - reader.GetString(1);
  - reader.GetBoolean(5);
  - Reader.GetName(2);