

Databaskonstruktion

Inlämning 3 “MVC”

Violeta Janikuniene
a22vioja

1. Visa innehållet från 2 olika databastabeller

För att visa innehållet från en databastabell med C# och användning MVC strukturen implementerades koden på tre olika ställen.

“**HomeController.cs**” (se figur 1.1) används för att hantera begäranden från användare och bestämma vilken vy som ska visas som svar på dessa begäranden. En funktion som heter “Index2” skapades för att hantera begäranden till en vy som ska visa DesinfoSpridare tabellen. “CustomerModel” används för att hämta data från en databastabell DesinfoSpridare agenter. Denna data hämtas med hjälp av “GetDesinfoSpridare” metoden som är beskriven in “CustomerModel” (se figur 1.2).

Resultatet sparas av datahämtningen “desinfoSpridareData” i en “DataTable”. “ViewBag” används för att lagra “desinfoSpridareData” som en variabel med namnet “DesinfoSpridare” och det gör det möjligt att överföra data från en kontroller till en vy.

```
public IActionResult Index2()
{
    CustomerModel desinfoSpridModel = new CustomerModel(_configuration);
    DataTable desinfoSpridareData = desinfoSpridModel.GetDesinfoSpridare();
    ViewBag.DesinfoSpridare = desinfoSpridareData;
    return View();
}
```

Figur 1.1

“**CustomerModel.cs**” (se figur 1.2) används för att hantera actions till olika tjänster på webbsidan. “MySqlDataAdapter” används för att utföra en SQL-fråga på databasen. I det här fallet hämtas alla rader från tabellen “DesinfoSpridare” och frågan utförs mot anslutningen dbcon som har öppnats i början av kod för att kunna avsluta till databas.

“DataSet” används för att lagra data som hämtas från databasen. Med hjälp av “adapter” utförs SQL fråga till databasen och resultatet av frågan fyller “DataSet” med data.

“DataTable desinfoSpridareTable = ds.Tables[“result”]” betyder att tabellen hämtas med namnet “result” från “DataSet” och tilldelas till en “DataTable” med namnet “desinfoSpridareTable”.

“dbcon.Close()” stänger connection till databasen.

Slutligen returneras “desinfoSpridareTable” som innehåller resultaten av SQL-frågan, från metoden.

```

public DataTable GetDesinfoSpridare()
{
    MySqlConnection dbcon = new
MySqlConnection(_connectionString);
    dbcon.Open();
    MySqlDataAdapter adapter = new MySqlDataAdapter("SELECT * FROM
DesinfoSpridare;", dbcon);
    DataSet ds = new DataSet();
    adapter.Fill(ds, "result");
    DataTable desinfoSpridareTable = ds.Tables["result"];
    dbcon.Close(); //stänger db konektion

    return desinfoSpridareTable;
}

```

Figur 1.2

View “Index2.cshtml” används för att visa resultatet av databashämtning på webbsidan (se figur 1.3). Här används html kod.”@” refererar till C# Razor-syntax i .NET-webbapplikationer för att blanda serverkod (C#) med HTML i vyer.

Första if satsen kontrollerar om “Viewbag.DesinfoSpridare” är inte tom och ifall så, skapas en HTML-tabell med CSS-klassen "my-table" för formatering. **“foreach” används** för att loopa igenom kolumnerna och för varje kolumn skrivs kolumnrubriken i tabellen.

En annan “foreach” loop används för att loopa igenom varje rad i ViewBag.DesinfoSpridare och “for” används för att loopa igenom varje kolumn i raden och för varje cell i raden skrivs värdet från den aktuella kolumnen.

“Else” används för att visa felmeddelandet “No search result found” ifall “DesinfoSPridare” tabellen är tom.

```


### Desinformation Spridare Agents



@if (ViewBag.DesinfoSpridare != null &&
((System.Data.DataTable)ViewBag.DesinfoSpridare).Rows.Count > 0)
{
    <table class="my-table">
        <tr>
            @foreach (var dataColumn in ViewBag.DesinfoSpridare.Columns)
            {
                <th>@dataColumn.ColumnName</th>
            }
            <th>Action</th>
        </tr>
        @foreach (var row in ViewBag.DesinfoSpridare.Rows)
        {
            <tr>
                @for (int i = 0; i < ViewBag.DesinfoSpridare.Columns.Count; ++i)
                {
                    <td>@row[i]</td>
                }
                <td>@Html.ActionLink("Delete", "DeleteAgent", "Home", new { namn
= row["namn"], nr = row["nr"] }, new { title = "Click to delete agent " +
row["namn"] })</td>
            </tr>
        }
    </table>
}
else
{
    <p>No search results found.</p>
}

```

Figur 1.3

Det här är webbsidans outputen för tabellen 'DesinfoSpridare' (se figur 1.4). För att göra vyn mer organiserad har jag lagt till CSS-kod (se figur 1.5). Här definieras en regel för alla element med klassen "my-table." Den används för att ge tabellerna med klassen "my-table" en viss visuell stil. De får svarta ramar runt sig, och det finns 5 pixlar stoppning runt cellinnehållet.

Desinformation Spridare Agents					
namn	nr	specialite	namnInc	nrInc	Action
A	1	skapar bra lögner	snabb prestanda	1	Delete
B	2	snurra	danser	1	Delete
C	3	blicka	snabb prestanda	2	Delete
M	5	sover alltid	konstig	6	Delete

Figur 1.4

```
.my-table {
    border: 1px solid #000;
}
.my-table th, .my-table td {
    padding: 5px;
    border: 1px solid #000;
}
```

Figur 1.5

Den andra databastabellen som visas på webbsidan är "Report" tabellen. Den har skapats på ett samma sätt som "DesinfoSpridare" utan att den visas på ett andra sidan (se figur 1.6).

HomeController:

Här skapades metod deklARATION som heter "Repport" och den implementerades på samma sätt som "Desinfospridare (se figur 1.6). Resultatet sparas av datahämtningen "repportData" i en "DataTable" och "ViewBag" används för att lagra "repportData" som en variabel med namnet "RepportResults". Den ska används senare i koden på "Views".

```
public IActionResult Repport()
{
    CustomerModel dm4 = new CustomerModel(_configuration);
    DataTable repportData = dm4.Repport();
    ViewBag.RepportResults = repportData;
    return View();
}
```

Figur 1.6

CustomerModel:

Samma logiken används för att skapa metoden "Repport" som deklarerar i "HomeController (se figur 1.6). "CustomerModel" ska hantera SQL- fråga i databasen (se figur 1.7).

```
public DataTable Repport()
{
    MySqlConnection dbcon = new MySqlConnection(_connectionString);
    dbcon.Open();
    MySqlDataAdapter adapter = new MySqlDataAdapter("SELECT * FROM
Rapport;", dbcon);
    DataSet ds4 = new DataSet();
    adapter.Fill(ds4, "result");
    DataTable repportTable = ds4.Tables["result"];
    dbcon.Close();

    return repportTable;
}
```

Figur 1.7

Views:

En ny vy har skapats i "Views" med namnet "Rapport" (se figur 1.8), som visar tabellen med hjälp av "ViewBag.RapportResult" som kommer från "HomeController" (se figur 1.6). HTML-koden implementerades på samma sätt som för "DesinfoSpridare"-tabellen i "Index2"-vyn.

```
@if (ViewBag.RepportResults != null &&
((System.Data.DataTable)ViewBag.RepportResults).Rows.Count > 0)
{
    <table class="my-table">
        <tr>
            @foreach (var dataColumn in ViewBag.RepportResults.Columns)
            {
                <th>@dataColumn.ColumnName</th>
            }
        </tr>
        @foreach (var row in ViewBag.RepportResults.Rows)
        {
            <tr>
                @for (int i = 0; i < ViewBag.RepportResults.Columns.Count; ++i)
                {
                    <td>@row[i]</td>
                }
            </tr>
        }
    </table>
}
else
{
    <p>No search results found</p>
}
```

Figur 1.8

Tabellen på webbsida ser ut så här (se figur 1.9):

Report Table			
datum	titel	namnIncident	nrIncident
1/1/2020 12:00:00 AM	mormor	konstig	6
1/2/2020 12:00:00 AM	farfar	inget	7
1/7/2020 12:00:00 AM	stora ögon	fuktigt	5
9/1/2020 12:00:00 AM	gröna kaniner	snabb prestanda	1
9/4/2020 12:00:00 AM	röda bananner	danser	1
9/5/2020 12:00:00 AM	gula roboter	snabb prestanda	2
9/10/2020 12:00:00 AM	vita ögon	inge	1
2/20/2022 12:00:00 AM	blommor med ögon	lalala	2
2/23/2022 12:00:00 AM	cc	inget	15
3/3/2023 12:00:00 AM	vänlig träff	sönder	10
9/18/2023 12:00:00 AM	stjärna	ljus	10

Figur 1.9

2. Ett formulär med fritextfält

För att lägga till desinformations spridare agent till tabellen et formulär med fritextfält skapades (se figur 2.1).



Add a new Agent:

Name:

Nr:

Specialite:

Incident name:

Incident nr:

Figur 2.1

HomeController:

I "HomeController" skapades metod deklaration med namnet "AddAgent" som tar in parametrar namn, nr, specialitet, namnInc och nrInc (se figur 2.2). "AddAgent" metoden är en del av en "CustomerModel" (se figur 2.3) som lägger till den nya agentinformationen i databasen. Efter det returnerar "RedirectToAction("Index2")". Detta innebär att användaren omdirigeras till en annan åtgärd med namnet "Index2" i samma kontroller efter att agentinformationen har lagts till.

```
public IActionResult AddAgent(string namn, string nr, string specialite, string namnInc, string nrInc)
{
    _customerModel.AddAgent(namn, nr, specialite, namnInc, nrInc);
    return RedirectToAction("Index2");
}
```

Figur 2.2

CustomerModel:

I "CustomerModel" implementerades metoden "AddAgent" för att utföra en SQL-fråga i en MySQL-databas för att lägga till en ny agent (se figur 2.3). Den tar emot agentens information som parametrar (samma som skapades i HomeController (se figur 2.2)). "string addSring" definierar en SQL-fråga som ska användas för att lägga till en ny rad i tabellen "DesinfoSpridare" och den också innehåller placeholders (markerade med "@"-tecknet) för

de värdena som kommer att infogas i frågan. Sedan utförs SQL-frågan. "sqlCmd.Parameters.AddWithValue" används för att säkerställa säker SQL-hantering och undvika SQL-injektioner.

```
public void AddAgent(string namn, string nr, string specialite, string namnInc,
string nrInc)
{
    MySqlConnection dbcon = new MySqlConnection(_connectionString);
    dbcon.Open();

    string addString = "INSERT INTO DesinfoSpridare
(namn,nr,specialite,namnInc,nrInc) VALUES (@namn, @nr, @specialite, @namnInc,
@nrInc);";
    MySqlCommand sqlCmd = new MySqlCommand(addString, dbcon);
    sqlCmd.Parameters.AddWithValue("@namn", namn);
    sqlCmd.Parameters.AddWithValue("@nr", nr);
    sqlCmd.Parameters.AddWithValue("@specialite", specialite);
    sqlCmd.Parameters.AddWithValue("@namnInc", namnInc);
    sqlCmd.Parameters.AddWithValue("@nrInc", nrInc);
    int rows = sqlCmd.ExecuteNonQuery();
    dbcon.Close();
}
```

Figur 2.3

Views:

För att visa datan på webbsidan finns redan en funktion som skapar tabellen "DesinfoSpridare". På grund av "return RedirectToAction" i "HomeController" (se figur 2.2) uppdateras "DesinfoSpridare" tabellen med ny data.

3. En dropdown

En dropdown implementerades på "Report" tabellen för att kunna hitta incidenten efter titel på rapporten (se figur 3.1).

Find incident name after the title:

mormor

▼

Show results

titel	namnIncident
gula roboter	snabb prestanda

Figur 3.1

HomeController:

En metod deklaration som heter "ChooseTitel" som tar parametern "Choose" skapades för att implementera en dropdown (se figur 3.2). TempData är en temporär datalagringsmekanism som används för att överföra data från en kontrolleråtgärd till en annan. I den här raden sätts värdet av "Choose" i TempData. Efter det utförs en omdirigering till "Repport" i samma kontroller.

När användaren väljer en titel i dropdown listan, metoden "ChooseTitel" anropas och tar den emot ett värde ("Choose") från användaren och lagrar det i "TempData".

```
public IActionResult ChooseTitel(string Choose)
{
    TempData["title"] = Choose;
    return RedirectToAction("Repport");
}
```

Figur 3.2

För att kontrollera om det finns någon data som har överförts från en tidigare åtgärd "ChooseTitel" med hjälp av "TempData", modifierades metoden "Repport" och skapades if satsen för att kontrollera om "TempData["title"]" inte är null (se figur 3.3). Om det är inte null, visas resultatet på webbsidan med hjälp av "ViewBag.TitelSearch" som definieras i Repport vyn.

```
public IActionResult Repport()
{
    CustomerModel dm4 = new CustomerModel(_configuration);
    DataTable repportData = dm4.Repport();
    ViewBag.RepportResults = repportData;
    if (TempData["title"] != null)
    {
        DataTable repportData2 =
        _customerModel.ChooseTitel(TempData["title"].ToString());
        ViewBag.TitelSearch = repportData2;
    }
    return View();
}
```

Figur 3.3

CustomerModel:

I "CustomerModel" implementeras metoden "ChooseTitel", som tar en titel som parameter för att anropa SQL procedure "getIncNamnByTitel2(@titel)" och returnerar en "DataTable" som heter "repportTable" med titel och incident namn (se figur 3.4).

```

public DataTable ChooseTitel(string titel)
{
    MySqlConnection dbcon = new MySqlConnection(_connectionString);
    dbcon.Open();
    MySqlDataAdapter adapter = new MySqlDataAdapter("CALL
getIncNamnByTitel2(@titel)", dbcon);
    adapter.SelectCommand.Parameters.AddWithValue("@titel", titel);
    DataSet ds4 = new DataSet();
    adapter.Fill(ds4, "result");
    DataTable repportTable = ds4.Tables["result"];

    dbcon.Close();
    return repportTable;
}

```

Figur 3.4

Views:

I "Repport" vyn skapades html kod som möjliggör för användare att välja ett värde från en dropdown-lista och skicka detta val till servern för att hämta ytterligare data (se figur 3.5)

Med den här del av koden ".<form method="post" action="@Url.Action("ChooseTitel", "Home")" skapades HTML-formulär som skickar data till servern med en HTTP POST-förfrågan när användaren skickar formuläret.

Med "<select name="Choose">:" skapas en HTML-dropdown lista med namnet "Choose".

"Foreach" används för att iterera över varje rad i "ViewBag.RepportResults".

Funktionen inuti loopen möjliggör att användaren får välja en titel i dropdown menyn. Om det inte finns några resultat att visa, så visas istället texten "No search results found."

```

@if (ViewBag.RepportResults != null &&
((System.Data.DataTable)ViewBag.RepportResults).Rows.Count > 0)
{
    <form method="post" action="@Url.Action("ChooseTitel", "Home")">
        <select name="Choose">
            @foreach (var row in ViewBag.RepportResults.Rows)
            {
                <option value="@row["titel"]">@row["titel"]</option>
            }
        </select>
        <input type="submit" value="Show results" />
    </form>
}
else{
    <p>No search results found</p>
}

```

Figur 3.5

Om det finns data som har överförs från en tidigare åtgärd "ChooseTitel" i "HomeController" (se figur 3.3), visas datan i en tabell med användarens valda titel på webbsidan (se figur 3.4). En "for"-loop itererar över varje kolumn i datatabellen tills den hittar användarens valda titel och visar raden med data som hämtats med hjälp av en SQL-procedure.

```
@if (ViewBag.TitelSearch != null &&
((System.Data.DataTable)ViewBag.TitelSearch).Rows.Count > 0)
{
    <table class="my-table">
        <tr>
            @foreach (var dataColumn in ViewBag.TitelSearch.Columns)
            {
                <th>@dataColumn.ColumnName</th>
            }
        </tr>
        @foreach (var row in ViewBag.TitelSearch.Rows)
        {
            <tr>
                @for (int i = 0; i < ViewBag.TitelSearch.Columns.Count; ++i)
                {
                    <td>@row[i]</td>
                }
            </tr>
        }
    </table>
}
else
{
    <p>No search results found</p>
}
```

Figur 3.4

4. Fritextsökning

Fritextsökning implementerades i "DesinfoSpridare" tabellen för att möjliggöra sökning av agenter efter namn (se figur 4.1). Användaren omdirigeras sedan till en ny sida som visar all information om agenten (se figur 4.2).

Search Agent by name:

Name:

Figur 4.1

Search Agent by name				
namn	nr	specialite	namnInc	nrInc
C	3	blicka	snabb prestanda	2
Home				

Figur 4.2

HomeController:

En ny insats av "CustomerModell" skapades (se figur 4.3). Efter det anropas "SearchCustomer" metoden i "CustomerModel" klassen med det namn som användaren har angivits. Resultatet av sökningen sparas i variabeln "desSpridData" och den sparas i "ViewBag.SearchResults", vilket möjliggör för vyn att få åtkomst till sökresultaten. Namnet som användaren har angivit i sökningen sparas i "ViewBag" under namnet "namn". Slutligen returnerar metoden vyn.

```
public IActionResult SearchCustomer(string name)
{
    CustomerModel dm2 = new CustomerModel(_configuration);
    DataTable desSpridData = dm2.SearchCustomer(name);
    ViewBag.SearchResults = desSpridData;
    ViewBag.namn = name;
    return View();
}
```

Figur 4.3

CustomerModel:

I "CustomerModel" en metod "SearchCustomer" som tar ett "name" som parameter skapades för att det blir möjligt att utföra fritextsökning (se figur 4.4). "@agentsnamn" sätts i SQL-frågan för att undvika SQL-injektioner och använda det angivna namnet. "adapter.SelectCommand.Parameters.AddWithValue("@agentsnamn", name);" har skapats för att definiera en parameter i SQL-frågan som är kopplad till det angivna parameter "name". SQL-frågan utförs och resultaten fylls i "DataSet" under namnet "result" och resultatet sparas i en DataTable "AgentensTable". Slutligen "AgentensTable" returneras med sökresultaten.

```
public DataTable SearchCustomer(string name)
{
    MySqlConnection dbcon = new MySqlConnection(_connectionString);
    dbcon.Open();
    MySqlDataAdapter adapter = new MySqlDataAdapter("SELECT * FROM
DesinfoSpridare WHERE namn=@agentsnamn;", dbcon);
    adapter.SelectCommand.Parameters.AddWithValue("@agentsnamn", name);
    DataSet ds2 = new DataSet();
    adapter.Fill(ds2, "result");
    DataTable AgentensTable = ds2.Tables["result"];
    dbcon.Close();
    return AgentensTable;
}
```

Figur 4.4

Views:

Först en HTML kod för fritextsökning i "Index2" vyn skapades (se figur 4.5). Den tar en användarens input som lagras som "name" och med hjälp av POST metoden omredigeras användaren till "SearcCustomer" vyn.

```
<h3>Search Agent by name:</h3>
<form method="post" action="@Url.Action("SearchCustomer", "Home")">
    <label>
        Name:
        <input type="text" name="name" value="" placeholder="Agents Name" />
    </label>
    <input type="submit" value="Search" />
</form>
```

Figur 4.5

En ny vyn som heter "SearcCustomer" (se figur 4.6) skapades för att visa resultatet på webbsidan. Den följer samma logiken som tidigare tabellen på "Report.cshtml" (se figur 1.8). Html Razor syntaxen omredigerar användaren till "Home" page som är en huvud sida.

```
<h2>Search Agent by name</h2>
@if (ViewBag.SearchResults != null &&
    ((System.Data.DataTable)ViewBag.SearchResults).Rows.Count > 0)
{
    <table class="my-table">
        <tr>
            @foreach (var dataColumn in ViewBag.SearchResults.Columns)
            {
                <th>@dataColumn.ColumnName</th>
            }
        </tr>
        @foreach (var row in ViewBag.SearchResults.Rows)
        {
            <tr>
                @for (int i = 0; i < ViewBag.SearchResults.Columns.Count; ++i)
                {
                    <td>@row[i]</td>
                }
            </tr>
        }
    </table>
}
else
{
    <p>No search results found</p>
}
@Html.ActionLink("Home", "Index", "Home", new { title = "Go back to search form" })
```

Figur 4.6

5. En förändring av innehållet i databasen med hjälp av DELETE (uppfyller krav till "generera en länk", 8)

För att implementera "delete" funktionaliteten en ny kolumn med namnet "Action" som innehåller länkar skapades (se figur 5.1).

Desinformation Spridare Agents					
namn	nr	specialite	namnInc	nrInc	Action
A	1	skapar bra lögner	snabb prestanda	1	Delete
B	2	snurra	danser	1	Delete
C	3	blicka	snabb prestanda	2	Delete
M	5	sover alltid	konstig	6	Delete

Figur 5.1

HomeController:

Här skapades metoden "DeleteAgent" för att ta bort en agent genom att anropa en metod i "CustomerModel" som hanterar databasoperationen. Sedan omdirigeras användaren till en samma sida. Det görs för att visa uppdaterat "Report" tabellen (se figur 5.2).

```
public IActionResult DeleteAgent(string namn, string nr)
{
    _customerModel.DeleteAgent(namn, nr);
    return RedirectToAction("Index2");
}
```

Figur 5.2

CustomerModel:

Metoden "DeleteAgent" är definierad i "CustomerModel" (se figur 5.2). Den tar parametrarna "namn" och "nr", vilka är primära nycklar i tabellen "DesinfoSpridare". "string deleteString" innehåller SQL-frågan som tar bort en rad med de angivna värdena för "namn" och "nr". Med "sqlCmd.Parameters..." sätts värdet av "@namn" till "namn" och värdet av "@nr" till "nr". Detta används för att förhindra SQL-injektioner.

```

public void DeleteAgent(string namn, string nr)
{
    MySqlConnection dbcon = new MySqlConnection(_connectionString);
    dbcon.Open();
    string deleteString = "DELETE FROM DesinfoSpridare WHERE namn=@namn
AND nr=@nr;";
    MySqlCommand sqlCmd = new MySqlCommand(deleteString, dbcon);
    sqlCmd.Parameters.AddWithValue("@namn", namn);
    sqlCmd.Parameters.AddWithValue("@nr", nr);
    int rows = sqlCmd.ExecuteNonQuery();
    dbcon.Close();
}

```

Figur 5.3

Views:

För att visa en ytterligare kolumn i "DesinfoSpridare" tabellen lades en "Action" kolumn till (se figur 5.4). En länk med texten "Delete" skapades, och när den klickas, leder den till "DeleteAgent" åtgärden (som är definierad i "CustomerModel") i "HomeController". Parametrarna "namn" och "nr" används för att identifiera den agent som ska tas bort.

```

th>Action</th>

<td>@Html.ActionLink("Delete", "DeleteAgent", "Home", new { namn = row["namn"],
nr = row["nr"] }, new { title = "Click to delete agent " + row["namn"] })</td>

```

Figur 5.4

6. En procedur från databasen

För att exekvera en procedur från databasen, skapades ytterligare en tabell "Field Reports" (se figur 6.1). I den här tabellen ska rapporter med typ "1" flytas till från "Report" tabellen.

Field Reports

datum	titel	komment	typ
1/7/2020 12:00:00 AM	stora ögon	fuktigt	1
9/1/2020 12:00:00 AM	gröna kaniner	springer runt	1

Figur 6.1

HomeController:

En metod som heter "MoveReport" deklarerades. Den tar "datumIn", "titelIn" och "typNr" som parametrar eftersom de är främmande nycklar till "Field reports" tabellen (se figur 6.2). Metoden utförs flyttningen av en rapport i databasen. Efter att flytten är klar, omdirigeras användaren till en annan sida "FaltRapport" som innehåller data från databasen.

```
public IActionResult MoveReport(string datumIn, string titelIn, string typNrIn)
{
    _customerModel.MoveReport(datumIn, titelIn, typNrIn);
    return RedirectToAction("FaltRapport");
}
```

Figur 6.2

CustomerModel:

Metoden "MoveReport" definierades för att utföra flyttningen av rapporten genom att anropa en lagrad procedur i databasen (se figur 6.3). "string deleteString" anropar procedur "flyttaFältRap" som tar in "@datumIn", "@titelIn", "@typNrIn" som parametrar.

```
public void MoveReport(string datumIn, string titelIn, string typNrIn)
{
    MySqlConnection dbcon = new MySqlConnection(_connectionString);
    dbcon.Open();
    string deleteString = "CALL
flyttaFältRap(@datumIn,@titelIn,@typNrIn)";
    MySqlCommand sqlCmd = new MySqlCommand(deleteString, dbcon);
    sqlCmd.Parameters.AddWithValue("@datumIn", datumIn);
    sqlCmd.Parameters.AddWithValue("@titelIn", titelIn);
    sqlCmd.Parameters.AddWithValue("@typNrIn", typNrIn);
    int rows = sqlCmd.ExecuteNonQuery();
    dbcon.Close();
}
```

Figur 6.3

Views:

En vy "FaltRapport" skapades för att visa data från "Field Reports" tabellen (se figur 6.4). HTML kod implementerades på ett samma sätt som "DesinfoSpridare" och "Repport" tabellerna.


```

@{
    ViewData["Title"] = "Field Reports";
}
<h1>@ViewData["Title"]</h1>

@if (ViewBag.AvslRapFältrapport != null &&
    ((System.Data.DataTable)ViewBag.AvslRapFältrapport).Rows.Count > 0)
{
    <table class="my-table">
        <tr>
            @foreach (var dataColumn in ViewBag.AvslRapFältrapport.Columns)
            {
                <th>@dataColumn.ColumnName</th>
            }
        </tr>
        @foreach (var row in ViewBag.AvslRapFältrapport.Rows)
        {
            <tr>
                @for (int i = 0; i < ViewBag.AvslRapFältrapport.Columns.Count; +
+i)
                {
                    <td>@row[i]</td>
                }
            </tr>
        }
    </table>
}
else
{
    <p>No search results found</p>
}

```

Figur 6.4

7. Visa innehållet från 2 olika databastabeller

Två tabellerna har beskrivits i kapitel 1.

Den tredje tabellen har beskrivits i kapitel 6.

8. Generera länkar utifrån databasinnehållet.

Kravet uppfylls i kapitel 5. En förändring av innehållet i databasen med hjälp av DELETE implementerades med en länk.

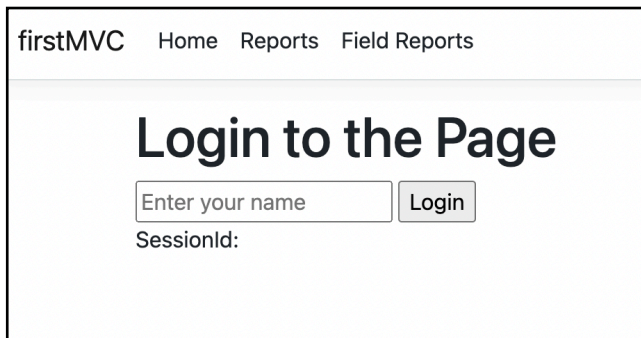
9. VG - implementera en procedur från din databas som tar emot en parameter

Kravet uppfylls i kapitel 3. I "CustomerModel" implementeras metoden "ChooseTitel", som tar en "titel" som parameter för att anropa SQL procedure "getIncNamnByTitel2(@titel)" (se figur 3.4).

Kravet uppfylls i kapitel 6. "string deleteString" anropar procedur "flyttaFältRap" som tar in "@datumIn","@titelIn","@typNrIn" som parametrar (se figur 6.3).

10. VG - En inloggnings funktionalitet som använder sig av C# sessionshantering

För att skapa inloggnings funktionalitet ändrades logiken i vyn "Index". En ändring var att byta namnet på huvudvyn till "Index2" för att göra "Index" till en inloggnings sida. Eftersom det inte längre finns några krav på inloggning, implementerades enkel inloggnings funktionalitet med användarnamn som input (se figur 10.1). När användaren skriver in sitt användarnamn och klickar på "Log in" knappen, omdirigeras användaren till huvudsidan "Index2".



firstMVC Home Reports Field Reports

Login to the Page

SessionId:

Figur 10.1

HomeController:

En metod "Index" deklarerades (se figur 10.2). Först hämtas värdet för användarnamn från en sessionsvariabel och tilldelas till "ViewBag.Username." Efter det hämtas sessionens ID från en sessionsvariabel och tilldelas till "ViewBag.SessionId."

"_logger.LogInformation" används för att logga information om användarens "Username" och session ID.

```
public IActionResult Index()
{
    ViewBag.Username =
HttpContext.Session.GetString(SessionVariables.SessionKeyUsername);
    ViewBag.SessionId =
HttpContext.Session.GetString(SessionVariables.SessionKeySessionId);

    _logger.LogInformation("Username:{session}",
HttpContext.Session.GetString(SessionVariables.SessionKeyUsername));
    _logger.LogInformation("SessionId:{session}",
HttpContext.Session.GetString(SessionVariables.SessionKeySessionId));

    return View();
}
```

Figur 10.2

Samma kod används i vyn "Index2" för att visa information om användarens namn och Session ID.

En till metod som heter "Login" implementerades i "HomeController" (se figur 10.3). Detta är en kontroll åtgärd som anropas när en användare försöker logga in i applikationen. Den tar en parameter "username", vilket är användarnamnet som användaren anger vid inloggningen.

Sessions används för att lagra "username" i sessionen. Detta gör att användarnamnet kan behållas över flera sidor.

"SessionVariables.SessionKeyUsername" används för att identifiera denna specifika session variabel.

Efter inloggning omredigeras användaren till "Index2".

```
public IActionResult Login(string username)
{
    HttpContext.Session.SetString(SessionVariables.SessionKeyUsername,
    username);
    HttpContext.Session.SetString(SessionVariables.SessionKeySessionId,
    Guid.NewGuid().ToString());

    _logger.LogInformation("Username:{session}",
    HttpContext.Session.GetString(SessionVariables.SessionKeyUsername));
    _logger.LogInformation("SessionId:{session}",
    HttpContext.Session.GetString(SessionVariables.SessionKeySessionId));

    return RedirectToAction("Index2");
}
```

Figur 10.3

"Logout" metoden deklarerades i "HomeController" (se figur 10.4). "Session.Clear" används för att rensas sessionen helt. Alla data som lagrats i sessionen tas bort. Detta är en typisk procedur för att logga ut en användare.

Efter ett användaren loggas ut, omredigeras den till "Index".

```
public IActionResult Logout()
{
    HttpContext.Session.Clear();

    _logger.LogInformation("Username:{session}",
    HttpContext.Session.GetString(SessionVariables.SessionKeyUsername));
    _logger.LogInformation("SessionId:{session}",
    HttpContext.Session.GetString(SessionVariables.SessionKeySessionId));

    return RedirectToAction("Index");
}
public static class SessionVariables
{
    public const string SessionKeyUsername = "Username";
    public const string SessionKeySessionId = "SessionId";
}
```

Figur 10.4

Views:

Html koden skapades i en vy som heter "Index"(se figur 10.5). Det finns en fritextfält som tar ett användarens namn och en knapp för att kunna logga in.

```
<h1>Login to the Page</h1>

<form method="post" action="@Url.Action("Login", "Home")">
    <input type="text" name="username" placeholder="Enter your name" />
    <input type="submit" value="Login" />
</form>
```

Figur 10.5

I vyn "Index2" skapades HTML koden för att kontrollera om användaren är inloggad (se figur 10.6) . Om användaren inte är inloggad, visas ett meddelande "You are not logged in!". Annars visas meddelandet "Welcome [username]!" och "Current Session ID" med alla data som finns i "Index2" vyn. Det finns också en "Logout" knapp som användaren kan klicka på för att logga ut och omdirigeras till inloggningssidan.

```
@if (ViewBag.username == null)
{
    <div>You are not logged in!</div>
}
else
{
    <h1>Welcome @ViewBag.username !</h1>
    <div>
        <div>Current Session Id: <span>@ViewBag.SessionId</span></div>
    </div>
    <form method="post" action="@Url.Action("Logout", "Home")">
        <input type="submit" value="Logout" />
    </form>
}
<hr />
```

Figur 10.6

Program.cs

Flera rader har lagts till "Program.cs" filen för att möjliggöra användning av sessions (se figur 10.7). Detta säkerställer att sessionshantering är aktiverad.

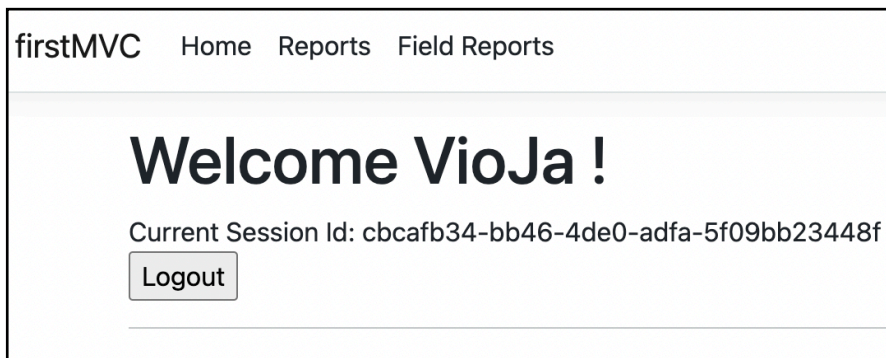
"IdleTimeout" betyder att 10 minuter kommer sessionen att vara aktiv. Om användaren inte interagerar med applikationen under den tiden kommer sessionen att avslutas.

"app.UseSession();" används för att aktivera sessionshantering och göra sessionsdata tillgängliga för applikation.

```
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(10);
});
.....
app.UseSession();
.....
```

Figur 10.7

Efter en lyckad inloggning visas den här vyn på webbsidan (se figur 10.8).



Figur 10.8