.NET (https://cloud.google.com/dotnet/) Guides

Using SQL Server with .NET

This part of the .NET Bookshelf tutorial shows how the sample app stores its persistent data in Microsoft SQL Server running on a Compute Engine VM.

This page is part of a multipage tutorial. To start from the beginning and read the setup instructions, go to .NET Bookshelf app (https://cloud.google.com/dotnet/docs/getting-started/tutorial-app).

Create the SQL Server instance

1. Create a new Compute Engine instance.

   GO TO THE VM INSTANCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/COMPUTE/INSTANCES)

2. For **Instance ID**, enter `library`.
3. For **Zone**, select `us-west1-a`.
4. In the **Boot Disk** section, click **Change**.
5. On the **Application Images** tab, select the **SQL Server 2014 Standard on Windows Server 2012 R2** image, and then click **Select**.
6. To create the VM instance, click **Create**.
7. When it's ready, click the name of the instance. It can take a few minutes for the instance to be ready. When the instance is ready, it is visible in the instances list.
8. On the **Instance details** page, click **Set Windows password**.
9. To create the user account on your instance, enter a username of your choice, and then click **Set**. Make a note of the provided password and close the dialog.

Create the SQL Server database

1. In the VM instances list, click the **RDP** link next to your SQL Server instance. Sign in with the username and password you set up during instance creation.
2. In the **Windows Start** menu, enter `SQL Server 2014 Manage`.
3. Right-click **SQL Server 2014 Management Studio** and select **Run as administrator**.
4. In the **Connect to Server** window, click **Connect**.
5. Right-click **Databases** and select **New database**.
6. Name the database `bookshelf`, and then click **OK**.

Configure the SQL Server

1. In SQL Server 2014 Management Studio, click the **Security** folder for the `library` SQL Server instance.
2. Right-click **Logins** and select **New login**.
3. For **Login name**, enter `dotnetapp`.
4. For the **Authentication** method, click **SQL Server Authentication**.
5. For **Password**, enter a password of your choice. Don't enable the **Enforce password policy** option.
6. Change the **Default Database** to be the `bookshelf` database you previously created.
7. In the left side of the **New login** dialog, click **User Mapping** and complete the following steps:
    a. For the `bookshelf` database, click the **Map** checkbox.
    b. Under **Database role membership for: bookshelf**, click all of the roles except for the following:
        - `db_denydatareader`
        - `db_denydatawriter`
8. To create the new database login account, click **OK**.
9. The user you created is set to use SQL Server authentication so SQL Server needs to be configured to allow this authentication method. In **SQL Server 2014 Management Studio** right-click the **Library** SQL Server instance you created and select **Properties**.

10. In the left-side menu of the Server Properties dialog, click **Security**.
11. For the **Server Authentication** setting, select **SQL Server and Windows Authentication mode**, and then click **OK**.
12. To restart the SQL Server service, right-click the **Library** SQL Server instance and select **Restart**.

Create a firewall rule for SQL Server

Configure a firewall rule to allow traffic on port 1433 so other clients can connect to the newly created SQL Server instance over the internet.

1. In the Google Cloud Console, go to the **Firewall rules** section.

   OPEN THE FIREWALL RULES (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/NETWORKING/FIREWALLS/

2. Click **Add firewall rule** and complete the following fields:
   a. For the new **Firewall Rule Name**, enter `allow-tcp-1433`.
   b. For **Source Filter**, select **IP Ranges**.
   c. To allow access for all IP addresses, for **Source IP Ranges**, enter `0.0.0.0/0`.

   > **Warning:** This configuration leaves your SQL Server instance open to traffic from everyone, everywhere. It is used only for demonstration purposes. In production environments, restrict access to only those IP addresses that need access.

   d. For **Allowed protocols and ports**, enter `tcp:1433`.
3. To create the firewall rule, click **Create**.

Configuring settings

1. To open the sample app in Visual Studio, in the `getting-started-dotnet\aspnet\2-structured-data` directory, double-click **2-structured-data.sln**.
2. In the **Solution Explorer** pane, click **Web.config**.
3. In `Web.config`, complete the following steps:
   a. Set `GoogleCloudSamples:ProjectId` to your project ID.

b. Set `GoogleCloudSamples:BookStore` to `sqlserver`.

c. Near the bottom of the file, under `<connectionStrings>`, find the `connectionStrings` XML sub-element with the attribute `name="LocalSqlServer"`. Update the **connectionString** value with the external IP address, database name, username, and password of your SQL Server instance. For example, the **connectionString** for a remote SQL Server on IP `104.155.20.171` with `database = bookshelf`, `user = dotnetapp` and `password = test` looks like this:

```
connectionString="Data Source=104.155.20.171;Initial
Catalog=bookshelf;Integrated Security=False;User
ID=dotnetapp;Password=test;MultipleActiveResultSets=True"
```

4. Save and close `Web.config`.
5. To build the solution, in the Visual Studio menu, click **Build** and then select **Build Solution**.
6. To create the database tables, in the Visual Studio menu, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**. At the `PM >` prompt, enter the following command:

```
Add-Migration Init
```

7. Create the tables in the SQL Server database that are used to store the books' data for the Bookshelf app. In the Package Manager Console, enter the following command:

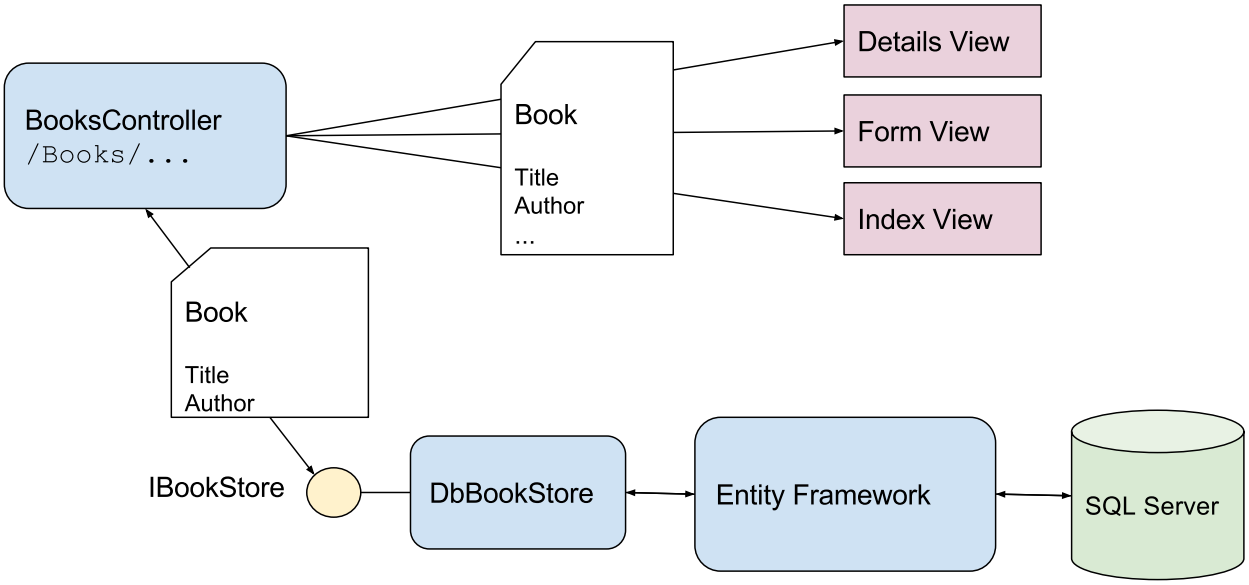
```
Update-Database
```

Running the app on your local machine

In Visual Studio, press **F5** to run the project. Now you can browse the app's web pages to add, edit, and delete books.

App structure

This diagram shows the app's components and how they fit together. The app follows the classic ASP.NET MVC (http://www.asp.net/mvc) pattern. An `IBookStore` interface between the `BooksController` and `DbBookStore` lets you switch to storing book data in Datastore without changing any code.



Understanding the code

This section walks you through the app's code and explains how it works.

The data model

The `Book` class contains information about one book as well as additional fields that are used in later tutorials:

aspnet/2-structured-data/Models/Book.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/2-structured-data/Models/Book.cs)

ATFORM/GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/2-STRUCTURED-DATA/MODELS/BOOK.CS)

```
[Bind(Include = "Title, Author, PublishedDate, Description")]
public class Book
{
    [Key]
    public long Id { get; set; }

    [Required]
    public string Title { get; set; }
```

```
        public string Author { get; set; }

        [Display(Name = "Date Published")]
        [DataType(DataType.Date)]
        public DateTime? PublishedDate { get; set; }

        public string ImageUrl { get; set; }

        [DataType(DataType.MultilineText)]
        public string Description { get; set; }

        public string CreatedById { get; set; }
    }
```

Entity Framework's **DbSet** (https://msdn.microsoft.com/en-us/library/gg696460(v=vs.113).aspx) converts LINQ (https://msdn.microsoft.com/en-us/library/bb397926.aspx) queries and **Create**, **Read**, **Update**, and **Delete** (CRUD) operations into SQL queries. The **ApplicationDbContext** class keeps a **DbSet** of Books.

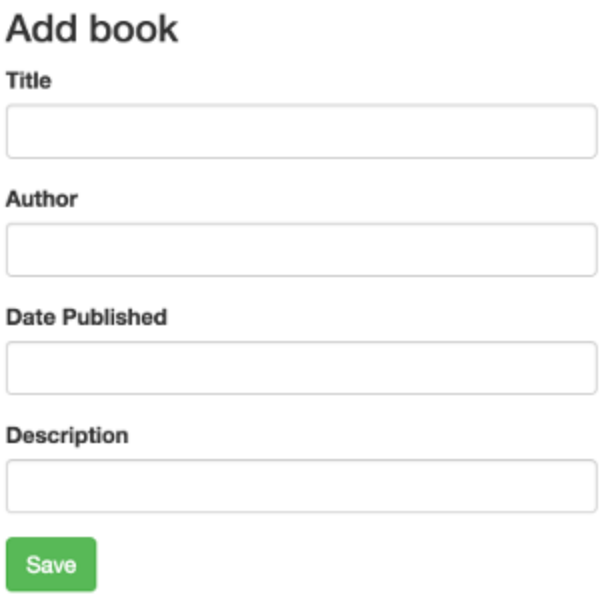
aspnet/2-structured-data/Models/ApplicationDbContext.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/2-structured-data/Models/ApplicationDbContext.cs)

TARTED-DOTNET/BLOB/MASTER/ASPNET/2-STRUCTURED-DATA/MODELS/APPLICATIONDBCONTEXT.CS)

```
public class ApplicationDbContext : DbContext
{
    // ...
    public DbSet<Book> Books { get; set; }
```

Handling user submissions with forms

The add/edit HTML form allows users to add and edit book submissions.



The HTML form is created using Razor (http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c), templates. This Razor template specifies that the form include text input fields for Title, Author, Date Published, and Description:

aspnet/2-structured-data/Views/Books/Form.cshtml (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/2-structured-data/Views/Books/Form.cshtml)

TTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/2-STRUCTURED-DATA/VIEWS/BOOKS/FORM.CSHTML)

```
<form action="/Books/@Model.FormAction/@Model.Book.Id" method="post" id="book-form"
    @Html.AntiForgeryToken()
    <div class="form-group">
        @Html.LabelFor(model => model.Book.Title)
        @Html.EditorFor(model => model.Book.Title, new { htmlAttributes = new { @cla
        @Html.ValidationMessageFor(model => model.Book.Title, "", new { @class = "te
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Book.Author)
        @Html.EditorFor(model => model.Book.Author, new { htmlAttributes = new { @cl
        @Html.ValidationMessageFor(model => model.Book.Author, "", new { @class = "t
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Book.PublishedDate)
        @Html.EditorFor(model => model.Book.PublishedDate, new { htmlAttributes = ne
```

```
        @Html.ValidationMessageFor(model => model.Book.PublishedDate, "", new { @cla
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Book.Description)
        @Html.EditorFor(model => model.Book.Description, new { htmlAttributes = new
        @Html.ValidationMessageFor(model => model.Book.Description, "", new { @class
    </div>

    <button type="submit" class="btn btn-success">Save</button>
</form>
```

Handling form submissions

When you click **Add Book**, the `BooksController.Create()` method displays the form. After the form is filled and you click **Save**, the `BooksController.Create()` method receives the form's contents and sends them to the SQL Server Database via the `IBookStore::Create()` method. Note that the `Create` method is annotated with `HttpPost`.

aspnet/2-structured-data/Controllers/BooksController.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/2-structured-data/Controllers/BooksController.cs)

ARTED-DOTNET/BLOB/MASTER/ASPNET/2-STRUCTURED-DATA/CONTROLLERS/BOOKSCONTROLLER.CS)

```
        // GET: Books/Create
        public ActionResult Create()
        {
            return ViewForm("Create", "Create");
        }

        // POST: Books/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create(Book book)
        {
            if (ModelState.IsValid)
            {
                _store.Create(book);
                return RedirectToAction("Details", new { id = book.Id });
            }
            return ViewForm("Create", "Create", book);
        }
```

The `DbBookStore` class invokes the `ApplicationDbContext` class to perform queries and CRUD operations for data stored in the SQL Server database. The SQL query is created using an Object-relational mapper (http://wikipedia.org/wiki/Object-relational_mapping) (ORM) called Entity Framework (https://msdn.microsoft.com/en-us/data/ef.aspx). Object-relational mappers let you write data models as simple C# classes, and they generate all the SQL for you.

`DbBookStore`'s CRUD methods, such as `Create()`, are simple calls to the `ApplicationDbContext` class.

aspnet/2-structured-data/Models/DbBookStore.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/2-structured-data/Models/DbBookStore.cs)

GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/2-STRUCTURED-DATA/MODELS/DBBOOKSTORE.CS)

```
public void Create(Book book)
{
    var trackBook = _dbcontext.Books.Add(book);
    _dbcontext.SaveChanges();
    book.Id = trackBook.Id;
}
```

Listing books

After you add books, click the **Books** link to go to the `/Books` page, which lists all the books currently stored in the SQL Server database. The `List()` method does the work of listing all the books by using data retrieved from the database.

aspnet/2-structured-data/Models/DbBookStore.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/2-structured-data/Models/DbBookStore.cs)

GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/2-STRUCTURED-DATA/MODELS/DBBOOKSTORE.CS)

```
public BookList List(int pageSize, string nextPageToken)
{
    IQueryable<Book> query = _dbcontext.Books.OrderBy(book => book.Id);
    if (nextPageToken != null)
    {
        long previousBookId = long.Parse(nextPageToken);
        query = query.Where(book => book.Id > previousBookId);
    }
```

```
    var books = query.Take(pageSize).ToArray();
    return new BookList()
    {
        Books = books,
        NextPageToken = books.Count() == pageSize ? books.Last().Id.ToString() : nul
    };
}
```

The `List()` method composes a LINQ query that reads books from the books' `DbSet`. The query gets a little complicated in order to implement paging. The query reads ten books and then stores the `Id` for the last book in `NextPageToken`. When you click the **More** button, the `List()` method unpacks the `nextPageToken` to get the `Id` of the last book and then queries for books with larger IDs.

< PREV (HTTPS://CLOUD.GOOGLE.COM/DOTNET/DOCS/GETTING-STARTED/USING-STRUCTURED-DATA)

NEXT > (HTTPS://CLOUD.GOOGLE.COM/DOTNET/DOCS/GETTING-STARTED/USING-CLOUD-STORAGE)

Last updated December 4, 2019.