Install

1. Unzip DemoApp.zip in desired folder
2. In root folder find files - SqlScript1_CreateDatabases.sql and SqlScript2_CreateDatabaseSchema.sql
3. Connect to SQL Server instance as administrator.
4. First execute SqlScript1_CreateDatabases.sql – this will create 3 databases
5. Next execute SqlScript2_CreateDatabaseSchema.sql for each one of the 3 databases
6. Open DemoApp.sln with Visual Studio
7. Open WebConfig file in root folder and if needed change the name of SQL Server instance in DefaultConnection, by default it is localhost\SQLEXPRESS. Please do not add database name it will be added dynamically.

Usage

1. Run the application;
2. Register new User with email and password;
3. Login with newly created user;
4. Go to Pictures page and click “Create new” link to add new image.
5. After adding a few images you can browse, edit and delete them.
6. To log off click the link in top right corner.

Solution

- As per requirements I am using ASP.NET MVC (classic approach, without JavaScript frameworks), which I hope is just fine for the purpose of this demo.
- As I suppose that we should deal with large data and I give up to use Entity Framework for data access, instead I am using ADO.NET command and DataReader in this demo, which is faster and may be more appropriate in this case.
- I choose to use Unity for IoC. In the demo I am using property injection which just saves me some typing to write Interfaces, but construction injection is also good choice.
- The main problem is which algorithm for distributing users evenly to the database instances to chose.

	advantages	drawbacks
We can Get user's email, hash it and compute the remainder after dividing of the number of databases which will give us the number of our database (for example - AppDemoDb2)	This approach is straightforward, it will distribute users almost evenly between databases and I implement this approach in the demo.	The drawback is when we want to scale and add another database we will need to import all data from old databases to new databases.
We can use something like regions. For example all users with emails from *.com domain will be in first db. Users with .eu, .fr, .de, .bg ect. Will be in second db and all other user in third db.	If second database grow we can move only users from .eu to new db and only change application configuration.	This approach will not distributing users evenly to the database we can end up with very large first db and small third db
We can give users URL with sub domain in addition to email and password and use sub domain to find out users database	You can always check the number of users and give the new user sub domain that point to db with min number of users that way we will distribute users evenly between databases	This approach may not fit in our case, because the customers will prefer to use own domain