Темы урока

На этом уроке мы **завершаем** подготовку слоя доступа к данным на базе MS SQL Server через ADO.NET и подключаем новую реализацию DAL к нашему чат-боту.

Вводная часть

Обзор результатов прошлого урока

На прошлом уроке мы реализовали 5 методов интерфейса `IReminderStorage`, начиная с хранимых на SQL Server процедур и заканчивая модульными тестами для проверки работоспособности нашей реализации:

- Schema.sql
- SPs.sql
- Data.sql

Список подготовленных методов:

- ✓ `Add(ReminderItem);`
- ✓ `Get(Guid)`
- ✓ `Get(ReminderItemStatus)`
- ✓ `UpdateStatus(ReminderItem, ReminderItemStatus);`
- ✓ `UpdateStatus(IEnumerable<ReminderItem>, ReminderItemStatus);`

Планы на сегодня

1. Мы подключим нашу сборку к чат-боту и убедимся, что всё работает в точности так, как мы задумали (ReminderItem-ы кладутся в базу данных MS SQL Server).
2. Мы реализуем *правильный* метод обновления статусов для группы напоминаний (не через `foreach` единичного апдейта): `UpdateStatus(IEnumerable<ReminderItem>, ReminderItemStatus)`.
3. Мы реализуем все оставшиеся методы интерфейса `IReminderStorage`.

Собственно, девелопмент :)

Подключаем новую реализацию DAL к чат-боту:

Подключим к нашему проекту `Reminder.Storage.WebApi` сборку `Reminder.Storage.SqlServer.ADO` вместо сборки `Reminder.Storage.InMemory`.

Вспоминаем, что, **поскольку мы реализовали грамотную архитектуру с низкой связностью компонентов**, наше приложение чат-бота вообще не потребует никаких изменений. Если посмотреть на диаграмму связей компонентов на презентации, видно, что со слоем данных приложение общается через выставленный Web API.

Так что изменения коснуться только сборки `Reminder.Storage.WebApi`, и то, в одном единственном месте: там где мы назначаем конкретную реализацию нашему интерфейсу IReminderStorage:

Класс `Startup`, метод `ConfigureServices`:

Заменить

```
services.AddSingleton<IReminderStorage>(new InMemoryReminderStorage());
```

на

```
services.AddSingleton<IReminderStorage>(new SqlReminderStorage(
        @"Data Source=localhost\SQLEXPRESS;Initial Catalog=Reminder;Integrated Security=true;"
));
```

Не забываем привести зависимости в актуальное состояние, ссылка на проект `Reminder.Storage.InMemory` больше не актуальна, зато надо добавить ссылку на проект `Reminder.Storage.SqlServer.ADO`.

Однако, хардкодить строку подключения — не самое изящное решение. Поэтому мы её вынесем в настройки. Добавим новую секцию в файл `appsettings.json`, в итоге он будет выглядеть как-то так:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection":
      "Data Source=localhost\\SQLEXPRESS;Initial Catalog=Reminder;Integrated Security=true;"
  }
}
```

При этом в файле `Startup.cs` необходимо обновить код с тем, чтобы считывать строку подключения из файла настройки конфигурации:

```
public void ConfigureServices(IServiceCollection services)
{
	services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

	services.AddSwaggerGen(c =>
	{
		c.SwaggerDoc(
			"v1",
			new OpenApiInfo
			{
				Title = "Reminders API",
				Version = "V1"
			});
	});

	string connectionString = new ConfigurationBuilder()
		.SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
		.AddJsonFile("appsettings.json")
		.Build()
		.GetConnectionString("DefaultConnection");

	services.AddSingleton<IReminderStorage>(new SqlReminderStorage(connectionString));
}
```

Запускаем наш Web API, потом приложение, тестируем, радуемся.

Доработка Reminder.Storage.SqlServer.ADO

Временные таблицы и обновление группы значений через SqlBulkCopy

1. Рассказать кратенько про временные таблицы с одной решёткой (можно совсем вскользь, лишь для возбуждения интереса, упомянуть, что есть временные таблицы с двумя решётками).
2. Пишем SQL-код: хранимую процедуру, которая рассчитывает на то, что существует временная таблица #ReminderItem:

```
DROP PROCEDURE IF EXISTS [dbo].[UpdateReminderItemsBulk]
GO
CREATE PROCEDURE [dbo].[UpdateReminderItemsBulk] (
	@statusId AS TINYINT
)
AS BEGIN
	SET NOCOUNT ON

	UPDATE R
		SET R.[StatusId] = @statusId,
		R.[UpdatedDate] = SYSDATETIMEOFFSET()
	FROM [dbo].[ReminderItem] AS R
	INNER JOIN #ReminderItem AS T
		ON T.Id = R.Id
END
GO
```

3. Пишем реализацию работы с такой хранимой процедурой:

```
public void UpdateStatus(IEnumerable<Guid> ids, ReminderItemStatus status)
{
	using (var sqlConnection = GetOpenedSqlConnection())
	{
		var cmd = sqlConnection.CreateCommand();

		// create temp table
		cmd.CommandType = CommandType.Text;
		cmd.CommandText = "CREATE TABLE #ReminderItem([Id] UNIQUEIDENTIFIER NOT NULL)";
		cmd.ExecuteNonQuery();

		// fill it with ids
		using (SqlBulkCopy copy = new SqlBulkCopy(sqlConnection))
		{
			copy.BatchSize = 1000;
			copy.DestinationTableName = "#ReminderItem";

			DataTable tempTable = new DataTable("#ReminderItem");
			tempTable.Columns.Add("Id", typeof(Guid));

			foreach (Guid id in ids)
			{
				DataRow row = tempTable.NewRow();
				row["Id"] = id;
				tempTable.Rows.Add(row);
			}

			copy.WriteToServer(tempTable);
		}

		// run query to bulk update
		cmd.CommandType = CommandType.StoredProcedure;
		cmd.CommandText = "[dbo].[UpdateReminderItemsBulk]";
		cmd.Parameters.AddWithValue("@statusId", (byte)status);
		cmd.ExecuteNonQuery();

		// drop temp table
		cmd.CommandType = CommandType.Text;
		cmd.CommandText = "DROP TABLE #ReminderItem";
		cmd.ExecuteNonQuery();
	}
}
```

Можно задать вопрос: “А будет ли это работать при параллельном вызове таких методов?” Правильный ответ: “Да, будет, так как временная таблица с одной решёткой - живёт изолированно в рамках сессии подключения. И хранимая процедура запускается в рамках этой же сессии. А поскольку соединение в каждом вызове нашего метода устанавливается новое — при каждом вызове будет создаваться своя “невидимая” никому за пределами этого подключения таблица #ReminderItem. И именно ней будет работать хранимая процедура вызванная из в этом же подключении”.

Пэйджинг в SQL-запросах: ROW_NUMBER() OVER()

```
--

DROP PROCEDURE IF EXISTS [dbo].[GetReminderItemsWithPaging]
GO
CREATE PROCEDURE [dbo].[GetReminderItemsWithPaging] (
	@count AS INT = NULL,
	@startPosition AS INT = NULL
)
AS BEGIN
	SET NOCOUNT ON

	IF @count IS NULL OR @count < 1
		SELECT @count = COUNT(*)
		FROM [dbo].[ReminderItem]

	IF @startPosition IS NULL OR @startPosition < 1
		SET @startPosition = 1;

	SELECT
		[Id],
		[ContactId],
		[TargetDate],
		[Message],
		[StatusId]
	FROM (
		SELECT
			[Id],
			[ContactId],
			[TargetDate],
			[Message],
			[StatusId],
			ROW_NUMBER() OVER (ORDER BY [TargetDate] DESC) AS RowNumber
		FROM [dbo].[ReminderItem]
	) AS TableWithRowNumbers
	WHERE RowNumber BETWEEN @startPosition AND @startPosition + @count - 1
	ORDER BY RowNumber ASC
END
GO
```

```
DROP PROCEDURE IF EXISTS [dbo].[GetReminderItemsByStatusWithPaging]
GO
CREATE PROCEDURE [dbo].[GetReminderItemsByStatusWithPaging] (
	@statusId AS TINYINT,
	@count AS INT = NULL,
	@startPosition AS INT = NULL
)
AS BEGIN
	SET NOCOUNT ON

	IF @count IS NULL OR @count < 1
		SELECT @count = COUNT(*)
		FROM [dbo].[ReminderItem]
		WHERE [StatusId] = @statusId

	IF @startPosition IS NULL OR @startPosition < 1
		SET @startPosition = 1;

	SELECT
		[Id],
		[ContactId],
		[TargetDate],
		[Message],
		[StatusId]
	FROM (
		SELECT
			[Id],
			[ContactId],
			[TargetDate],
			[Message],
			[StatusId],
			ROW_NUMBER() OVER (ORDER BY [TargetDate] DESC) AS RowNumber
		FROM [dbo].[ReminderItem]
		WHERE [StatusId] = @statusId
	) AS TableWithRowNumbers
	WHERE RowNumber BETWEEN @startPosition AND @startPosition + @count - 1
	ORDER BY RowNumber ASC
END
GO
```

Стремиться нужно к результату тут:
https://github.com/ago-cs/cs-course-q4/tree/master/Lessons/33/ClassWork/Final.

Домашнее задание

—