

MySQL Login and Registration

For GTA:Network - by AndreasB

Throughout this tutorial you'll learn to

- set up a new GTA Network Server project in Visual Studio,
- add references and NuGet packages to your project,
- use the [Insight.Database ORM](#) for MySQL communication,
- set up a new database and tables in MySQL using phpMyAdmin,
- code a secure user account system with login and registration using Bcrypt for password security
- learn how to deal with database stored procedures/routines

NOTE: I'm using [WAMPserver](#) to get MySQL and phpMyAdmin easily installed. It's a prerequisite that you have phpMyAdmin and a MySQL server running or understanding of how to apply this tutorial to another database and management tool. This tutorial will not go through setup of such.

Creating a New Project

Open Visual Studio, create a new project based on the Class Library template. Name the project **NewServer** (or anything else, but this is used throughout the tutorial). Hit OK.

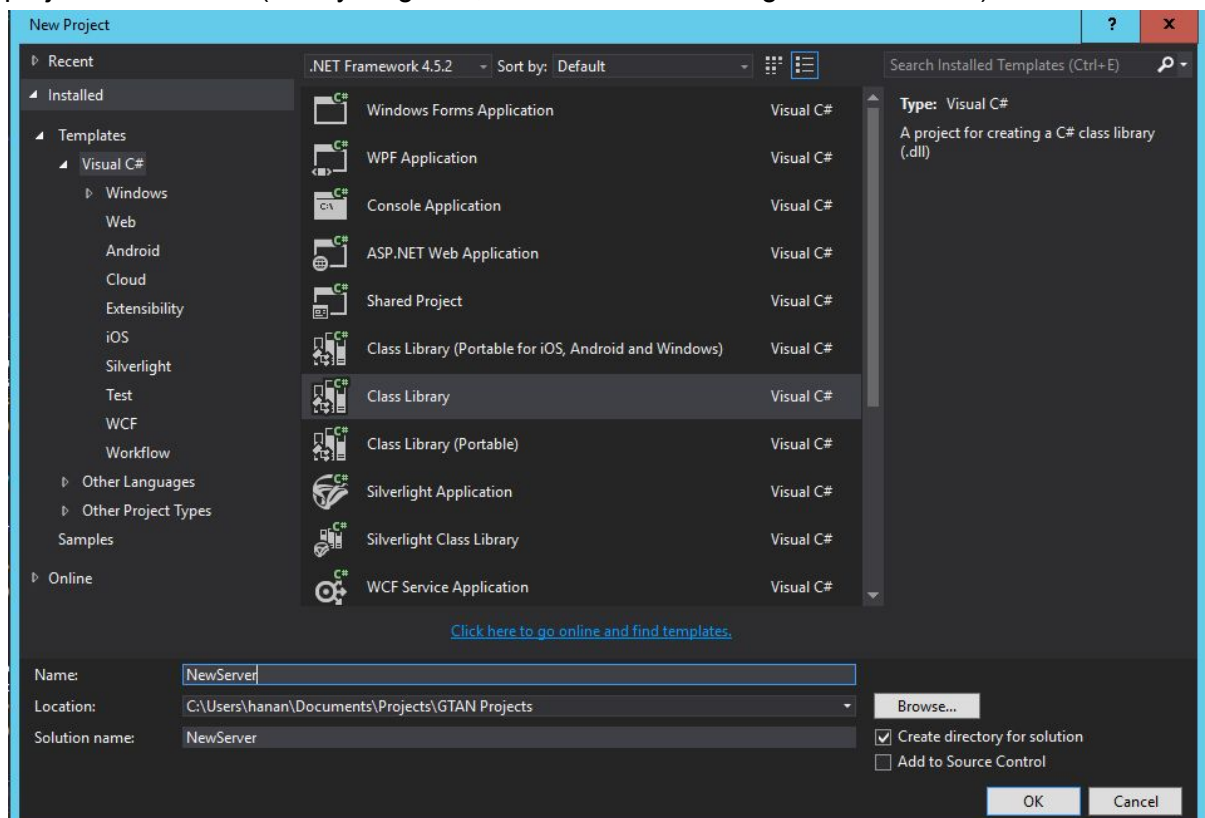


Figure 1: Project creation

After the new project has been made, we want to add two references to it. Both references are located in your **GTA Network Server** folder.

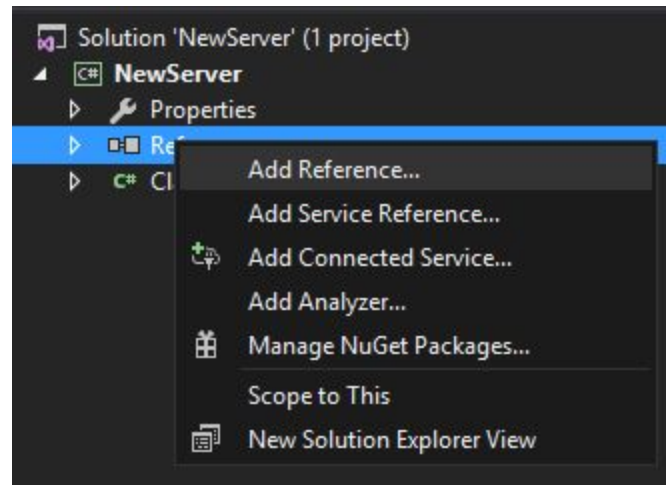


Figure 2: Adding new references

Go to the **Browse** view, hit the **Browse...** button, locate and add the GTANetworkServer.exe and GTANetworkShared.dll files, *tick* both of them, and click OK.

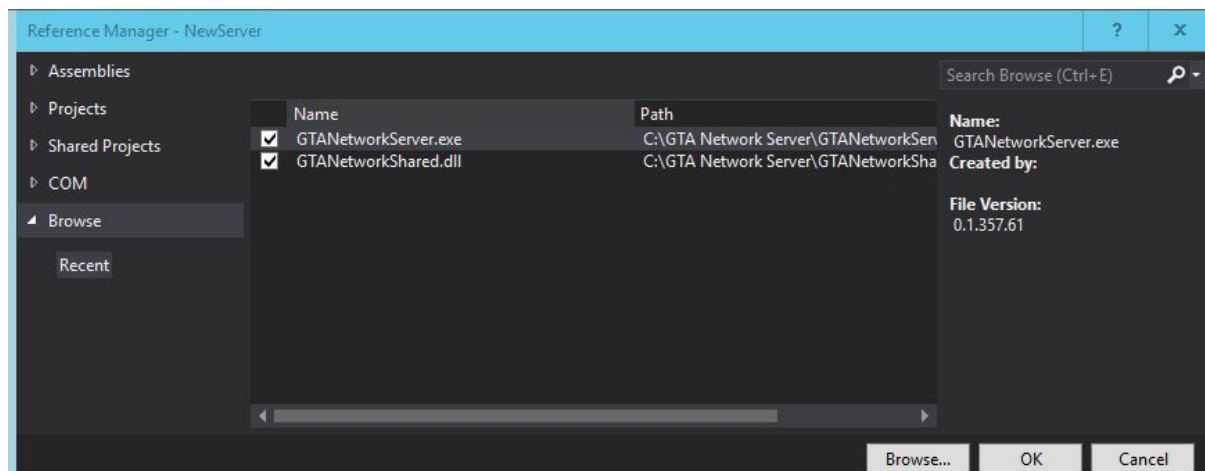


Figure 3: Adding references from GTA:Network

After adding the references, rename the default **Class1.cs** file into **Main.cs**. Hit **Yes** if prompted with the message in figure 5.

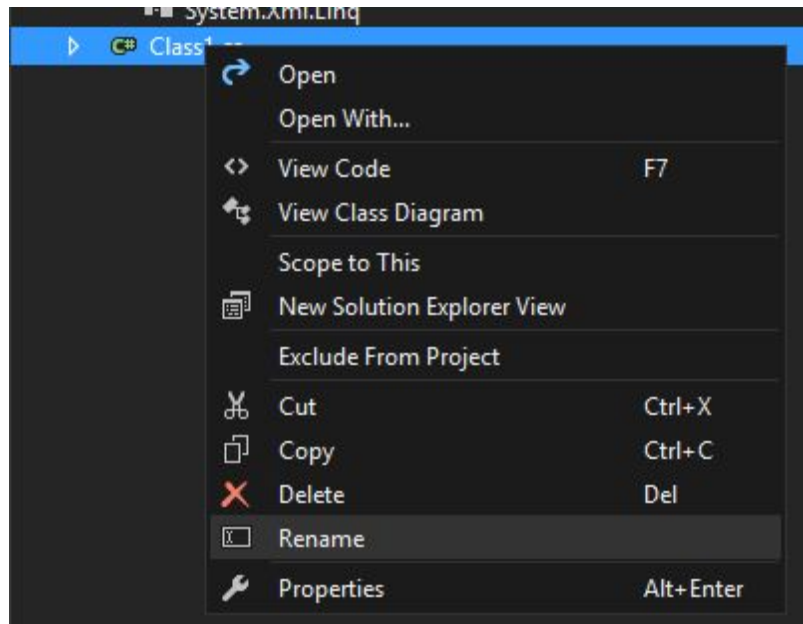


Figure 4: Renaming the Class1.cs file

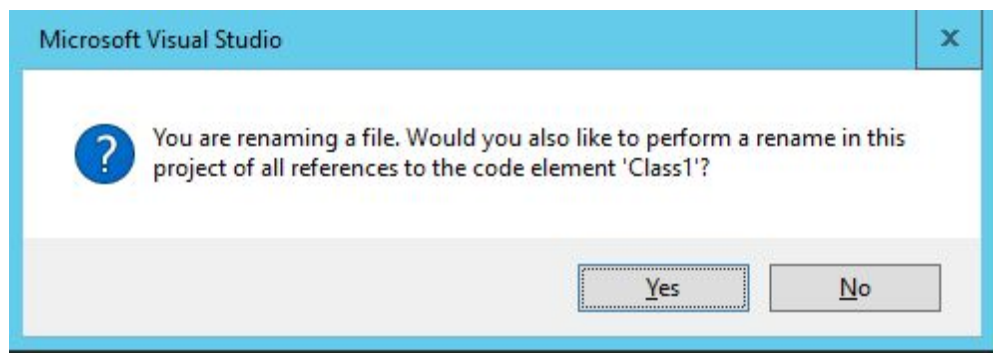


Figure 5: "Yes"

After renaming the main file, let's import the GTA:Network dependencies by adding

```
: Script
```

to our Main class. It'll show a red squiggly line, so click it and press `CTRL+.` (dot) and hit `ENTER` to add the **GTANetworkServer** using like below:

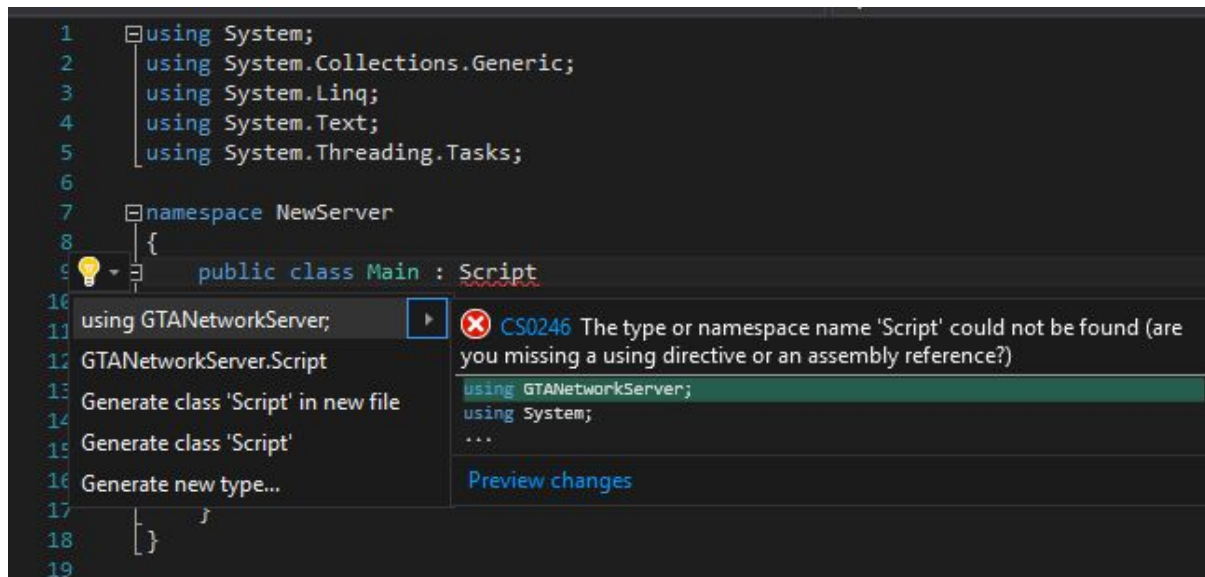


Figure 6: Adding the GTANetworkServer using

Add a corresponding `Main` method to the script, this will be used later.
Now let's start on the two commands,

- User login: `/ul <password>`
- User registration: `/ur <password>`.

Both methods are decorated with `[Command("...", GreedyArg = true)]` which allows GTA Network to understand that the following method will be called when the specified command is used.

```

namespace NewServer
{
    public class Main : Script
    {
        public Main()
        {
        }

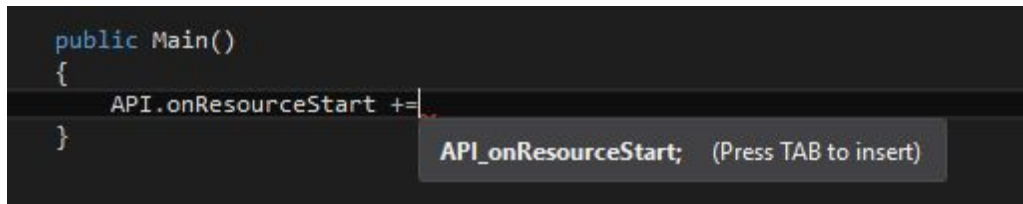
        [Command("ul", GreedyArg = true)]
        public void CMD_UserLogin(Client player, string password)
        {
        }

        [Command("ur", GreedyArg = true)]
        public void CMD_UserRegistration(Client player, string password)
        {
        }
    }
}

```

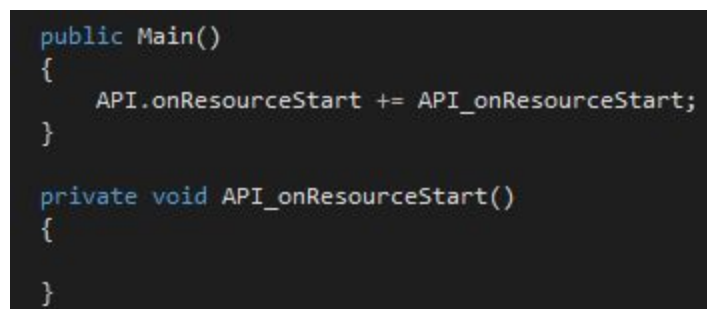
Figure 7: Boilerplate command methods

Inside our `Main()` method we specify a method for handling the `onResourceStart` event. We will later use this method to connect to our MySQL database.



```
public Main()
{
    API.onResourceStart +=
}
API_onResourceStart; (Press TAB to insert)
```

Figure 8: Creating the `onResourceStart` method (hit TAB for auto complete)



```
public Main()
{
    API.onResourceStart += API_onResourceStart;
}

private void API_onResourceStart()
{
}
```

Figure 9: The finished `onResourceStart` event handler

Adding our ORM (Insight.Database)

We can locate the Insight.Database ORM from the NuGet package manager. Start by opening the manager, then search for **Insight.Database**. Install both the highlighted packages (figure 11). The first package will install the Insight.Database ORM. The second package is the MySQL provider and connector.:

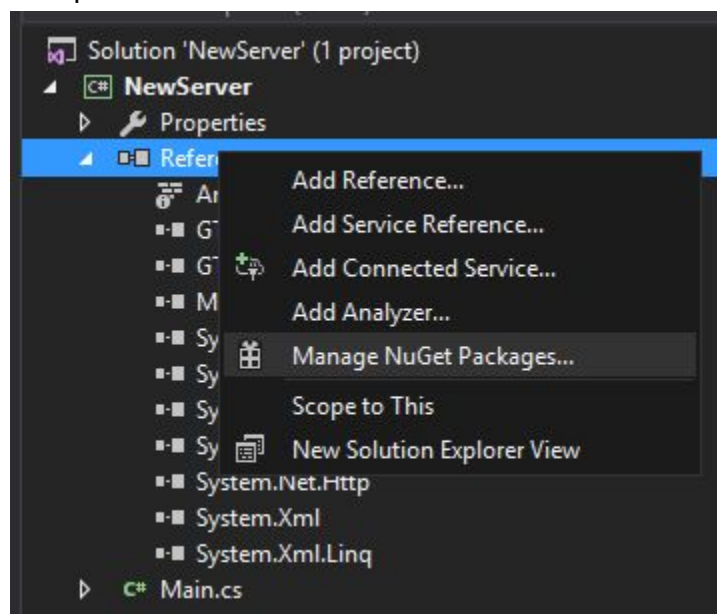


Figure 10: Open the NuGet Package Manager

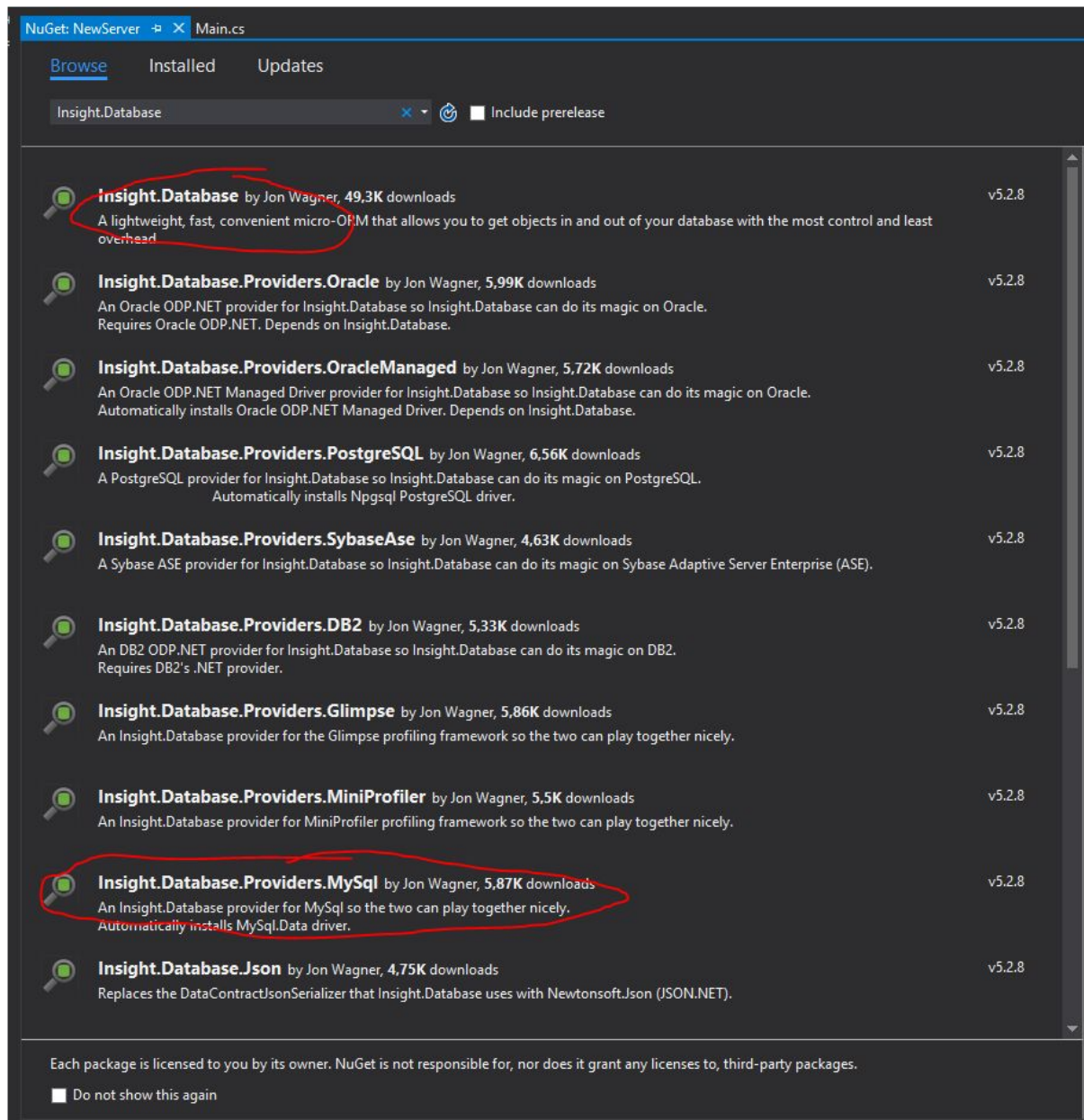


Figure 11: Search results for *Insight.Database*

Defining our repository and user account model

To keep it simple for the sake of the tutorial, the `UserAccount` model and the `User` repository are kept inside the same `Main.cs` file (but below and outside the `public class Main`). The `IUserRepository` interface is mapped to our database's stored procedures (more on that later), so we will make two methods inside our interface: `RegisterAccount` and `GetAccount`. Later we will make two stored procedures in MySQL with the same names.

The `UserAccount` class holds two properties: a player's `Username` and `Hash`. More on the `Hash` later, but think of it as a user's password in an unreadable form (*security*).

```

namespace NewServer
{
    public class Main ...

    public interface IUserRepository
    {
        UserAccount RegisterAccount(UserAccount userAccount);
        UserAccount GetAccount(string name);
    }

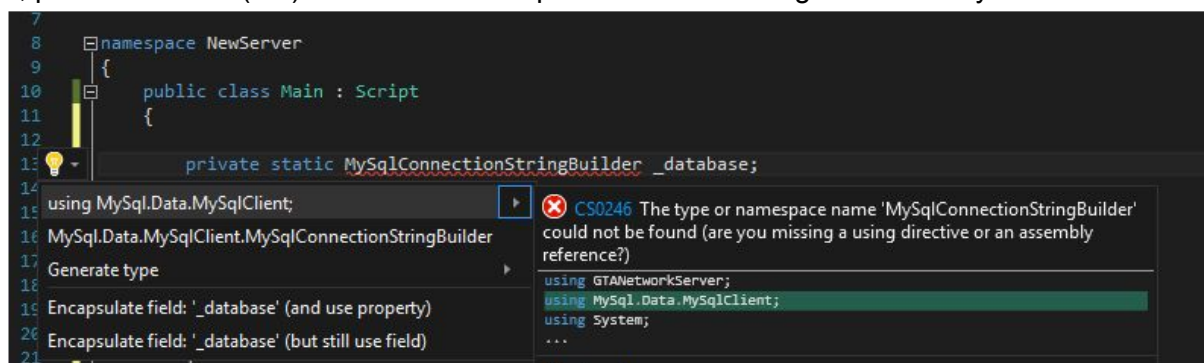
    public class UserAccount
    {
        public string Username { get; set; }
        public string Hash { get; set; }
    }
}

```

Figure 12: User repository and Account model

More on MySQL

To be able to use our database connection throughout the **Main.cs** file, we define it just inside the class definition. We call it `_database` and it's of type `MySQLConnectionStringBuilder`. Again, the red squiggly line means we can click on it, press `CTRL+. (dot)` and `ENTER` to import the correct using automatically.



```

7
8 namespace NewServer
9 {
10     public class Main : Script
11     {
12
13         private static MySQLConnectionStringBuilder _database;
14
15         using MySQL.Data.MySqlClient;
16         MySQL.Data.MySqlClient.MySQLConnectionStringBuilder
17         Generate type
18         Encapsulate field: '_database' (and use property)
19         Encapsulate field: '_database' (but still use field)
20
21

```

CS0246 The type or namespace name 'MySQLConnectionStringBuilder' could not be found (are you missing a using directive or an assembly reference?)

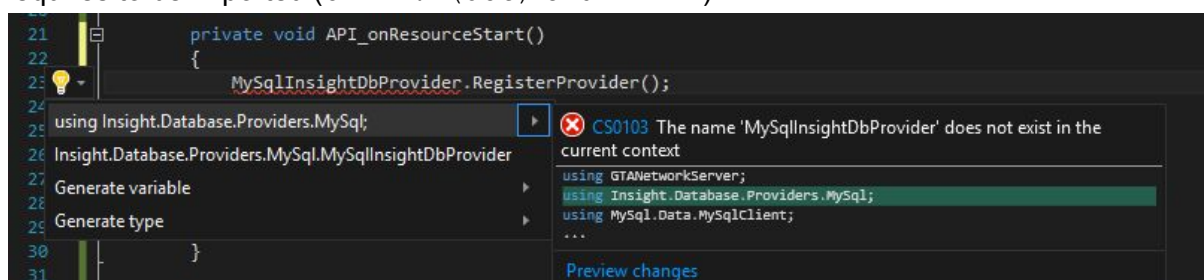
```

using GTANetworkServer;
using MySQL.Data.MySqlClient;
using System;
...

```

Figure 13 Defining our `_database` variable and importing the `MySQL` using

To tell our ORM (`Insight.Database`) that we want to use `MySQL`, we use the `registerProvider()` method available on `MySQLInsightDbProvider` which also requires to be imported (`CTRL+. (dot)` and `ENTER`):



```

21 private void API_onResourceStart()
22 {
23     MySQLInsightDbProvider.RegisterProvider();
24
25     using Insight.Database.Providers.MySql;
26     Insight.Database.Providers.MySql.MySQLInsightDbProvider
27     Generate variable
28     Generate type
29
30 }
31

```

CS0103 The name 'MySQLInsightDbProvider' does not exist in the current context

```

using GTANetworkServer;
using Insight.Database.Providers.MySql;
using MySQL.Data.MySqlClient;
...

```

Figure 14: Importing and registering the `MySQL` provider from `Insight.Database`.

Below the `_database` variable defined just inside our `Main` class from figure 13, define a `_userRepository` based on our `IUserRepository` from figure 12:

```
public class Main : Script
{
    private static MySqlConnectionStringBuilder _database;
    private static IUserRepository _userRepository;
```

Figure 15: Defining the `_userRepository`.

Now we build up the MySQL connection string using some default connection values. The database is named `newserver`, which we will create later in the tutorial. We assign the connection string builder to our previously `_database` variable.

Then we connect and map our `_userRepository` (`IUserRepository` interface) to the database, as illustrated below:

```
private void API_onResourceStart()
{
    MySqlInsightDbProvider.RegisterProvider();

    _database = new MySqlConnectionStringBuilder("server=localhost;user=root;database=newserver;port=3306;password=");

    _userRepository = _database.Connection().As<IUserRepository>();
}
```

Figure 16 Setting up connection string and database connection through interface.

You're likely to see some red squiggly lines... You should know what to do with those by now!

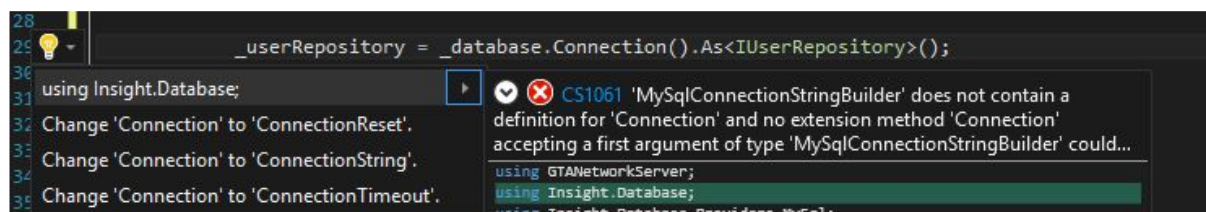


Figure 17: Importing `Insight.Database`.

Still you might see some squiggly lines. This is because we need `System.Configuration` to be referenced. Let's do so by opening the reference manager (similar to figure 2), and go to `Assemblies -> Framework` (instead of `browse` like in figure 2). Locate the `System.Configuration` assembly and tick it in the checkbox (not illustrated, but do so either way):

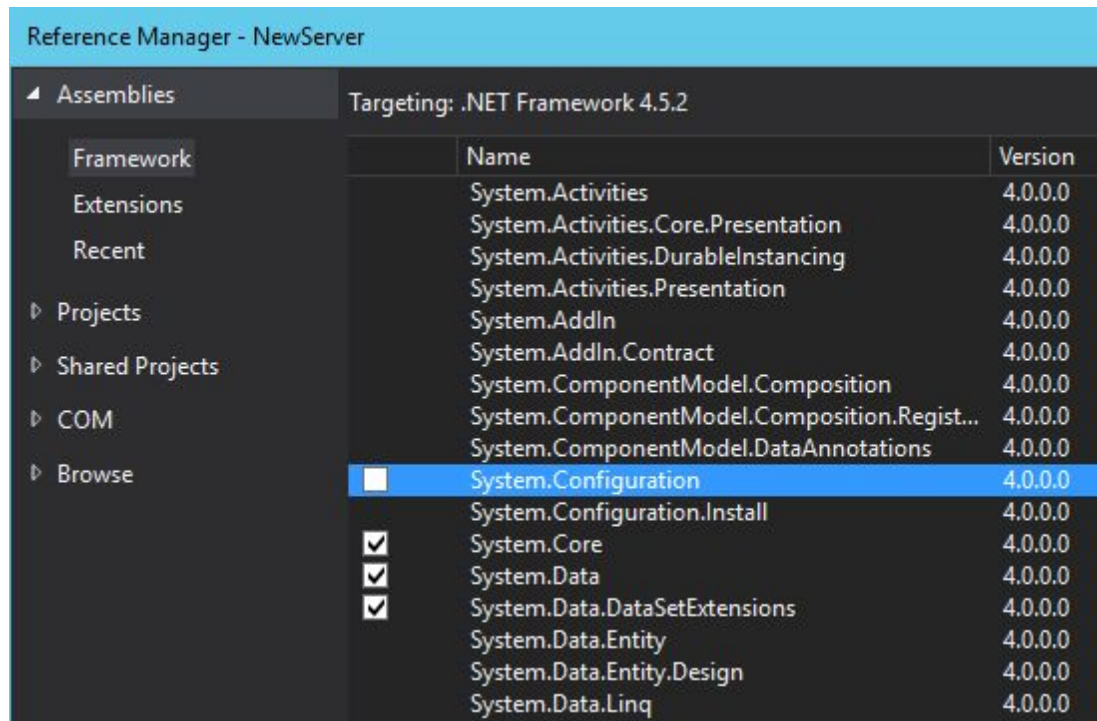


Figure 18: Adding a reference to System.Configuration

Adding BCrypt for password security

Open NuGet again (like in figure 10). Search for **bcrypt** and install the **BCrypt.Net** package:

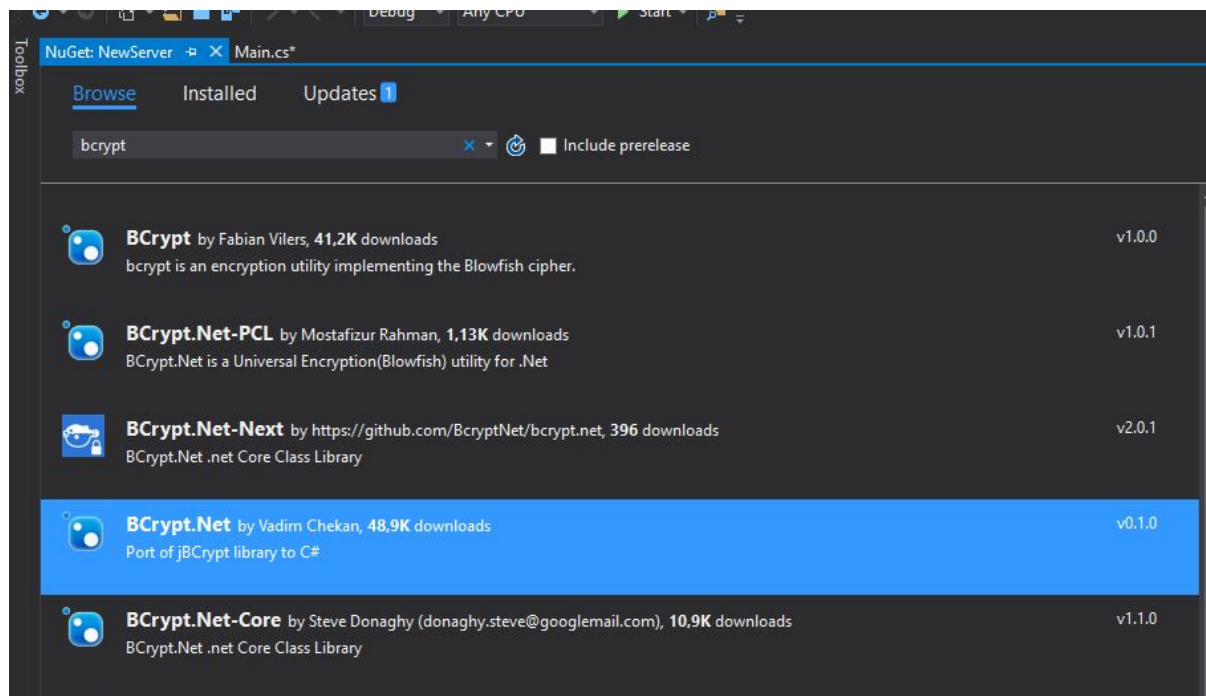
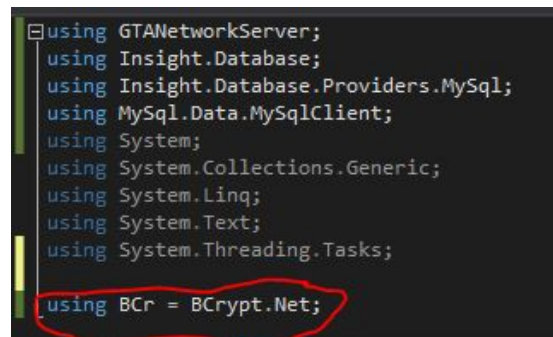


Figure 19: Install the highlighted package **BCrypt.Net**.

After installing the package, add the Bcrypt using to the top of the Main.cs file:

```
using BCr = BCrypt.Net;
```

...so it looks like this:



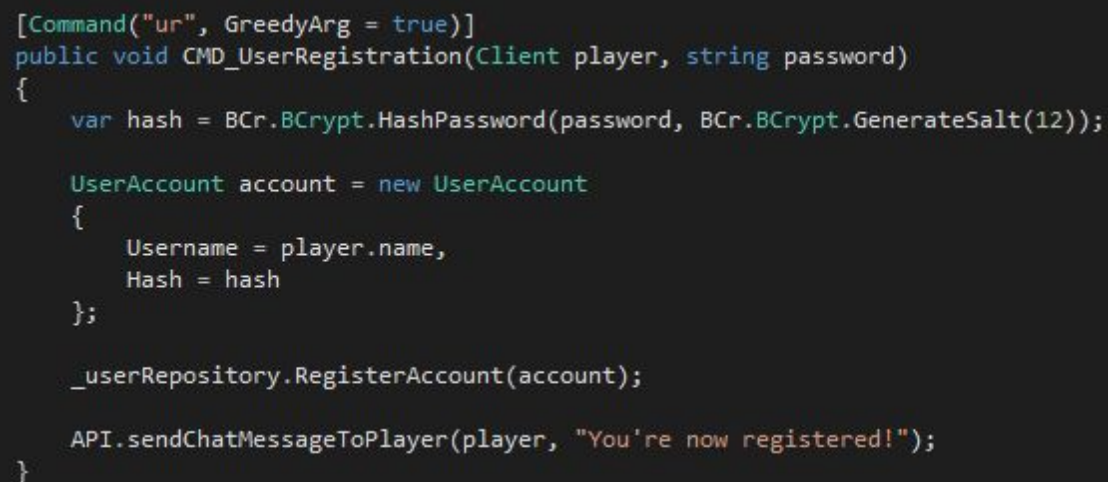
```
using GTANetworkServer;  
using Insight.Database;  
using Insight.Database.Providers.MySql;  
using MySql.Data.MySqlClient;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using BCr = BCrypt.Net;
```

Figure 20: Adding the BCrypt.net using

User Registration

Back to our `/ur` command handler! Since we do not want to store passwords in clear text, we use Bcrypt to generate a salted hash based on the user's inputted password and store the result in `var hash`. Then we build a new user account with our `UserAccount` model based on the player's name and the newly generated hash. Then we call our `_userRepository` and its `RegisterAccount` method where we pass in our new account. Then we display a chat message to the player, welcoming them as a registered user.

*NOTE: To keep this tutorial simple, we do not check for existing users with the same username, nor do we have any exception handling or password validation. You **should** do all of this.*



```
[Command("ur", GreedyArg = true)]  
public void CMD_UserRegistration(Client player, string password)  
{  
    var hash = BCr.BCrypt.HashPassword(password, BCr.BCrypt.GenerateSalt(12));  
  
    UserAccount account = new UserAccount  
    {  
        Username = player.name,  
        Hash = hash  
    };  
  
    _userRepository.RegisterAccount(account);  
  
    API.sendChatMessageToPlayer(player, "You're now registered!");  
}
```

Figure 21: Code to register a new user using the `/ur <password>` command in-game.

User Login

Back to our `/ul` command handler! We start by fetching the account from the database using the user repository's `GetAccount()` method passing in the name of the player doing

the `/ul` command. Then we use Bcrypt's `Verify()` method and pass in the password inputted by the user as well as the hash stored in the database for the respective user fetched. Bcrypt will figure out if the inputted password equals to the hash stored. The result of that operation is stored in the `isPasswordCorrect` bool. We send feedback to the player based on the result, either positive (You're logged in!) or negative (Wrong password).

NOTE: To keep things simple for the tutorial, no validation or exception handling is included here either.

```
[Command("/ul", GreedyArg = true)]
public void CMD_UserLogin(Client player, string password)
{
    UserAccount account = _userRepository.GetAccount(player.name);

    bool isPasswordCorrect = BCrypt.Verify(password, account.Hash);

    if (isPasswordCorrect)
    {
        API.sendMessageToPlayer(player, "You're now logged in!");
    }
    else
    {
        API.sendMessageToPlayer(player, "Wrong password!");
    }
}
```

Figure 22: Code to log in an existing user using the `/ul <password>` command in-game.

Your Main.cs file should now look like this:

<https://pastebin.gtanet.work/?a24ac30fc9dd91e8#YAulpcgQSn4t/hev1B2nGBCJO7j/mNtlnZv7W7G3iNE=>

Database - MySQL Setup

So far we've coded the login and registration part, but we've had no actual database to connect to. Log in to your phpMyAdmin dashboard (often located at <http://localhost/phpmyadmin/>), click the **New** button on the left, type `newserver` into the database name field and hit **Create**.



Figure 23: Database creation

Create a new table called `users`. For this tutorial we only need three columns (`id`, `username`, `hash`).

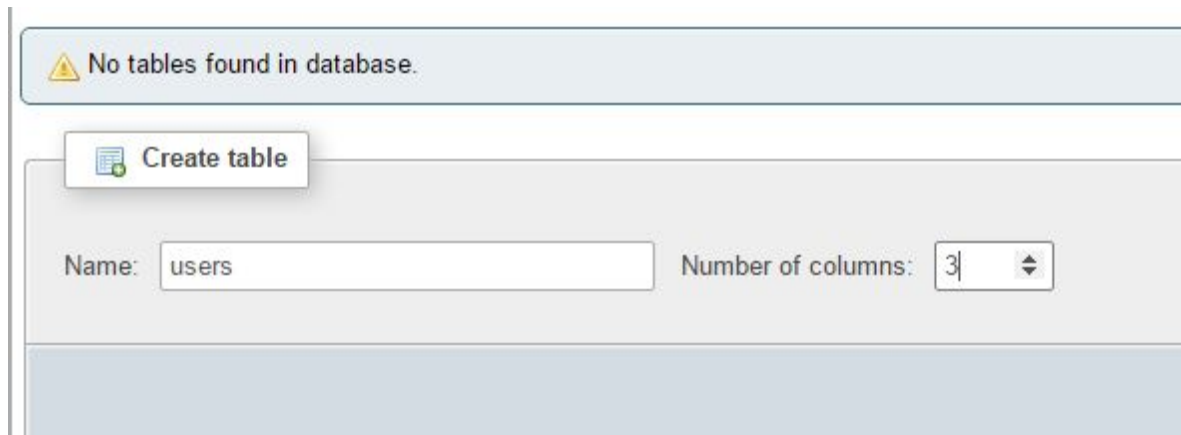
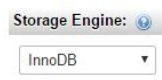


Figure 24: Users table creation

In the next step we define the table, which should look something like this :

- `id` as a primary index with A_I enabled (auto increment),
- `username` as a **varchar** with a certain length (best practice may vary),
- `password` as a **varchar** with a certain length. You can also use a binary type and way shorter length, but for the sake of tutorial simplicity you can rather do optimizations later



- Set your Storage Engine to *InnoDB*:

Hit **save**!

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I
id	INT		None			<input type="checkbox"/>	PRIMARY	PRIMARY <input checked="" type="checkbox"/>
username	VARCHAR	255	None			<input type="checkbox"/>	---	<input type="checkbox"/>
hash	VARCHAR	255	None			<input type="checkbox"/>	---	<input type="checkbox"/>

Figure 25: Users table should look like this ^

Database - Stored Procedures (Routines) Setup

Stored Procedures, also known as **Routines** are available at *database level* (not *table level*) in phpMyAdmin. This is where we define what the `RegisterAccount` and `GetAccount` methods from the `IUserRepository` should actually do, as these methods are automatically mapped to stored procedures in our database through the Insight.Database ORM. For reference, the mapping is displayed in figure 16.

Go here to start defining stored procedures:

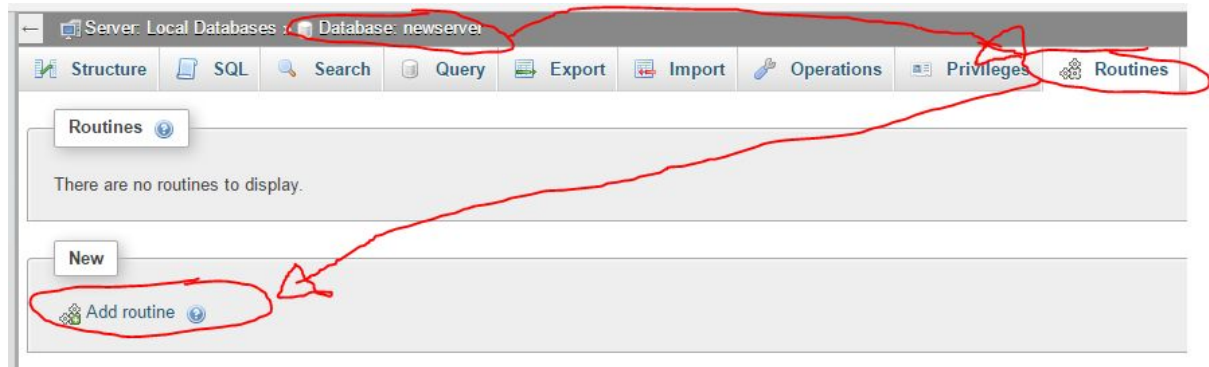


Figure 26: Go to your database, Routines tab and click “Add routine”

Routine 1: GetAccount

Name the procedure `GetAccount` and define a `name` parameter with `varchar` type and 255 length. The parameter names must match those from the repository methods (see figure 12). We insert the SQL query necessary to fetch the correct player based on their (user)name. Hit **Go** to store it.

A screenshot of the 'Details' tab in phpMyAdmin for defining a routine. The 'Routine name' field contains 'GetAccount'. The 'Type' dropdown is set to 'PROCEDURE'. Under the 'Parameters' section, a table shows one parameter: 'name' of type 'VARCHAR' with a length of '255'. Below this, the 'Definition' field contains the SQL query: '1 SELECT * FROM users WHERE username = name'.

Direction	Name	Type	Length/Values	Options
IN	name	VARCHAR	255	Char

Figure 27: Defining our GetAccount procedure/routine.

Routine 2: RegisterAccount

Name the procedure `RegisterAccount` and define `name` and `hash` parameters, both of type `varchar` with length 255. Insert the SQL query necessary to store our new user. As you might understand by now, these parameters are passed into the routine by our ORM which maps our `UserAccount` model's properties into the parameters defined in the routine. Hit **Go** to save.

The screenshot shows a 'Details' tab for a routine named 'RegisterAccount'. The 'Type' is set to 'PROCEDURE'. Under the 'Parameters' section, there is a table with two parameters: 'username' and 'hash', both of type 'VARCHAR' with a length of 255. Below the parameters table is an 'Add parameter' button. The 'Definition' section contains the following SQL code:

```
1 INSERT INTO users (username, hash)
2 VALUES (username, hash)
```

Figure 28: Defining our `RegisterAccount` procedure/routine.

Back to Visual Studio

meta.xml

For GTA:Network to understand that we're trying to (in the end) run a server *resource*, we must define a `meta.xml` file with some content. Start out by adding a new xml file in Visual Studio (*Right click project -> Add -> New Item*):

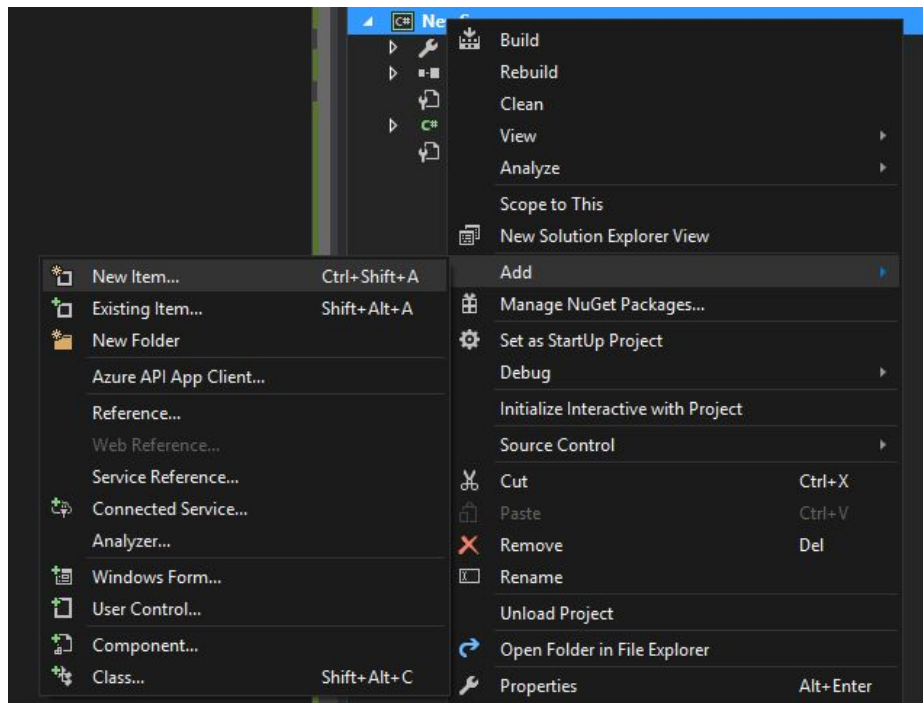


Figure 29: Opening the “New Item” window.

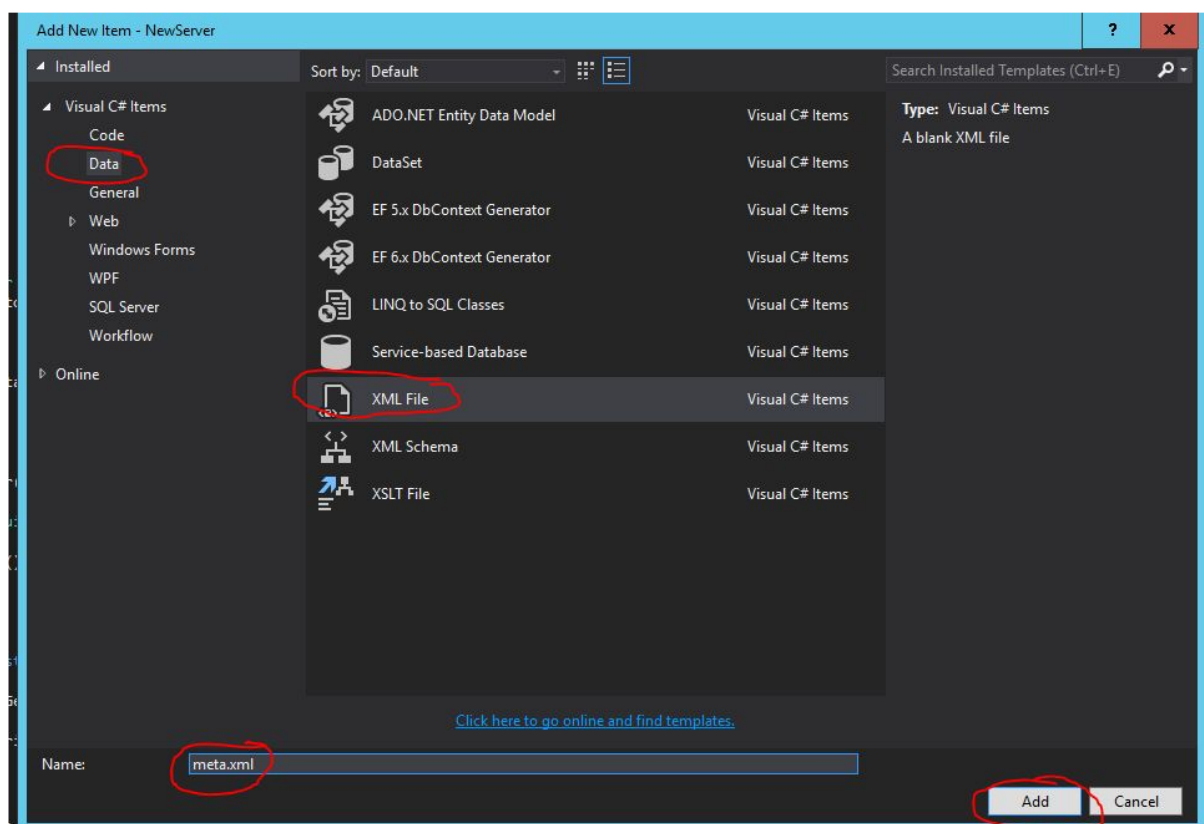


Figure 30: Go to Data, locate XML File, call it **meta.xml** and hit **Add**.

The meta.xml file should contain (for this tutorial) the following (change `author=""`):

```

<meta>
  <info name="newserver" description="Boilerplate MySQL with registration and
login" author="AndreasB" type="script" />

  <script src="Main.cs" type="server" lang="csharp" />

  <assembly ref="Insight.Database.Configuration.dll" />
  <assembly ref="Insight.Database.dll" />
  <assembly ref="Insight.Database.Providers.Default.dll" />
  <assembly ref="Insight.Database.Providers.MySql.dll" />
  <assembly ref="MySql.Data.dll" />
  <assembly ref="System.Data.dll" />
  <assembly ref="System.Globalization.dll" />
  <assembly ref="System.Configuration.dll" />
  <assembly ref="BCrypt.Net.dll" />
</meta>

```

Figure 31: Insert the above content into your meta.xml file.

Create Resource, Build Project, Move Files

- 1) Go to your GTA Network Server folder
 - a) Go into the /resources/ folder and make a new folder there called **newserver**,
 - b) Go back to the server root folder and open **settings.xml**.
 - c) Insert the following line into the settings file near the bottom together with the other resources: `<resource src="newserver" />`
- 2) Inside Visual Studio, hit F6. This will build the project and output the files we want.
- 3) Inside Visual Studio, open the project folder (right click on your project in the Solution Explorer and click "Open Folder in File Explorer").
- 4) Copy the **Main.cs** and **meta.xml** files into your newly created **newserver** resource folder from step **1a** above.
- 5) Go to the project folder which you opened in step 3. Navigate to the /bin/Debug folder located within it. Then copy the files listed below into your **newserver** resource in the GTA Network Server/resources/ folder from step **1a**.
 - a) BCrypt.Net.dll
 - b) Insight.Database.Configuration.dll
 - c) Insight.Database.Providers.Default.dll
 - d) Insight.Database.Providers.MySql.dll
 - e) Insight.Database.dll
 - f) MySql.Data.dll

Your resource folder should now look like this:

This PC > Local Disk (C:) > GTA Network Server > resources > newserver

Name	Date modified	Type	Size
BCrypt.Net.dll	04.05.2011 18.49	Application extens...	14 KB
Insight.Database.Configuration.dll	09.10.2016 12.51	Application extens...	9 KB
Insight.Database.dll	09.10.2016 12.50	Application extens...	403 KB
Insight.Database.Providers.Default.dll	09.10.2016 12.51	Application extens...	25 KB
Insight.Database.Providers.MySql.dll	09.10.2016 12.51	Application extens...	8 KB
Main.cs	08.01.2017 11.39	Visual C# Source f...	3 KB
meta.xml	08.01.2017 11.37	XML File	1 KB
MySql.Data.dll	02.10.2015 17.37	Application extens...	433 KB

Figure 32: The current content of your server resource folder

Let's start the server!

- 1) Go back to your **GTA Network Server** folder and start the server executable.
- 2) Launch the **GTA Network** game client, and go to the CONNECT -> Local Area Network tab:

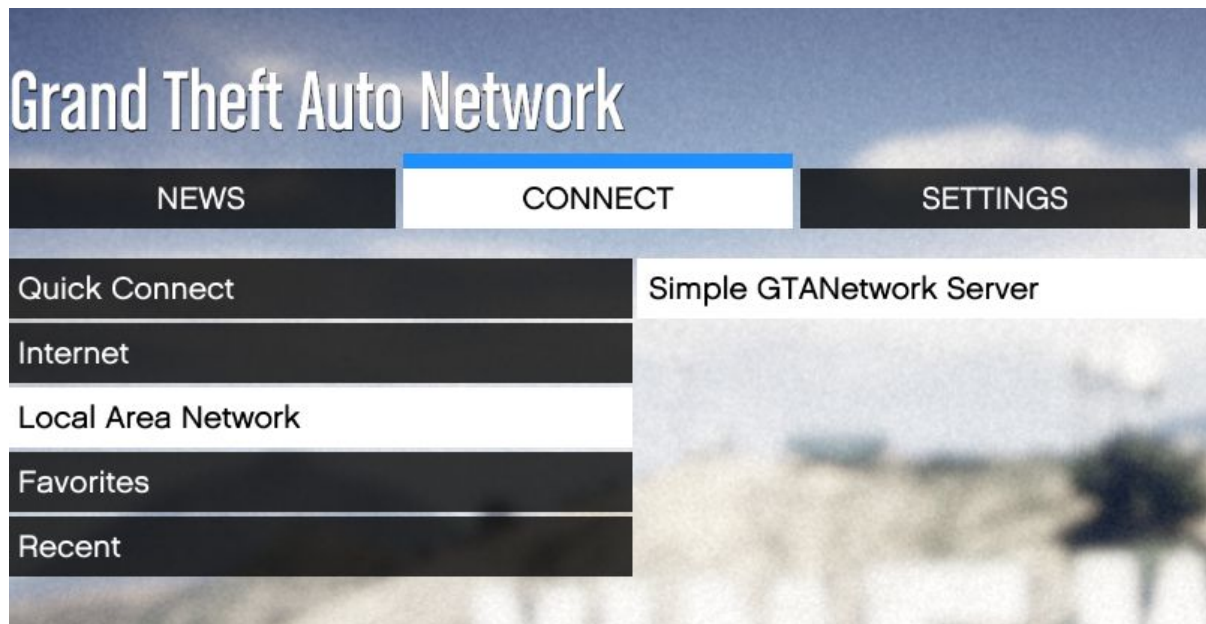


Figure 33: Connect to your local game server

- 3) Once you're on the server, press **T** to open the chat, then type **/ur password**.



- a) Witness greatness #1:

- 4) After registering, type **/ul password**



- a) Witness greatness #2:

The database has now stored:

id	username	hash
3	YourUsername	\$2a\$12\$ZpkAw6hqUEUSdF2i3DijuOeO0P87uiglOwYUJo0HOzs...

Figure 34: The user is registered in the database.

Congrats, you now have a basic understanding of how to fetch and insert data stored in MySQL using an ORM and stored procedures :-)