# Lab: Creating PizzaMore Web Site using C#, Bootstrap and CGI

This **tutorial** provides step-by-step guidelines to build a **“PizzaMore” app** in C#, HTML, CSS, Bootstrap and CGI. The app should implement **sign in** / **sing up** / **main** / **menu** pages.

## Project Specification

Design and implement a **“PizzaMore” web application** in C#, HTML, CSS, Bootstrap and CGI. Create 4 HTML pages with the following functionality:

* **Home**
  + Home page of the pizza site
  + Should be able to redirect to all other pages
  + Should send a **cookie** to the browser
  + Option to change language via **cookie**
* **Sign Up**
  + Register a new user in a separate file.
* **Sign In**
  + In case of success, a session key should be stored somewhere in a separate file.
  + Log out option that will delete the session
* **Menu**
  + Information about the pizzas
  + Should be accessible only when signed in

## Logger

First off, we need to make sure we have an easy way to debug if some unexpected behavior is being experienced in our application. That’s why we can make a logger, to make logs for when we would like to see some state of our program variable.

Create a class library called PizzaMore.Utility and in it, create a new class called Logger. It should be a static class with only one static method called Log that takes a string message and appends it as a text on a new line to a log.txt file.

## Cookie class

Next thing we need to do is create our custom Cookie class. Again create it in the PizzaMore.Utility class library. It should have two properties: name of the cookie and the value of the cookie which are both strings. Therefore, you should make two constructors. One with no parameters which sets the properties to null and one with both parameters. Finally make the ToString method return the cookie representation which is needed for the Set-Cookie header, which is name=value.

## **Constants class**

Next up, make a constants class in PizzaMore.Utility. In it we are going to put all the shared constants of our application.

## ICookieCollection interface

Now what you need to do is set up the specs for what we will call a CookieCollection. In the interface implementation, you should use a Dictionary<string, Cookie> so think of how you are going to implement the methods that come as a specification from the ICookieCollection, which are:

void AddCookie(Cookie cookie);

void RemoveCookie(string cookieName);

bool ContainsKey(string key);

int Count {get;}

Cookie this[string key] {get; set;}

## CookieCollection class

Now that you know what a collection of cookies should have as a behavior, you should implement it, by implementing the ICookieCollection class.

## Creating the data context.

Next, it’s time to make the database. First create a new class library, called PizzaMore.Data. In it, create a new EntityFramework context, called PizzaMoreContext. Since a problem will appear while trying to connect using the standard Windows authentication login, we will need to make a new user for the SQL Server that has the rights to access the database engine and make changes to DBs. To do that open SSMS with the user you’ve been currently using and right-click on Security -> New -> Login. Check the SQL Server authentication radio button. **Add** a **new** login name (**username**) and **password**. **Uncheck** “**Enforce** **password** **policy**”. Now go to the upper left corner and **select** **Server** **Roles** and **give** **all** the **roles** **to** the **current** **user**, so that you don’t have any problems later.

## Modifying the connection string

Now, go to the App.config and change the connection string, to be suitable for the new user and the database you would like to create. It should look something like this:

<add name="PizzaMoreContext" connectionString="data source=.;initial catalog=PizzaMoreContext;integrated security=False;User Id=appUser;Password=1234;MultipleActiveResultSets=True;App=EntityFramework" providerName="System.Data.SqlClient" />

## Creating the models for our app

Since we are going to save many things in the database, we need to create models for them. The three models we need so far will be one for the User, one for its Session and one for the Pizzas. Finally, don’t forget to make a new DbSet for each of the models in the context.

## Implementing the User class

The user has a few properties. The first one of course is ID, an Email, a Password and finally a collection of Pizzas which are the Suggestions for pizzas from the user.

## Implementing the Pizza class

The pizza class has some more properties. Those are the Id, the Title, the Recipe(string), ImageUrl(string), UpVotes and DownVotes(int) which are indicator, for how many people have voted for the pizza. Finally, there is an OwnerId and an Owner which is some user.

## Implementing the Session class

Despite the Id that is common for models, the only thing we have right here is the UserId and a reference to the User. Something more is also important and that is the ToString method. You should implement it and it should return the following:

$"{this.Id}\t{this.User.Id}"

## Creating a WebUtil class

### Identifying request method

You need to create the WebUtil class in the Utilities class library. It is going to be a static class and so, all of its methods will also be static. Let’s create the first two. They are going to be IsPost() and IsGet(). Since you know you can get the string for the request method using the environment variable “REQUEST\_METHOD”, you should be able to implement them both.

### Retrieving request parameters

The next pair of methods are RetrieveGetParameters() and RetrievePostParameters() which both return an IDictionary<string, string>. The first one gets them as an environmental variable as you know and the second one gets them from the standard input. However, either of them are Decoded using the UrlDecode method from the WebUtility (System.Net). Since both of the methods use the same strategy to parse the parameters, you may use a single method RetrieveRequestParameters. If you are not sure how to parse the parametersString that comes from the Post/Get, you can log it in a file and see the format and implement it later.

### Retrieving cookies

What you need to do now is define a method called GetCookies() which returns an ICookieCollection. Its purpose is to check if any cookies were passed to the program and if so, to parse them.

First off take the Cookie header using the environmental variable “HTTP\_COOKIE”. Next you need to parse them if there are any present and finally return the filled CookieCollection. Note that the format of the cookies passes is cookieName1=cookieValue1;cookieName2=cookieValue2…

### Retrieving sessions

Next up, create a new method GetSession(), that returns a Session. As we’ve talked, in order to get the session from the database, we need to receive its ID from the cookie. That’s why first we need to GetCookies(). First you would like to check, whether the given ICookieCollection, contains the key “sid” (not for [Sid the sloth](http://iceage.wikia.com/wiki/Sid) from Ice age, but “sid”, for Session ID). If it does not, return null, else get the corresponding cookie and using its value, extract the session from the database and return it.

### Printing content of file

Since we are going to read files from the HDD, we should make it in a centralized place. That’s why you need to create a PrintFileContent(string path) that reads all the text from the file with the given path and writes it on a new line in the console.

### Returning page not allowed

Since there will be some pages that should not be accessed if there is no user logged, we should return some kind of a special page that is for this occasion. This page is given to you in the resources for the current exercise. You might want to check it out. All you have to do in the PageNotAllowed() is call the PrintFileContent(), giving it the relative path to the index.html of the game.

## Implementing a Header class

Since every time, we get some sort of a request to our app, we need to return some header, we should make a Header class. Again, create it in the Utilities class library. For now, its properties are going to be a Type(string), a Location(string) and Cookies(ICookieCollection).

In the only constructor, which is empty, you should set the Type to "Content-type: text/html" and initialize the Cookies.

It has two methods: void AddLocation(string location), which sets the location to $"Location: {location}" and void AddCookie(Cookie cookie), which adds the passed cookie, to the collection of cookies.

In order to generate a meaningful header that we can send, we have to override the ToString() method. In it create a new StringBuilder to which we will append the components of the header class. First append a new line with the type of the returned header. Next, if the cookies collection isn’t empty, for each cookie inside it, append a new line like the following $"Set-Cookie: {cookie.ToString()}". After that we should append the location property on a new line, but only if it is not null.

Finally append two empty lines and return the string from StringBuilder

## Implementing the Home class

Since we are going to have to call our home.html somehow, we will do this through the home.exe. So first off, create a new Console App called Home and in it, name the class Home.

It should have some static fields which are: RequestParameters(IDictionary<string, string>), Session, Header which is initialized at the declaration line and a Language(string).

Since if you have not accessed the web page before, you’ll probably need to set the cookies for the user for the first time, we will do that in the void AddDefaultLanguageCookie(). In it, you should check whether there is a cookie with key “lang” from the cookies that came from the user request (WebUtil.GetCookies). If there isn’t such one, you should add one to the header of the current home page with a name of “lang” and a value of “EN”. Also, set the Language property to “EN”. This way we will know which language of the page to serve and also the program will set an appropriate Set-Cookie header.

Now call this method in the begging of the Main.

Next it’s time to settle our reaction to the client and it is based on the request method that the user agent sent to us.

If the method is get, we should retrieve the get parameters from the WebUtil and set the language to the one that comes from the cookie from the collection at index “lang”.

If the method is post, then the user has probably changed the language from the UI off the app and has sent us the new language. That means that we should again retrieve the request parameters but this time from the post. Since the user has probably set a new cookie, we will take the “language” parameter from the RequestParameters and it should be the value of the new cookie that we need to add to the header. The key is “lang”. After adding the cookie to the header, all we need to do is change the Language property to the one that comes from the request parameters.

Finally, in the Main, independently if the request is get post, we should call the ShowPage().

You might think, “Where is the ShowPage method?”. Well dinner isn’t going to cook itself – you need to create it with a return type of void.

In it call the Header.Print(). After that simply check if the Language equals “DE” or “EN” and depending on which one it is, call ServeHtmlDe or ServeHtmlEn. Their only job is to call the WebUtil.PrintFileContent() with the according path.

## Pray to the CGI goddess

If you’ve done everything right, you might want to test the whole app.   
**NOTE**: You need to copy not only the home.exe, but all of its dependencies. For now, this is only the dll from the Utilities.

Stay tuned, because next time we are going to implement the sign in and sign up and also the pizza menu.