## Documentation:

The desired learning outcome was defined with the words: “To develop a Database Driven Website prototype using the Microsoft .NET platform. “

I created a demo website for presenting a list of games that were released. The website fulfils all desired aspects and provides three kind of roles: Users, Employees and Admins. The user can only view the list of games and edit his own personal information. Employees can additionally use the CRUD operations on the game list while the admin also can change the user list completely.

Technology:

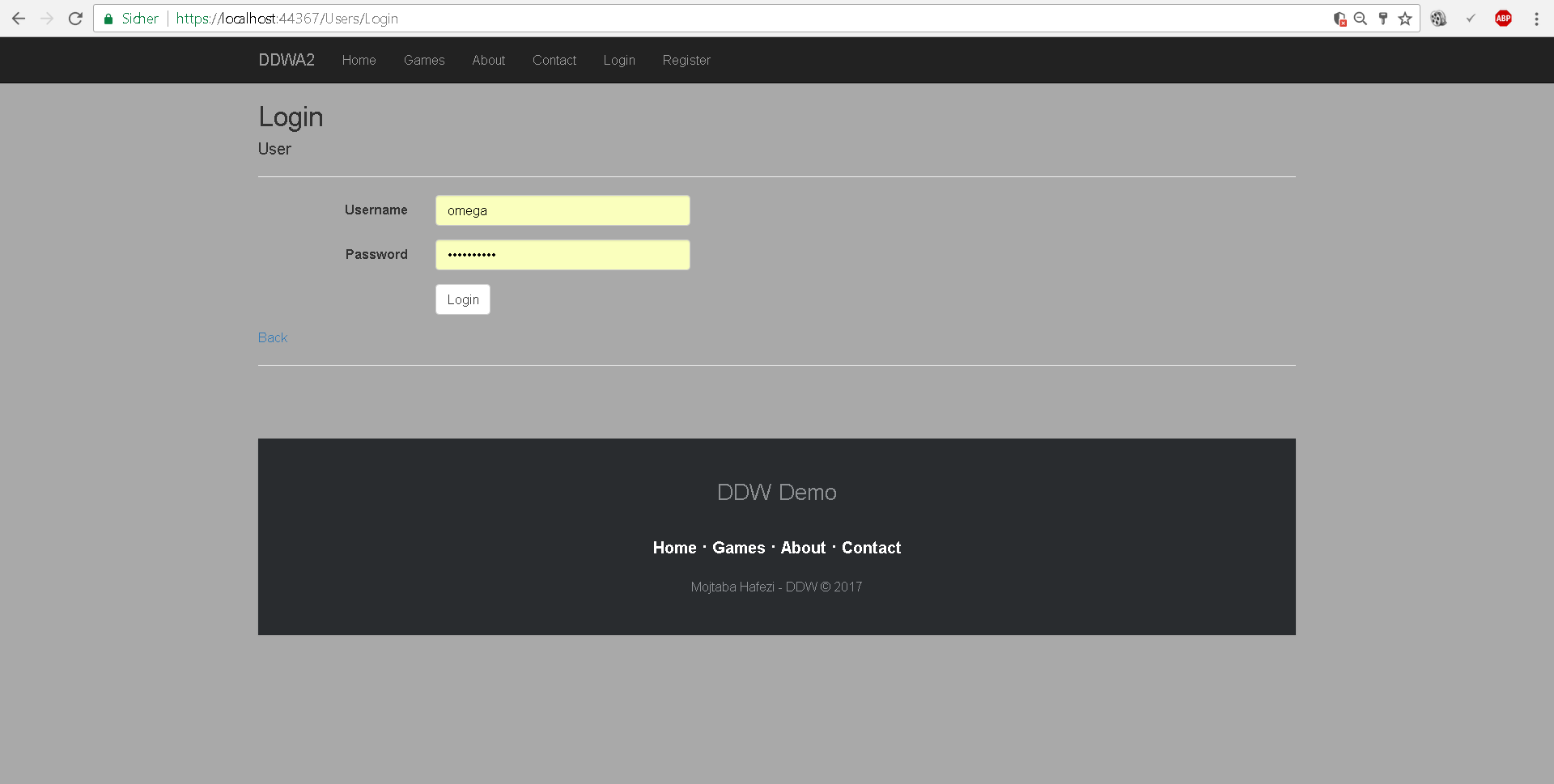
Visual Studio 2015 + Microsoft .NET platform + MVC5 + Razor

Requirements:

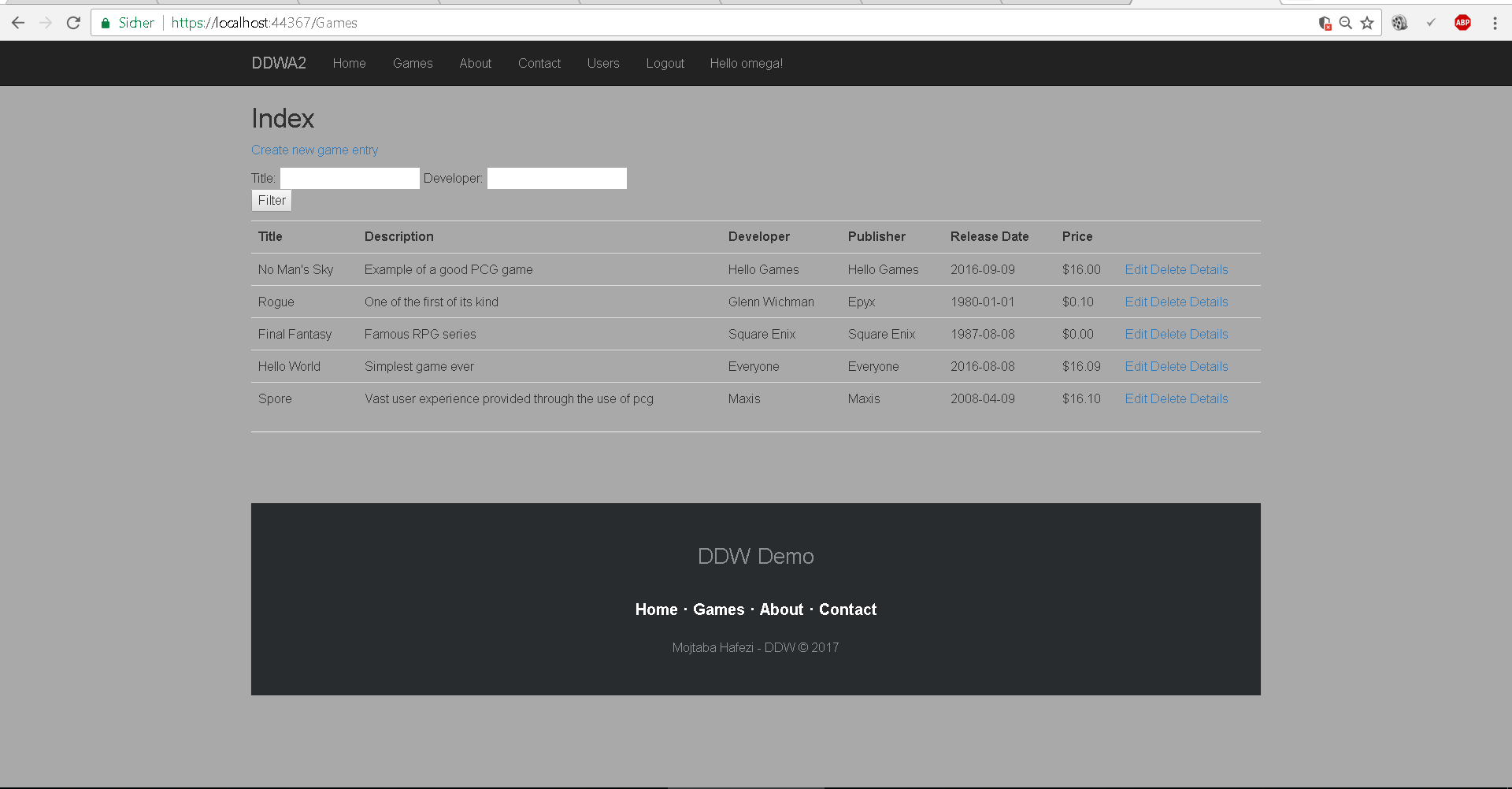
1. Existing user should be able to login with username and password:

The first page a user sees shows the basic functionalities in the navigation bar.

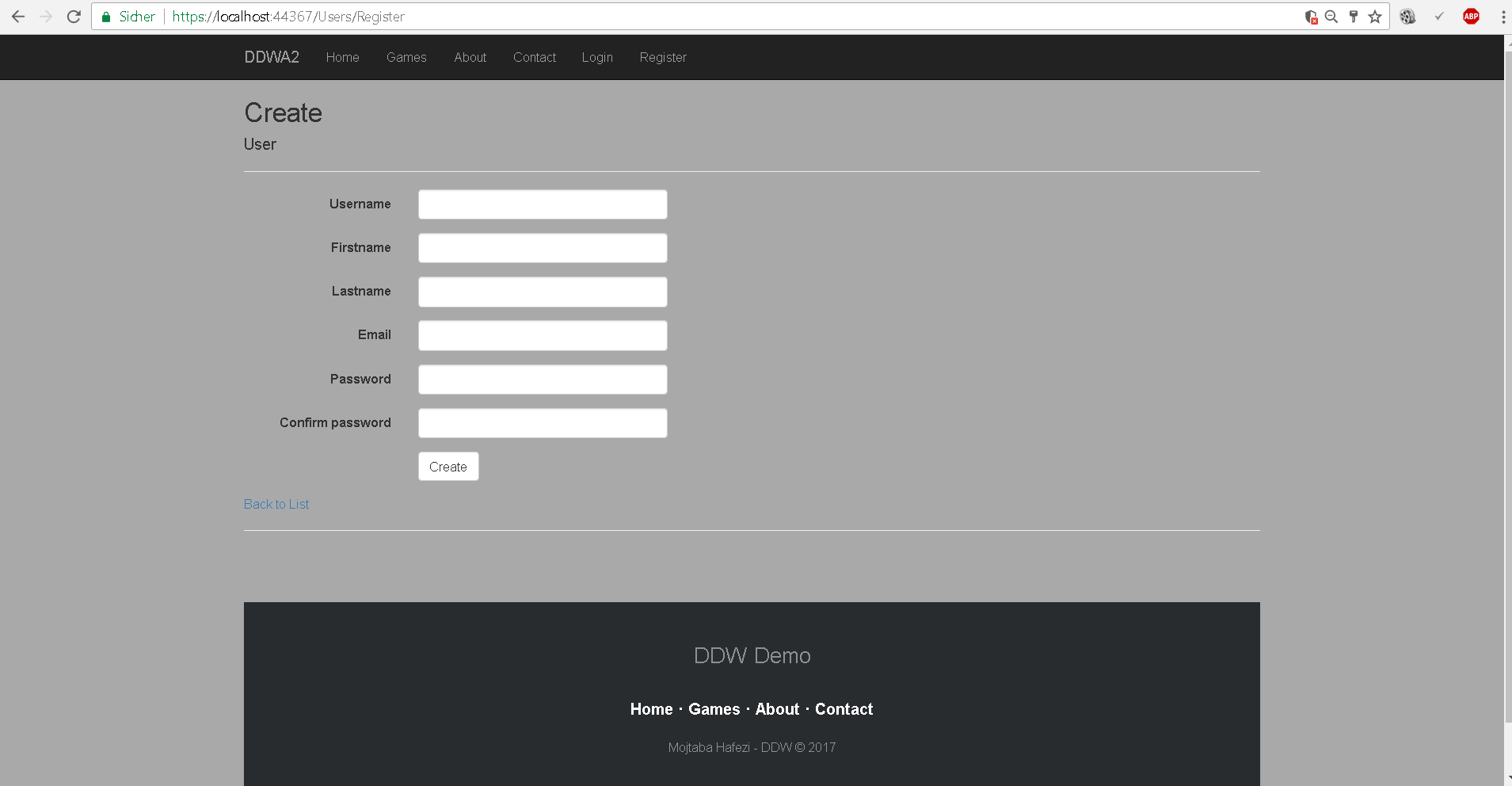


An unauthorized user can see the main page which leads to every necessary link. By clicking the Login link on the navigation bar or the carousel, the following page is visible. 

After successful login, the user gets redirected to the games. On the navigation bar one can see the username used to login.



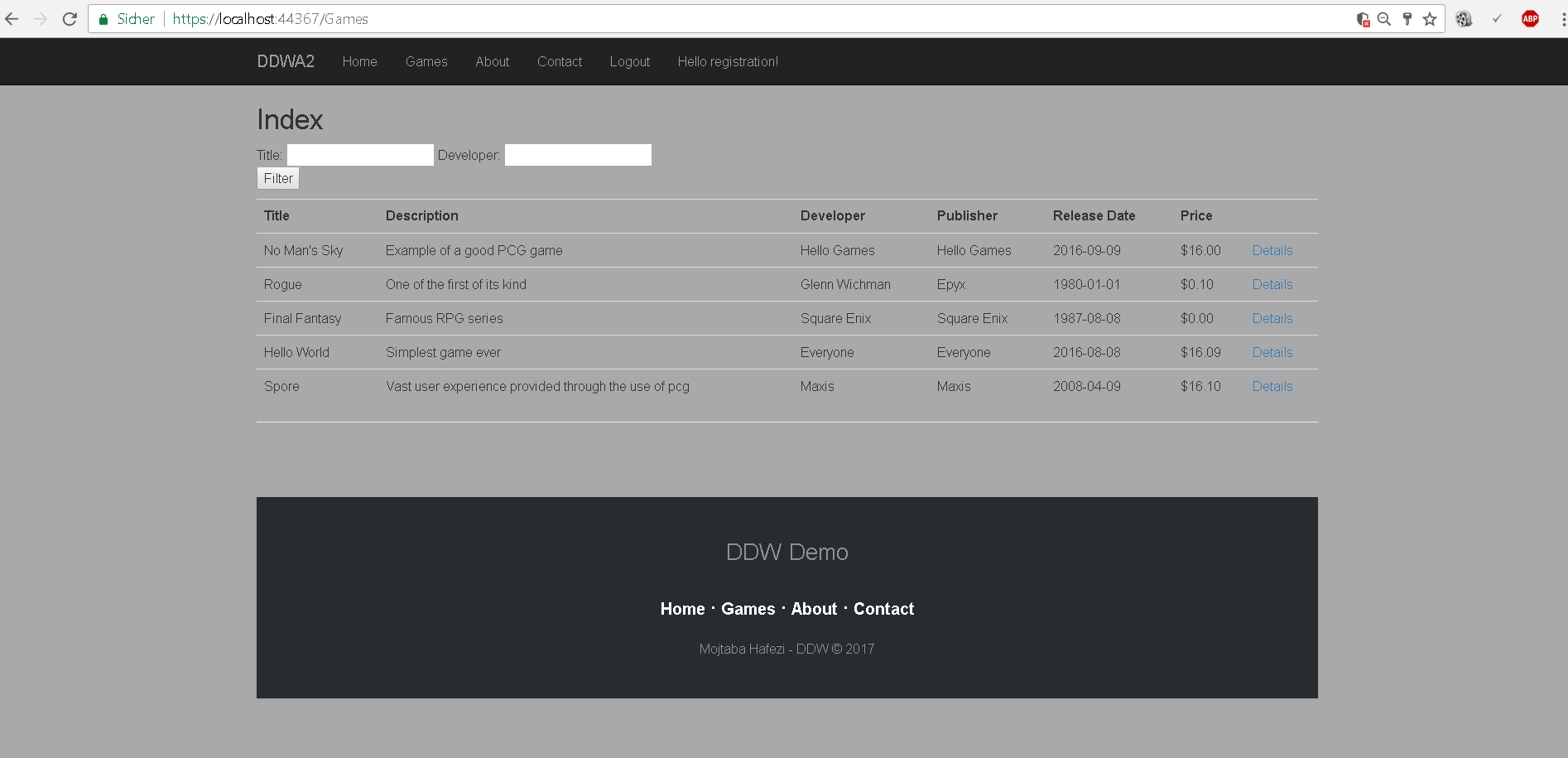
1. Registration of users with username and password:



After successful registration, the user gets redirected to the Login page. There he can log in with the newly created account.

1. Persistent state between pages:

After login, the user stays logged in and on every page the identity persists. This is not only achieved using session but additionally through help by the application cookie and the Microsoft Identity.



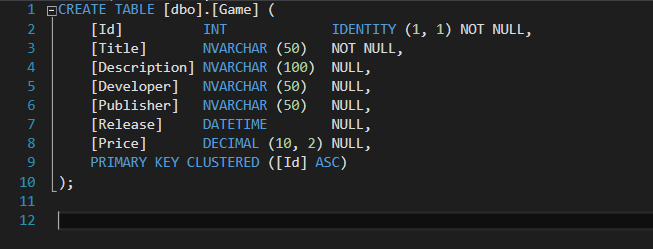
Code snippet of Session and Identity implementation. The reason to use the Identity is due to being able to give the users their roles and authorize the access to specific pages before the GET request was called.

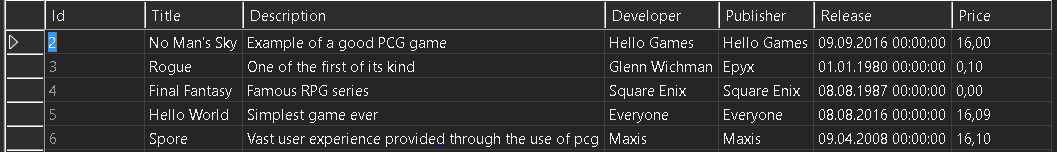


1. Minimum of 2 MYSQL tables for data:

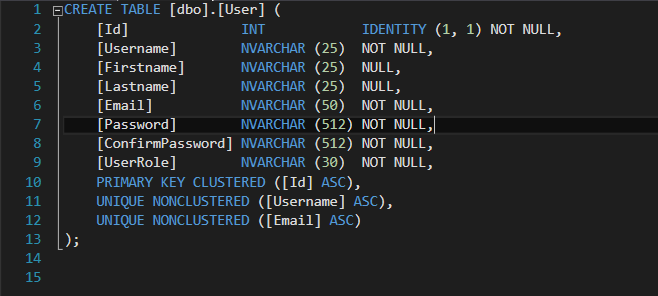
The MYSQL database has 2 tables which are for the user and the games.

Games:





Users:



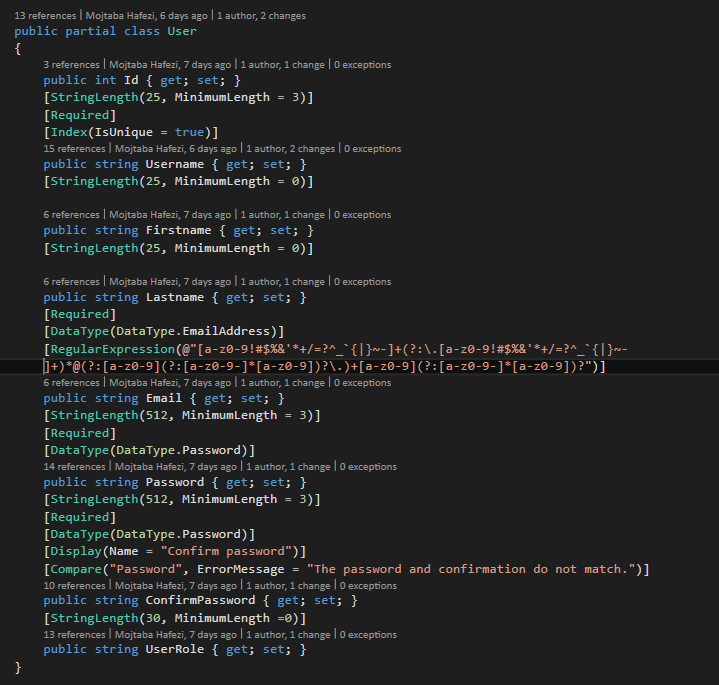


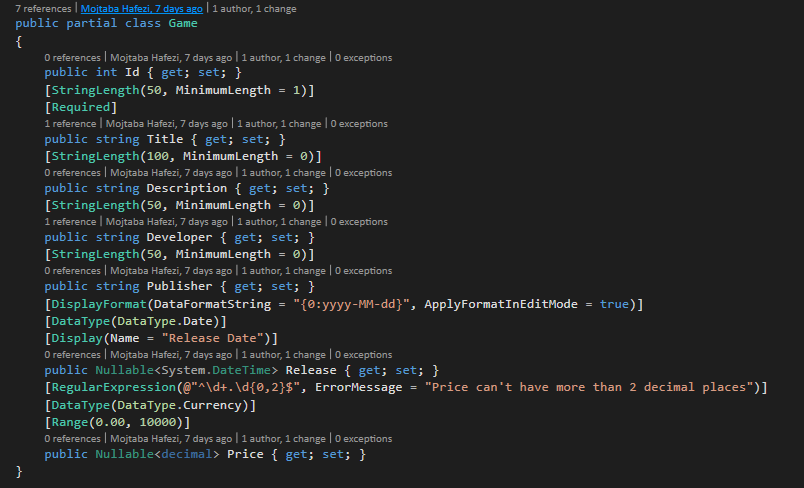
1. Server and client side validation:

The validation is mainly provided by the Framework and is an excellent demonstration how to use the DRY principle. Every validation rule is declared in the Model class but applied to the whole application. Server and client-side validation are done through attributes specified in the model classes as for example a “Required” attribute means “NOT NULL” in SQL. Regular Expressions, minimum length, specific data values (date, currency…) and a lot more can be chosen in this context.

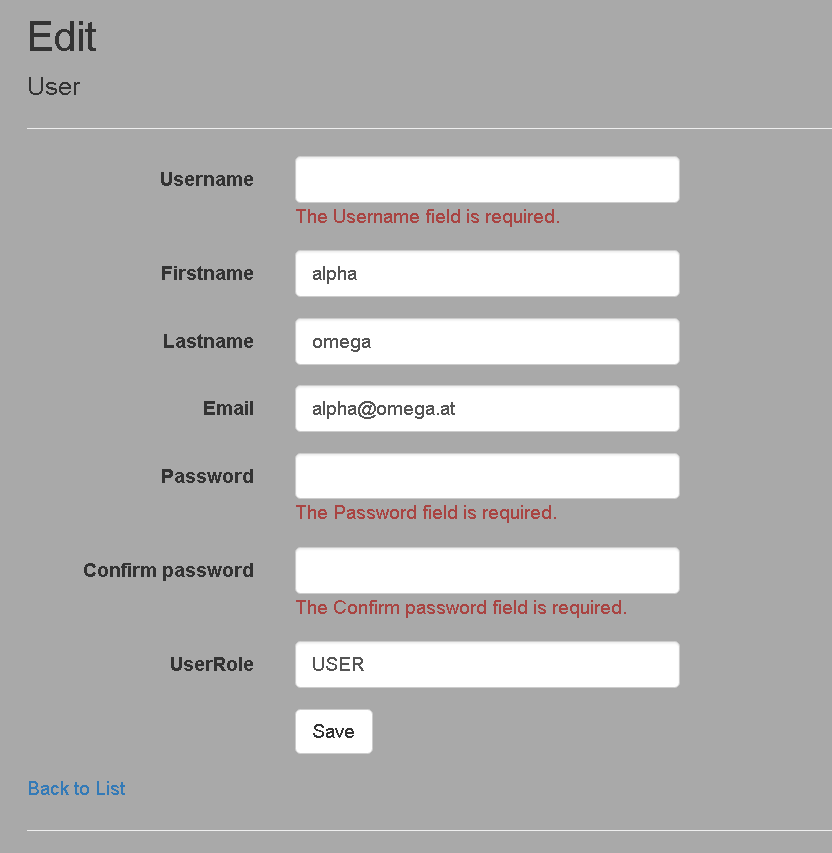
Should an error occur the message gets displayed both on the client-side through JavaScript and server-side (when JavaScript is disabled) at the point when “ModelState.IsValid” gets called. In both cases the errors are shown on the view side through the Razor engine.

The user and game model classes and their validation:





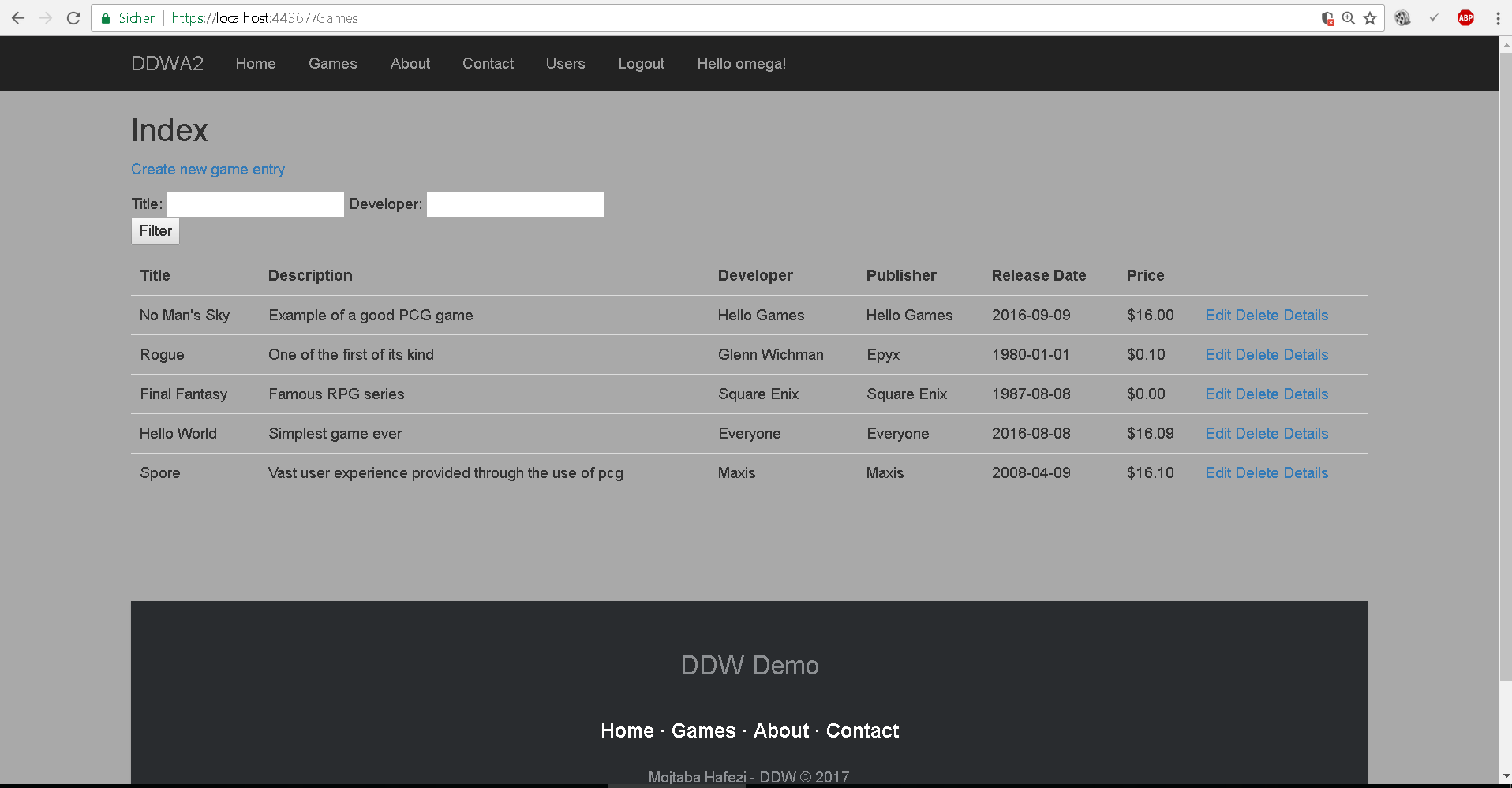
Demo of wrong or missing input while editing user details:

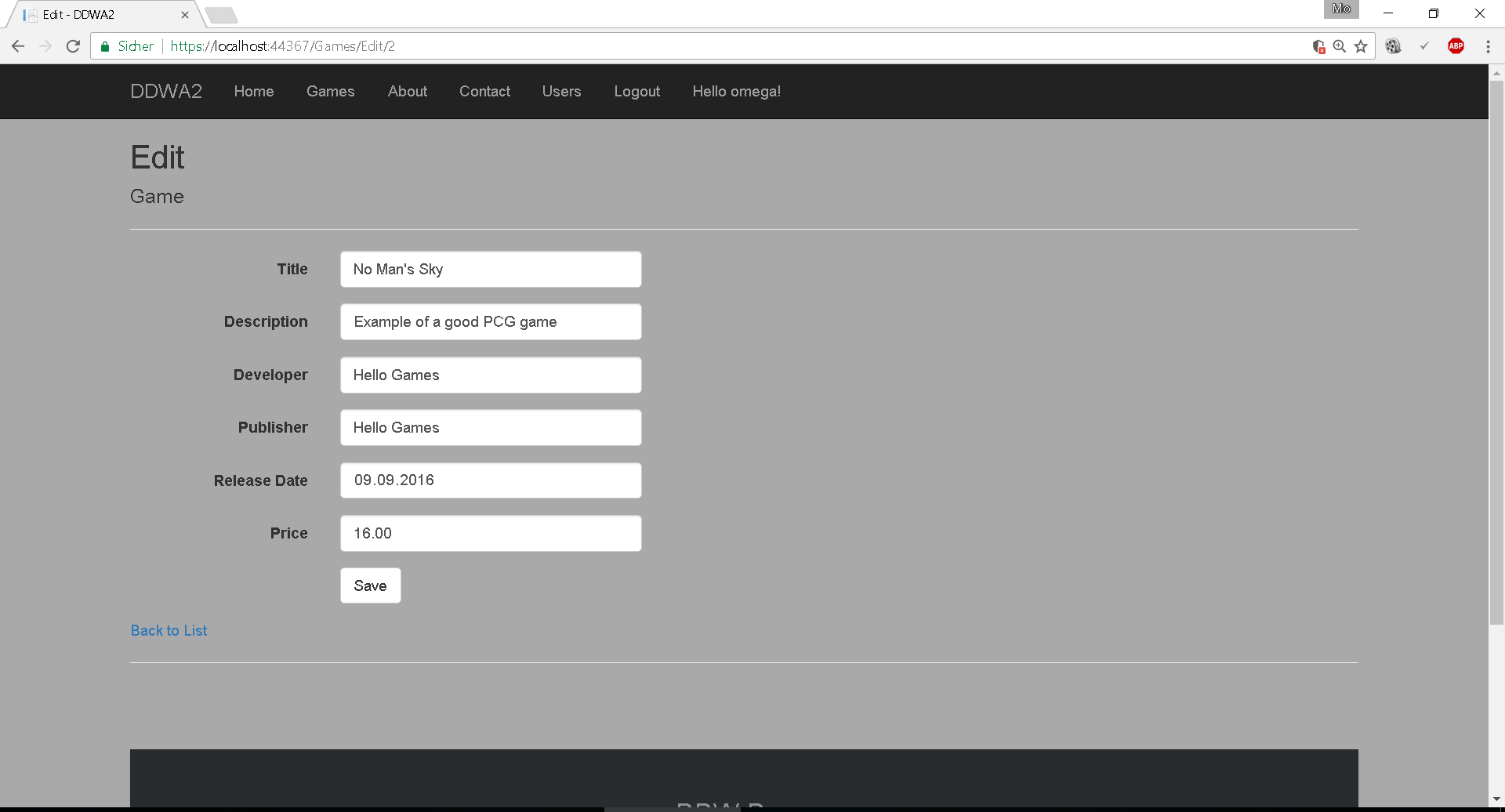


1. CRUD operations on data (user and games):

A **broad range** of information can be changed. The employees and administrators can operate all CRUD operations on the products, while the users can only be viewed and changed by the admin (and the user can edit his own credentials).

Upon viewing the list of games on the right side the crud operations appear for the employees and admins.



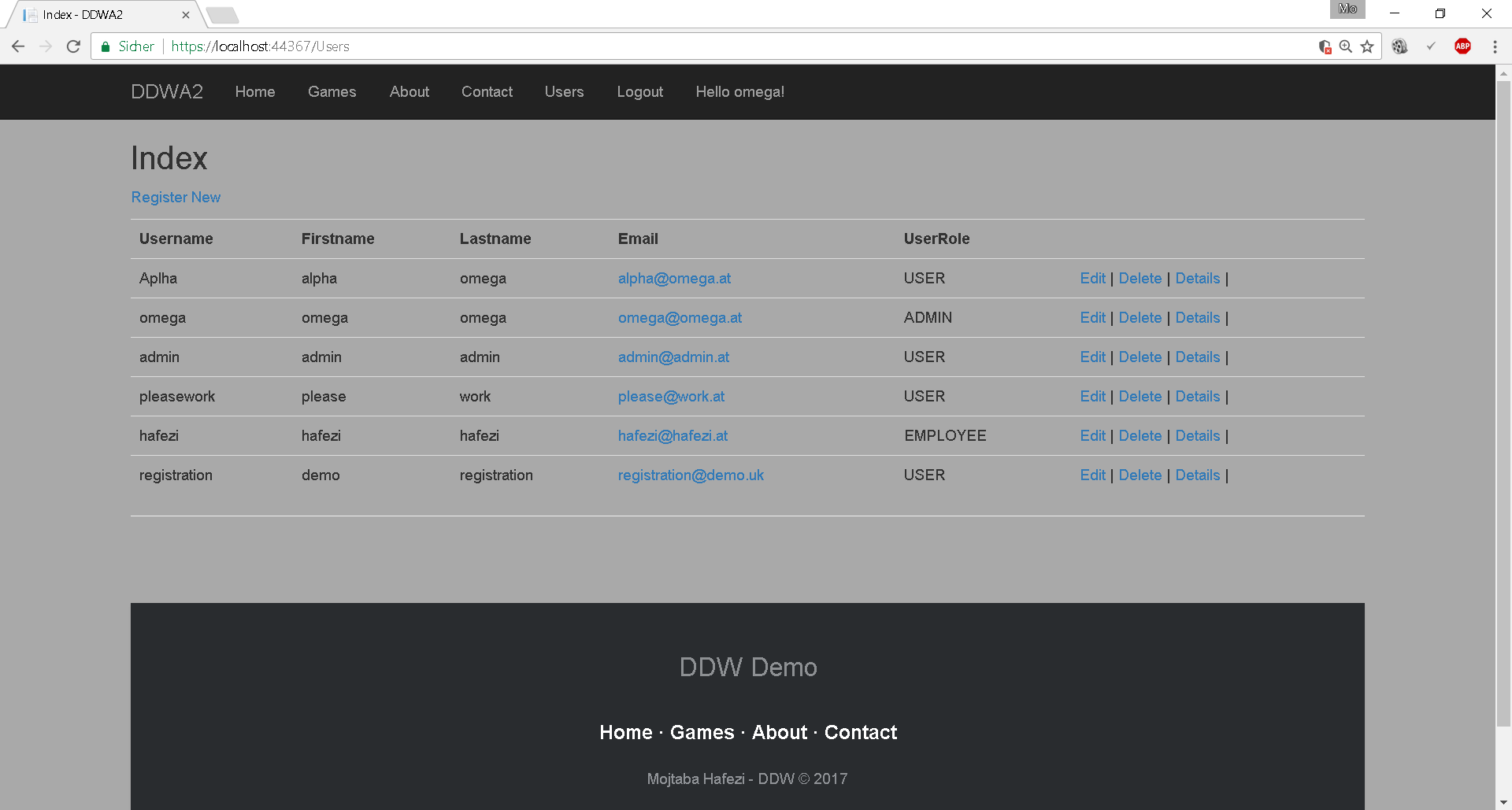


Most of the fields get **auto-populated**. This is provided mostly through the server to take off some workload for the users. The code for the CRUD operations, the Login/Registration etc. are to be found in the controller classes, while validation happens in model classes (found in the model directory).

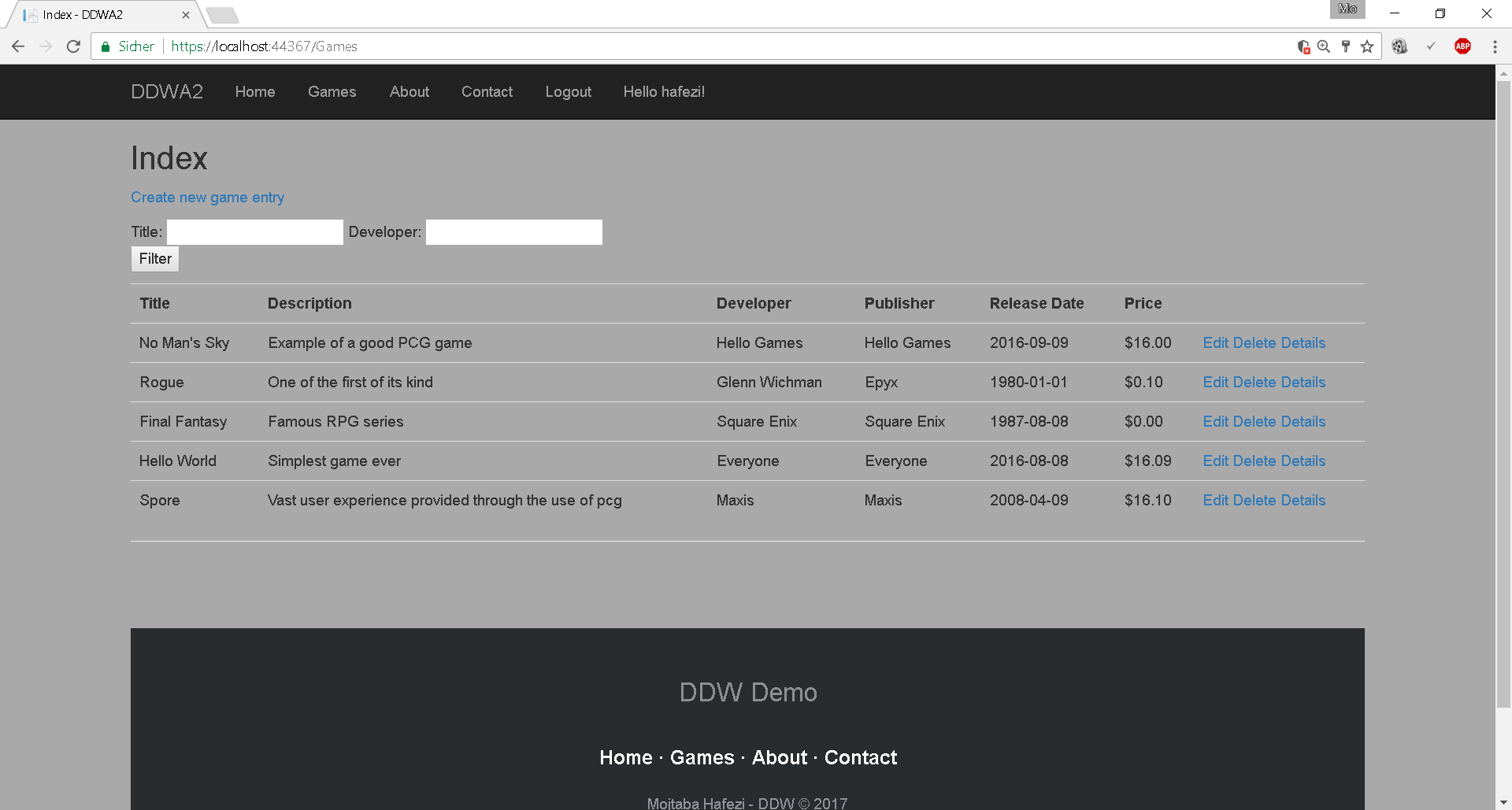
1. Different users with different rights/views:

To begin with I provide three roles: User, Admin and Employee. The employee can change game data and all their own data. A normal user has no CRUD rights on any data but his own login credentials. The admin has CRUD rights on all tables and can see the list of users and edit them etc.

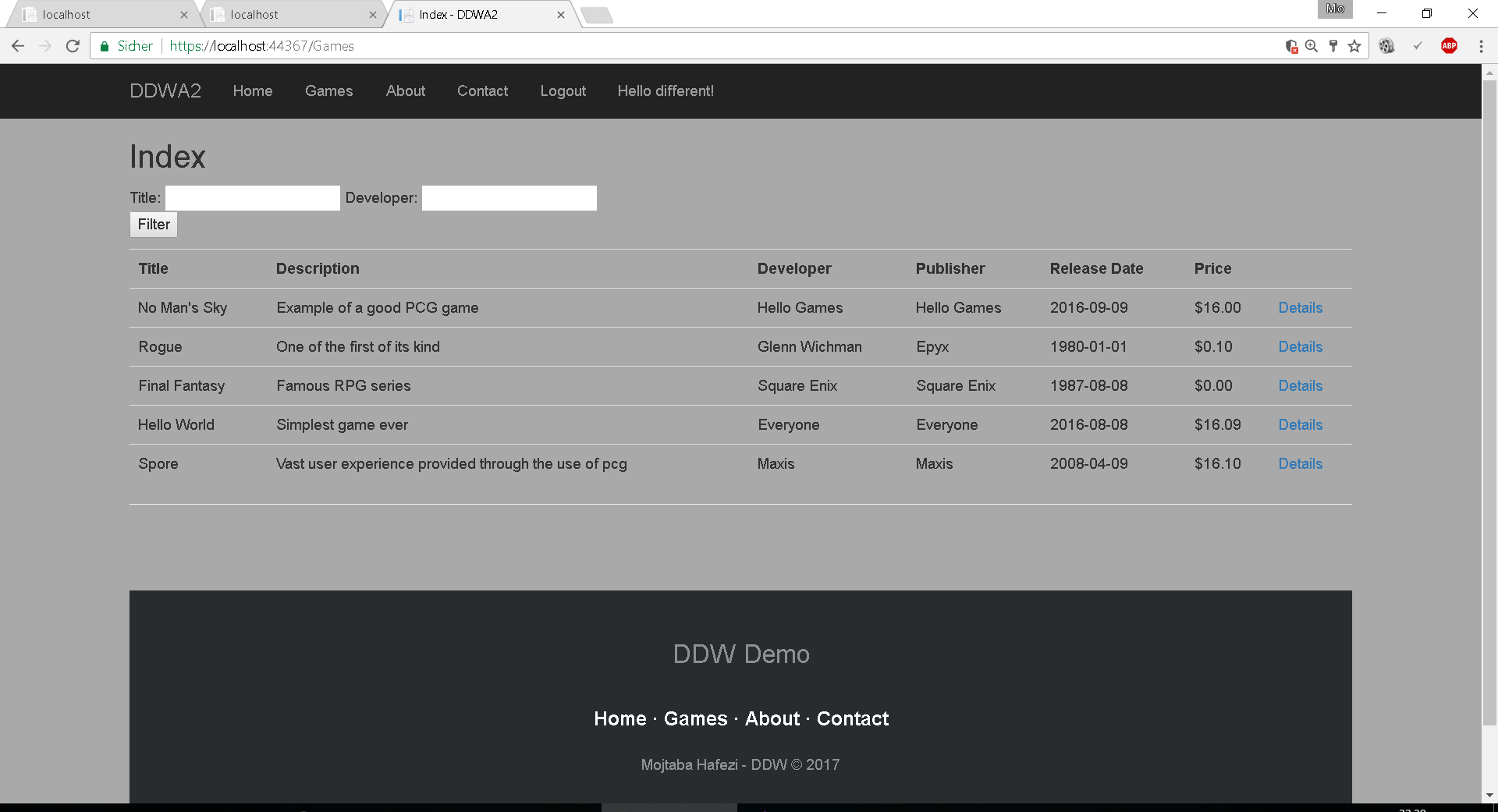
ADMIN with access to the user list:



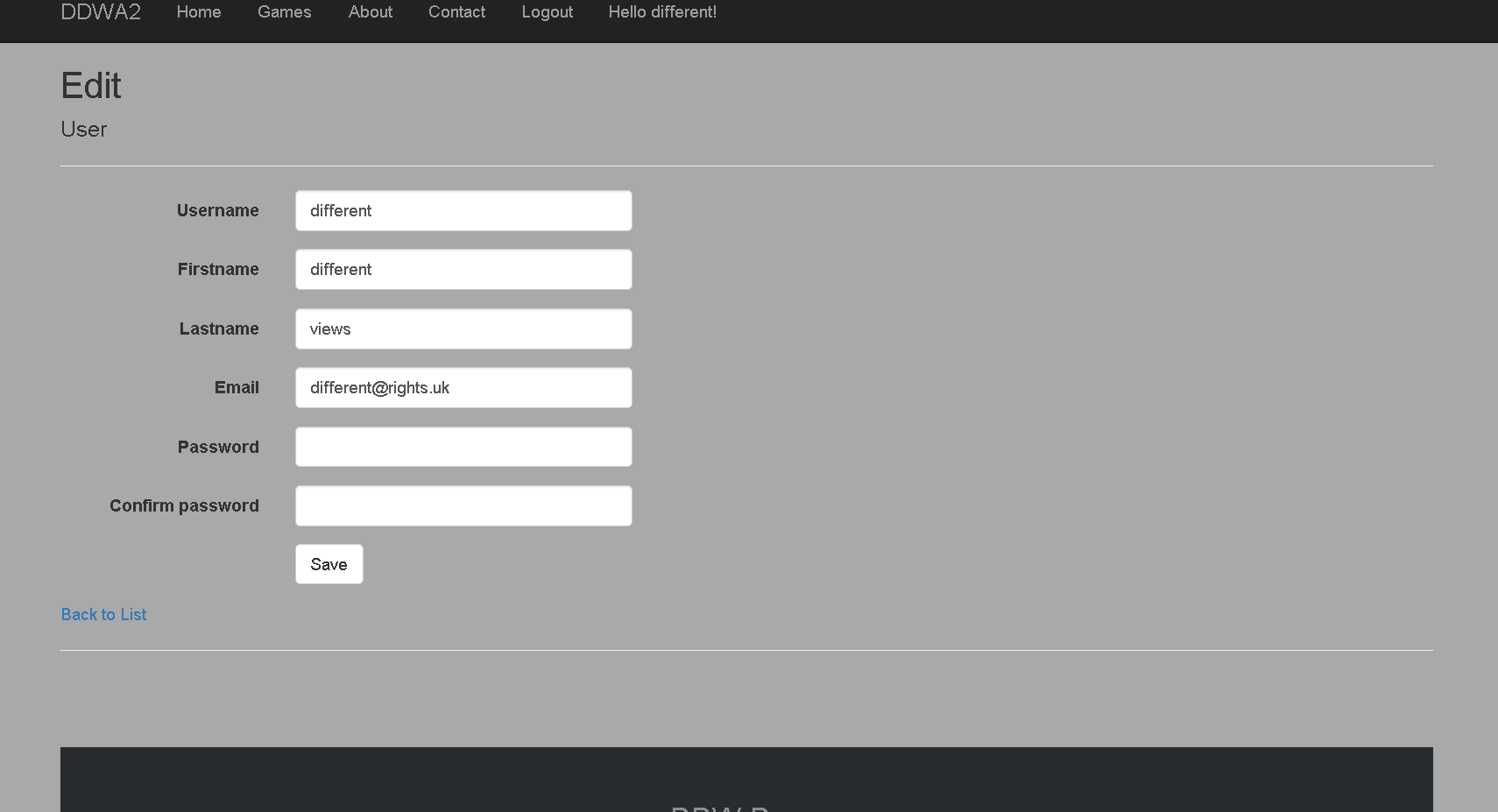
EMPLOYEE with access to CRUD on the game list:



USER with no CRUD rights except for his own data:

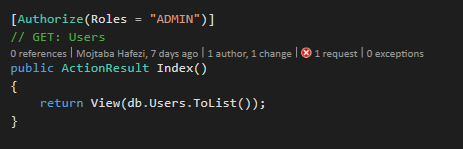


By clicking on “Hello, {username}!” the user can access his or her profile and change specific data:

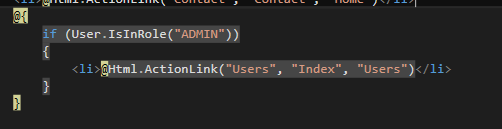


To make this possible I used the Identity feature of the .NET framework to pass the role of the user along with the username and the id. By using the Identity implementation, the “Authorize” attribute can be used more thoroughly and one can decide upon Users and Roles to have the rights to call a method in the controller or a specific part of the view. The Razor engine also provides the possibility to check for conditions and makes it easier.

For example, only the admin can view the list of users:



In combination with the razor options:



These two lines of code prevents any other users to see the “Users” link on the navigation bar and restricts them from calling the link to the view.

1. Secure password hashing with additional salting:

After researching many hashing algorithms and possibilities I concluded to use the BCrypt hashing algorithm. The implementation has a salting function, which makes the passwords very safe and is added dynamically. There is no way to decode the hashed algorithm so the “Verify” function gets used to check for correctness of the passwords.

Upon Registration or Login, the password gets hashed before being saved into the database/compared with the existent hashed password.





In the configure method I demand the BCrypt to be used for encoding all passwords in the checks. To use the BCrypt implementation the NUGET packet manager was used to find a good implementation. This makes the application safer and the implementation of the algorithm is not necessary anymore.

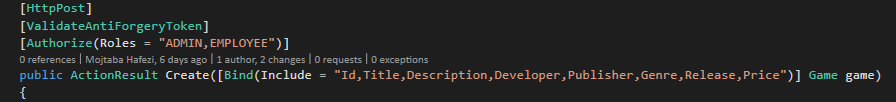
1. Additional security features:

XSS:

XSS works by tricking the application into inserting the <script> tag into the view or similar undesired actions. The best prevention measures are to either constrain the input or decode the output. To prevent the input I used the @Range attribute as validation for example or the regular expression for the e-mail. For decoding cases the Razor provides the @HTML tags and their appropriate implemented encoders like “HttpUtility.HtmlEncode”.

CSRF:

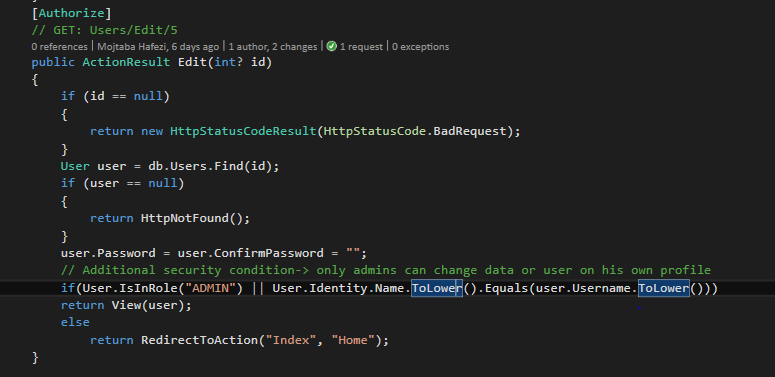
To help prevent CSRF attacks, ASP.NET MVC uses anti-forgery tokens, also called request verification tokens. Anti-forgery tokens work because the malicious page cannot read the user's tokens, due to same-origin policies (Same-origin policies prevent documents hosted on two different sites from accessing each other's content).

Controller:

View:

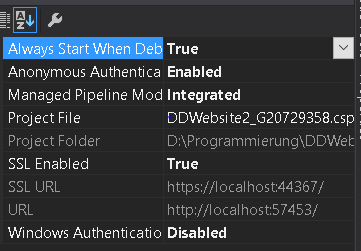


Authorization:

Through different roles, specific authorization has been added and additionally every user can only change his or her own data. It is not desired that a user changes the destination id in the URL and gets the right to change someone else’s data.

SSL:

SSL has been enabled to provide a better security feature. Although for the presentation only the localhost certificate has been changed to “trusted”. For deployment, a real certificate would be necessary.



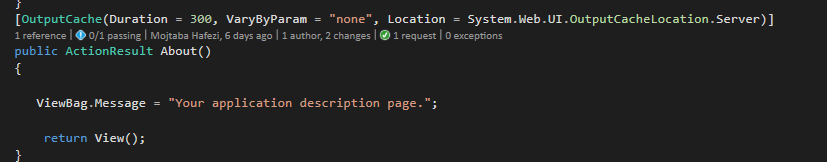
One can see that the localhost is accessed through https and the deployed cookies on the browser are the Application Cookie (Identity), the Session and the AntiForgeryToken.

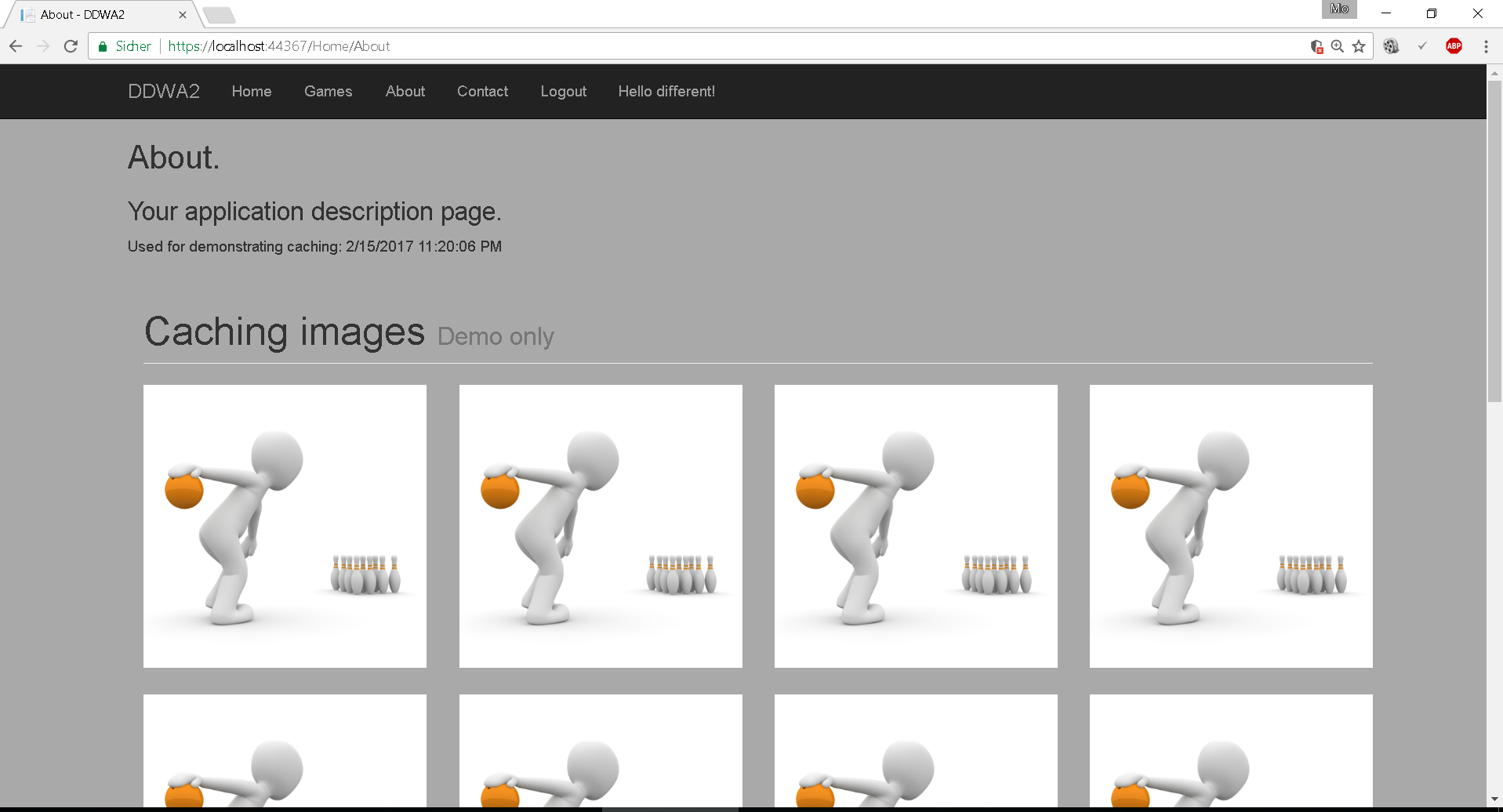


1. Additional performance feature:

Caching:

To provide a better performance the output caching has been implemented on the “About” link of the navigation bar. This is just for presentation and could be used in a better manner. For demonstration, some images were put into the view which then get cached for a specific amount of time.

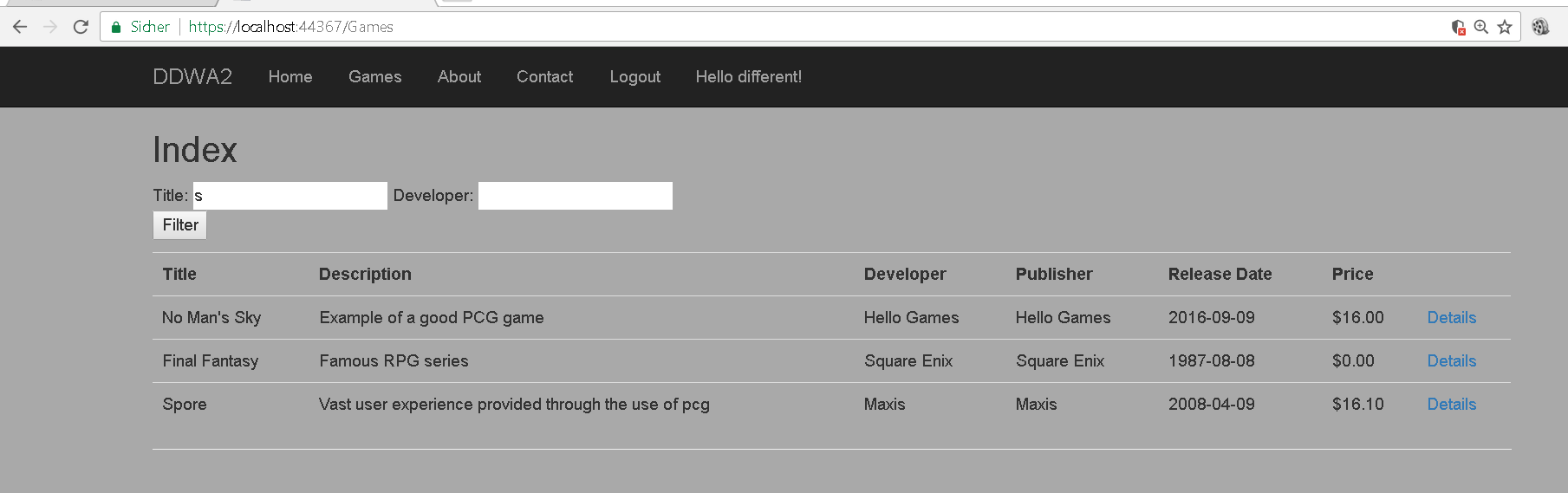


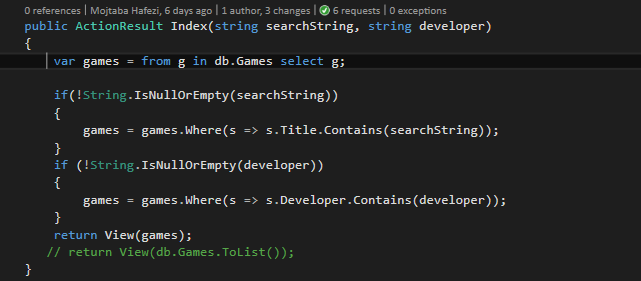


The small text shows the time it was cached and can be compared to with the current time. The second time this page gets loaded it is a lot faster than the first time thanks to the caching. Should anything change in the meantime (user logs in etc.) and returns to „About“ then the navigation bar will show the old version, although the session and cookie were created and the user is logged in.

Filter:

Another thought was to use filters for searching through a list of games for example. It could be very hard to find a specific item through an unsorted list. This is easy scalable for every desired aspect including specific prices or dates (greater, smaller than etc.).



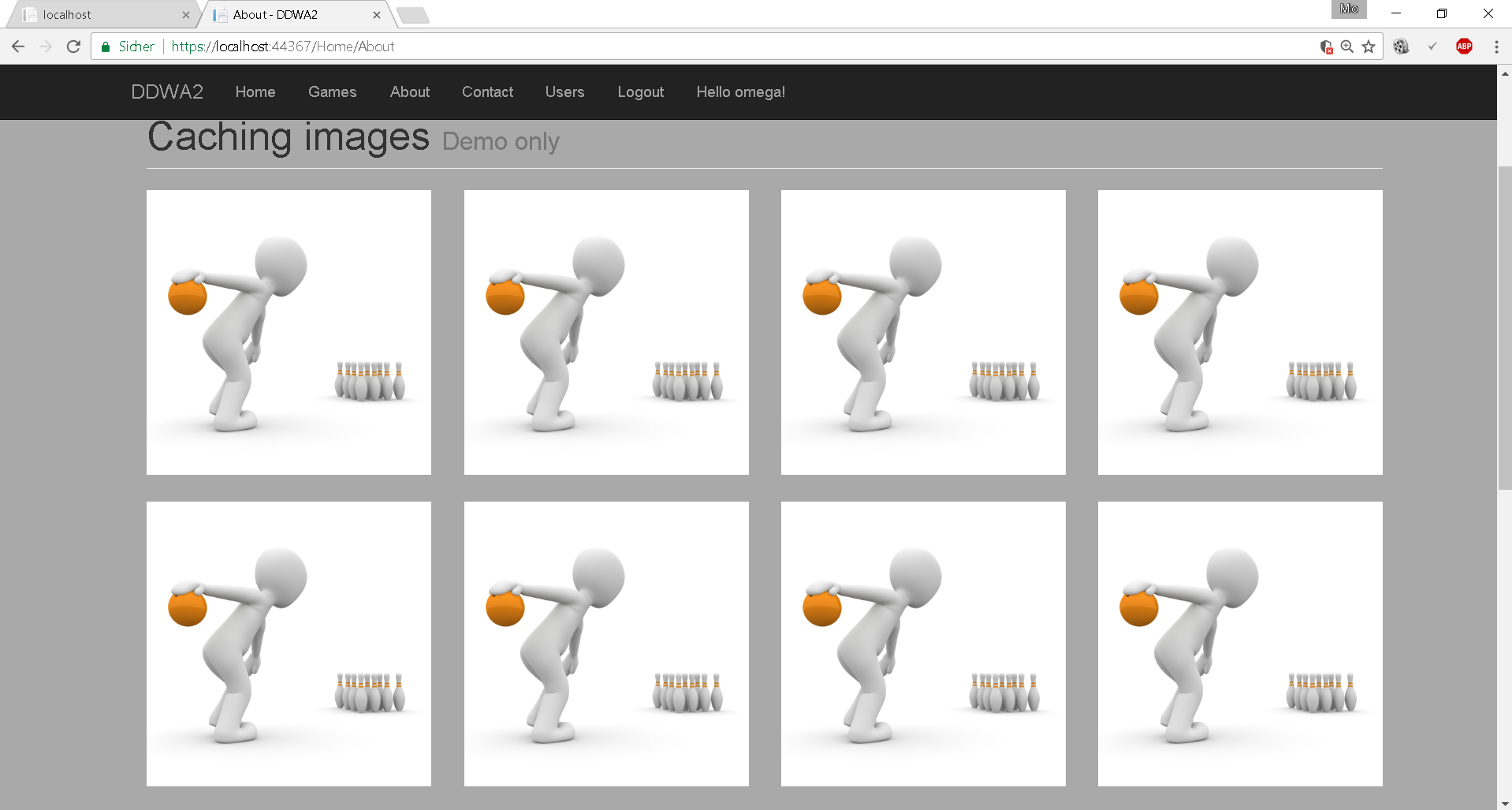


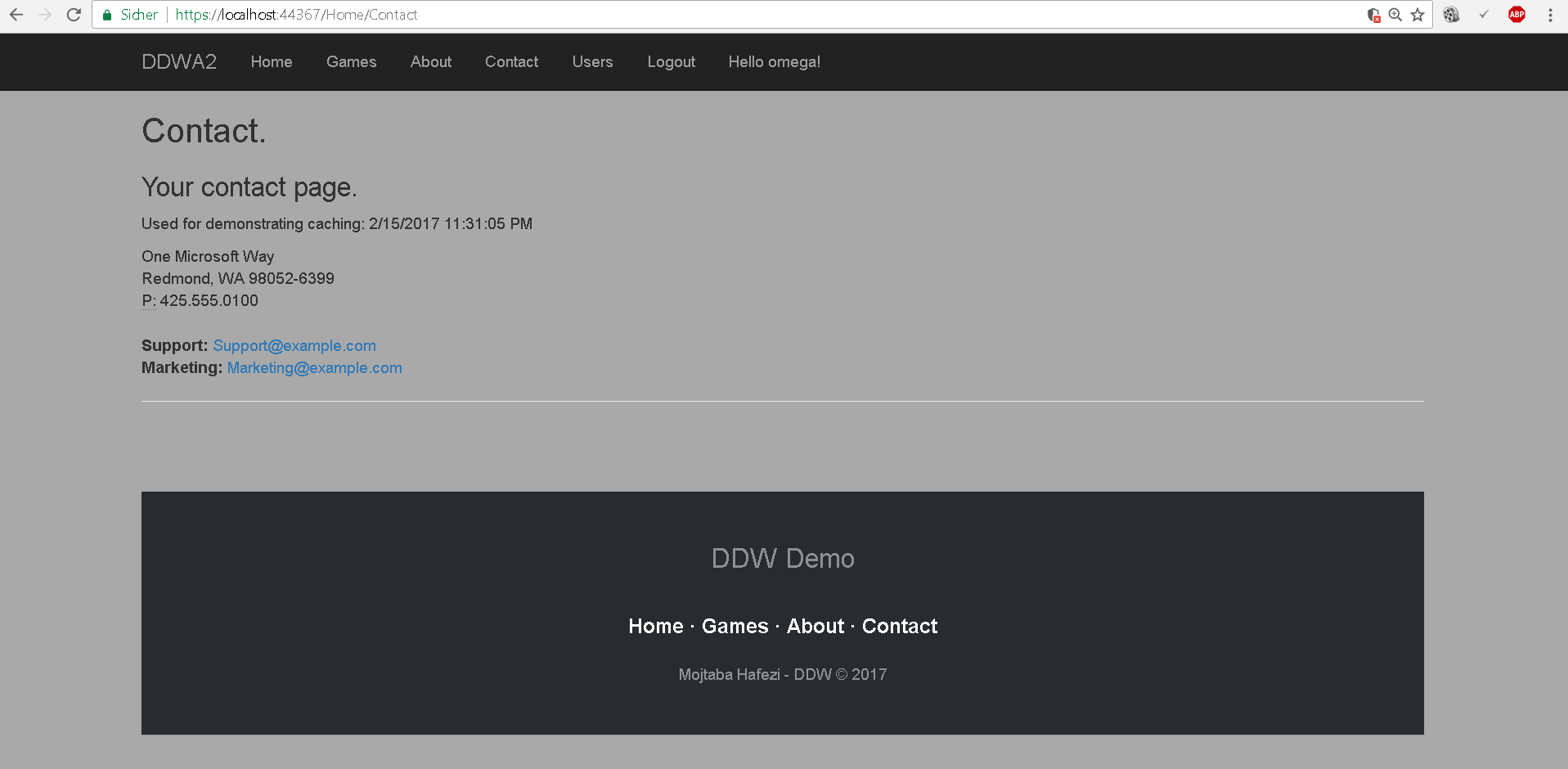
Filtering was implemented through the usage of **LINQ**.

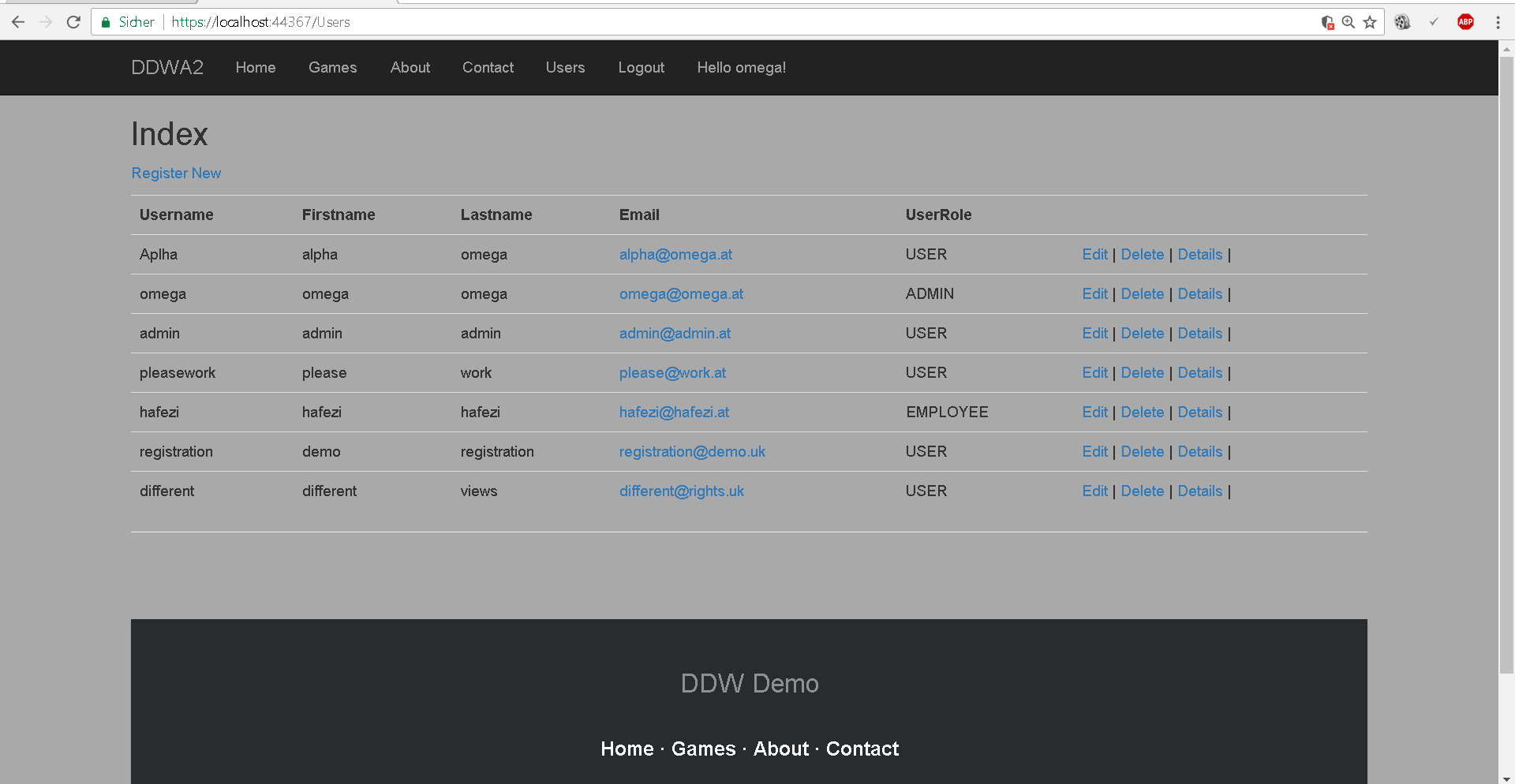
1. Design, usability and layout:

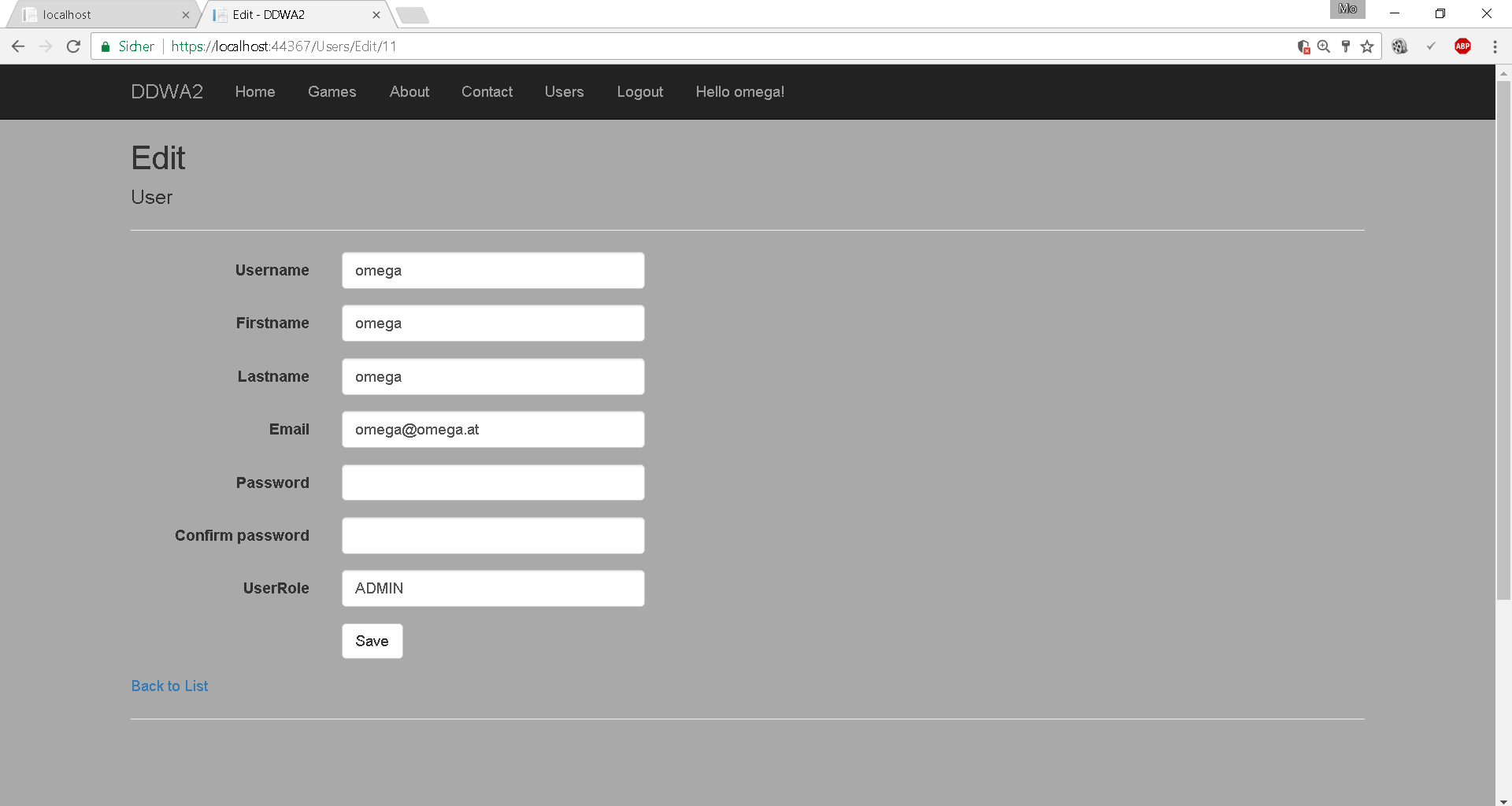
A well-designed website is provided with responsiveness and usability. From here onwards only additional screenshots of the website are shown.

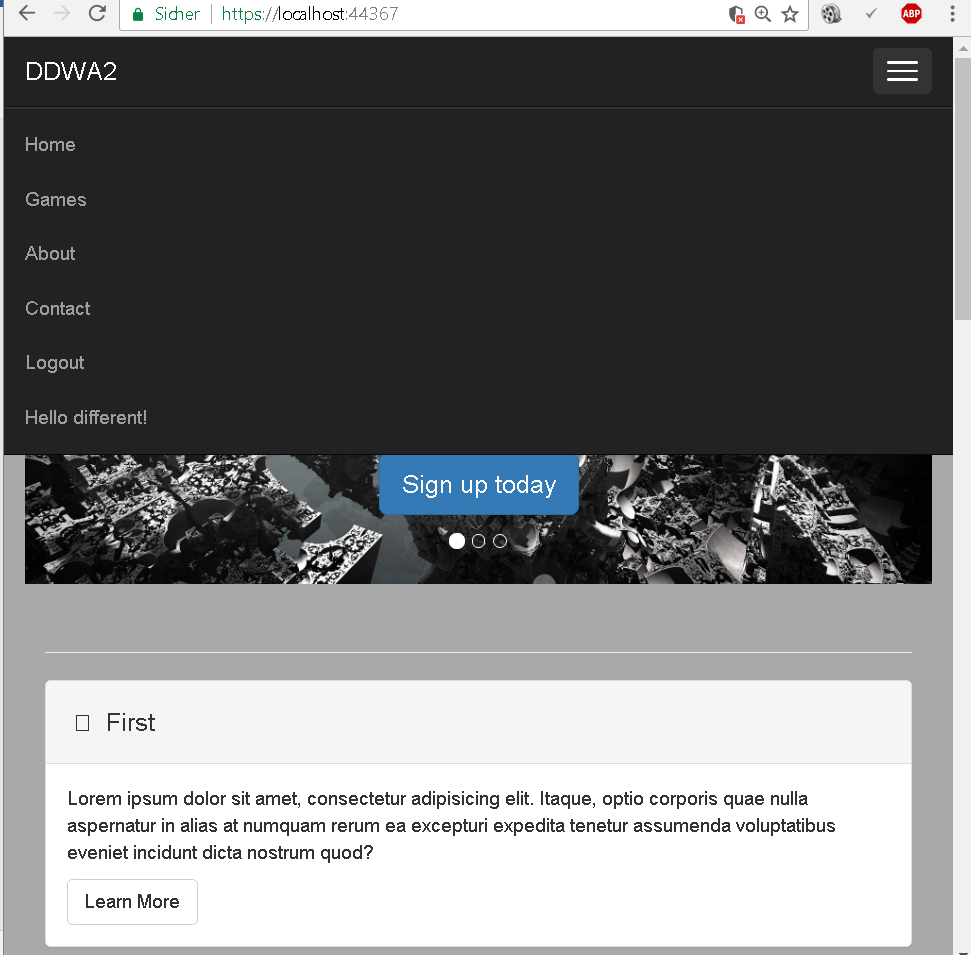












Last picture: Upon resizing the window until it is not wide enough the layout changes.