Examentraining 1

Realiseren

hoofdstuk

1

Authenticatie binnen een applicatie

Algemene informatie

Onderwerp	Authenticatie binnen een applicatie
Leerdoel(en)	1. De student realiseert een authenticatiesysteem 2. De student realiseert een rollensysteem 3. De student realiseert password hashing
Vereiste voorkennis	1. De student heeft uitgebreide OOP voorkennis. 2. De student heeft uitgebreide MVC architectuur voorkennis
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Je hebt in alle voorgaande realiseren lessen geleerd hoe je een goedwerkende applicatie kunt bouwen waarin de gebruiker alle functionaliteiten beschreven kan uitvoeren.

Echter missen we een laatste en belangrijk deel van onze applicatie: Authenticatie van gebruikers en het bepalen van hun rechten (wie mag wat?). Zo mag een reguliere gebruiker bijvoorbeeld geen settings aanpassen en mag een administrator dit juist weer wel.
Wellicht mag een caissière alleen producten scannen en mag een baliemedewerker weer de prijzen van een product aanpassen? Dit zijn allemaal dingen die je kunt programmeren.

Een gebruiker moet dus inloggen voordat de applicatie gebruikt kan worden. Daarna wordt er per scherm bepaald of de gebruiker deze specifieke actie wel of juist niet mag gebruiken.

Het maken van een loginscherm en daaromheen toegangscontrole noemen we een authenticatiesysteem

Inhoud

→ Je kunt onderstaande stappen volgen om zo een authenticatiesysteem in te bouwen. Het is echter geen stappenplan, want in de les gaan we dieper op de stof in. Je dient dus de lessen te volgen om deze stappen volledig te begrijpen.

1. Basisbeginselen

Autorisatie binnen een applicatie is een relatief simpel proces. Een gebruiker voert een username en password in, de server/database controleert of er een match is. Zo ja, dan geeft de server/database het inlogniveau (in de vorm van een getal, zie 1.1 rechtentabel) en kun je per formulier bepalen of deze gebruiker toegang heeft tot die functionaliteit.

We nemen in de komende hoofdstukken de werking van een supermarkt als voorbeeld.

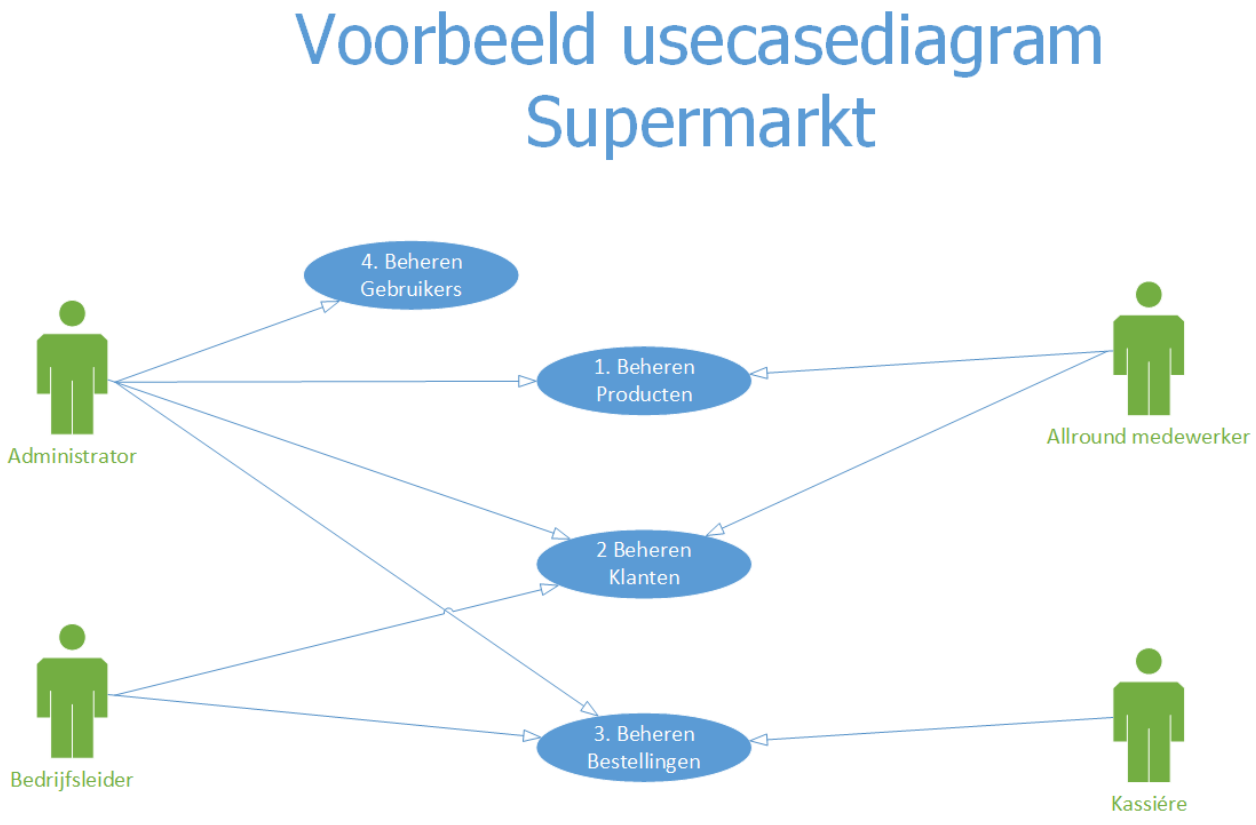
1.1 Usecasediagram

Bepaalde gebruikers mogen meer in jouw applicatie dan andere gebruiker. Een kassière mag bijvoorbeeld minder dan de bedrijfsleider. Je moet dus vooraf nadenken over de rechten en wie wat mag. Je kunt deze informatie uit de userstories en usecasediagrammen halen waardoor je kunt bepalen wie wat mag. Bijvoorbeeld mag een kassière alleen een actie op Bestellingen doen, terwijl een bedrijfsleider weer juist Bestellingen, Klanten en Producten mag beheren. Dit noemen we de rol van een gebruiker of ook authenticatierol genoemd.

Authenticatierol

We gebruiken dus het woord authenticatierol om te beschrijven welke rechten een bepaalde gebruiker heeft. De authenticatierol medewerker heeft dus bepaalde rechten (zie usecasediagram hieronder).

Hieronder zie je de usecasediagram van onze supermarkt:

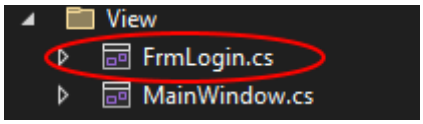


2. Basis: Een loginscherm maken

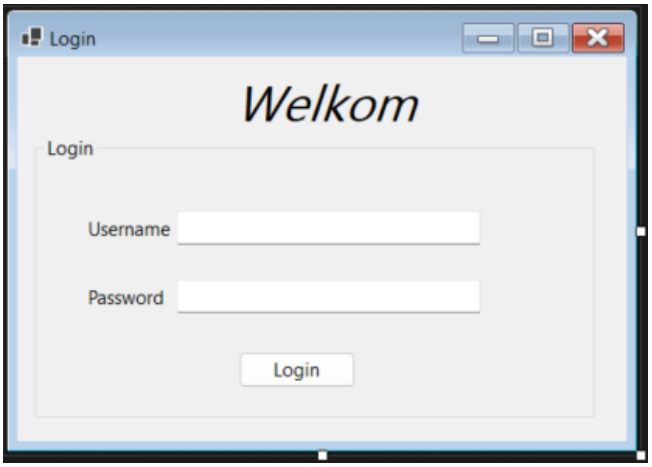
2.1 View

Het loginscherm dient als eerste scherm van de applicatie getoond te worden. Daarna mag de gebruiker pas verder gaan naar het hoofdmenu.

→ Maak in de View map een nieuw WinForm aangenaamd *FrmLogin* aan.



→ Daarin maken we het login scherm. Hieronder een voorbeeld.



- Maak het passwordveld geheim door de property *PasswordChar* te gebruiken.

Program.cs

In Program.cs kun je *Main()* method vinden. Deze methode is het startpunt van je applicatie. In de Main methode definieer je bijvoorbeeld welk formulier als eerste getoond dient te worden.

→ Zorg ervoor dat je loginformulier als eerste opgestart wordt.

```
// Show login screen
Application.Run(new View.FrmLogin());
```

Je kunt nu **Taak 1** van **oefening 1.1** maken.

2.2 Model en database

Database

De inloggegevens van de gebruikers zijn opgeslagen in een speciale tabel (*UserLogin* genoemd). Ook slaan we de authenticatierollen (zie kopje "Authenticatierollen") op van alle gebruikers in deze UserLogin table.

We slaan dus de **username**, **email**, **volledige naam** en **authenticatierol** op in deze tabel. Je ziet hieronder een overzicht:

Columnname	Datatype	Lengte	PK?	Not Null?
LoginUserName	Varchar	64	Ja	Ja
LoginPassword	Varchar	60		Ja
LoginUserEmail	Varchar	255		Ja
LoginUserRealName	Varchar	255		Ja
LoginUserRole	Varchar	64		Ja

Let op: De kolom LoginPassword heeft een max lengte van 60 karakters. Dit is bewust gedaan, zie het hoofdstuk "Password Hashing".

Vanuit bovenstaande gegevens kun je de *CREATE TABEL* SQL-code maken om deze UserLogin tabel te maken.

Model

Vanuit deze UserLogin ga je daarna het Model bouwen in C#-code. Dit hebben we al zo vaak gedaan, dat we hier nu verder op in gaan 😊

 Je kunt nu **Taak 2** van **oefening 1.1** maken.

2.3 Controller

De Controller is de plek waar alle puzzelstukjes gaan samenvallen wat betreft het inloggen, ingelogd blijven en het bijhouden van de rechten van de gebruiker. Daarom dat we deze controller wat meer laten doen dan je tot nu toe gewend bent.

→ Maak een nieuwe Controller genaamd **LoginUserStaticController** aan.

Static Controller

Deze Controller is speciaal omdat hij data dient te bewaren die door de gehele applicatie heen opvraagbaar/benaderbaar moet zijn. De "normale" OOP programmeerfilosofie kun je hiervoor niet gebruiken omdat deze vereist dat je steeds een nieuw *Object* aanmaakt (en dus je data weer kwijt bent). Daarom maken we van de Controller een zogeheten *Static* Controller. Meer informatie over *static* in de lessen en in het hoofdstuk "Static classes, methods en properties".

```
public static class LoginUserStaticController
{
```

→ Maak nu de Controller static.

De controller is nu klaar om *static* data (properties in ons geval) te beheren. Extra *public* (dus zichtbaar voor de gehele applicatie) properties/data die we gaan inbouwen in de controller:

Propertienaam	**Datatype**	**Opmerking**
public static IsLoggedIn	boolean	Is er een gebruiker ingelogd in de applicatie?
public static CurrentUser	LoginUserModel	De LoginUser die momenteel is ingelogd

Zoals je misschien gemerkt hebt, maken we de properties *public static*. De term public/private ken je, maar static hebben we nog niet eerder behandeld. Het is cruciaal dat we de properties static maken (om de data voor de gehele applicatie beschikbaar te maken)

In de les leer je meer hierover. Zie het kopje "Static classes, methods en properties" voor een korte introductie.

Daarnaast bevat deze Controller ook de onderstaande methodes. Meer informatie over CheckLogin() lees je verderop in deze reader.

Methodenaam	**Parameters**	**Opmerking**
static *bool* **CheckLogin**()	*string* password, *string* username	Controleer in de DB op bestaande gegevens en geef true/false terug. Sla daarnaast *IsLoggedIn* en *CurrentUser* op.

→ Voeg de gegeven properties en de CheckLogin methods toe.

De CheckLogin() methode

```
public static LoginUserModel CheckLogin(string username, string password)
```

In de CheckLogin() methode doe je al het harde werk wat betreft het controleren van de gegevens in de database (heeft de user een correcte of incorrecte inlogpoging gedaan).

Zodra er een juiste username/password combinatie gevonden is, dient de controller dit te onthouden (de user is immers correct ingelogd!). Hiervoor hebben we de twee static properties genaamd *IsloggedIn* en *CurrentUser* aangemaakt. Doordat deze *static* zijn, zijn deze gegevens door de gehele applicatie heen te gebruiken. Hier komen we later op terug.

SQL-Query

Uiteindelijk moet de onderstaande SQL-query uitgevoerd worden. Hoe dit werkt, gaan we niet nog eens uitleggen 😜.

```
// Laat de database bepalen of de inlogpoging correct is
string query = "SELECT * FROM LoginUser WHERE Username = @usernameValue AND Password = @passwordValue";
```

Is er wel/niet correct ingelogd?

Zodra bovenstaande query is uitgevoerd kunnen we via de *HasRows* property bepalen of een inlogpoging geslaagd is (of niet). Want zodra er een database row is gevonden (via HasRows), weten we dat.

Hiernaast zie je een codevoorbeeld:

```
// Query uitvoeren
SqlDataReader reader = command.ExecuteReader();

// Lees één record
reader.Read();

// Is er een correcte inlog?
if(reader.HasRows == true)
{
    // De gebruiker is correct ingelogd!
}
else
{
    // Foutieve poging
}
```

Opslaan van data in de statics

Ok, we hebben een correcte inlogpoging te pakken! Omdat deze data (ingelogde gebruiker) door de globaal door de gehele applicatie beschikbaar dient te zijn hebben we statics aangemaakt om ze in op te slaan. Hoe doe je dit? Simpel: Exact hetzelfde als met normale "Object" variabelen:

```
// Sla deze gegevens op in de static properties
IsLoggedIn = true;
CurrentUser = new LoginUserModel();

CurrentUser.LoginUserEmail = (string) reader["LoginUserEmail"];
CurrentUser.LoginUserRole = (string)reader["LoginUserRole"];
CurrentUser.LoginUserName = (string)reader["LoginUserName"];
CurrentUser.LoginUserRealName = (string)reader["LoginUserRealName"];
```

Je bent nu in staat om deze CheckLogin() methode te bouwen. Dit ga je oefenen in oefening 1.1.

 Je kunt nu **Taak 3** van **oefening 1.1** maken.

2.4 De static Controller binnen Views gebruiken

Nu je geleerd hebt hoe je een static property/method/property bouwt is het tijd om te leren hoe je dit kunt gebruiken binnen de Views (formulieren dus) kunt gebruiken.
Belangrijk is om te weten, dat je van een static controller niet het statement *= new()* mag/kunt gebruiken (want een static is namelijk geen Object). Je spreekt een static direct aan via de naam van de class (Inclusief hoofdletters). Zie onderstaand voorbeeld:

```
// Welkoms melding aan de gebruiker
string realName = LoginUserStaticController.CurrentUser.LoginUserRealName;
MessageBox.Show("Welkom " + realName);
```

Het opvragen en tonen van de "echte naam" van de ingelogde gebruiker via de static property CurrentUser

→ Maak in het **load event** (dus niet in de *constructor*) van dit formulier de onderstaande code aan om te bepalen of een user ingelogd is:

```
// Via de IsLoggedIn static kunnen we bepalen of de gebruiker is i
if (UserLoginStaticController.IsLoggedIn == false)
{
    // Huidig formulier sluiten
    MessageBox.Show("U bent niet ingelogd. Login voor toegang");
    this.Close();
}
```

Het gebruik van de static property binnen een if/else statement

Je ziet hierboven dat je de static properties / methods als reguliere variabelen kunt gebruiken. Je dient de exacte naam van de static controller (inclusief hoofdletters) te gebruiken, hierdoor krijgt de naam van de class een ander kleurtje.

 Je kunt nu **Taak 4** van **oefening 1.1** maken.

2.4 Toegangscontrole via rollen

Je hebt hierboven geleegd hoe je een inlogscherm bouwt, met daarna een check of er wel correct is ingelogd. Alleen weet je nog niet hoe je een bepaald formulier alleen voor de bedrijfsleider of Administrator laat zien. Een normale kassamedewerker mag bijvoorbeeld niet het formulier met de financiële gegevens zien.

Bepalen per formulier welke rol toegang heeft

Eerst moeten we bepalen welke rollen wel toegang hebben tot het formulier. Daarom maken we onderstaande Array aan binnen het formulier:

```
public partial class FrmSecret : Form
{
    // Wie mag deze pagina bezoeken?
    string[] allowedRoles = { "Administrator", "Bedrijfsleider" };
```

In de Array hierboven definiëren we welke rollen toegang hebben tot dit formulier.

Omdat we een Array gedefinieerd hebben kunnen we via de .**Contains()** methode controleren of de rol van de gebruiker voorkomt in de Array. Is dit niet het geval, dan mag de gebruiker de pagina niet bezoeken. Je krijgt dan de volgende code:

```
// Komt de rol van de CurrentUser voor in allowedRoles Array?
if(allowedRoles.Contains(LoginUserStaticController.CurrentUser.LoginUserRole) == false)
{
    MessageBox.Show("U mag hier helemaal niet in! Wegwezen.");
    this.Close();
}
```

Bekijk hoe hierboven via de .Contains() methode gecontroleerd wordt de CurrentUser role in de eerder gemaakte Array bestaat.

Belangrijk:

Bovenstaande code kun je alleen uitvoeren als je zeker weet dat CurrentUser niet een nullwaarde bevat. Check dus altijd eerst of de *IsLogged* in wel true is en daarna pas de **Contains()** uitvoeren.

 Je kunt nu **Taak 5** van **oefening 1.1** maken.

3. Wat zijn statics?

We gebruiken/misbruiken statics om data op te slaan, die door de gehele applicatie heen beschikbaar dient te zijn. Een static is exact het tegenovergestelde van OOP-programmeren (Object Georiënteerd Programmeren). Want van een static zal altijd maar één van bestaan, terwijl objecten bedoeld zijn om er juist heel veel van te hebben.

Zowel een Class als Property als ook een Method kunnen static gemaakt worden. Zodra we bijvoorbeeld een Class als static definiëren betekend dit, dat er maar slechts 1 "kopie" van de class in de applicatie kan zijn.

Een static kan niet via =new() geïnstantieerd worden.

```
// Dit kan en mag dus niet:
LoginUserStaticController contr = new LoginUserStaticController();
```

 Wil je meer leren over statics? Bekijk onderstaande video:

https://www.youtube.com/watch?v=eHU2vdw9OHI

4. Security: Password hashing

In oefening 1.1 heb je de database gevuld met een aantal medewerkers. Zodra we de database eens gaan bekijken ontdekken we een nogal groot beveiligingsrisico:

Als je nu in de LoginUser database gaat kijken zie je nogal een flink securityrisk ontstaan:

Results Messages

	LoginUserName	LoginPassword	LoginUserEmail	LoginUserRealName	LoginUserRole
1	JKeesma	Vjfg67Je!!	j.keesma@dominos.nl	Jan Keesma	Ploegleider
2	JvdWetering	jvdwe1987!	jvdwetering@dominos.nl	Jeroen van de Wetering	Bedrijfsleider
3	KJansma	keesjuuuuu12	k.jansma@dominos.nl	Kees Jansma	Bedrijfsleider
4	KW1CCollege	P@ssword4321	administratie@kw1c.nl	ICT-Academie	Administrator
5	RvanEsch	HteJehb!Qt4v	r.vanesch@dominos.nl	Rob van Esch	Ploegleider
6	Spierings01	Hondje2008	ro.spierings@kw1c.nl	Ron Spierings	Administrator

De wachtwoorden zijn vrij leesbaar (unencrypted genoemd)! Zodra je database uitlekt (compromised genoemd) zijn de emailadressen, usernames EN passwords uitgelekt en heb jij ervoor gezorgt dat jouw users een mega groot probleem hebben doordat kwaadwillende via deze inloggegevens ook op andere services kunnen proberen in te loggen (veel mensen gebruiken hetzelfde wachtwoord voor veel andere services).

 Lees meer over hashing:

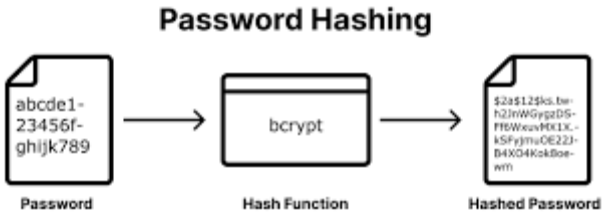
https://www.okta.com/uk/blog/2019/03/what-are-salted-passwords-and-password-hashing/#:~:text=Password%20hashing%20is%20defined%20as,series%20of%20numbers%20and%20letters

 Een goed artikel, met Python codevoorbeelden over enkele hashing methodes

https://auth0.com/blog/hashing-passwords-one-way-road-to-security/

4.1 BCrypt library

We gebruiken de BCrypt hashing methode en library om wachtwoorden te husselen (encrypten genoemd), zodat de wachtwoorden dusdanig veranderen, zodat ze niet meer terug te lijden zijn naar je originele wachtwoord. Dit proces noemen we Hashing.



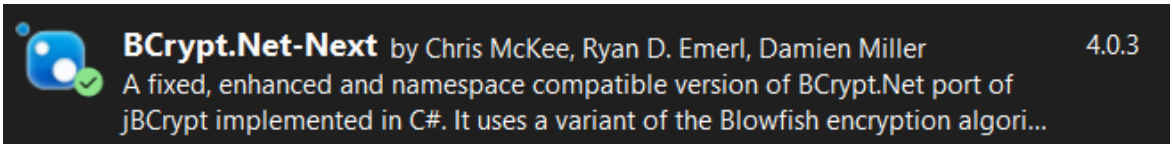
Het wachtwoord **P@ssword1234** wordt in de database dan:

$2a$10$jw.k9eMKCBCRNle1EIC4OOuO2RAK55J90kC9VAiE3/fW00v/fWVkC

Zodra je database toch uitlekt, dan kan een hacker nooit meer jouw originele wachtwoord achterhalen.

We gebruiken de encryptiemanier genaamd genaamd BCrypt. Deze is momenteel het meest gebruikt en relatief erg veilig. Maar er zijn meerdere hashingmethodes (zoals SHA2, SHA256 en het onveilige MD5) die je zou kunnen gebruiken.

→ Voeg onderstaande NuGet library (de versie kan inmiddels anders zijn):



Daarna kun je een hash maken via onderstaande code. Deze hash sla je dus op in de database:

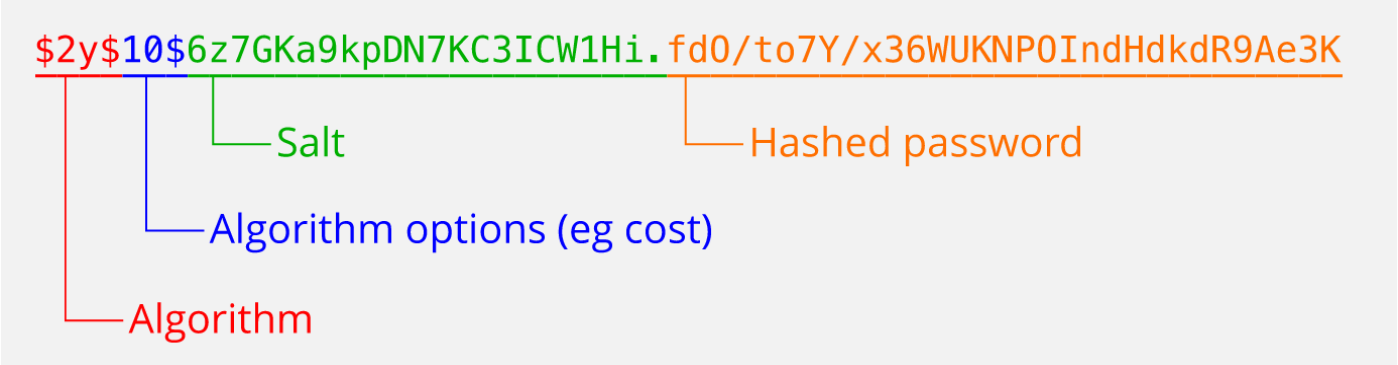
```
// Een pasword hashen via BCrypt
string passwordHash = BCrypt.Net.BCrypt.HashPassword(txtPassword.Text);
```

Of via een online tool.
https://bcrypt-generator.com/

→ Verander nu alle wachtwoorden in je *UserLogin* table naar een gehashte variant.

Opbouw van een BCrypt hash:

De opbouw van een BCrypt hash heeft een hele speciale opbouw. Alle karakters die je ziet, hebben een specifieke positie en functie. Zie afbeelding hieronder:



Twee hashes vergelijken met elkaar

Je wachtwoorden heb je inmiddels in je Table gehasht. Nu is het tijd om je CheckLogin() methode om te bouwen, zodaning dat de **Verify()** methode (uit het BCrypt library) gebruikt kan worden om de invoer te gaan checken.

Begin met het aanpassen van de SQL-query in de CheckLogin() methode, zodat deze de password niet meer uit de database haalt, maar alleen de Username. Zie code hieronder.

```
// Laat de database bepalen of de inlogpoging correct is
string query = "SELECT * FROM LoginUser WHERE LoginUserName = @usernameValue";
```

Bouw daarna onderstaande if/else statement in, zodat binnen de HasRows if/else ook gaat controleren of de hash van het password ook overeen komt met de hash van het ingevoerde password:

```
// Lees één record
reader.Read();

// Is er een correcte inlog?
if(reader.HasRows == true)
{
    // Komt het wachtwoord overeen met de hash?
    if(BCrypt.Net.BCrypt.Verify(password, (string)reader["LoginPassword"]) == true)
    {
        // Sla deze gegevens op in de static properties
        IsLoggedIn = true;
```