

DotNet Advanced – Examenvoorbereidingsopgave:

Inleiding

We gaan een applicatie maken die een soort game moet voorstellen, in deze opgave wordt gebruik gemaakt van Code-First en Ado.Net om handelingen uit te voeren op de databank.

Deze opgave zal van scratch opgezet worden om tot het eindresultaat te komen.

Voor ieder stuk van de opgave zal er op tijd gezegd worden wanneer je een nieuw project moet aanmaken en welke naam je dit project geeft.

De delen volgen elkaar op maar zijn zo opgesteld dat de zolang je de tabellen hebt kunnen genereren je verder kan met het volgende deel.

Doel:

We gaan de volgende systemen uitwerken:

- Loginsysteem door gebruik te maken van het EF.
- Game-systeem die Characters en accounts met elkaar verbindt.
- Monster-gedeelte dat we uitwerken met behulp van het 3-lagen-model (we komen hier later op terug).

We zullen werken met de volgende tabellen:

Account	Character	Monster
AccountId	CharacterId	MonsterId
CharacterId	AccountId	HP
Gebruikersnaam	Naam	Attack
Wachtwoord	HP	Alive
	Attack	Name
	Mana	
	Alive	

Deel 1: Entity Framework

We passen het concept van Code-First toe om de databank voor deze opgave op te zetten.

Je DbContext noem je **GameContext**

Het schema van de databank moet als volgt zijn:

ExamenJanuari.Account
ExamenJanuari.Character
ExamenJanuari.Monster

1. Leg de nodige relaties tussen de tabellen
Maak gebruik van de Fluent API om de volgende relaties tussen de tabellen te leggen, voeg properties en klassen toe die nodig zijn om dit tot een goed einde te brengen:
 - Een account kan slechts 1-of-geen character hebben.
 - Een character moet altijd een account hebben (1-op-1)
2. Zorg ervoor dat je je databank aanmaakt door gebruik te maken van Migrations.
3. Gebruik de Fluent-API om de Primary Key en de Foreign Key aan te duiden.
4. Gebruik dataAnnotations om de properties aan te passen:
 - Properties van het type string mogen een maximumlengte van 60 hebben en moeten verplicht ingevuld worden.
5. Zorg ervoor dat je in de Seed -methode van je programma een Account aanmaakt met de data:

AccountID	Gebruikersnaam	Wachtwoord
1	Test1	Test1
2	Test2	Test2
3	Test3	Test3
4	Test4	Test4
5	Test5	Test5

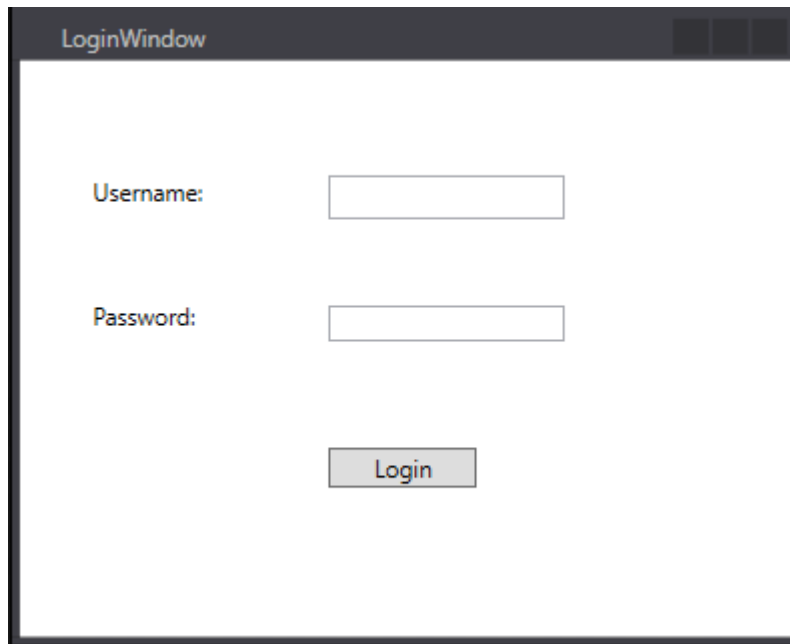
6. Maak ook 3 Characters aan en link deze aan de Account1, Account3 en Account5

CharacterId	Alive	Attack	HP	Mana	Naam
1	True	20	100	100	Char1
2	False	50	0	0	Char2
3	True	100	1000	500	Char3

Zorg zelf voor de juiste navigatieproperties die nodig zijn.

Layout & functionaliteit:

Maak een window genaamd **LoginWindow** hier geef je een username in en een wachtwoord, wanneer je op Login klikt, moet je door middel van het EF controleren of de gegevens voorkomen in de databank (schrijf hier dus een methode voor).



Deel 2: ADO.NET

We gaan ervoor zorgen dat we monsters kunnen toevoegen aan onze databank, we gaan dit volledig doen door gebruik te maken van ADO.NET.

Maak een nieuw project aan en noem dit **Ado3LagenMonster**.

Maak hierin de nodige klassen (Monster, MonsterDB → Je kan ook de Monster klasse van EF gebruiken) aan en zorg voor een klasse **ConnectDB** die een methode `GetConnection()` zal bevatten.

Zorg ervoor dat je deze klasse kunt oproepen door `ConnectDB.GetConnection()` te gebruiken in je code.

Schrijf een methode (`GetNextMonsterId`) in de `MonsterDB` klasse die de volgende ID van een nieuw monster kan berekenen.

Schrijf de methoden die ervoor zorgen dat je een Monster kunt:

- Toevoegen
- Verwijderen
- Updaten

Layout:

Maak een Window genaamd **MonsterWindow** dat er als volgt uitziet:
Het onderstaande scherm heeft 3 buttons en een Datagrid.

Zorg ervoor dat als je succesvol kan inloggen (zie EF) je dit scherm voor je krijgt.

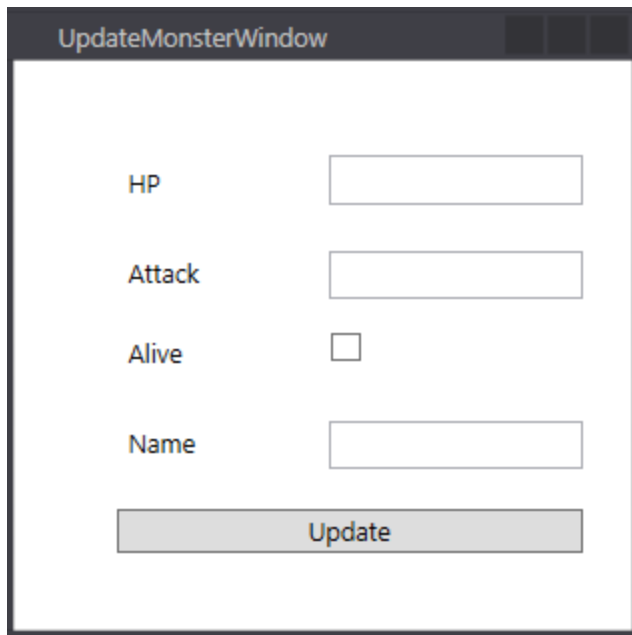
Bij het opstarten van het scherm moet je alle Monsters uit de databank ophalen en deze via databinding in de Datagrid plaatsen.



Add Monster (wanneer je op de knop drukt, opent er zich een dialogwindow):

The image shows a dialog window titled "Add Monster". It contains four input fields: "HP" (a text box), "Attack" (a text box), "Alive" (a checkbox), and "Name" (a text box). Below these fields is a button labeled "Add".

Update Monster:



Zorg ervoor dat al deze windows automatisch van data voorzien worden door gebruik te maken van Databinding

Indien je data aanpast in het Update-window, dan moet zich dat automatisch aanpassen in de Datagrid en natuurlijk weggeschreven worden naar de databank.

Je kan bv een ObservableCollection in samenwerking met het DataContextChanged event van de datagrid en wat databinding gebruiken om de datagrid te updaten

Ook als je delete moet de DataGrid aangepast worden en moet de record verwijderd worden zowel in de datagrid als in de databank.

Voor het deleten zorg je ervoor dat je slechts 1 rij in de datagrid kunt selecteren en vervolgens geef je dat record door aan de delete methode die je in het Ado.Net gedeelte hebt geschreven.

Voorbereiding op Unit-Tests

Schrijf een Fight(Monster m) methode in de klasse Character die een monster kan aanvallen. Als je een monster aanvalt, gaat zijn HP naar beneden met de waarde van de Attack van het character.

Hierna controleer je of het monster nog leeft, zoniet, pas je de Alive waarde aan.

Je doet hetzelfde voor de Fight methode in de klasse Monster.

Schrijf voorts nog een Heal(int x) methode in de Character klasse die het hp van je character met x omhoog doet.

Unit-testen:

Maak een nieuw project en noem dit **GameUnitTests**

Maak de volgende unit-testen door gebruik te maken van NUnit:

Werk de volgende unit-testen uit in een aparte klasse **MonsterTest**

- ShouldCreateMonsterTest → Controleert of een monster aangemaakt wordt.

- ShouldDeleteMonsterTest → Controleert of een monster uit de db verwijderd wordt.

Werk de volgende unit-testen uit in een aparte klasse **LoginTest**

- ShouldLoginUser → Controleert of een gebruiker ingelogd kan worden.
- ShouldNotLoginUser → Controleert of een gebruiker niet ingelogd kan worden.

Werk de volgende unit-testen uit in een aparte klasse **FightTest**:

- MonsterShouldTake20DamageTest
- CharacterShouldTake10DamageTest
- CharacterShouldDieTest
- MosterShouldDieTest
- CharacterShouldHeal50Test

Werk de volgende unit-testen uit in een aparte klasse **AfterSeedTest**

- ShouldHave3Characters → Deze test run je nadat je je migration gedaan hebt, als je migratie succesvol is verlopen zal je 3 characters in je Characterstabel hebben
- ShouldHave5Accounts → Deze test run je nadat je je migration gedaan hebt, als je migratie succesvol is verlopen zal je 5 accounts in je Accountstabel hebben

Op andere momenten zullen deze testen falen omdat je nieuwe data insert en delete.