**Implementatie details**

Bij deze opgave dien je de voorgestelde aanpak strikt op te volgen!  
In de solution vind je de volgende projecten:

* ChatBot.Web -> de eigenlijke (MVC) applicatie waarin ook een API verweven zit.
* ChatBot.Web.Tests -> bevat testen voor het Web project
* ChatBot.Data -> de datalaag. Via deze laag loopt de opslag van data in de database.

**Datalaag (3pt)**

Zorg ervoor dat Enitiy Framework de class ‘ChatBotDbContext’ kan gebruiken om een model in het geheugen op te bouwen. De domainclass ‘Feedback’ (in folder DomainClasses) zou moeten resulteren in een ‘Feedback’ tabel in de database. De database zou ook de tabellen moeten genereren die gebruikt worden door het Identity framework. Deze tabellen zouden gebaseerd moeten zijn op de domainclass ‘User’ (in folder ‘domainclasses’).

Schakel (automatische) migraties in. Zorg er, in de ‘Seed’ methode, voor dat een dummy user in de databank wordt gestopt. Maak hierbij gebruik van de ‘UserManager’ class van het Identity framework. De dummy user heeft volgende eigenschappen:

* UserName = [dummy@pxl.be](mailto:dummy@pxl.be)
* Email = [dummy@pxl.be](mailto:dummy@pxl.be)

Stel het wachtwoord van de gebruiker in op ‘dummy123!’

Vervolledig de methode ‘SetInitializer’ van ‘ChatBotDbContext’. De methode moet er voor zorgen dat entity framework de database aanmaakt indien hij nog niet bestaat en migreert naar de laatste versie indien de database reeds bestaat. Zorg er ook voor dat ‘SetInitializer’ op de juiste plaats in de applicatie opgeroepen wordt.

Zorg er voor dat de class ‘FeedbackdbRepository’ de interface ‘IFeedbackRepository’ implementeert. Deze class maakt gebruik van ‘ChatBotDbcontext’ om feedback toe te voegen aan de database.

Tips:

* Async await
* ‘SaveChangesAsync’ i.p.v. ‘Savechanges()’

**MVC – ChatroomController (4pt)**

Op de startpagina (‘Index’ actie van de ‘HomeController’) worden 2 linken getoond naar 2 chatrooms met hun bijhorende chatbot. De url’s van de linken zien er als volgt uit:

* /ChatRoom/HostedBy/BenLamb
* /ChatRoom/HostedBy/Francis

Je gaat er nu voor zorgen dat deze linken effectief werken. Zorg ervoor dat requesten naar bovenstaande url’s afgehandeld worden door de ‘ChatBot’ actie van de ‘ChatRoomController’. Maak hiervoor gebruik van een ‘RouteAttribute’. Zorg er ook voor dat het laatste deel van de url (de naam van de bot) kan weggelaten worden. Als er geen naam in de url gevonden wordt, dan wordt de naam ‘Frank’ aangenomen.

Ga naar de ‘ChatRoomControllerTests’ class in het test project (onder ‘Controllers’). Vervolledig 1 voor 1 de testen zodanig ze zinvol zijn. De ViewModel class ‘ChatBotViewModel’ krijg je cadeau. Merk op dat de class ‘ChatroomController’ een publieke constante heeft die zowel in testcode als productiecode gebruikt kan worden. Schrijf de testen zonder gebruik te maken van mocking. Voor de tweede test moet je ervoor zorgen dat het mogelijk is om de ‘ChatBot’ action aan te roepen zonder parameter. Maak de testen groen. Refactor indien mogelijk.

De View ‘ChatBot.cshtml’ krijg je cadea. Alleen moet hier nog een link naar het javascript bestand ‘chatbot.js’ toegevoegd worden. Doe dit, rekening houdend met volgende richtlijnen:

* De script-tag moet net voor de sluittag van de <body>-tag komen
* De script-tag mag enkel voorkomen in de ‘chatBot’ view
* Gebruik de static methode ‘Render’ van de ‘Scripts’ class om de script-tag te genereren. Als parameter geef je ‘~/Scripts/chatbot.js’ mee.

**Verwijder of wijzig geen bestaande code uit de view. Voeg enkel code toe!**

**API – ChatBotController (4pt)**

Het script ‘chatbot.js’ zorgt ervor dat er een http POST request gestuurd wordt naar de url ‘/api/chatbot’ als je op de knop ‘Send’ klikt. De body van de request bevat de inhoud van de textbox (in json-formaat). In de body van de response zou het antwoord van de chatbot moeten zitten samen met satus code 200 (OK). Het script zorgt ervoor dat dit antwoord in de html getoond wordt. Het zou ook kunnen dat je een slechte vraag hebt gesteld. In dat geval zou de API moeten antwoorden met een status code 400 (bad request) en een foutboodschap in de body. Het script zorgt er voor dat deze foutboodschap in de html getoond wordt. Als er iets anders mis gaat bij de http request, dan zorgt het script ervoor dat er een samenvatting van de http response getoond wordt in de html. **Aan het script ‘chatbot.js’ hoef je dus niets te wijzigen!** Waar je wel nog voor moet zorgen is dat de API werkt. In dit project is de WebApi verweven in het ChatBot.Web project ( Dit is dus zowel een MVC als een WebApi project).

Ga naar de ‘ChatBotControllerTests, maak 1 voor 1 de testen ………………………………… Het generen van het antwoord mag niet door de ‘ChatBotController’ zelf gebeuren. De ‘ChatBotcontroller moet gebruik maken van een (geïnjecteerd) instantie van de interface ‘IAnswerGenerator’. Deze interface krijg je cadeau en vind je in de folder “Models/Services”. Maak de testen groen. Refactor indien mogelijk.

**PxlAnswerGenerator (3pt)**

De ‘PxlAnswerGenerator’ class (‘Models/Services’ folder) is een implementatie van de ‘IAnswerGenerator’ interface. Ga naar de ‘PxlAnswerGeneratorTests’ class in het test project (onder ‘Services’). Vervolledig de asserts in de testen zodat ze zinvol zijn. Maak beide testen groen. Refactoren mag je overslaan. Merk op dat de class ‘PxlanswerGenerator’ 2 publieke constanten heeft die zowel in testcode als productiecode gebruikt kunnen worden.

Gebruik het Simple Injector Framework om er voor te zorgen dat er gedurende de hele levensduur van de appliatie slechts 1 instantie van ‘PxlAnswerGenerator’ wordt aangemaakt. Deze instantie wordt geïnjecteerd in de ‘ChatBotController’ wanneer het nodig is. Vervolledig hiervoor de ‘RegisterWebApiDependencies’ methode van de ‘DependencyConfig’ class in de ‘App\_Start’ folder.

**Layout (1pt)**

Voeg in de navigatiebalk van de applicatie een link toe naar de “Index” actie van de HomeController en een tweede link naar de “Feedback” actie van de HomeController. Maak hierbij gebruik van een (bestaande) html helper.

Zorg dat in de footer het huidige jaar gevolgd door ‘@PXL’ getoond wordt. (2018@PXL)

**MVC – HomeController (5pt)**

Ga naar de ‘HomeControllerTests’ class in het test project (onder ‘Controllers’). De test ‘Feedback\_Get\_ShouldReturnViewWithNewFeedbackModel’ krijg je als cadeau. Maak hem groen. Het is wel mogelijk dat je deze test moet wijzigen als je aan het refactoren bent.

Vervolledig 1 voor 1 de testen.

Maak de ‘Feedback’ post action asynchroon door te werken met ‘async’ en ‘wait’. Zorg ervoor dat de HomeController gebruik maakt van de ‘IFeedbackRepository’ interface uit de datalaag. De ‘Naughty’ view krijg je cadeau. Deze vind je terug in de folder ‘Views/Shared’. Maak de testen groen. Refactor indien nodig.

Zorg ervoor dat enkel aangemelde gebruikers feedback kunnen geven.

Gebruik het Simple Injector framework om er voor te zorgen dat er bij elke nieuwe http request een nieuwe instantie van ‘FeedbackDbRepository’ wordt aangemaakt. Deze instantie wordt geïnjecteerd in de ‘HomeController’ wanneer het nodig is. Vervolledig hiervoor de ‘RegisterMvcDependencies’ methode van de ‘DependencyConfig’ class in de ‘App\_start’ folder.

Maak een view aan voor de Feedback actions. TIP: schrijf de code van de view niet zelf, maar maak gebruik van de ‘Edit’ template.

Controleer of je via de applicatie effectief een feedbackformulier kan invullen en posten naar de applicatie.

**VEEL SUCCES!!**