https://youtu.be/wfHplJmvXWM

Aplikacja jest dostępna w środowisku AZURE pod linkiem:
http://issuetracker20170807105203.azurewebsites.net/
(w ramach 30-dniowego trial (począwszy od 06.08.2017) w profilu podstawowym nie ma możliwości bezpłatnego dodania SSL więc aplikacja jest wystawiana po http ☹☹☹)

Konto (niższe uprawnienia) umożliwiające zalogowanie do aplikacji:
Login: kiepskiz
Hasło: alamakota

Końcowy opis projektu IssueTracker

[AUTOR]:
Adam Arciechowski

[NAZWA PROJEKTU]:
IssueTracker

[CEL PROJEKTU]:
Stworzenie aplikacji WEB za pomocą której HelpDesk w małej firmie może rejestrować i obsługiwać zgłoszenia (Issues) w podstawowym zakresie.

[GŁÓWNA UŻYTA TECHNOLOGIA]:
Asp.NET MVC

[DODATKOWE TECHNOLOGIE I BIBLIOTEKI (poza automatycznie dodawanymi przez framework]:
1. EF - Entity Framework v6.1.3 (framework ORM <object relational mapping> w projekcie użyty w trybie CODE FIRST)
2. BCRYPT (do generowania salt, hashowania haseł i weryfikacji/odhashowywania haseł użytkowników)
3. PagedList.Mvc (biblioteka zastosowana do stronicowania listy zgłoszeń)
4. JQUERY FILE UPLOAD (zestaw skryptów które w połączeniu z CSS3 umożliwiły stworzenie upload załączników poprzez Drag&Drop)

[BAZA DANYCH]

Bazę danych (MSSQL) zapewnia wirtualny host (issuetracker.database.windows.net) w środowisku AZURE. Do bazy danych aplikacja podłącza się za pomocą connection stringa zdefiniowanego w głównym pliku konfiguracyjnym WebConfig:

*<add name="IssueTrackerDatabase" connectionString="Data Source=issuetracker.database.windows.net;Initial Catalog=IssueTrackerDB;Persist Security Info=True;User ID=altano;Password=******" providerName="System.Data.SqlClient" />*

Natomiast w środowisku developerskim (oraz w finalnej wersji projektu wrzuconej na GITHUBA) baza danych serwowana jest poprzez engine lokalny (LocalDb)\MSSQLLocalDB.
Użycie standardowych mechanizmów Entity Frameworka (m.in. MIGRATIONS) umożliwiło prostą migrację bazy (w późniejszych fazach produkcyjnego wdrożenia projektu) do pełnej instancji MSSQL SERVER (deploy do środowiska AZURE).

Graficzna prezentacja modelu bazy danych i powiązań pomiędzy tabelami wygenerowana automatycznie z użyciem narzędzia Microsoft SQL Server Management została wrzucona do repozytorium https://github.com/altano01/akademia_C-2017 GitHub w pliku: database_graph.pdf.

Ogólna uwaga co do stylu programowania w tym projekcie: jako że jest to mój pierwszy „większy" projekt w C# w różnych miejscach różne funkcjonalności były realizowane różnymi metodami po to by przećwiczyć możliwe różnorodne sposoby realizacji. Przykładem może być zastosowanie złożonego modelu ViewModels/ IssueAddViewModel.cs (w przypadku formatek związanych z edycją i tworzeniem nowych issue) a w innym miejscu zastosowanie modelu bazodanowego bezpośrednio aplikowanego do widoku (tam gdzie komplikacja formatki była relatywnie niższa). Zrobiłem tak chociaż wiem że zgodnie z zaleceniami każdy widok powinien mieć swój własny dedykowany ViewModel…

[TECHNICZNY OPIS FUNKCJONOWANIA APLIKACJI]

Generalnie aplikacja pracuje w zgodności z koncepcją MVC (Model <->View <->Controller) czyli requesty z przeglądarki zgodnie ze zdefiniowanym routingiem trafiają do kontrolera który odbiera request (POST/GET) i odpowiada za przyjęcie parametrów, uruchomienie zaprogramowanej logiki biznesowej i przygotowanie modelu który wysłany zostanie do widoku zawierającego szablon strony odsyłanej w odpowiedzi do przeglądarki.
Dane prezentowane są w blokach stanowiących zaprogramowane modele Biznesowe (katalog Models) lub techniczne (katalog ViewModels).

1. W warstwie prezentacji używane są standardowe widoki (RAZOR VIEWS) z dołączonymi CSS'ami oraz zestawem skryptów JavaScript.
   Co ważne, skrypty i css'y są bundlowane (czyli łączone w paczki-zestawy konfigurowane w BundleConfig.cs) dzięki czemu przeglądarka ściąga je za pomocą pojedynczego requestu GET a nie pojedyncze pliki.

Powtarzalne części layout aplikacji renderowane są w widoku _Layout wspólnego dla wszystkich stron serwowanych przez aplikację (wyjątkiem jest odrębny _LoginLayout.cshtml stworzony specjalnie dla strony logowania).
Widoki są zbudowane w oparciu o klasy modeli (Model Entities) opisujące globalny model biznesowy.
Cały Layout (za wyjątkiem strony logowania) warstwy prezentacji (HTML/CSS/JAVASCRIPT) aplikacji zbudowany został w oparciu o domyślny TEMPLATE który Visual Studio oferuje przy każdym nowym projekcie MVC.
Chodziło o to aby w projekcie skupić się na programowaniu logiki i możliwie wszechstronnym użyciu funkcjonalności i mechanizmów oferowanych przez MVC oraz sam C# a nie tworzenie wymyślnego UIX z użyciem najnowocześniejszych trendów CSS3/HTML5/JS .

Wyjątkiem jest strona logowania (/Authentication/Login) dla której widok i cały layout (bardzo prosty zresztą ☺) został wykonany w oderwaniu od domyślnego TEMPLATE.
Pola na wszystkich formatkach są walidowane na podstawie definicji adnotacji (DataAnnotations) ustawionych w klasach modeli. Walidacja pól na formatkach wszędzie jest dwuetapowa:

a. Walidacja po stronie klienta (przeglądarki) – odbywa się za pomocą skryptów javascript (jQuery Validate unobtrusive)
b. Niezależnie od tego wartości z pól są też walidowane już po stronie aplikacji serwera, po wysłaniu formularza metodą POST. Kod odpowiedzialny za tą walidację jest umieszczony w kontrolerach. Walidacja po stronie serwera zabezpiecza przed próbą wysłania niezwalidowanych danych w momencie gdy przeglądarka ma wyłączoną/nie obsługuje JavaScript.
   Dodatkowym feature walidacji po stronie serwera jest zabezpieczenie przed atakiem typu OVER-POSTING (próba wysłania do serwera aplikacji pól i wartości nie istniejących na oryginalnym formularzu) kod w kontrolerach przyjmuje wartości jedynie z podanej listy parametrów (Model Binding), przykładowo:

```
//*****CREATE [POST]*****
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include =
"Id,Status,Category,Priority,UserId,GroupId,Title,Description")] Issue issue)
```

   inne parametry nie wymienione w dyrektywie Bind są ignorowane.

2. Requesty GET/POST wysyłane z przeglądarki użytkownika przejmowane są przez kontrolery (Controler) (w zależności od kontekstu URL którego żąda przeglądarka) w których zaimplementowana jest cała logika biznesowa, odbierająca wysłane dane, weryfikująca te dane, przetwarzająca je i zapisująca w bazie z użyciem API dostarczanego przez EntityFramework wreszcie renderująca zwrotny widok wysyłany do przeglądarki.

3. Za interakcję z bazą danych w pełni odpowiada EntityFramework który samodzielnie bez ingerencji developera przejmuje na siebie m.in. wymienione niżej zadania:
    a. Automatyczne wygenerowanie tabel na bazie danych na podstawie modeli biznesowych (Entity Models)
    b. Utworzenie odpowiednich definicji kolumn w tabelach na podstawie adnotacji zdefiniowanych w modelach (DataAnnotations)
    c. Utworzenie odpowiednich relacji pomiędzy tabelami, dodanie indeksów, constraint, ustawienie foreign keys itp. Itd. to również odbywa się automatycznie

Jak już wspomniałem wyżej w aplikacji EntityFramework jest używany „w trybie CODE FIRST" co oznacza że najpierw wymyślony został model biznesowy aplikacji, potem został on zaprogramowany w klasach (Models), następnie klasy te zostały dodane do kontekstu entity managera, który przezroczyście wykonał w/w zadania.
Podejście CODE FIRST i sam ENTITY FRAMEWORK skutecznie separuje warstwę logiki biznesowej od warstwy składowania danych i pozwala na OGROMNĄ oszczędność czasu który jest używany na projektowanie funkcjonalności a nie na nisko poziomowe programowanie API dostępu do bazy danych.
Do komunikacji z bazą danych Użycie standardowych mechanizmów EntityFrameworka zapobiega również atakom SQL-INJECT bowiem pośredniczy w KAŻDEJ interakcji pomiędzy aplikacją a bazą danych (nigdzie w aplikacji nie ma od tego wyjątku).

[PRZEGLĄD ZAIMPLEMENTOWANYCH FUNKCJONALNOŚCI]:

1. **Moduł zarządzania użytkownikami / grupy / role**

W aplikacji praktycznie od zera zaimplementowany został moduł obsługi kont użytkowników, dostępny z pod linku "Users" (w górnym menu) jednak TYLKO I WYŁĄCZNIE gdy do aplikacji zaloguje się użytkownik przypisany do grupy-roli "Admins".
Zarówno konta użytkowników jak i powiązane grupy trzymane są w osobnych tabelach na bazie danych. Hasła kont użytkowników przetrzymywane są w bazie w formie bezpiecznej (hashowanej).
Użyty został algorytm hashujący dostarczany z biblioteką BCRYPT.
W funkcji generującej salt do hasła ustawiono relatywnie długi salt:
string mySalt = BCrypt.Net.BCrypt.GenerateSalt(13);
co powoduje że weryfikacja hasła następuje po około 1s interwale czasu -> zabezpiecza to przed atakami typu Brute-Force, chociaż po trzech nieudanych próbach zalogowania (poprzez podanie nieprawidłowego hasła) konto użytkownika automatycznie blokuje się i odblokować je może wyłącznie użytkownik przypisany do grupy Admins.
W ramach tego modułu użytkownicy uprzywilejowani (przypisani do grupy Admins) mają możliwość:
    a. tworzenia kont użytkowników
    b. resetowania-ustawiania haseł użytkowników

c. blokowania/odblokowywania kont (aplikacja nie przewiduje możliwości usuwania użytkowników ponieważ naruszałoby to constraint'y utworzone automatycznie przez EntityFramework) zamiast tego konto można zablokować uniemożliwiając tym samym zalogowanie i ewentualnie odblokować gdy użytkownik ma mieć ponownie dostęp) jest to bardziej elastyczne rozwiązanie.

Funkcjonalności te obsługuje kontroler UserController (oraz PasswordController) w powiązaniu z widokami z katalogu Views/User.

2. Mechanizm logowania i obsługa sesji

Mechanizm logowania w oparciu o w/w konta użytkowników zaimplementowany został od zera i udostępniany jest przez kontroler Authentication oraz akcję Login.

Sesja użytkownika wykorzystuje standarową-wbudowaną autentykację „Forms" zgodnie z konfiguracją w pliku Web.config:

```
<authentication mode="Forms">
  <forms loginUrl="~/Authentication/Login" name=".ISSUETRACKER" timeout="60" />
</authentication>
```

Czas wygasania sesji ustawiony jest na 60 minut, sesja jest podtrzymywana/weryfikowana w oparciu o COOKIE (o nazwie name=".ISSUETRACKER").

Wszystkie akcje w kontrolerach (za wyjątkiem strony logowania) poprzez zdefiniowanie globalnego filtra w App_Start/FilterConfig.cs:

```
filters.Add(new AuthorizeAttribute());
```

wymagają weryfikacji tokena (z COOKIE) i tym samym uniemożliwiają dostęp użytkownikom niezautoryzowanym. Jednocześnie rozwiązanie takie jest standardowym zabezpieczeniem przed atakami typu Cross-Site Request Forgery (CSRF).

Przy wylogowaniu (akcja LogOff z kontrola Authentication) sesja jest czyszczona:

```
FormsAuthentication.SignOut();
Session.Abandon();
```

jednocześne aplikacja wysyłając do przeglądarki użytkownika zwrotne request zapobiega jego cache'owaniu poprzez ustawienie nagłówków (w pliku konfiguracyjnym Global.asax):

```
Response.Cache.SetCacheability(HttpCacheability.NoCache);
Response.Cache.SetExpires(DateTime.UtcNow.AddHours(-1));
Response.Cache.SetNoStore();
```

Taka konstrukcja powoduje, że po wylogowaniu próba użycia przycisku WSTECZ w przeglądarce spowoduje automatyczne przekierowanie użytkownika do strony logowania.

Na stronie logowania istnieje możliwość wybrania checkbox „Remember my session" co powoduje że algorytm ustawia persistent COOKIE i dzięki temu po zamknięciu przeglądarki i ponownym otwarciu użytkownik nie musi się logować ponownie o ile sesja nie przekroczyła timeout przy czym w przypadku aktywności użytkownika framework automatycznie przesuwa czas wygasania sesji za sprawą ustawionego parametru:

```
slidingExpiration="true"
```

3. Główny moduł zgłoszeń (Issues)

Pod linkami „My Issues” oraz „All Issues” aplikacja serwuje główną funkcjonalność systemu czyli obsługę zgłoszeń (Issues).
W zakresie tej głównej funkcjonalności w aplikacji jest możliwość:

a. Przeglądania zgłoszeń ze stronicowaniem wyświetlanej listy (liczba elementów na jednej stronie listy jest hardcodowana w aplikacji i wynosi: 7).
b. Możliwość wyszukania zgłoszenia po numerze (z użyciem górnego box’a szybkiego wyszukiwania) albo po słowach w tekście (wyszukiwanie jest case-insensitive).
c. Możliwość przefiltrowania listy zgłoszeń za pomocą zdefiniowanego wcześniej filtra (definicje filtrów zapisywane są w bazie danych w kontekście każdego pojedynczego użytkownika).
d. Możliwość szybkiego przypisania danego zgłoszenia do zalogowanego aktualnie użytkownika(loginu).
e. Możliwość szybkiego przejścia do Edycji, Szczegółów, Dodania komentarza z użyciem linków dostępnych przy rekordzie ze zgłoszeniem
f. Możliwość dodania nowego zgłoszenia (poprzez przycisk w górnym menu lub link dostępny w widokach My Issues/All Issues).
g. Do zgłoszenia można dodać załączniki. Formant umożliwiający dodanie został zaimplementowany za pomocą AJAX i obsługuje Drag&Drop. Załączniki są uploadowane do dedykowanego katalogu (Uploads) (w clustrowej implementacji na produkcji byłby to zapewne share sieciowy współdzielony pomiędzy serwerami).
   Max wielkość pojedynczego załącznika (20MB) jest determinowana wielkością requesta zdefiniowaną w WebConfigu:

```
<httpRuntime targetFramework="4.5.2" maxRequestLength="20480" />
```

4. Moduł dodawania filtrów zgłoszeń

Aplikacja umożliwia definiowanie i edytowanie swoich własnych filtrów zgłoszeń, które są zapisywane w bazie w kontekście każdego użytkownika z osobna.
Na formatce definicji filtra oprócz standardowych editBox’ow użyte zostały listy wielokrotnego wyboru, a w kontrolerze (IssueFilter) do wyciągnięcia listy zgłoszeń spełniających kryteria użyto poniższej konstrukcji LINQ dzięki której warunek z danym kryterium jest procesowany o ile wartość warunku nie jest nulle’m albo kolekcja wybranych wartości jest > 0 :

```
var issuesS = db.Issues
                .Where(p => (FilDef.FTitle == null || p.Title.Contains(FilDef.FTitle)))
                .Where(p => (FilDef.FDescription == null || p.Description.Contains(FilDef.FDescription)))
                .Where(p => (FilDef.UserId == null || p.UserId == FilDef.UserId))
                .Where(p => (sts.Count() == 0 || sts.Any(m => m.Contains(p.Status.ToString()))))
                .Where(p => (cts.Count() == 0 || cts.Any(m => m.Contains(p.Category.ToString()))))
                .Where(p => (pts.Count() == 0 || pts.Any(m => m.Contains(p.Priority.ToString()))))
                .Where(p => (gts.Count() == 0 || gts.Any(m => m.Contains(p.Group.GroupName)))).OrderBy(o=>o.Id);
```

Utworzony filtr można zaaplikować do listy zgłoszeń dostępnej pod linkiem “My Issues” lub „All Issues”. Jeśli użytkownik nie posiada żadnych filtrów, combobox z filtrami jest automatycznie ukrywany (steruje tym warunek if w widoku).

5. Moduł raportowy

Pod górnym linkiem "Reports" w aplikacji zaimplementowana została skromna (ale jednak :) funkcjonalność raportowa zapewniający podstawowe raporty:

1. zgłoszeń per użytkownik
2. zgłoszeń per grupa
3. zgłoszeń per priorytet zgłoszenia
4. zgłoszeń per kategoria zgłoszenia

Funkcjonalność jest dostarczana przez kontroler UserController w powiązaniu z widokami z katalogu Views/Report. Widok jest zorganizowany nieco inaczej niż inne widoki w aplikacji ponieważ opiera się on na generowania (renderowania) widoków częściowych (partial views) w zależności od tego który raport użytkownik wybiera z comboboxa.
W kontrolerze użyto zapytań grupujących LINQ.

6. Plan projektu vs realizacja

Wymaganie	Realizacja
rejestracja zgłoszeń (issues) + prosty issue life cycle (workflow)+ definiowanie podstawowych parametrow issue (m.in severity)	Zrealizowane.
obsługę zgłoszeń przez użytkowników aplikacji pogrupowanych w zespoły	Zrealizowane. Lista grup zahardcodowana ale Admin może przypinać/odpinać userów od grup a także przepinać ich pomiędzy grupami.
w ramach jednego issue aplikacja umożliwi dodawanie wpisów (comments)	Zrealizowane. Dodatkowo do zgłoszenia można dodawać załączniki.
aplikacja umożliwi ryglowanie issue (zablokowanie issue na wyłączność przez jednego użytkownika)	To nie zostało zaimplementowane. Uznałem że wystarczy tutaj standardowy mechanizm WIN-FIRST.
parametry techniczne i merytoryczne będą przechowywane w bazie danych	Zrealizowane. Jest to m.in. definicja grup.
loginy użytkowników przechowywane w bazie danych	Zrealizowane. Nie tylko loginy ale hasła i dane kont użytkowników przetrzymywane są w bazie danych.
hasła użytkowników będą przechowywane w bazie danych w formie zahashowanej (wybrany algorytm hashowania+salt)	Zrealizowane. Hasła są przechowywane w bezpiecznej postaci (hash BCRYPT’em).

7. Wymagania do technologii vs realizacja

Wymaganie	Realizacja
Sugerowane techonologie : Główne: WPF , ASP.Net , Console	**Zrealizowane.** Projekt w technologii ASP .Net MVC
Wyrażenia LINQ/ Wyrażenia lamda	**Zastosowane.** Wyrażenia używane są w wielu różnych postaciach i odmianach do komunikacji z Entity Framework.
Dziedziczenie	**Zastosowane.** Używane praktycznie we wszystkich kontrolerach: `public class IssueController : Controller`
Polimorfizm statyczny.	**Zastosowane.** Przede wszystkim w przeciążeniach wywołań metod w kontrolerach.

Enumy	**Zastosowane.**
	Kolekcje enum użyte zostały np.do przygotowania listy możliwych statusów issue (Models/Issue.cs):
	`public enum Status`
	`{ New, InProgress, Suspended,`
	`ClosedFixed, ClosedInvalid }`

Podział kodu w taki sposób, aby metody w miarę możliwości nie przekraczały jednego ekranu	**Spełnione.**
Opis kodu	**Spełnione.** Kod jest solidnie i dość dokładnie (w niektórych miejscach nawet do przesady ☺) opisany komentarzami. Wynika to nie tylko z wymogów projektu ale przede wszystkim z tego że zaprogramowanie niektórych algorytmów zajęło mi sporo czasu i „rozkminiania" dlaczego tak a nie inaczej. W przyszłości komentarze pozwolą przypomnieć sobie tok rozumowania. Na GITHUB wrzucona dokumentacja (Help) wygenerowana z użyciem GhostDoc (IssueTracker_GhostDoc.chm)