KekManager – implementacja

Daniel Bider  
Szymon Barańczyk

Spis treści

[Dostęp 3](#_Toc518228759)

[Realizacja wymagań niefunkcjonalnych 3](#_Toc518228760)

[Wsparcie systemów Unixowych 3](#_Toc518228761)

[B4: Logowanie on-site 4](#_Toc518228762)

[Microsoft ASP.NET Identity 4](#_Toc518228763)

[JWT Tokens 4](#_Toc518228764)

[U1. Responsywny interfejs utrzymany w kolorystyce PWR 5](#_Toc518228765)

[WS1. Skrypty tworzące bazę w przypadku relacyjnego modelu danych 5](#_Toc518228766)

[Dodatkowe mechanizmy 7](#_Toc518228767)

[ORM – Entity Framework Core, separacja kontekstów 7](#_Toc518228768)

[Programowa rejestracja zależności 7](#_Toc518228769)

[Automapper 8](#_Toc518228770)

[Wydzielenie miejsca wejściowego aplikacji 8](#_Toc518228771)

[Continuous Integration i Code Quality 8](#_Toc518228772)

[Travis 8](#_Toc518228773)

[SonarCloud 8](#_Toc518228774)

[Azure i BitBallon 8](#_Toc518228775)

[Postman 9](#_Toc518228776)

[Visual Studio Code Maps 9](#_Toc518228777)

# Dostęp

Repozytorium - <https://github.com/Eragra3/architecture>

Aplikacja - <http://brave-keller-2a24f4.bitballoon.com>

Serwer - <https://kekmanager-regular.azurewebsites.net/>

Przykładowy użytkownik

* Email - admin@kekmanager.com
* Hasło - 1qaz@WSX

# Realizacja wymagań niefunkcjonalnych

## Wsparcie systemów Unixowych

Wszystkie projekty zostały napisane w oparciu o wieloplatformowe biblioteki. Aplikacje Webowe wykorzystują framework *ASP.NET Core 2.1* (najnowszy) wykorzystujący środowisko uruchomieniowe *.NET Core*. Aplikacje biblioteczne (logika, modele domenowe itd.) wykorzystują z kolei standard *.NET Standard 2.0* (również najnowszy). Jest to zbiór bibliotek (standard) określający zakres bibliotek standardowych. **Nie zawiera żadnego środowiska uruchomieniowego**, taki standard powstał w celu tworzenia aplikacji opartych o abstrakcyjny standard, który następnie jest implementowany przez różne środowiska uruchomieniowe (patrz Tabela 1).

Tabela 1 - obsługa standardu .NET Standard

|  |
| --- |
|  |
| **.NET Standard** | [**1.0**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.0.md) | [**1.1**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.1.md) | [**1.2**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.2.md) | [**1.3**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.3.md) | [**1.4**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.4.md) | [**1.5**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.5.md) | [**1.6**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard1.6.md) | [**2.0**](https://github.com/dotnet/standard/blob/master/docs/versions/netstandard2.0.md) |
| .NET Core | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 2.0 |
| .NET Framework 1 | 4.5 | 4.5 | 4.5.1 | 4.6 | 4.6.1 | 4.6.1 | 4.6.1 | 4.6.1 |
| Mono | 4.6 | 4.6 | 4.6 | 4.6 | 4.6 | 4.6 | 4.6 | 5.4 |
| Xamarin.iOS | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.14 |
| Xamarin.Mac | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.8 |
| Xamarin.Android | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 8.0 |
| Universal Windows Platform | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0.16299 | 10.0.16299 | 10.0.16299 |
| Windows | 8.0 | 8.0 | 8.1 |  |
| Windows Phone | 8.1 | 8.1 | 8.1 |  |
| Windows Phone Silverlight | 8.0 |  |  |

Dzięki temu aplikacja może być nie tylko wdrażana na Unixowych systemach, ale również kontenerach, Macintoshach i innych. Dodatkowym plusem jest możliwość przeprowadzenia tzw. *Self-Contained Deployment*. Taki rodzaj wdrożenia nie wymaga preinstalowania **żadnych** aplikacji w środowisku, minusem jest bardzo duży rozmiar paczki wdrażania (aplikcja „Hello World” ma więcej niż 30 MiB).

W celu zweryfikowania tego wymagania najlepiej przejrzeć pliki *.csproj*.

## B4: Logowanie on-site

Uwierzytelnianie oraz logowanie zostało zaimplementowane wykorzystując kilka bibliotek.

### Microsoft ASP.NET Identity

Oficjalna biblioteka frameworku ASP.NET do tworzenia tzw. tożsamości. Zawiera logikę hashowania haseł, logowania, zarządzania użytkownikami (tworzenie, usuwanie, blokowanie) oraz rozbudowany system autoryzacji. Chociaż autoryzacja wykracza poza nasze wymagania – zaproponowana implementacja umożliwa wykorzystanie modelu autoryzacji na podstawie ról, uprawnień („claims”), zasobów i innych. Biblioteka zapewnia również bezpieczne i sprawdzone obsługiwanie haseł i kont.

Bezpośrednie „wpięcie” biblioteki można zobaczyć w pliku KekManager.AppStartup.Startup.cs:

services.AddIdentity<SecurityUser, IdentityRole>()

.AddEntityFrameworkStores<FullDatabaseContext>()

.AddDefaultTokenProviders();

### JWT Tokens

Standard uwierzytelniania użytkowników na podstawie tokenów. Bardzo popularny model autentykacji z chęcią stosowany w bezstanowym protokole http oraz aplikacjach uwierzytelniających się przez aplikacje trzecie.

Ponownie miejsce rejestracji biblioteki widać w pliku KekManager.AppStartup.Startup.cs:

services

.AddAuthentication(options =>

{

(…)

})

.AddJwtBearer(options =>

(…)

Samo sprawdzenie uwierzytelnienia odbywa się poprzez dekorowanie kontrolerów (lub akcji) atrybutem „Authorize”, tym samym bezpośrednio integruje się z frameworkiem ASP.NET.

## U1. Responsywny interfejs utrzymany w kolorystyce PWR

Responsywny interfejs powstał dzięki wykorzystaniu (nader) często używanej biblioteki *Bootstrap* w najnowszej, czwartej odsłonie oraz frameworka *Angular 5*. *Bootstrap* jest dobrze zintegrowany z *Angularem* (natywne wrażenie podczas używania dyrektyw, pełna obsługa możliwości), dzięki temu interfejs może wykorzystywać bardzo rozbudowane możliwości przeglądarek (animacje, filtry, gesty). Biblioteka komponentów jest bardzo bogata, a każdy komponent jest przygotowany w responsywny sposób, często z obsługą gestów i animacji.

Rejestracja biblioteki odbywa się w pliku „package.json” znajdującym się w folderze „Web”.

"dependencies": {

(…)

"@angular/core": "^5.2.0",

"@angular/forms": "^5.2.0",

"@angular/http": "^5.2.0",

"@angular/platform-browser": "^5.2.0",

"@angular/platform-browser-dynamic": "^5.2.0",

"@angular/router": "^5.2.0",

"bootstrap": "^4.1.1",

"core-js": "^2.4.1",

"jquery": "^1.9.1",

"ngx-bootstrap": "^2.0.0"

(…)

Aplikacja kliencka została również w całości oddzielona od serwera, dzięki temu może również być konteneryzowana i rozwijana niezależnie.

## WS1. Skrypty tworzące bazę w przypadku relacyjnego modelu danych

Obsługa bazy danych została zaprojektowana w dosyć nietypowy sposób. Szcz **TODO**

Wykorzystanie tej biblioteki umożliwia wygenerowanie skryptów bazodanowych na podstawie migracji. Do tego celu potrzebujemy narzędzia konsolowe *Entity Framework Core CLI*. W folderze bazodanowym możemy wywołać polecenie[[1]](#footnote-1):

dotnet ef migrations script -s ../KekManager.Startup > Scripts\create.sql

Polecenie domyślnie wypisuje wszystko na standardowe wyjście, więc możemy wykorzystać możliwości przekierowywania strumieni w systemie, aby zapisać skrypty do pliku. Konsola umożliwia również wygenerowanie skryptów przyrostowych (od migracji x do migracji y) oraz idempotentnych. Pełna dokumentacja znajduje się na stronie Microsoftu <https://docs.microsoft.com/pl-pl/ef/core/miscellaneous/cli/dotnet#dotnet-ef-migrations-script>.

Pełen skrypt znajduje się w pliku „Database/Scripts/create.sql”.

# Dodatkowe mechanizmy

## ORM – Entity Framework Core, separacja kontekstów

Aplikacja wykorzystuje najnowszą odsłonę ORM’a *Entity Framework*. Chociaż biblioteka nadal jest w trakcie tworzenia (brakuje wielu możliwości, jak relacje wiele do wielu, procedury składowane) jest dosyć wygodna w użyciu, dodatkowo oferuje bardzo dużo dostawców (baz danych). Niestety baza danych jest często monolitycznym miejscem w aplikacji, dlatego zaproponowaliśmy kilka sposobów modularyzacji i separacji projektów.

Części aplikacji wykorzystujące dostęp do danych (DAL, repozytoria) zawierają u nas tylko interfejsy kontekstów, wymagające istnienia pewnych zbiorów danych (np. DbSet<Subject>). Nie odpowiadają natomiast za tworzenie bazy danych i nie mają decydującego głosu nad jej kształtem. Aplikacyjne części bazy („KekManager.Dal.Data” i „KekManager.Security.Dal.Data”) zawierają wspomniane interfejsy, modele (wykorzystujące adnotacje walidacyjne „DataAnnotations”) oraz konfiguracje wykorzystującą *FluentAPI*. Pozwala to na stworzenie wymagań na bazę, bez wskazywania konkretnej bazy. W szczególności, jeżeli jakiś moduł nie wskazywałby potrzeby relacyjnej bazy (brak powiązań, duże klasy, być może dynamiczne) w pewnym momencie architekci mogą zadecydować o przeniesieniu danych do nierelacyjnej bazy – wystarczy wtedy wstrzyknąć inny kontekst.

Główny projekt bazodanowy to „KekManager.Database”. Znajduje się tam fizyczny kontekst FullDatabaseContext.cs oraz implementacja migracji (i migracje oczywiście). Warto zauważyć, że nie wymagamy tworzenia osobnych baz dla każdego modułu. Większość projektów startuje z jedną relacyjną bazą, potem decyduje się na denormalizacje itp. Jeżeli zajdzie taka potrzeba, możemy usunąć tabele jednego modułu przy pomocy migracji i utworzyć osobny kontekst.

Częściowy model można zobaczyć w pliku „database\_model.xps”.

## Programowa rejestracja zależności

Wstrzykiwanie zależności poprzez konwencję lub plik konfiguracyjny ukrywa zależności kodu między sobą. Uniemożliwia również wykorzystanie kompilatora to przeprowadzenia dodatkowych walidacji. W tym celu rejestracja odbywa się programowo.

Jednym z podejść jakie chcieliśmy zaproponować jest tworzenie modułów zależności, osobnych dla różnych funkcjonalnych części systemu. Jak się okazało – taki model już istnieje <https://autofaccn.readthedocs.io/en/latest/configuration/modules.html>.

Utworzyliśmy dwa moduły (główny i security), aczkolwiek warto spróbować umieścić moduły Autofaca bezpośrednio w modułach funkcjonalnych. Takie podejście uprościłoby podpinanie modułów (funkcjonalnych) do innych aplikacji (moduł sam się rejestruje, lub podpowiada jakich typów potrzebuje).

Rejestracja odbywa się w projekcie „KekManager.DependancyRegistration” oraz pliku „KekManager.AppStartup.Startup.cs” (zastępuje domyślny model IoC):

// Add Autofac

var harvester = new DependancyHarvester();

var containerBuilder = harvester.Harvest(Configuration);

containerBuilder.Populate(services);

var container = containerBuilder.Build();

return new AutofacServiceProvider(container);

## Automapper

Prosty framework umożliwiający automatyczne mapowanie encji. Projekt w klasyczny sposób wykorzystuje jego funkcjonalności, więc wskażemy tylko miejsca gdzie są wykorzystywane:

Rejestracja - „KekManager.AppStartup.Startup.cs”:

services.AddAutoMapper(Assembly.Load("KekManager.Logic"));

Profile – “KekManager.Data.DataMapperProfile.cs”, “KekManager.Logic.LogicMapperProfile.cs”.

Zachęcam do wykorzystywania profili oraz uczulam na sposób ich rejestracji, my musieliśmy wymusić wczytanie odpowiednich assembly’i (fragment „Assembly.Load”).

## Wydzielenie miejsca wejściowego aplikacji

Projekty zawierające kontrolery (API) nie są u nas miejscem wejściowym, ma to na celu oddzielenie konfiguracji samego serwera (routing, IoC, „connection strings”, logi) od logiki kontrolerów. Upraszcza to wdrażanie, zarządzanie aplikacją (jeden punkt konfiguracyjny) oraz sprawia, że projekty zawierające API są lżejsze. Łatwiej również integrować wiele API na jednym serwerze, ponieważ możemy nadpisywać „routy”, wyłączać polityki wyświetlania strony z błędem, serwowania statycznych plików itd.

Punkt wejściowy to aplikacja „KekManager.AppStartup”.

## Continuous Integration i Code Quality

### Travis

Aktualnie wykorzystywany tylko do budowania aplikacji i klienta oraz uruchamiania skanera *SonarQube*. W przyszłości może być wzbogacony o testy itp. Link <https://travis-ci.com/Eragra3/architecture>.

### SonarCloud

Darmowe, chmurowe rozwiązanie oferujące możliwości znanego (i lubianego) *SonarQube’a*. Link <https://sonarcloud.io/dashboard?id=KekManager>. Wykorzystaliśmy również możliwość połączenia *SonarQube’a* z lokalnym skanerem *SonarScanner*, w celu lokalnych analiz.

### Azure i BitBallon

Usługi hostingowe, pierwsza dla serwera i bazy danych, druga dla aplikacji klienckiej.

Klient - <http://brave-keller-2a24f4.bitballoon.com>

Sewer - <https://kekmanager-regular.azurewebsites.net/>

## Postman

Do projektu zostały przygotowane testowe żądania http to testowania API. Można je zaimportować do programu *Postman* z pliku „KekManager.postman\_collection.json”. Żądania mogą wymagać edycji adresu i wklejenia nowego tokena (można go uzyskać poprzez zapytanie „Login”).

## Visual Studio Code Maps

Najwyższa wersja VS (Enterprise) oferuje możliwość tworzenia tzw. map kodu. Wykorzystywaliśmy te narzędzia podczas modelowania/walidowania architektury. Wygenerowane (z kodu) diagramy zostały załączone w formacie xps.

* dal\_model.xps – implementacja jednego z funkcjonalnych projektów bazodanowych
* full\_model.xps – ogólne spojrzenie na całą solucję
* database\_model.xps – implementacja separacji kontekstów bazodanowych.

Przeglądanie przez Visual Studio (przeglądać można również w niższych wersjach) jest interaktywne. Jest tam również pełna legenda. Zachęcam do zapoznania się z tym narzędziem, nie widzę żadnej możliwości generowania kodu, aczkolwiek można wygodnie przeglądać zależności między nie tylko projektami, ale również klasami, metodami a nawet atrybutami!

1. Parametr „-s” jest wymagany ze względu na specyfikę projektu i działania narzędzia. Wskazuje on projekt, który może być wykorzystany jako wejściowy (jest wykonywalny), szczegółowe informacje można znaleźć [tutaj](https://docs.microsoft.com/pl-pl/ef/core/miscellaneous/cli/dotnet#using-the-tools). [↑](#footnote-ref-1)