KekManager

Architecture Notebook

There is guidance within this template that appears in a style named InfoBlue. This style has a hidden font attribute that allows you to toggle whether it is visible or hidden in this template. Use the Microsoft® Word® menu **Tools > Options > View > Hidden Text** check box to toggle this setting. There salso an option for printing: **Tools > Options > Print**.

Spis treści

[1. Cel 2](#_Toc514623808)

[2. Cele architekturalne i ograniczenia 2](#_Toc514623809)

[3. Decyzje i uzasadnienia 2](#_Toc514623810)

[4. Mechanizmy architekturalne 3](#_Toc514623811)

[ORM 3](#_Toc514623812)

[Code-First 3](#_Toc514623813)

[Odwracanie zależności 3](#_Toc514623814)

[Autentykacja wykorzystująca standard *JWT* 4](#_Toc514623815)

[Gruby klient 4](#_Toc514623816)

[Separacja kontekstów Entity Framework 4](#_Toc514623817)

[Programowa rejestracja wstrzykiwania zależności 4](#_Toc514623818)

[5. Główne abstrakcje 5](#_Toc514623819)

[6. Widoki architektoniczne 6](#_Toc514623820)

[7. Realizacje przypadków użycia (dla wybranych przypadków użycia) 8](#_Toc514623821)

[C1.4 Ustalanie opiekuna przedmiotu 8](#_Toc514623822)

# Cel

Celem dokumentu jest przybliżenie wymagań i wyzwań jakie mogą się pojawić podczas realizacji projektu oraz omówienie decyzji które zostały podjęte w celu rozwiązania powstałych problemów.

# Cele architekturalne i ograniczenia

Projekt aplikacji wspomagającej zarządzanie kursami jest problemem złożonym, dlatego duży nacisk został położony na testowalność oraz rozszerzalność aplikacji.

Podczas etapu zbierania wymagań oraz analizy biznesowej systemu wyróżniono następujące wymagania, które wymagają zaprojektowania oraz implementacji:

Funkcjonalne:

* C1.2. CRUD programu kształcenia w tym planu i programu studiów oraz macierzy śladowania
* C1.3. Przydzielanie efektów kierunkowych do przedmiotów i budowanie macierzy śladowania
* C1.4. Ustalanie opiekunów przedmiotów
* C1.6. Weryfikacja realizacji efektów kierunkowych przez przedmioty (śladowanie w dół)

Niefunkcjonalne:

* Wsparcie systemów Unixowych
* B4: Logowanie on-site
* U1. Responsywny interfejs utrzymany w kolorystyce PWR
* WS1. Skrypty tworzące bazę w przypadku relacyjnego modelu danych

# Decyzje i uzasadnienia

* Wsparcie systemów Unixowych - W celu ułatwienia pielęgnacji aplikacji (dostęp do narzędzi wspomagających wytwarzanie oprogramowania, jak np. Docker) oraz kosztów utrzymania infrastruktury hostującej aplikacja (razem z bazą danych), aplikacja musi wspierać systemy *Unixowe*. Środowiska deweloperskie mogą wykorzystywać system Windows, natomiast narzędzia klasy *DevOps* służące budowaniu, testowaniu czy wdrażaniu aplikacji, muszą wykorzystywać systemy *Unixowe.*
* B4: Logowanie on-site – dla zachowania wysokiego standardu bezpieczeństwa, autentykacja będzie opierać się na tokenie. Pozwoli to również w prosty sposób rozpiąć stronę kliencką od serwera. Autoryzacja nie może wykorzystywać niestandardowej (własnej) implementacji, musi wykorzystywać istniejące, sprawdzone rozwiązania i ewentualnie rozszerzać je według instrukcji autorów.
* U1. Responsywny interfejs utrzymany w kolorystyce PWR – ponieważ interfejs aplikacji musi być przejrzysty i używalny na różnych urządzeniach, aplikacja kliencka będzie dynamicznie dopasowywać się do urządzenia klienta. Uniemożliwia to wykorzystanie frameworków silnie powiązanych z platformą, jak aplikacje Androidowe, iOS’owe czy desktopowe jak WinForms. W celu uproszczenia i przyspieszenia implementacji, aplikacja kliencka będzie aplikacją webową w stylu tzw. „grubego klienta”.
* WS1. Skrypty tworzące bazę w przypadku relacyjnego modelu danych – aplikacja będzie wykorzystywać relacyjną bazę danych, przemawia za tym integralność danych, wsparcie dla transakcji oraz wieloletnie doświadczenia innych projektów. Dla uproszczenia wdrażania w repozytorium kodu aplikacji muszą znajdować się skrypty tworzące i ewentualne skrypty populujące bazę.

|  |  |
| --- | --- |
| Cel | Jak osiągnąć (taktyka) |
| Wsparcie platform Unixowych | Wykorzystanie platformy ASP.NET Core 2.0.  Wykorzystanie bazy danych My-SQL.  Zakaz wykorzystywania pakietów dedykowanych wyłącznie na system Windows. |
| Logowanie on-site | Wykorzystanie standardu JWT.  Wykorzystanie mechanizmów tożsamości ASP.NET Core 2.0*.*  Zakaz implementacji dodatkowych mechanizmów, w szczególności niezgodnych z wytycznymi OWASP i innymi dobrymi praktykami bezpieczeństwa. |
| Responsywny interfejs | Wykorzystanie biblioteki graficznej Semantic UI i wdrożenie zmian kolorystyki.  Wykorzystanie frameworka Angular – wspólnie z przestrzeganiem zasad tworzenia stron zarówno na urządzenia mobilne jak i desktopy.  Zakaz bazowania podstawowych funkcjonalności interfejsu na rozwiązaniach konkretnej platformy (np. Android) – dopuszczalne jest natomiast rozszerzanie o takie rozwiązania, o ile nie wpływają negatywnie na działanie aplikacji na innych platformach. |
| Skrypty bazodanowe | Wykorzystanie frameworka Entity Framework Core – generowanie skryptów na podstawie klas stworzonych w kodzie.  Absolutny zakaz ręcznej modyfikacji wygenerowanych skryptów w repozytorium. |

# Mechanizmy architekturalne

## ORM

Zapewnia automatyczne mapowanie encji bazodanowych (tabel) na klasy/obiekty definiowane w logice biznesowej. Pozwala również zachować ograniczenia/relacje w jednym miejscu.

* Uproszczenie integracji bazy danych z logiką biznesową
* Dodatkowa walidacja zapytań na poziomie typów

## Code-First

Podejście pozwalające utworzyć bazę danych na podstawie klas zdefiniowanych w kodzie oraz ewentualnych metadanych w postaci dodatkowych adnotacji. Znacznie upraszcza proces tworzenia bazy danych.

* Unikanie duplikacji pracy w postaci tworzenia encji biznesowej i tabeli bazodanowej
* Generowanie skryptów bazodanowych w jednolitej postaci (formatowanie, nazewnictwo)
* Obsługa migracji – iteracyjnego budowania bazy danych oraz możliwości odtworzenia starszej wersji

## Odwracanie zależności

Pozwala odizolować elementy aplikacji od siebie i ich niezależnie rozwijanie. Części aplikacji są wymienne (bez dużego nakładu pracy), w szczególności na tzw. “mocki” w celach testowania.

* Możliwość testowania elementów niezależnie od siebie
* Możliwość szybkiej wymiany wadliwych, przestarzałych czy wolnych elementów

## Autentykacja wykorzystująca standard *JWT*

*JWT* (*JSON Web Token*) – otwarty standard opisujący sposób autentykacji poprzez wymianę tokena.

* Standard wykorzystywany z powodzeniem w instniejących rozwiązaniach
* Mechanizm wbudowany w framework *ASP.NET Core 2.0*

## Gruby klient

Część kliencka (interfejs) może zostać odizolowana od części serwerowej (logika biznesowa) dzięki implementacji tzw. “grubego klienta”. Pozwala to nie tylko na lepsze skalowanie, ale również większe możliwości pod względem UX (“user experience”*)*.

* Dodatkowa walidacja po stronie klienta
* Bogate możliwości rozbudowania interfejsu
* Możliwość zastosowania innego frameworka niż w części serwerowej, który jest projektowany specjalnie dla klienta aplikacji
* Separacja części klienckiej od części serwerowej, która uproszcza dodawanie obsługi nowych urządzeń

## Separacja kontekstów Entity Framework

Większość aplikacji wykorzystuje jeden kontekst dla repozytoriów, utrudnia to oddzielenie funkcjonalności od siebie, np. przy przechodzeniu na architekturę mikroserwisową. Aby rozwiązać ten problem, nowsze wersje Entity Framework (<6) wspierają wykorzystywanie wielu kontekstów. Takie podejście pozwala zmodularyzować aplikację, wprowadza natomiast problem z generowaniem migracji oraz obsługą transakcji między kontekstami.

W celu rozwiązania tego problemu, aplikacja wykorzystuje kilka kontekstów, natomiast wszystkie są gromadzone w dodatkowym projekcie, gdzie znajduje się kontekst zbiorczy (implementujący wszystkie wymagane interfejsy kontekstów). Gwarantuje to bezproblemowe generowanie migracji, jednolitą obsługę transakcji oraz jeden punkt tworzenia bazy danych w przypadku wykorzystania podejścia „Code First”.

## Programowa rejestracja wstrzykiwania zależności

Ponieważ odwracanie zależności powoduje odwrócenie referencji między projektami, aby uniknąć cyklicznych odwołań wykorzystuje się deklaratywne rejestrowanie typów. Jest to realizowane na ogół w postaci plików XML lub JSON, które są czytane w punkcie startowym aplikacji i tam, na ich podstawie, konstruowane są kontenery. Takie podejście uniemożliwia weryfikację rejestracji podczas kompilacji oraz zaciemnia zależności między komponentami (plik z rejestracjami znajduje się na ogół w najwyższej warstwie i zawiera również rejestracje typów z niższych warstw).

Aby umożliwić programowe rejestrowanie typów do projektu został wprowadzony dodatkowy projekt, dzięki któremu unikamy cyklicznych zależności.

# Główne abstrakcje

Poniżej zamieszczono diagram klas przedstawiający kluczowe abstrakcje w aplikacji.

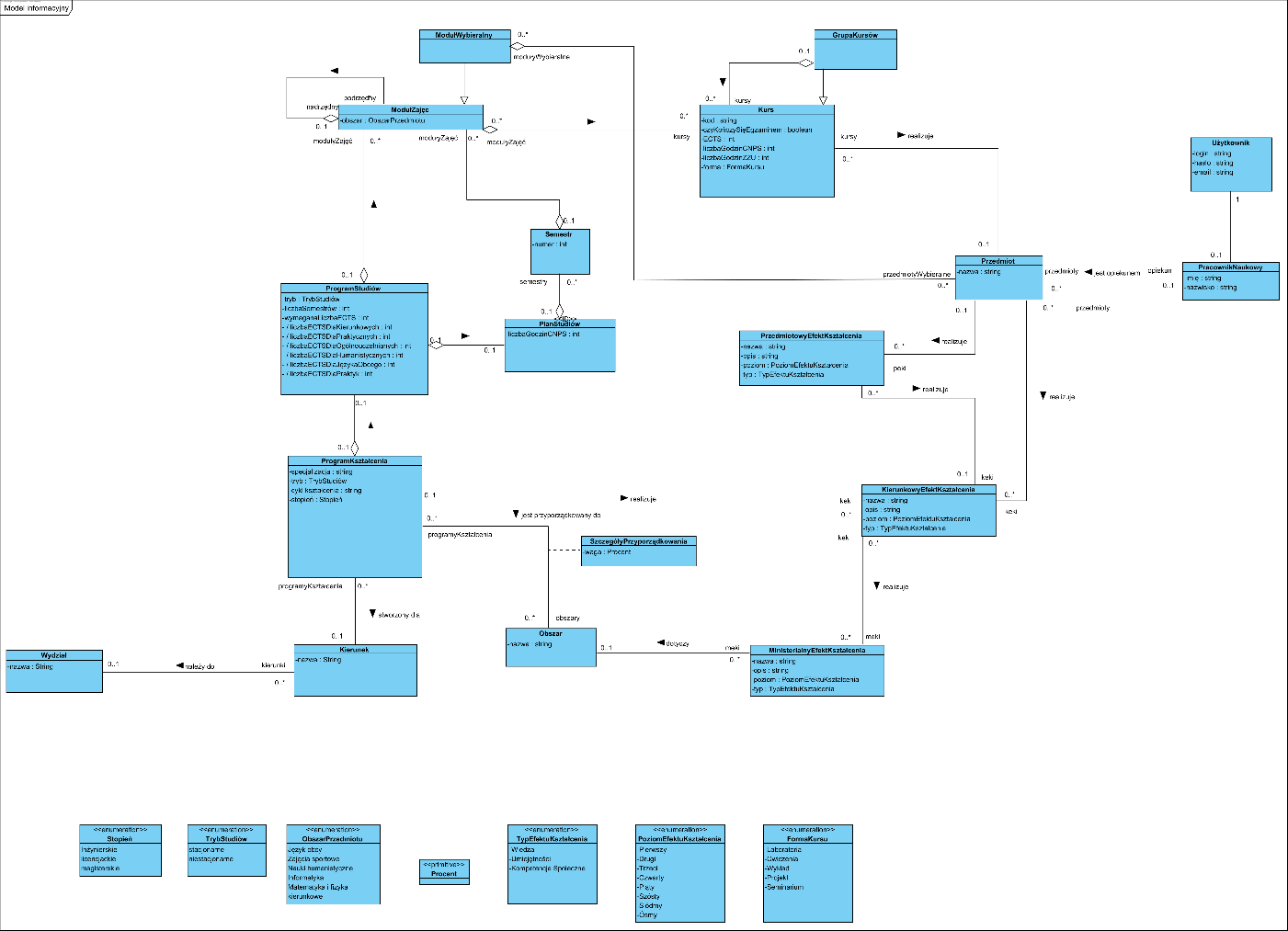


Figure 1 - diagram klas - perspektywa informacyjna

# Widoki architektoniczne

Poniżej zamieszczono widoki architektoniczne omawianej aplikacji:

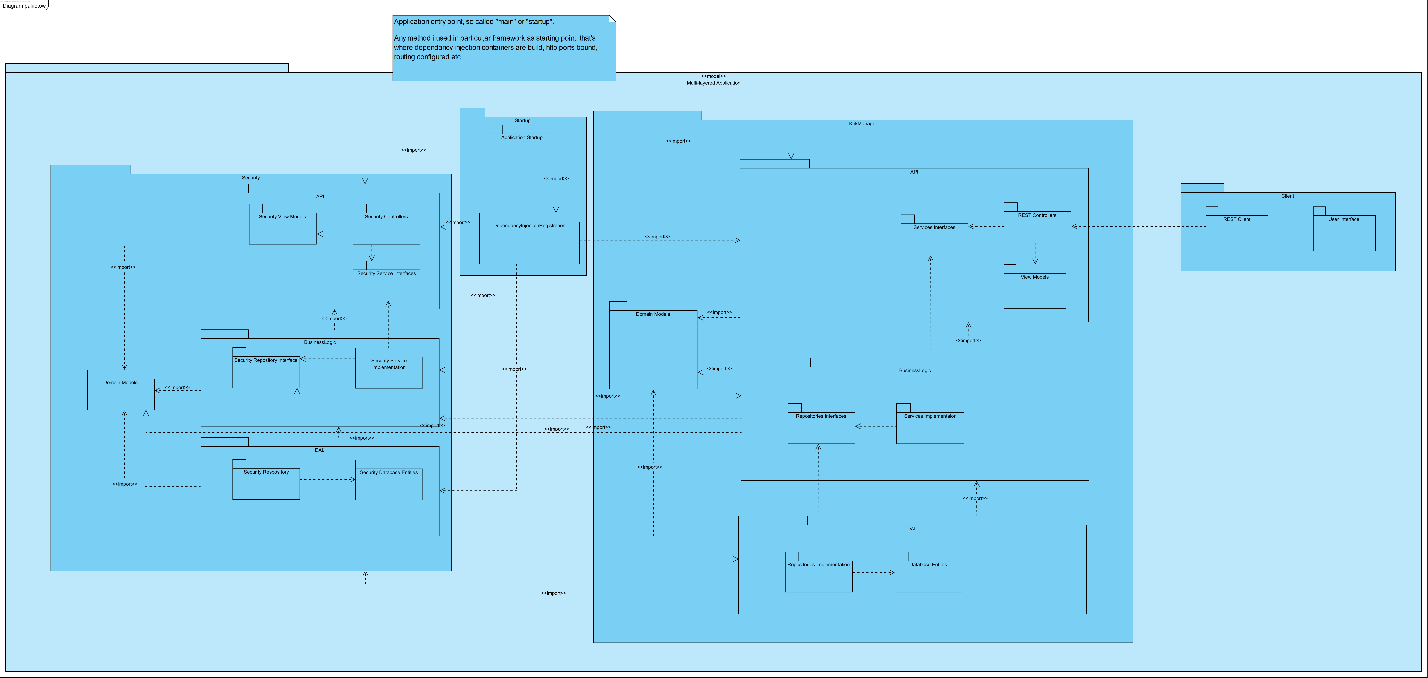


Figure 2 - diagram pakietów - perspektyuwa wytworzenia

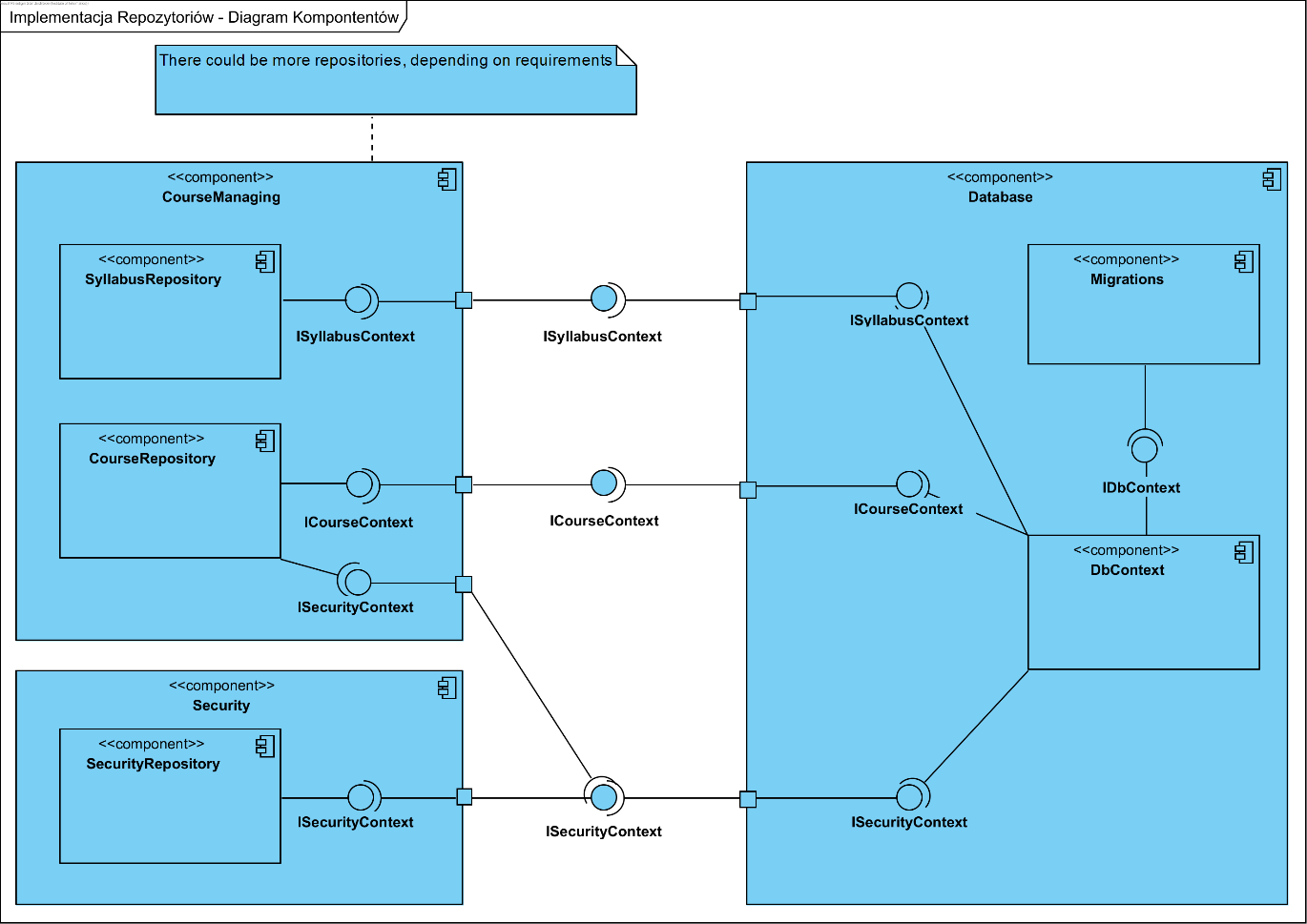


Figure 3 diagram komponentów fragmentu warstwy dostępu do danych - perspektywa wytwarzania

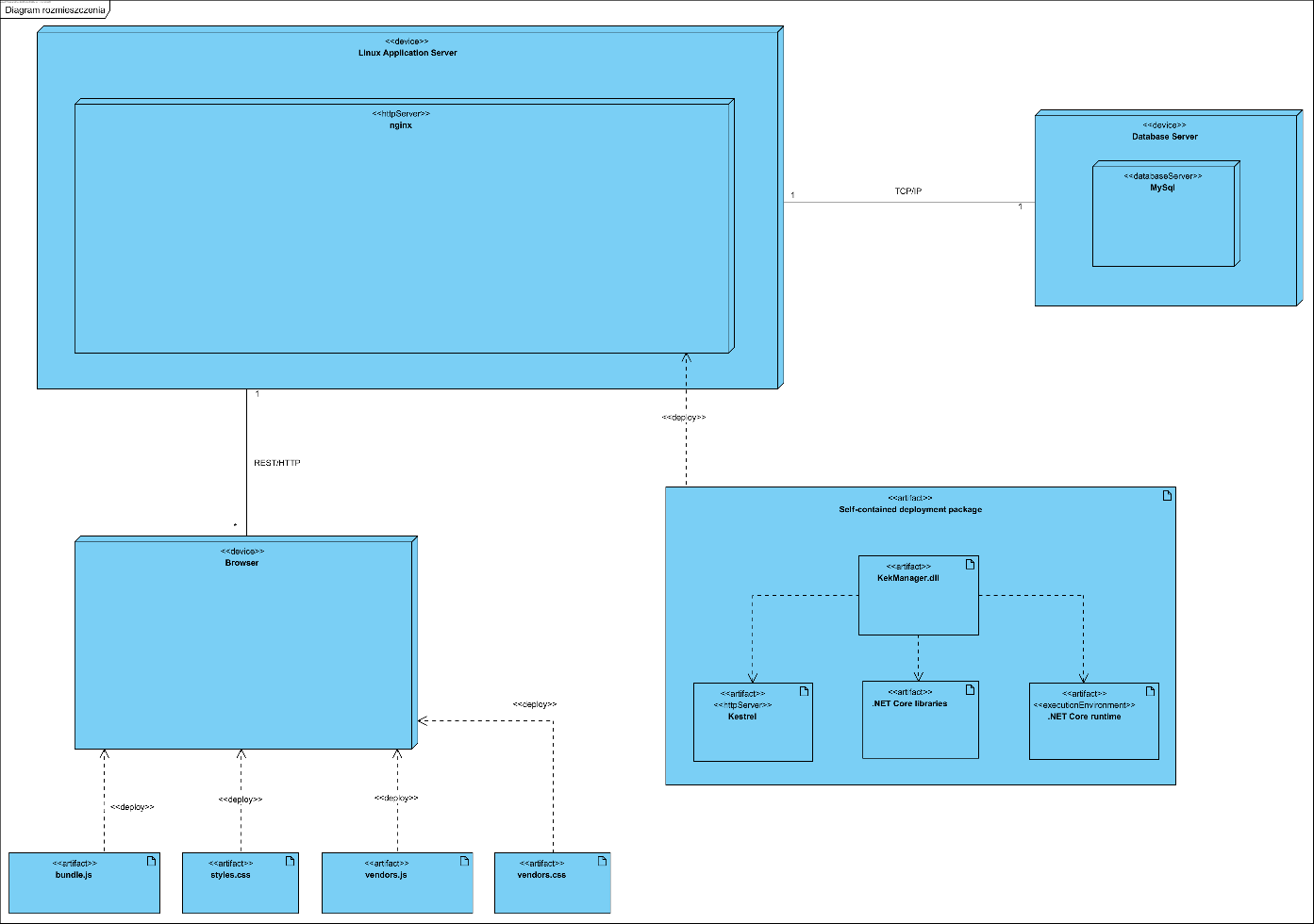


Figure 4 - diagram rozmieszczenia - perspektywa wdrożenia

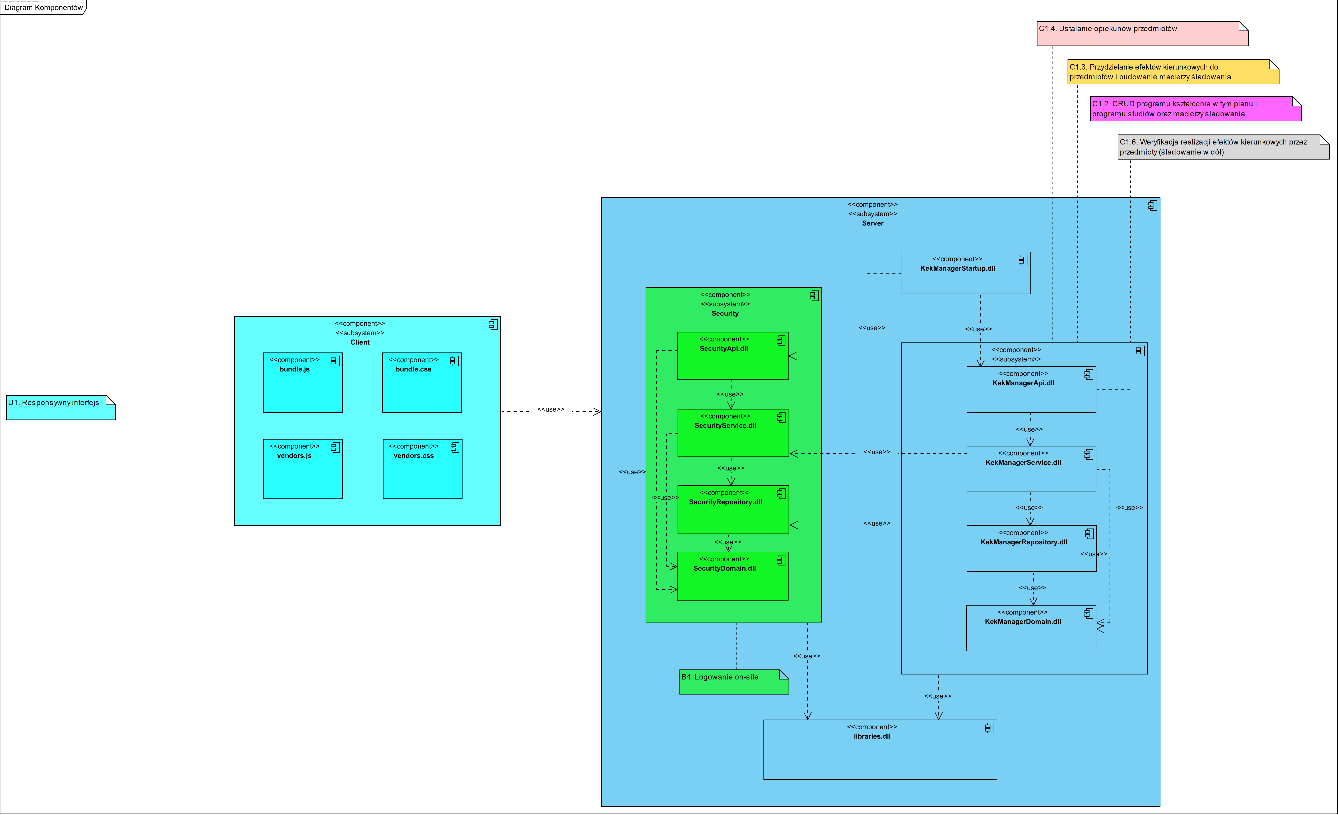


Figure 5 - diagram komponentów - perspektywa wytworzenia

# Realizacje przypadków użycia (dla wybranych przypadków użycia)

Kolorem czerwonym oznaczone zostały elementy klienta aplikacji.

## C1.4 Ustalanie opiekuna przedmiotu

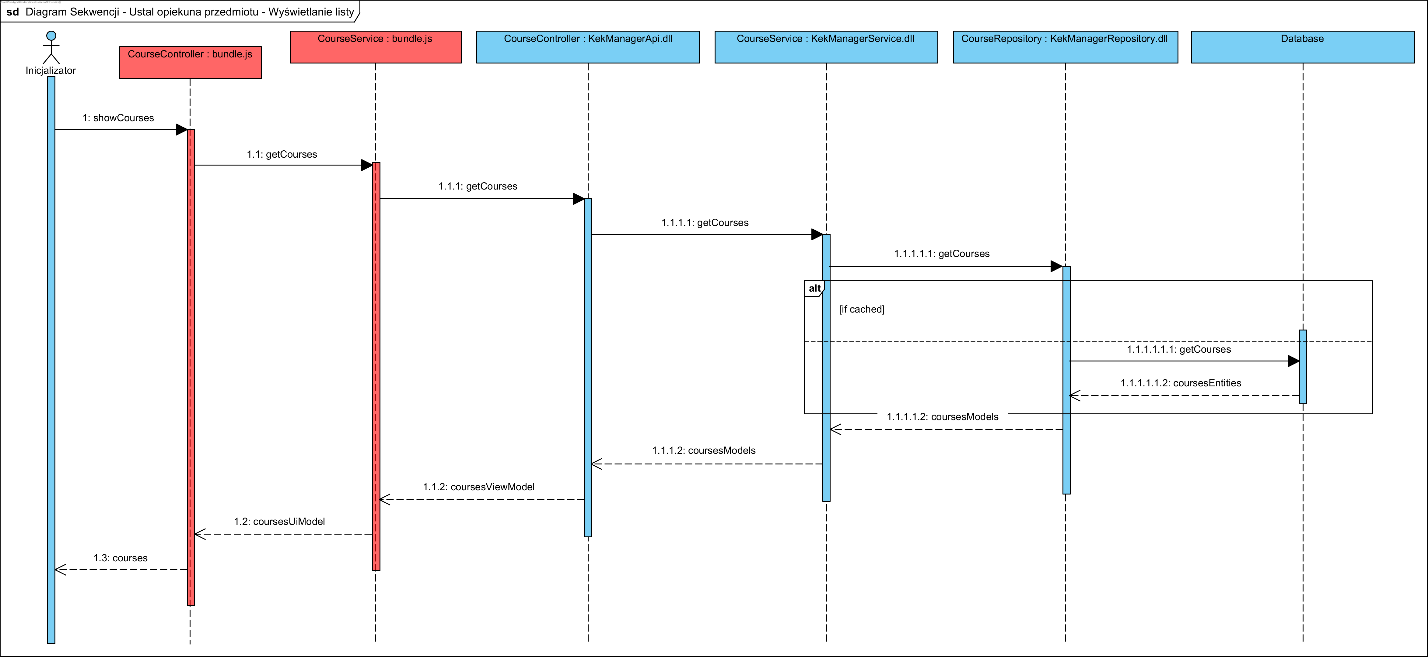


Figure 6 - diagram sekwencji - wyświetlanie listy kursów

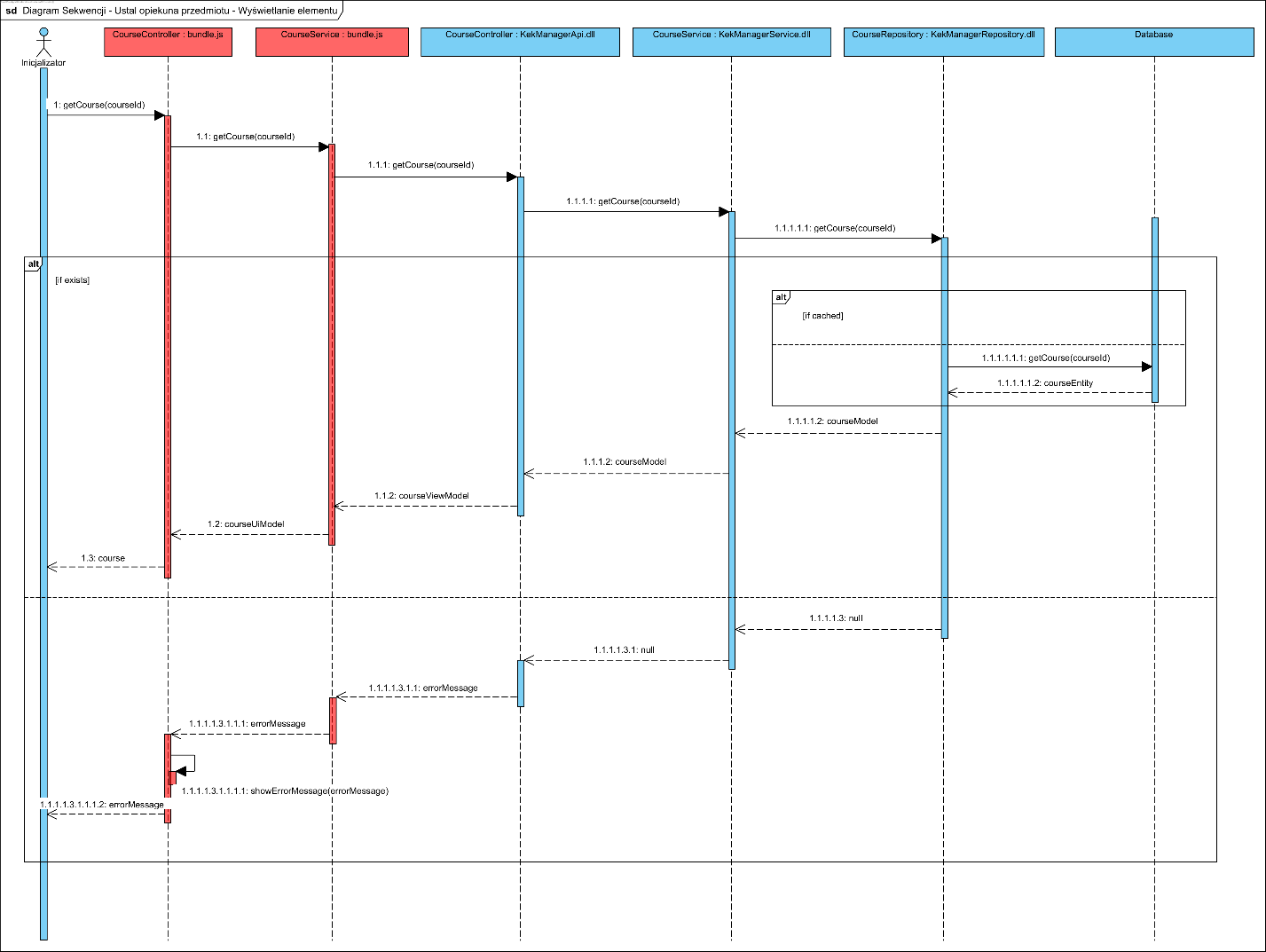


Figure 7 - diagram sekwencji - wyświetlanie kursu

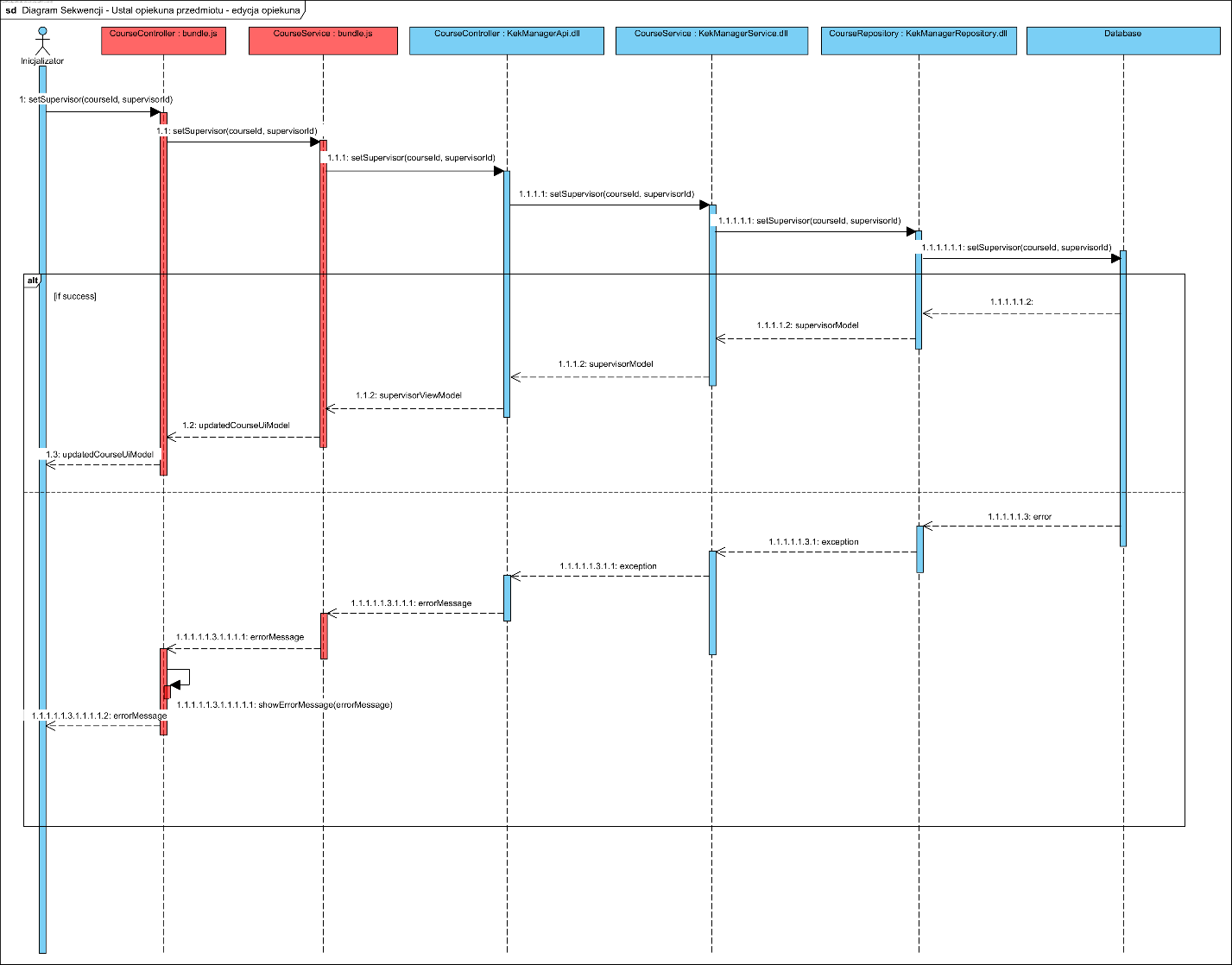


Figure 8 - diagram sekwencji - ustalanie opiekuna kursu