POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: AUTOMATYKA I ROBOTYKA (AIR)
SPECJALNOŚĆ: TECHNOLOGIE INFORMACYJNE W AUTOMATYCE (ART)

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja desktopowa wspomagająca pracę przychodni pediatrycznej.

Desktop application supporting the pediatric outpatient clinic.

AUTOR:

Kacper Weiss

PROWADZĄCY PRACĘ:

Dr inż. Jarosław Mierzwa

OCENA PRACY:

WROCŁAW, 2020

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wprowadzenie

1.1. Wstęp

W postępującej epoce cyfryzacji, coraz więcej różnych instytucji i przedsiębiorstw zaczyna korzystać z wszelkiego rodzaju oprogramowania wspomagającego, lub automatyzującego pracę i służącego do bezpiecznego przechowywania danych niezbędnych dla biznesu. Zwiększenie znaczenia i zainteresowania rozwiązaniami IT zarówno ze strony klientów, jak i inżynierów i developerów z całego świata zmieniło sposób w jaki funkcjonują wszyscy. Zaczynając od pojedynczych użytkowników, po działalności wielkich korporacji. Rozwiązania takie znajdziemy m.in. w takich dziedzinach jak:

- Telekomunikacja
- Transport
- Automatyka budynkowa
- Rekreacja
- Edukacja
- Medycyna
- i wiele innych

1.2. Cel i zakres pracy

Celem pracy jest projekt i implementacja aplikacji desktopowej służącej do wspomagania pracy przychodni pediatrycznej, tj. aplikacji typu klient-serwer umożliwiającej przetwarzanie i przechowywanie danych pacjentów, dziecięcej dokumentacji medycznej, a także wystawianie

wszelkiego rodzaju druków takich jak recepty, czy skierowania. Projekt został zrealizowany wyłącznie z myślą o systemie operacyjnym Windows, z tego powodu oprogramowanie napisane zostało w języku C# wykorzystującym platformę .NET. Z tego powodu niezbędne było zapoznanie się z metodami rozwoju oprogramowania na platformie .NET, zdobycie podstawowej wiedzy z zakresu tworzenia baz danych z wykorzystaniem takich narzędzi dostępnych na tejże platformie, a także struktury takiego oprogramowania. Do realizacji pracy konieczne było opanowanie narzędzi takich jak:

- Środowisko programistyczne Microsoft Visual Studio 2017
- Material Design dla Windows Presentation Foundation
- Metodyka tworzenia baz danych Code First w Entity Framework 6

Ponadto należało opracować sposób realizacji wszystkich założeń projektu w sposób jak najbardziej spójny i otwarty na dalsze rozszerzanie jego funkcjonalności. Do wykonania projektu kluczowa okazała się wiedza i umiejętności nabyte w trakcie trwania studiów, m.in. z dziedziny baz danych, oraz programowania w językach wysokiego poziomu.

1.3. Układ pracy

Praca składa się z sześciu rozdziałów. W pierwszym rozdziale przedstawione zostały wprowadzenie do tematu, cel, zakres i układ pracy. W rozdziale drugim przedstawione zostały wymagania funkcjonalne i niefunkcjonalne projektu. Rozdział trzeci to omówienie zastosowanych w trakcie pracy technologi i narzędzi. Rozdział czwarty opisuje strukturę projektu, oraz sposób jego implementacji. W piątym rozdziale przedstawione zostały testy aplikacji wraz z ich rezultatami. W ostatnim szóstym rozdziale zamieszczone zostało podsumowanie wykonanej pracy, wnioski z zakończonej pracy, a także dalsze możliwości rozwoju projektu. Na koniec zostały zamieszczone spis literatury, a także załącznik w postaci instrukcji obsługi.

Rozdział 2

Wymagania funkcjonalne i niefunkcjonalne

Podstawowe cele projektu pozwoliłem sobie przedstawić w formie tabel, a założenia podzieliłem na wymagania funkcjonalne (tab. 2.1) i wymagania niefunkcjonalne (tab. 2.2).

2.1. Wymagania funkcjonalne

Tab. 2.1: Założenia funkcjonalne projektu

Założenia funkcjonalne
Tworzenie kont nowych pracowników
Logowanie pracowników do systemu
Przechowywanie dokumentacji medycznej pacjentów
Rejestracja wizyt
Dodawanie nowych usług do oferty
Wystawianie druków takich jak recepty i skierowania

Powyższe wymagania definiują podstawowe funkcje programu, jakie na projekt powinien móc wykonywać w chwili zakończenia pracy nad nim, aby mógł zostać uznany za zakończony. Wymagania te pozwalają nam również na określenie przypadków użycia aplikacji.

2.2. Wymagania niefunkcjonalne

Tab. 2.2: Założenia niefunkcjonalne projektu

Założenia niefunkcjonalne
Poprawnie funkcjonowanie oprogramowania na urządzeniach działających w systemie Windows
Logowanie i rejestracja użytkowników muszą być przeprowadzone w sposób bezpieczny
Interfejs graficzny musi być intuicyjny i łatwy w użyciu
Aplikacja musi być w miarę możliwości zoptymalizowana na tyle, aby użytkownik nie odczuwał dyskomfortu
Bezpieczna obsługa błędów i wyjątków
Bezpieczny dostęp do danych
Całość oprogramowania implementowana za pomocą platformy .NET w języku C#
Aplikacja musi wykorzystywać niekomercyjną bazę danych

Powyższe wymagania definiują podstawowe wymagania techniczne, ograniczenia przy których system musi nadal spełniać swoje podstawowe funkcje, a także jakie udogodnienia może oczekiwać użytkownik ze strony aplikacji.

Rozdział 3

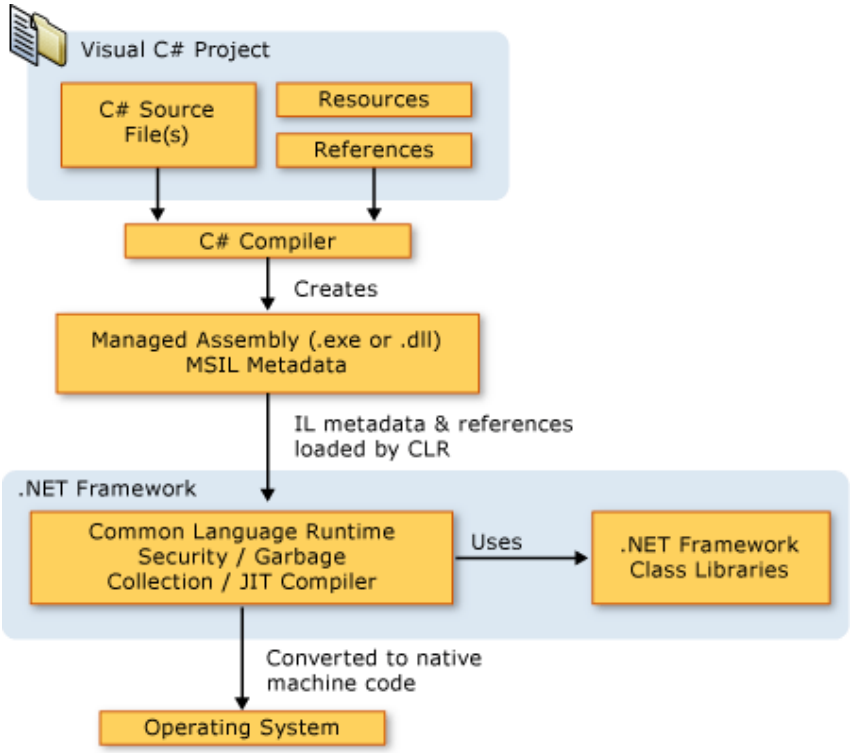
Zastosowane technologie i narzędzia

W tym rozdziale opisane są wszelkiego rodzaju programy, narzędzia i technologie, które zostały wykorzystane w procesie tworzenia aplikacji. Zostanie opisana platforma .NET, usługa Microsoft SQL Server 2017 w wersji Developer w której skład wchodzi użyta baza danych, a także narzędzia developerskie takie jak Microsoft SQL Server Management Studio 17. Krótko zostanie zaprezentowane zintegrowane środowisko developerskie Microsoft Visual Studio 2017 w wersji Community, wykorzystywany tam język programowania C# i język opisu interfejsu użytkownika eXtensible Application Markup Language (XAML).

3.1. C#

Cała logika aplikacji została wykonana w języku C#. C# jest powszechnie znanym językiem programowania wysokiego poziomu, zorientowanym wokół programowania obiektowego. Jego pierwsza wersja powstała na pod koniec lat dziewięćdziesiątych ubiegłego wieku, a projekt pod okiem Andersa Hejlsberga początkowo nosił nazwe COOL, która miała oznaczać wtedy Ć like Object Oriented Language". Ostatecznie nazwę tę zmieniono na C# w celu podkreślenia jego korzeni w języku C++, a wybór znaku # był spowodowany jego podobieństwem do 4 znaków "+". Obecnie język C# coraz prężniej się rozwija, m.in. w ramach część jej otwarto-źródłowej implementacji zwanej .NET Core, która poza Windowsem pozwala tworzyć aplikacje również na platformy Linux, oraz macOS, czyniąc język C# coraz bardziej uniwersalnym narzędziem.

Do poprawnego uruchomienia, działania, czy tworzenia aplikacji pisanych w języku C# konieczne jest posiadanie platformy .NET Framework, która obecnie jest integralnym składnikiem systemu Windows, lub w wypadku systemu Linux i macOS platformy .NET Core. Dzieję się tak ponieważ kod napisany w języku C# jest kompilowany do CIL (Common Intermediate Language), który następie jest uruchamiany przez CLR (Common Language Runtime) i interpretowany przez kompilator JIT (Just-In-Time), a tam po konwersji na natywny kod maszynowy, program wykonywany jest już przez system operacyjny.



Rys. 3.1: Proces uruchamiania oprogramowania w języku C# [3].

3.2. XAML

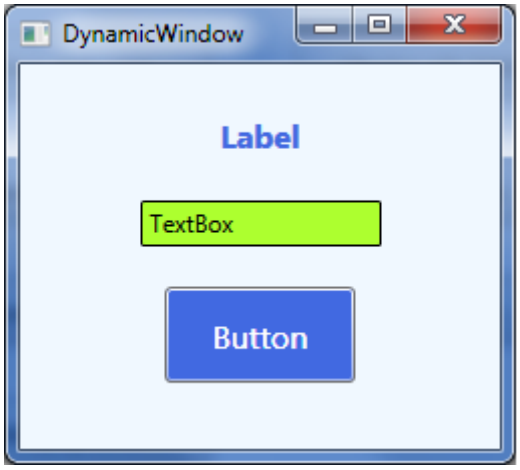
Język XAML oparty jest w dużej mierze o język XML. Jest on wykorzystywany do tworzenia interfejsu użytkownika w technologii WPF (Windows Presentation Foundation). Wykorzystanie języka XAML umożliwia developerom rozdzielenie kodu logicznego, od definicji kodu źródłowego. Dzięki postawieniu granicy pomiędzy interfejsem, a logiką uzyskuje się większą spójność i czytelność kodu. Kod odpowiadający za logikę często znajduje się w tym samym module dzięki zastosowaniu plików code behind. Poszczególne elementy interfejsu można bindować z odpowiednimi danymi, które przy ustawieniu DataContext na odpowiednie okno, oraz wykorzystanie odpowiednich zdarzeń, będzie na bieżąco aktualizowane z niewielkim kosztem

dla aplikacji. Pracę z językiem XAML ułatwiają różne narzędzia wchodzące w skład Microsoft Visual Studio m.in. Microsoft Blend. Umożliwia to pracę nad interfacem zarówno za pomocą designera graficznego, jak i edytora tekstowego z bezpośrednim podglądem ostatecznego wyglądu interfejsu aplikacji. Na listingu 3.1 zamieściłem przykładowy element interfejsu napisany w języku XAML.

```
<StackPanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Name="stackPanel1" Orientation="Vertical" Background="#FFF0F8FF">
    <Label Name="label1" Foreground="#FF4169E1" FontSize="16" FontWeight="Bold"
        HorizontalContentAlignment="Center" Width="80" Height="28" Margin="0,20,0,0">
        Label
    </Label>
    <TextBox Name="textBox1" BorderBrush="#FF000000" Background="#FFADFF2F"
        Width="120" Height="23" Margin="0,20,0,0">
        TextBox
    </TextBox>
    <Button Name="button1" Background="#FF4169E1" Foreground="#FFFFFFFF"
        FontSize="16" Width="95" Height="48" Margin="0,20,0,0">
        Button
    </Button>
</StackPanel>
```

Listing 3.1: Przykład treści pliku „StackPanel.xaml”.
Źródło: Strona internetowa Developer Notes [2].

Poniżej zamieściłem rysunek 3.2 przedstawiający utworzony za pomocą powyższego kodu element interfejsu.



Rys. 3.2: Rezultat przykładowego elementu interfejsu w XAML.
Źródło: Strona internetowa Developer Notes [2].

3.3. Microsoft Visual Studio 2017

Microsoft Visual Studio jest zintegrowanym środowiskiem programistycznym, jego pierwsze wersje powstały już pod koniec XX wieku. Microsoft Visual Studio 97 znacznie różni się od wersji użytej w projekcie, ponieważ to IDE powstało jeszcze przed powstaniem platformy .NET, a na celu miała przede wszystkim połączenie istniejących już środowisk w jedno uniwersalne. Wersja Visual Studio 7.0 z 2002 roku wprowadziła platformę .NET jako podstawę działania środowiska, co już zostało do najnowszych wersji. Obecnie najnowsza wersja Visual Studio 2019, która różni się od Visual Studio 2017 bardzo niewielkim stopniu, są to zmiany głównie co do wersji używanych narzędzi, tj. zaktualizowane zostały .NET Core do wersji 3.0, F# do wersji 4.6, a także C# do wersji Preview 8.0. Visual Studio jest środowiskiem które udostępnia użytkownikowi wiele opcji szybkiej edycji, refaktoryzacji kodu, oraz bardzo wygodny, zintegrowany debugger i narzędzia pozwalające na bardzo wygodną pracę w języku C#. Poza wsparciem dla języka C# środowisko to wspiera m.in.:

- Microsoft Visual Basic,
- Microsoft Visual C++,
- Microsoft Visual J#,
- Microsoft Visual F#,
- a także wiele innych po zainstalowaniu dodatkowych zasobów np. JavaScript.

3.4. Microsoft SQL Server 2017

Microsoft SQL Server (MS SQL) jest głównym bazodanowym produktem firmy Microsoft. Pełni funkcje systemu zarządzania bazą danych, a jego charakterystyczną cechą jest wykorzystanie języka zapytań Transact-SQL stanowiącego rozwinięcie standardu ANSI/ISO. Istnieją wersje tego oprogramowania przeznaczone do zastosowań zarówno komercyjnych, jak i niekomercyjnych, w zależności od wykorzystywanej przez użytkownika edycji programu. Od wersji 2005 wraz z edycją Developer możemy pobrać graficzne narzędzia do obsługi bazy danych, takie jak Microsoft SQL Server Management Studio.

3.5. .NET Framework

.NET Framework jest platformą programistyczną stworzoną i zaprojektowaną przez firmę Microsoft. Obejmuje ona środowisko uruchomieniowe CLR (Common Language Runtime)

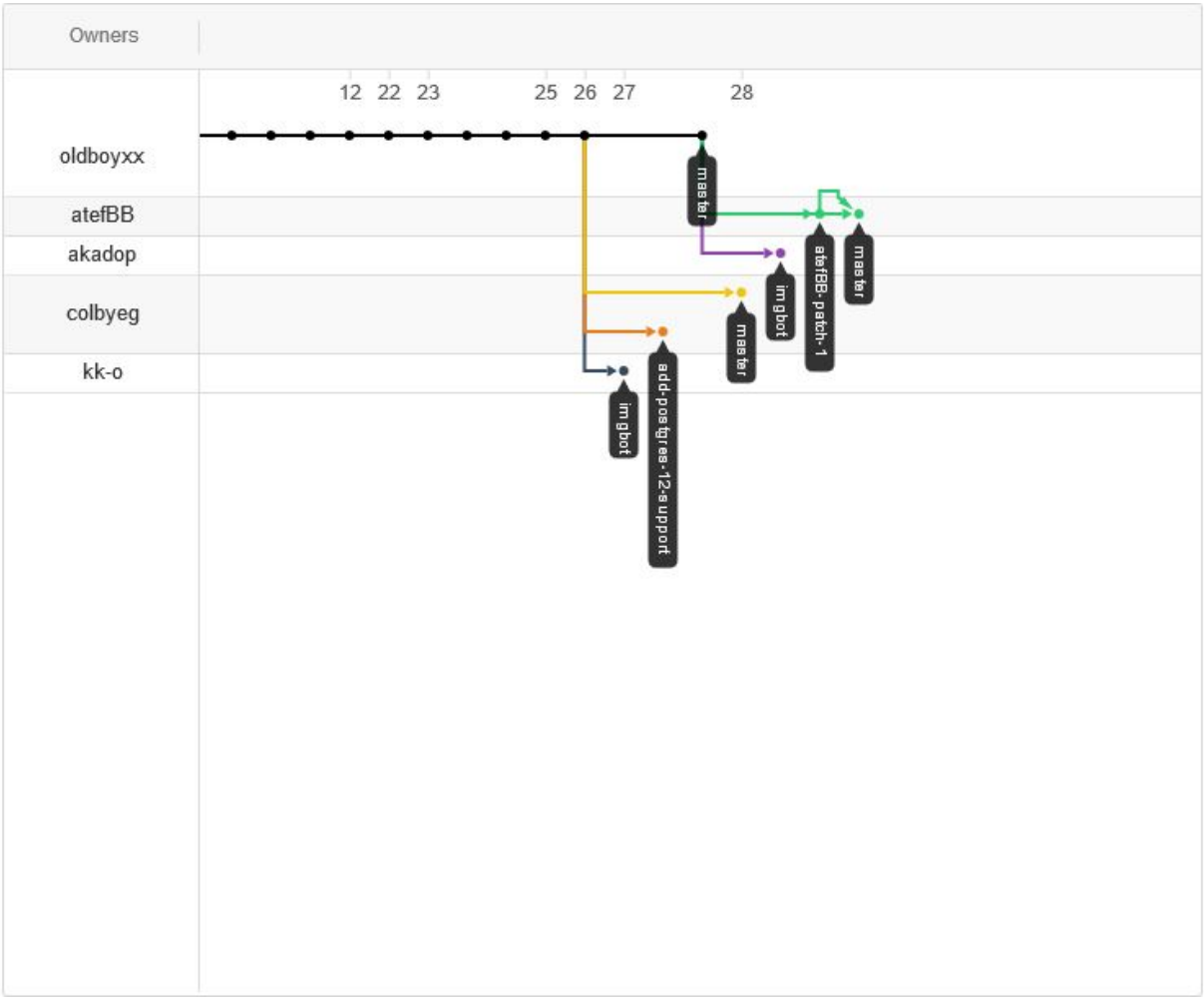
które zarządza aplikacjami przeznaczonymi na tą platformę. Zapewnia ono odpowiednie zarządzanie pamięcią, usługi systemowe takie jak garbage collector, szeroki wybór wbudowanych bibliotek klas dających deweloperom dostęp do zaufanego, zoptymalizowanego kodu dla większości dziedzin rozwoju oprogramowania. Istotną zaletą środowiska platformy .NET jest interoperacyjność języków programowania. Znaczy to, że kod pisany w językach kompilowalnych do CIL (Common Intermediate Language) mogą ze sobą współpracować w ramach platformy za pomocą narzędzi i technologii takich jak między innymi COM Interop.

3.6. Git

Niezbędnym narzędziem dla każdego projektu zajmującym się wytwarzaniem oprogramowania jest system kontroli wersji. W tej pracy zdecydowałem się użyć systemu kontroli wersji Git stworzonego przez Linusa Torvaldsa. Jest to oprogramowanie typu Open-Source. Praca z tym systemem jest bardzo wygodna, pomaga usystematyzować pracę nad projektem, a także przy wykorzystywaniu serwisów takich jak GitHub zapewnia bezpieczne miejsce do przechowywania danych, a także historii zmian w kodzie, czy tzw. „`commit'ów`". Git sprawdza się zarówno w pracy samodzielnej jak i zespołowej. Istnienie tzw. „`branch'y`" pozwala na przechowywanie niezależnych od siebie gałęzi zmian kodu. Poniżej pozwoliłem sobie zamieścić przykładową strukturę „`branch'y`" na rys. 3.3. W trakcie pracy nad kodem możemy w dowolnej chwili wrócić do wybranego „`commit'a`", np. w przypadku gdy podejmiemy decyzje o zmianie podejścia do sposobu rozwiązania jakiegoś problemu w projekcie. Git udostępnia wiele komend z których może korzystać deweloper, nalerzą do nich:

- git init - służący do inicjalizacji repozytorium,
- git add - dodawanie plików obsługiwanych przez kontrole wersji w projekcie,
- git commit - zapisanie zmian w repozytorium,
- git push - wysyłanie zmian do zdalnego repozytorium,
- git pull - ściąganie zmian ze zdalnego repozytorium,
- git log - wyświetlanie historii zmian repozytorium.

Oprogramowanie to można za darmo pobrać z oficjalnej strony wraz z narzędziami wspomagającymi przyszłą pracę z tym systemem kontroli wersji dla dowolnej platformy.



Rys. 3.3: Przykładowy fragment drzewa „`commit'ów`” z repozytorium „`jira_clone`” na stronie GitHub[4] z dnia 28.01.2020r.

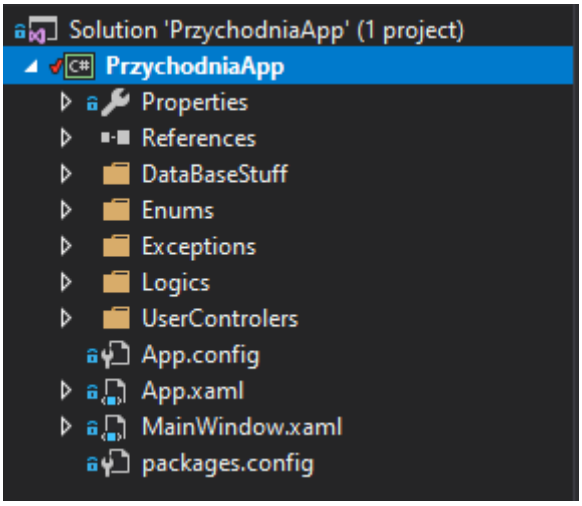
Rozdział 4

Projekt i implementacja

Rozdział ten przeznaczony jest do zaznajomienia czytelnika z projektem, jego działaniem i implementacją. Szczególna uwaga zostanie poświęcona aspektom funkcjonalnym aplikacji („Back-End"), oraz części wizualnej projektu („Front-End"). Opis techniczny rozpocznę od omówienia struktury projektu, najważniejszych jego komponentów - folderów i plików, a następnie przejdę do bardziej szczegółowych opisów funkcjonalności.

4.1. Struktura projektu

Projekt prowadzony był przy jak najbliższym przestrzeganiu ogólnie przyjętych standardów, które można odnaleźć na oficjalnej stronie z dokumentacją sporządzoną przez firmę Microsoft [1]. Ze względu na dotychczasowe stosunkowo niewielkie doświadczenie o charakterze wyłącznie akademickim, zdecydowałem się na nie wprowadzanie do projektu zbędnych komplikacji, a projekt zrealizowałem w sposób jak najbardziej uporządkowany. Najbardziej ogólną strukturę projektu przedstawiam na rys. 4.1.
W folderze „*DataBaseStuff*" zostały zamieszczone stworzone przez kreator ADO.NET Entity Data Model komponenty odpowiadające za kontekst bazy danych Entity Framework 6 dla podejścia Code First, modele przedstawiające oczekiwane struktury znajdujące się później w bazie danych, a także statyczne klasy służące do modyfikowania relacji pomiędzy nimi.



Rys. 4.1: Ogólna struktura projektu.

Rozdział 5

Testy aplikacji

Rozdział 6

Podsumowanie

6.1. Wnioski

6.2. Możliwości rozwoju

Literatura

[1] Konwencje kodowania c#. https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/inside-a-program/coding-conventions.

[2] Strona internetowa developer notes. https://mndevnotes.wordpress.com/2012/06/18/wpf-dynamiczne-tworzenie-i-wczytywanie-kodu-xaml/.

[3] Strona internetowa z dokumentacją języka c#. https://docs.microsoft.com/pl-pl/dotnet/csharp/.

[4] I. Reic. A simplified jira clone built with react/babel (client), and node/typescript (api). auto formatted with prettier, tested with cypress. https://github.com/oldboyxx/jira_clone.

Dodatek A

Instrukcja obsługi

Dodatek B

Opis załączonej płyty CD/DVD

Tutaj jest miejsce na zamieszczenie opisu zawartości załączonej płyty. Należy wymienić, co zawiera.