



UNIWERSYTET  
IM. ADAMA MICKIEWICZA  
W POZNANIU

Wydział Matematyki i Informatyki

Krystian Osiński  
Numer albumu: 444820

Platforma e-learningowa Brainboost  
Brainboost e-learning platform

Praca inżynierska na kierunku **informatyka**  
napisana pod opieką  
**dr Patryka Żywicy**

Poznań, styczeń 2024

## Streszczenie

Platforma e-learningowa Brainboost to praca powstała na bazie projektu inżynierskiego, w ramach którego przygotowana została aplikacja webowa do tworzenia, udostępniania i realizowania kursów edukacyjnych, w szczególności kursów programowania. Praca umożliwia zapoznanie się z trzema kluczowymi kwestiami procesu wytwarzania aplikacji: 1) wdrażania systemu z wykorzystaniem elementów podejścia DevOps; 2) tworzenia aplikacji z wykorzystaniem frontendowego frameworka Angular; 3) zarządzania projektem z wykorzystaniem zwinnych metodyk zarządzania projektem (Kanban). Każde z tych zagadnień zostało omówione w osobnym rozdziale pracy – a rozdziały te stanowią główną część pracy. W poszczególnych rozdziałach zamieszczono zarówno teoretyczne ujęcie omawianych zagadnień, jak i przykłady implementacji danych metodyk/technologii/narzędzi w projekcie Brainboost (w tym kod źródłowy). Dlatego zakres wątków poruszanych w pracy jest szeroki: od Kubernetesa, CI/CD i GitOpsa, przez Angulara, dyrektywy, serwisy i mechanizm DI, aż po przepływ, WIP i wizualizację. Pozwoliło to jednak w pełniejszy sposób przedstawić proces realizacji projektu Brainboost.

**Słowa kluczowe:** Kubernetes, CI/CD, GitOps, Angular, dyrektywa, serwis, mechanizm DI, Kanban, wizualizacja, WIP, przepływ

# Abstract

The Brainboost e-learning platform is a work based on an engineering project in which a web application was prepared for creating, sharing and implementing educational courses, in particular programming courses. This thesis allows to become familiar with three key issues of the application development process: 1) system deployment using elements of the DevOps approach; 2) creating applications using the front-end Angular framework; 3) project management using Agile project management methodologies (Kanban). Each of these issues was discussed in a separate chapter of the thesis - and these chapters constitute the main part of the work. The individual chapters contain both a theoretical approach to the issues discussed, as well as examples of the implementation of given methodologies/technologies/tools in the Brainboost project (including the source code). Therefore, the range of topics covered in the work is wide: from Kubernetes, CI/CD and GitOps, through Angular, directives, services and the DI mechanism, to flow, WIP and visualization. However, this allowed for a more comprehensive presentation of the Brainboost project implementation process.

**Keywords:** Kubernetes, CI/CD, GitOps, Angular, directive, service, DI mechanism, Kanban, visualization, WIP, flow

# Spis treści

Wstęp . . . . .	6
<b>Rozdział 1. Zastosowanie Kanban w zarządzaniu procesem wytwarzania oprogramowania . . . . .</b>	<b>8</b>
1.1. Wstęp . . . . .	8
1.1.1. Zarys . . . . .	8
1.1.2. Kontekst . . . . .	8
1.1.3. Cel . . . . .	9
1.2. Historia powstania metodyki Kanban . . . . .	9
1.2.1. Etymologia . . . . .	9
1.2.2. Toyota Production System . . . . .	10
1.2.3. Filary systemu TPS . . . . .	12
1.2.4. Przełom w wytwarzaniu oprogramowania . . . . .	14
1.2.5. Kanban jako metodyka Agile . . . . .	19
1.3. Metodyka Kanban . . . . .	21
1.3.1. Wizualizacja . . . . .	21
1.3.2. Praca w toku . . . . .	24
1.3.3. Zarządzanie procesem . . . . .	26
1.3.4. Ewolucja procesu . . . . .	29
1.4. Zastosowanie metodyki Kanban w projekcie Brainboost . . . . .	31
1.4.1. Wybór metodyki . . . . .	31
1.4.2. Organizacja pracy zespołu projektowego . . . . .	31
1.4.3. Wizualizacja procesu . . . . .	32
1.5. Zarządzanie procesem i jego ewolucja . . . . .	40
1.5.1. Cykliczne spotkania członków zespołu . . . . .	40
1.5.2. Retrospektywa . . . . .	40
1.5.3. Przepływ a praca zespołu projektowego . . . . .	43
1.5.4. Obszary do udoskonalenia . . . . .	46
1.6. Podsumowanie . . . . .	48
1.6.1. Wynik projektu . . . . .	48
1.6.2. Wnioski . . . . .	48
Zakończenie . . . . .	50
Bibliografia . . . . .	51

Spis treści	5
Załącznik nr 1: Brainboost - wizja i wymagania projektowe . . . . .	57
Załącznik nr 2: Dokumentacja użytkownika . . . . .	62

## WSTĘP

Brainboost to aplikacja webowa służąca do tworzenia, publikowania i realizowania kursów z różnych dziedzin wiedzy, przede wszystkim jednak z dziedzin technicznych (w szczególności: programowania). Została stworzona we współpracy z konkretnym klientem – firmą SkyBlue, zajmującą się edukacją młodszych uczniów i prowadzącą kursy m.in. z programowania w Pythonie czy robotyki dla dzieci.

Chociaż na rynku oferta edukacyjna związana z kursami programowania (i pokrewnymi) jest dość szeroka, Brainboost był tworzony konkretnie pod wymagania klienta, mając wpasować się w dotychczasowy model działalności – a także uzupełnić go. Sam projekt zmieniał się koncepcyjnie wraz z rezygowaniem z części wymagań (np. gry przeglądarkowe) i pojawianiem się nowych wymagań (np. obsługa wirtualnych terminali).

Efektem finalnym jest aplikacja, która pozwala na stworzenie kursu, z nastawieniem na kursy programowania, a także na zarządzanie różnymi użytkownikami aplikacji. Pojedynczy kurs może zostać podzielony na kilka modułów, a moduły na mniejsze lekcje. W ramach lekcji można udostępniać materiały tekstowe (choćby z teorią dotyczącą danego zagadnienia, np. jakiegoś języka programowania), jak i audiowizualne (np. nagrany wykład z danego języka programowania). Można również załączać materiały dodatkowe. Do każdej lekcji istnieje możliwość dodania komentarzy – tak, aby komunikacja odbywała się między uczestnikami kursów, jak i między uczestnikami i nauczycielami/twórcami kursu.

Kursy są interaktywne, a więc możliwe jest dodanie zadań, nad którymi uczniowie będą pracować w przeglądarce. W tym momencie aplikacja obsługuje języki Javascript i Python; możliwe jest także tworzenie stron z wykorzystaniem HTML i CSS. Ideą udostępnienia takich możliwości było jak najbardziej praktyczne podejście do nauki. Uczniowie mają więc możliwość uruchamiania napisanego kodu (w odpowiednio przygotowanych do tego edytorach) w przeglądarce – i obserwowania rezultatów swojej pracy natychmiastowo, bez konieczności konfiguracji własnych środowisk. Było to również sporym atutem dla samego klienta, który swoje kursy może realizować z wykorzystaniem zwykłych przeglądarek.

Aplikacja składa się z 4 głównych komponentów: backend (napisany z wykorzystaniem technologii SpringBoot), frontend (napisany głównie w Angularze), terminala (napisanego z użyciem Next.js; ten komponent umożliwia zdalne wykonywanie kodu uczniów), a także z bazy danych (PostgreSQL).

W kolejnych rozdziałach pracy omówione zostały trzy zagadnienia związane

z realizowaniem projektu Brainboost. W pierwszym rozdziale omówione zostały zagadnienia związane z wdrożeniem i utrzymaniem aplikacji Brainboost. Przeanalizowane zostały m.in. zagadnienia konteneryzacji, orkiestracji czy CI/CD. Rozdział pierwszy podzielony jest na dwie zasadnicze części: 1) teoretyczne wprowadzenie do wspomnianych zagadnień; 2) omówienie sposobu praktycznej realizacji tych zagadnień w projekcie.

Tematem drugiego rozdziału jest framework Angular. W rozdziale tym została opisana historia Angulara, a także interesujące komponenty tego języka, m.in. injector, tworzenie zależności, wstrzykiwanie zależności. Trzeci rozdział skupia się na zastosowaniu Kanbana w procesie wytwarzania oprogramowania. Przedstawiono w nim zarówno zarys historyczny i teoretyczny metodyki Kanban, jak i praktyczne wykorzystanie tej metodyki, w szczególności w kontekście projektu Brainboost.

Poza przytoczonymi rozdziałami do pracy dołączono również załączniki (Załącznik-nr-1, Załącznik-nr-2). W pierwszym przedstawiono dokument wymagań, którymi kierowaliśmy się w czasie realizacji całego projektu. Drugi załącznik to dokumentacja użytkownika, stworzona na potrzeby nauczycieli i uczniów, pozwalająca lepiej poznać system Brainboost i sposoby na jego wykorzystanie.

## ROZDZIAŁ 1

# Zastosowanie Kanban w zarządzaniu procesem wytwarzania oprogramowania

Autor: Krystian Osiński

### 1.1. WSTĘP

#### 1.1.1. Zarys

W pierwszej połowie tego rozdziału, przedstawiono historię i kluczowe założenia metodyki Kanban. Następnie szczegółowo opisano praktyczne wdrożenie oraz wykorzystanie tej metodyki w projekcie *Brainboost*. Na końcu podkreślono ewolucję procesu oraz zidentyfikowano błędy i wyzwania, które pojawiły się w trakcie projektu.

#### 1.1.2. Kontekst

Globalizacja, wzrastająca cyfryzacja oraz dynamika środowiska biznesowego i technologicznego wymusza na organizacjach ciągły rozwój w zakresie zarządzania projektami programistycznymi. Projekty te na przestrzeni lat stały się co raz bardziej obszerne, skrócił się czas dostarczania, wymagania ewoluują, a zespoły stały się rozproszone. Aby odnieść sukces w realizacji projektu, przedsiębiorstwa muszą zastosować właściwą strategię i narzędzia, które wspomogą zespoły w efektywnym wytwarzaniu oprogramowania pomimo zmieniającego się otoczenia i założeń. Jednym z podejść, które zdobywa coraz większe uznanie w dziedzinie zwinnego zarządzania projektami jest metodyka Kanban.



### 1.1.3. Cel

Celem i wartością dodaną niniejszego rozdziału jest zaprezentowanie praktycznego wdrożenia i wykorzystania metodyki Kanban w procesie wytwarzania oprogramowania oraz podkreślenie znaczenia stałego doskonalenia procesu. Zostanie to zademonstrowane na przykładzie projektu: "Brainboost".

Rozpocząłem pracę jako manager tylko z podstawową wiedzą na temat metodyki Kanban. Wybrałem podejście zgodne z jej założeniem: *"zaczynij tam, gdzie jesteś"* (ang: "Start where you are now") co okazało się doskonałym punktem wyjścia do dalszej pracy i zgłębienia tematu. Stworzony przeze mnie system ewoluował wraz z moim rozwojem i nabytym doświadczeniem. W rozdziale przedstawię zmiany i usprawnienia, jakie zachodziły w procesie w miarę upływu czasu, skupiając się szczególnie na punkcie kulminacyjnym, którym był okres międzysemestralny. Właśnie wtedy podjąłem intensywne studiowanie literatury oraz blogów poświęconych tematyce zwinnego zarządzania projektami.

Skoncentruję się również na aspektach, które wymagają poprawy oraz na popełnionych błędach i problemach, które pojawiły się w trakcie zarządzania projektem i zespołem.

## 1.2. HISTORIA POWSTANIA METODYKI KANBAN

### 1.2.1. Etymologia

Etymologia słowa *"kanban"* sięga początku XVII wieku. W tym okresie, w Japonii zakończyła się wojna domowa, dzięki czemu kraj jak i jego gospodarka zaczęły się odbudowywać i rozwijać [76].

W wyniku chęci pozyskania większej ilości klientów, ówcześni przedsiębiorcy umieszczali na budynkach specjalne dobrane szyldy. Niosły one jasny i klarowny przekaz poprzez wizualizację przechodniom takich rzecz jak rodzaj oferowanych produktów bądź usług [67]. Przyjmowały one postać wywieszonych przedmiotów lub tablic informacyjnych zawierających tekst i namalowane lub wryte obrazki [66]. W tamtym czasie zostały one nazwane "kanban'ami". Z biegiem kolejnych wieków, ewoluowały, stając się coraz bardziej artystycznymi i stylowymi elementami.

Wyraz ten wywodzi się z połączenia dwóch japońskich słów: *kan* (看) oznaczający: "patrzenie", "oglądanie" oraz *ban* (板) znaczący: "deska", "płyta", "tablica". *Kanban* (看板) można tłumaczyć na język polski jako "szyld" lub "tabliczkę informacyjną"<sup>1</sup>



**Rysunek 1.1.** Kanban z okresu Edo (XVII wiek)

kolekcja Kenichi Higasa, Kobe.

Źródło: [blog.lostartpress.com/2020/04/03/](http://blog.lostartpress.com/2020/04/03/)

### 1.2.2. Toyota Production System

Korzenie metodyki kanban sięgają lat 40. XX wieku. W tamtym czasie Toyota mierzyła się z licznymi wyzwaniami i problemami. Biznes notował liczne straty, a produktywność i wydajność zakładów była na niskim poziomie. Firma nie mogła konkurować z innymi światowymi przedsiębiorstwami z branży motoryzacyjnej, w których to proces produkcji był znacznie lepszej jakości [77]. Założyciel Toyoty, Kiichiro Toyoda, stanął przed ciężkim wyzwaniem polegającym na poprawie ówczesnej sytuacji.

Pod koniec II wojny światowej, w 1943 roku do Toyota Motor Company dołączył ambitny inżynier Taiichi Ōno. W wyniku swojej determinacji, zaangażowania, spostrzegawczości i kreatywności, bardzo szybko awansował w strukturach firmy. Zaledwie po 11 latach - w 1954 roku, został mianowany dyrektorem [87]. Przez cały ten czas, głównym celem Taiichi Ōno było usprawnienie procesu produkcyjnego poprzez zwiększenie wydajności i efektywności pracy. W wyniku jego działań powstał innowacyjny system. Nazwano go *TPS*

<sup>1</sup> [www.japonski-pomocnik.pl](http://www.japonski-pomocnik.pl)

(Toyota Production System). Jego fundamentalnymi elementami są, ograniczanie marnotrawstwa, ciągła nauka oraz dążenie do doskonałości procesów i ludzi.



**Rysunek 1.2.** Taiichi Ōno

Źródło: [global.toyota/en](http://global.toyota/en)

Dzięki uważnym obserwacjom, Taiichi Ōno zidentyfikował oraz skategoryzował 7 rodzajów strat:

1. *przestój ludzi i maszyn*
2. *nadprodukcja*
3. *bezwartościowe procesy*
4. *zbędny transport*
5. *zbędny ruch*
6. *nadmierne magazynowanie*
7. *defekty wyrobów [88]*

Przestoje zarówno w pracy maszyn jak i pracowników wpływają negatywnie na wydajność i efektywność firmy. Inwestycje w postaci zakupu nowej maszyny czy zatrudnienia nowej osoby w wyniku przestojów mogłyby się nie zwrócić. Koszty związane z płaceniem za brak efektów są znaczną stratą. Nadprodukcja wiązała się z nadmiernym magazynowaniem, zarówno półproduktów jak i wyrobów końcowych. Zwiększało to koszty ich przechowywania, utrzymania odpowiedniej infrastruktury oraz wiązało się to z ryzykiem utraty ich wartości i jakości (np.: terminu ważności użycia). W dodatku, każdą z tych rzeczy należało przenieść z jednego miejsca do drugiego, co przyczyniło się do ich zbędnego transportu i ruchu pracowników. Każde przemieszczanie elementów wiązało się z ryzykiem ich uszkodzenia oraz wprowadzało logistyczny chaos, albowiem każdy transport należało zaplanować. Usterki i defekty często były odkrywane

na końcowym etapie produkcji lub dopiero u klienta. Wiazało się to z dodatkowymi kosztami związanymi z naprawami i reklamacjami [81]. Wszystkie te elementy, prowadziły do ogromnej straty czasu i pieniędzy.

Poznanie i zrozumienie tych problemów otworzyło możliwość opracowania adekwatnych rozwiązań i docelowo przyczyniło się to do ewolucji pracowników i procesów produkcyjnych.

### 1.2.3. Filary systemu TPS

Odpowiedzią na wyzwania, które codziennie trapiły fabryki Toyoty, było wprowadzenie wydajnego systemu produkcji *Just-In-Time* (dokładnie na czas). Podejście to zakładało ograniczenie magazynowania zapasów do minimum, a jednocześnie zachowanie ciągłego i efektywnego przepływu pracy w całym procesie [71]. Wprowadzenie tej metody wymagało, jednakże zastosowania odpowiednich narzędzi, które pozwoliłyby na kontrolę i efektywne zarządzanie systemem. W odkryciu tego rozwiązania, kluczową rolę odegrała podróż Taiichi Ōno do USA w 1956 roku. Właśnie tam zainspirował się lokalnymi supermarketami, w których to klienci sami wybierali produkty i zabierali je z półek. Zaimponował mu sposób w jaki właściciele sklepów kontrolowali ich stan. Regały te były uzupełniane w momencie, gdy zaczynało brakować wyrobów (Just-In-Time) [87]. Po powrocie z wyprawy, Taiichi Ōno wprowadził do systemu TPS karteczki z informacjami o zapotrzebowaniu na dany produkt. Nazwał je "*kanbanami*".



**Rysunek 1.3.** Magazyn kanban w zakładzie Motomachi

Źródło: [global.toyota/en](http://global.toyota/en)

Były one przyczepione do półek z produktami. W momencie jego pobrania, karteczka została zdjęta i przekazywana do innego działu, którego zadaniem było uzupełnić wskazane braki. Karty zapotrzebowania były w ciągłym ruchu przez cały proces produkcji, co zapewniało ciągły przepływ wyrobów [67]. "Podejście to reprezentuje system 'pull'. Oznacza to, że produkcja opiera się na zapotrzebowaniu klientów, a nie na standardowej praktyce 'push', polegającej na produkowaniu towarów i wypychaniu ich na rynek" [24].

Wprowadzanie kart sygnałowych "kanban" było przełomowym momentem. Spowodowało to polepszenie systemu *Just-In-Time*. Dzięki nim, zwiększyła się wydajność, przejrzystość całego procesu oraz efektywność pracy. Ograniczono również ilości zapasów i magazynowania gotowych wyrobów. Było to tak skuteczne rozwiązanie, że system ten został wdrożony we wszystkich zakładach Toyoty w 1963 roku, a do 1965 roku, zaczęto pozyskiwać w ten sposób produkty od dostawców [69].

Taiichi Ōno poradził sobie również z problemem defektów wyrobów. Wprowadził system *Jidoka* (自化) którego głównym założeniem było identyfikowanie i eliminowanie niedociągnięć oraz odstępstw od ustalonych norm. W momencie wykrycia nieprawidłowości, w każdym etapie procesu, operatorzy maszyn mogli zatrzymać całą linię produkcyjną aż do momentu usunięcia usterki. Dzięki temu zapobiegano dalszemu powstawaniu wadliwych wyrobów [70]. Usprawniono ten system poprzez wprowadzanie kolejnej wizualizacji. Na każdej maszynie umieszczono guzik, który po kliknięciu włączał sygnał świetlny. W ten o to sposób następowała sygnalizacja i informowanie przełożonych jak i innych pracowników o wystąpieniu problemu. Narzędzie to nazwano *Andon* (安灯) [68].



**Rysunek 1.4.** Andon (linia montażowa w fabryce Kamigo)

Źródło: [global.toyota/en](http://global.toyota/en)

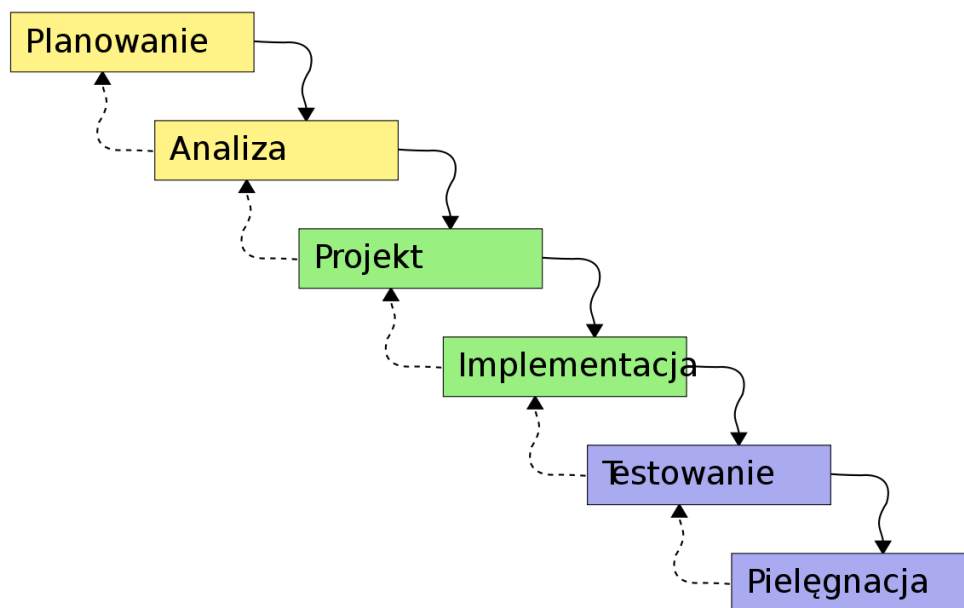
Wprowadzono go w 1966 roku w fabryce Kamigo. Wprowadzenie systemu *Jidoka*, pozwoliło na zwiększenie jakości wyrobów oraz wyeliminowanie strat związanych z wadliwymi produktami.

System *TPS* stał się jedną z głównych przyczyn światowego sukcesu Toyota Motor Corporation. Dzięki czemu, śmiało można nazwać go fundamentem, na którym wzorowano się w tworzeniu metodyki *Kanban* którą znamy dzisiaj.

#### 1.2.4. Przełom w wytwarzaniu oprogramowania

##### Model kaskadowy

Proces wytwarzania oprogramowania uległ standaryzacji. Przez dziesiątki lat był on prowadzony w sposób *kaskadowy*. Model ten został oficjalnie zaprezentowany w 1970 roku w artykule Winstona Royce'a [61]. Zakładał on podział całego procesu na odrębne fazy projektowe, które to należało wykonywać jedna po drugiej. Nie było możliwości przejścia do następnego etapu aż do momentu, w którym wcześniejszy nie został zakończony [90].



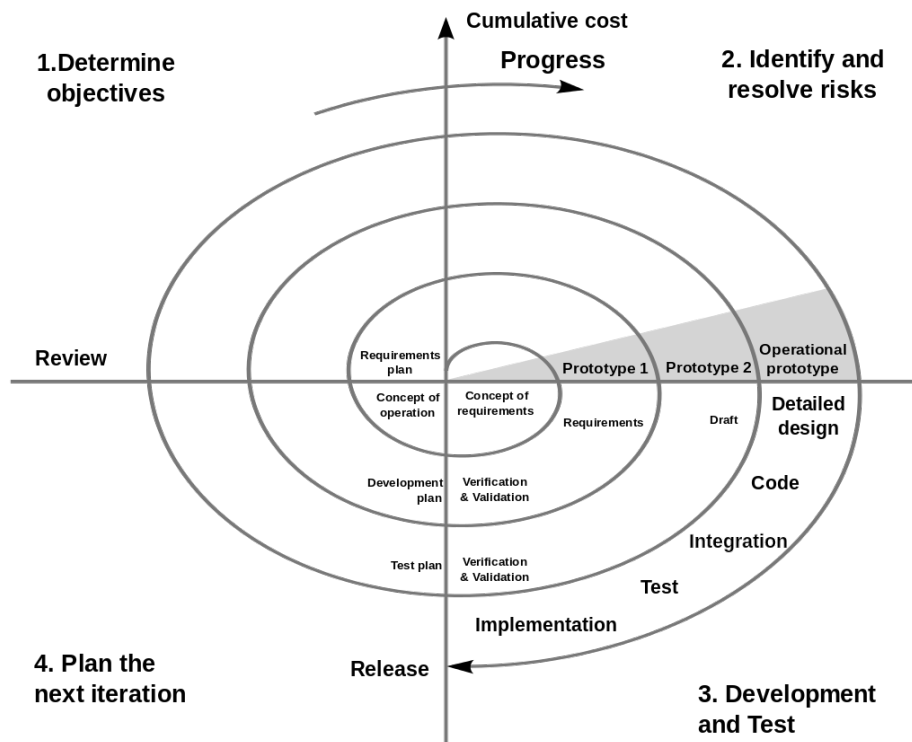
**Rysunek 1.5.** Fazy modelu kaskadowego  
Źródło: [pl.wikipedia.org/wiki/Model\\_kaskadowy](http://pl.wikipedia.org/wiki/Model_kaskadowy)

Wymagania funkcjonalne i pozafunkcjonalne były tworzone na samym początku. Wystąpienie nowych kryteriów, wymagało rozpoczęcia przechodzenia procesu od nowa. Podobny przypadek dotyczył się wykrycia błędów powstałych w trakcie realizacji poprzednich faz. Częstym problemem był również brak poczucia odpowiedzialności za cały system. Działania każdego zespołu, koncentrowały się na jak najszybszym przerzuceniu odpowiedzialności poprzez wypchnięcie swojej części i przekazanie jej do kolejnego działu.

Te wydarzenia powodowały, że proces był bardzo kosztowny oraz czasochłonny a wytwarzanie oprogramowania było bardzo trudne i nieefektywne. Wraz z rozwojem technologii i rosnącym doświadczeniem w działaniu w metodyce kaskadowej, rosła frustracja w branży. Zauważono powtarzające się problemy i niedociągnięcia wynikające z korzystania z tego podejścia. Kryzys ten wystąpił w wyniku wzrostu wielkości systemów, dokumentacji i ilości wymagań oraz ich zmienności w czasie. Zmieniły się również oczekiwania biznesu, który dążył do jak najszybszego otrzymania produktu końcowego. Cykl życia projektu się rozciągał [33]. Model kaskadowy nie był w stanie sprostać tym wyzwaniom.

#### Model spiralny

Odpowiedzią na powstały problem było powstanie nowych podejść do wytwarzania oprogramowania. W 1986 roku, Barry Boehm świadomy ograniczeń modelu kaskadowego, zaprezentował w swoim artykule model *spiralny* [21]. Był on pierwszym podejściem iteracyjnym. Zakładał on, że cały proces wytwarzania oprogramowania przyjmuje postać spirali. Każdy pełen obrót oznacza rozpoczęcie następnej iteracji procesu. Jeden taki cykl podzielony jest na 4 główne etapy: planowanie, analizę ryzyka, inżynierie i ewaluację. Przejście do następnego etapu oznaczało zakończenie prac w poprzednim. Tworzenie oprogramowania skupiało się na kreowaniu kolejnych prototypów. Dużą zaletą tego podejścia jest większa elastyczność i nacisk na analizę ryzyka. Umożliwia to dostosowanie się do zmieniających się wymagań systemowych oraz pozwala na dostarczenie bardziej dopasowanego i niezawodnego systemu [86].



**Rysunek 1.6.** 4 główne etapy modelu spiralnego

Źródło: [en.wikipedia.org/wiki/Spiral\\_mode](http://en.wikipedia.org/wiki/Spiral_mode)

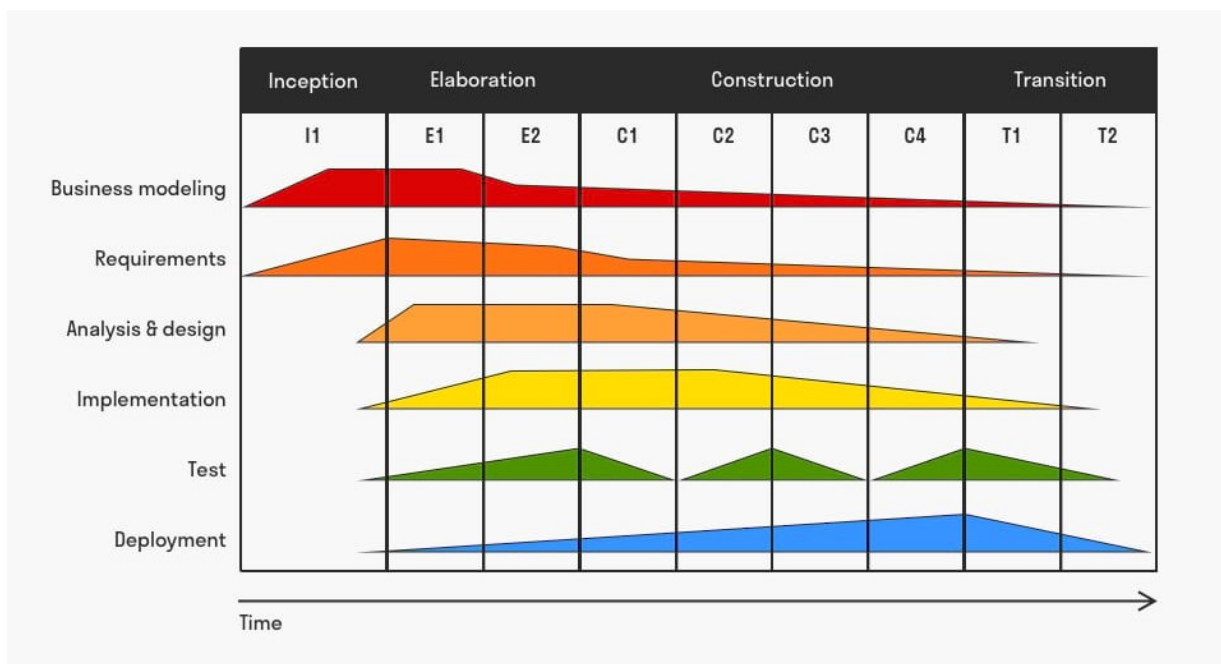
### Model RUP

Następnie w 1995 roku powstała wczesna wersja metodyki, obecnie rozpoznawanej jako *RUP*, czyli *Rational Unified Process*. Początkowo został opracowany przez firmę *Rational Software*, która została przejęta przez *IBM*. Twórcy i osoby rozwijające tę metodykę analizowali nieudane projekty, a dokładniej przyczyny ich niepowodzenia. Rezultatem tych badań było opracowanie skutecznych praktyk, które składały się na tę ustrukturyzowaną metodykę. *RUP* bazuje na modelu spiralnym. Jest to model iteracyjno-przyrostowy. Dzieli projekt na 4 fazy, które to składają się z iteracji. Każda z nich zakłada dostarczenie klientowi kolejnej części systemu. W każdej fazie wykonywano wszystkie niezbędne czynności, skupiano się jednak na zdefiniowanych głównych celach, które należało osiągnąć w trakcie jej trwania. Zostały one przedstawione na rysunku 1.7. Wyróżniamy:

1. fazę rozpoczęcia (skupiona na zbieraniu wymagań i planowaniu)
2. fazę opracowywania (skupiona na analizie i projektowaniu)
3. fazę konstrukcji (skupiona na implementacji)
4. fazę przekazania systemu (skupiona na testowaniu i wdrożeniu)



Wprowadzono w nim również nowe elementy, takie jak: reużywalne komponenty, przypadki użycia (sposób zbierania wymagań) i diagramy *UML* (sposób graficznej reprezentacji wymagań, funkcjonalności i architektury systemów) [85].



Rysunek 1.7. Fazy modelu RUP i czynności w nich wykonywane

Źródło: [www.toolshero.com](http://www.toolshero.com)

### Manifest programowania zwinnego

Odkrywamy nowe metody programowania dzięki praktyce w programowaniu i wspieraniu w nim innych. W wyniku naszej pracy, zaczęliśmy bardziej cenić:

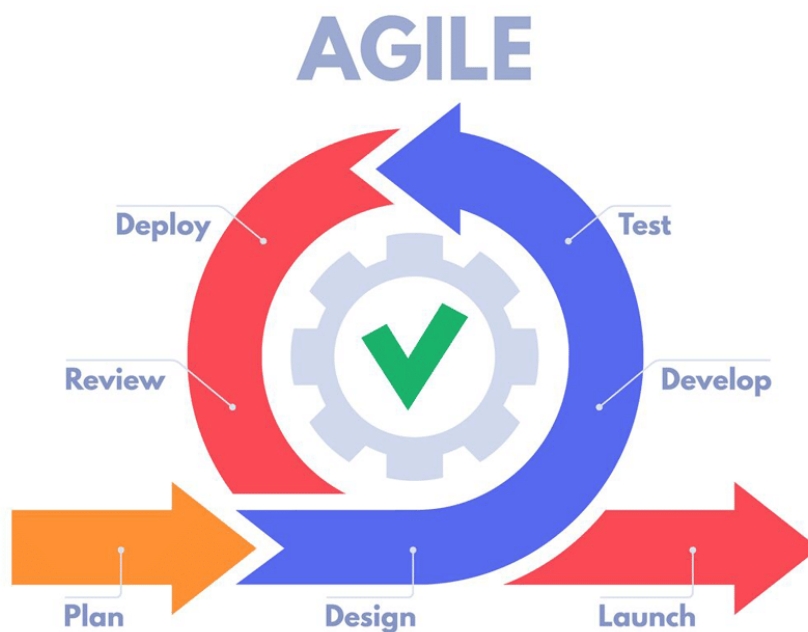
1. Ludzi i interakcje ponad procesy i narzędzia
2. Działające oprogramowanie ponad obszerną dokumentację
3. Współpraca z klientem ponad negocjowaniem umów
4. Reagowanie na zmiany ponad realizacją założonego planu

Oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej [5].

Pod koniec lat 90. XX wieku zapoczątkowano nowe podejście do wytwarzania systemów informatycznych. Był to przełom w historii inżynierii oprogramowania. Sposób ten docelowo nazwano *modelem Agile* (zwinnym). W porównaniu

do RUP i modelu *spiralnego*, nowe podejście było łatwiejsze do wdrożenia oraz oferowało znacznie większą elastyczność. Wymagało zdefiniowania mniejszej ilości ról i zasad oraz ograniczało dokumentację. Jego główną koncepcją jest iteracyjne i jak najszybsze dostarczenie klientom biznesowym wartościowej części produktu. Wynika to z chęci pozyskania informacji zwrotnej na temat systemu w celu jego udoskonalenia i sprecyzowania wymagań. System końcowy jest wynikiem kolejnych przyrostów. Rewolucją był fakt, że w każdym z nich równoległe występują wszystkie fazy wytwarzania oprogramowania. Zmieniono sposób podchodzenia do zmian. Zauważono, że są one nieuniknione i powodują one wzrost jakości dostarczanego produktu. Zaczęto je akceptować. Wprowadzono także idee ciągłego doskonalenia zespołu oraz procesu [38].

W 2001 roku został opracowany *manifest agile*. Ustanawia on uniwersalne standardy które stały się fundamentem dla zwinnego wytwarzania oprogramowania. Został sporządzony przez grupę cenionych przedstawicieli nowych metodyk kreowania systemów informatycznych. Grono to było znane jako "The Agile Alliance" [4]. Do zbioru najbardziej renomowanych i znanych technik, zalicza się *Scrum* oraz *programowanie ekstremalne*. Podejścia te znacząco zyskiwały na popularności. Stały się atrakcyjnymi i modnymi metodykami zarządzania projektami.



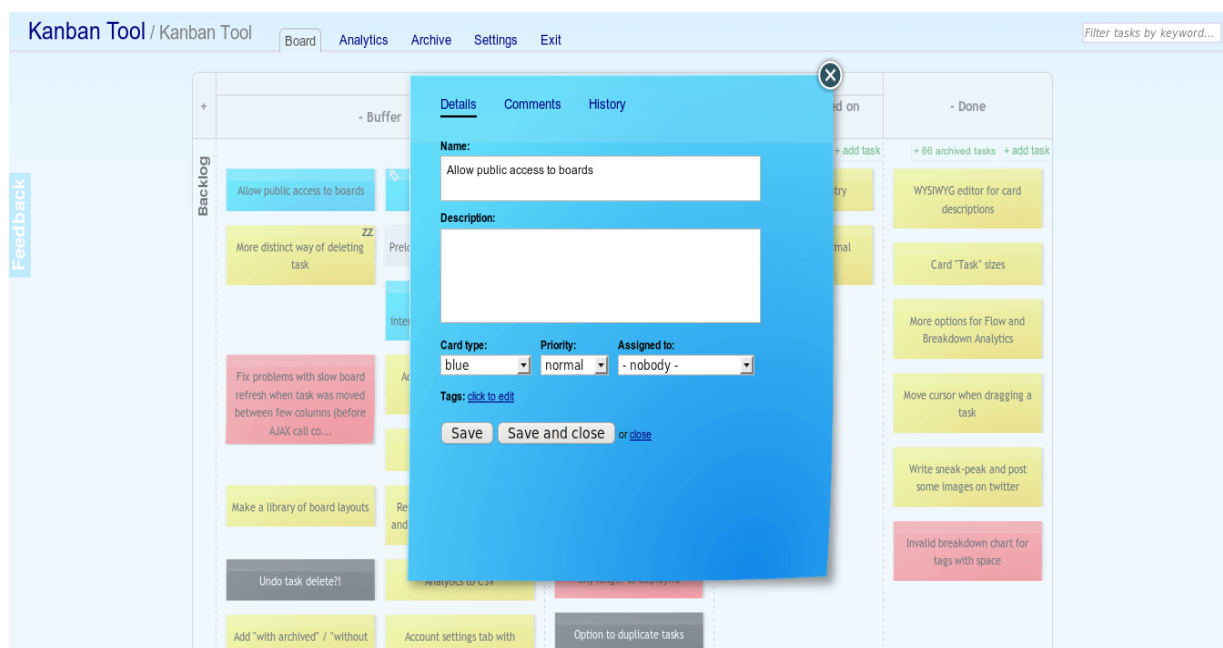
Rysunek 1.8. Pojedyncza iteracja w metodykach zwinnych

Źródło: [www.wimi-teamwork.com](http://www.wimi-teamwork.com)

### 1.2.5. Kanban jako metodyka Agile

Jako pomysłodawcę i pioniera metodyki Kanban w zarządzaniu projektami, wyznacza się Davida Andersona. Źródłem jego inspiracji były osiągnięcia Systemu Produkcyjnego Toyota (TPS). Przekształcił i dostosował jego główne zasady do specyfiki inżynierii oprogramowania [80]. Pierwsze wykorzystanie tego innowacyjnego podejścia, miało miejsce w 2004 roku, w firmie Microsoft. Projekt ten osiągnął ogromny sukces. Stanowiło to bodziec do dalszych prac Andersona nad rozwojem koncepcji i technik związanych z Kanbanem. Jego wysiłki przyczyniły się do zastosowania Kanban w nowych projektach w firmie Corbis w latach 2006-2008 [48].

Anderson postanowił szeroko dzielić się swoją wiedzą i spostrzeżeniami. Co docelowo doprowadziło go do założenia własnej firmy w 2008 roku. Jego następnym krokiem było zorganizowanie konferencji "Lean Kanban 2009" na której podzielił się z audiencją swoimi spostrzeżeniami. W tym samym roku ukazały się pierwsze programy do prowadzenia projektów z wykorzystaniem metodyki Kanban. Na przykład "Kanban Tool" [36].



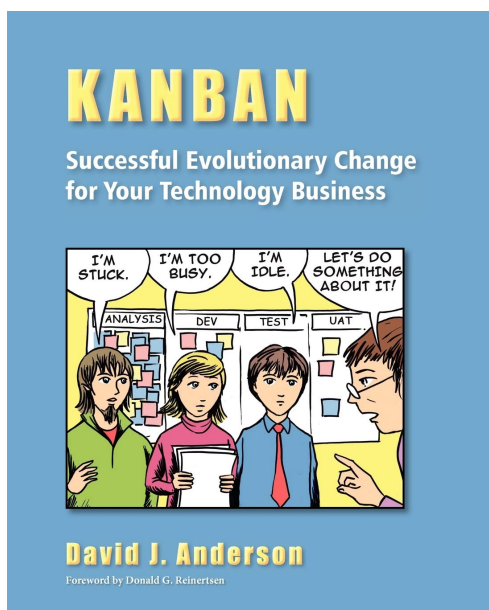
Rysunek 1.9. Wczesna wersja Kanban Tool 2009 rok

Źródło: *kanbantool.com*

Bazując na zdobytym doświadczeniu, Anderson w 2010 roku, "Opublikował książkę [6], która znalazła się w pierwszej piątce książek o zwinności, jakie

kiedykolwiek opublikowano" [7]. Opisał w niej swoje praktyki i koncepcje metodyki Kanban.

Z czasem Kanban zaczął nabierać popularności. Powstały na jego temat liczne wątki na forach społecznościowych. Został zauważony przez praktyków. Zaobserwowano jego zalety w zarządzaniu nieprzewidywalnymi procesami.



**Rysunek 1.10.** Okładka książki Davida Andersona:

"Kanban, Successful Evolutionary Change for Your Technology Business" 2010

Źródło: [www.amazon.pl](http://www.amazon.pl)

Zaczęto stosować go również w innych dziedzinach, które wymagały pracy intelektualnej i miały ustrukturyzowany proces. Należały do nich między innymi: HR, marketing i finanse [20]. W procesie wytwarzania oprogramowania, Kanban w wielu firmach stał się rozszerzeniem dotychczasowo stosowanych praktyk. Wprowadzono go jako uzupełnienie metodyk Scrum (*Scrumban*) i programowania ekstremalnego.

Opierając się na badaniach *State of Agile* z 2020 roku, sponsorowanych przez firmę *digital.ai*, można wywnioskować, że aż 15% firm wykorzystuje koncepcje metodyki Kanban w zarządzaniu projektami, z czego 9% pracuje w hybrydzie Scrum i Kanban, (*ScrumBan*) a 6% stosuje Kanban bez dodatkowych modyfikacji [1]. Ten wynik doskonale ilustruje jego popularność.

## 1.3. METODYKA KANBAN

### Zasady Kanban

Kanban stanowi zestaw dobrych praktyk, które można dostosować do ustrukturyzowanego procesu zachodzącego w projekcie lub firmie. Nie jest to ściśle narzucona metodyka ze sztywnymi regułami. To raczej podejście, które umożliwia stopniowe wprowadzanie zmian. Kluczowym aspektem jest ciągła adaptacja i doskonalenie, które postępują w tempie zespołu i organizacji. Jedną z głównych i kluczowych zasad Kanban, jest "*Zaczynij tam, gdzie jesteś*". Oznacza to że, można przyjąć tę metodykę bez wymogu wprowadzania nowych ról czy uciążliwych transformacji. Kanban, dzięki swojej elastyczności, stał się niezwykle przystępny i prosty do wdrożenia [8].

Głównym zadaniem Kanban, jest dostarczenie wartości biznesowej. Odbywa się to poprzez zapewnienie szybkiego przepływu pracy w ustalonym procesie. Jest to możliwe dzięki zastosowaniu sześciu głównych praktyk. Należą do nich:

1. wizualizacja
2. ograniczenie pracy w toku
3. zarządzanie przepływem
4. ustalenie jasnych zasad dotyczących pracy
5. implementowanie pętli opinii
6. poprawianie kolektywne, ewolucja eksperymentalna [36]

#### 1.3.1. Wizualizacja

W metodyce Kanban, wizualizacja polega na uwidocznieniu wszelkich niezbędnych i wartościowych danych na temat naszej pracy. Jest to kluczowy element. Zwiększa on zrozumienie przebiegu procesu oraz obrazuje szerszą perspektywę stanu projektu. Doprowadza to do zwiększania wydajności. Jak wykazano w artykule [56], ludzki mózg przetwarzana grafikę znacznie szybciej i efektywniej niż tekst. Należy jednak pamiętać, aby unikać przeładowania informacjami. Może to doprowadzić do utrudnienia zrozumienia i zniechęcenia do korzystania z tej techniki.

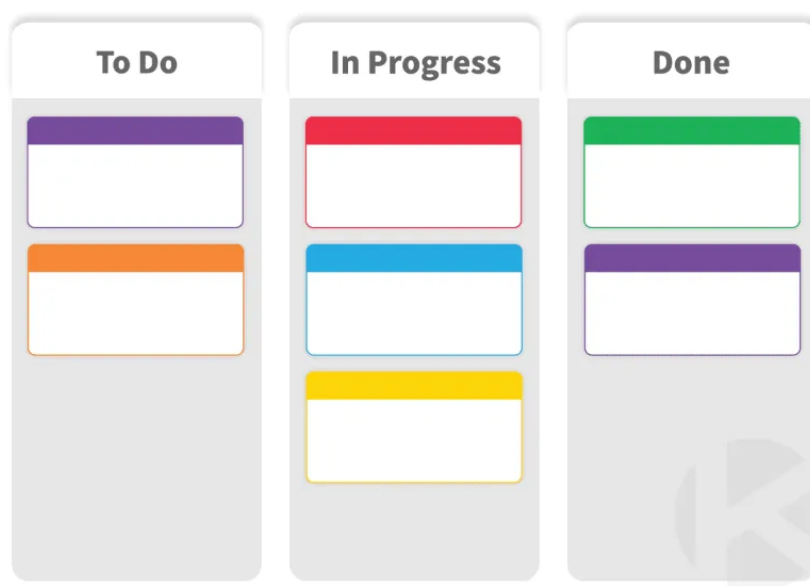
Ważne jest transparentne przedstawienie danych oraz wspólne ustalenie zrozumiałych i jasnych dla każdego zasad. Nie wolno niczego ukrywać. Jest to konieczne, aby móc rzetelnie usprawniać proces i zarządzać jego przepływem. Wizualizacja w metodyce Kanban odbywa się przy wykorzystaniu kilku prostych, ale potężnych narzędzi.

### **Tablica Kanban**

Fizyczna lub cyfrowa tablica dostępna dla całego zespołu. Jest graficznym przedstawieniem zmapowanego faktycznego przebiegu procesu. Wizualizuje aktualny postęp projektu. Należy ją na bieżąco aktualizować, aby każdy mógł podejmować słuszne decyzje. Zbudowana jest z kolumn, których nazwy reprezentują poszczególne fazy pracy. W wyniku ewolucji, jej struktura można jednak zmieniać się w czasie.

Pojedyncze zadania są reprezentowane przez karty Kanban. Gdy spełnią odpowiednie kryteria (wyznaczone i zaakceptowane przez zespół), przemieszcza się je między kolumnami. Ich ruch odzwierciedla postęp. Można również zauważyć ich blokadę lub wystąpienie wąskich gardeł (zatrzymania się większej ilości zadań w danym momencie).

Tablica Kanban pomaga śledzić natężenie pracy. Usprawnia komunikację i pomaga zrozumieć, co w danym momencie robią inni członkowie zespołu. Na jej podstawie dostrzegane i identyfikowane są elementy do poprawy. Ułatwia natychmiastowe wykrycie problemów w procesie.



**Rysunek 1.11.** Prosta tablica Kanban

Źródło: [kanbanzone.com/resources/kanban/what-is-kanban-board/](http://kanbanzone.com/resources/kanban/what-is-kanban-board/)

### **Karta Kanban**

Graficzna reprezentacja pojedynczego wykonywanego zadania. Znajdują się na niej użyteczne informacje, konieczne do jej poprawnego wykorzystania i podejmowania decyzji. Nie ma jasno zdefiniowanej struktury i zakresu danych

który powinien się na takiej karcie znaleźć. Zależy to od specyfiki projektu i indywidualnych potrzeb zespołu. Przykładowymi informacjami znajdującymi się na karcie Kanban są:

1. tytuł - ułatwiający identyfikację zadania i komunikację w zespole
2. opis - rzeczowy i dokładny, określający co ma zostać wykonane
3. typ pracy - pomocny do określenia priorytetów
4. osoby odpowiedzialne - do kogo kierować pytania, sugestie i komentarze
5. estymata - do określenia czasu potrzebnego do wykonania zadania
6. ważne terminy - pomagają ustalać pierwszeństwo zadania
7. blokada - w celu wyróżnienia wystąpienia problemu w przepływie
8. identyfikatory referencyjne - do powiązania z innymi systemami [36]

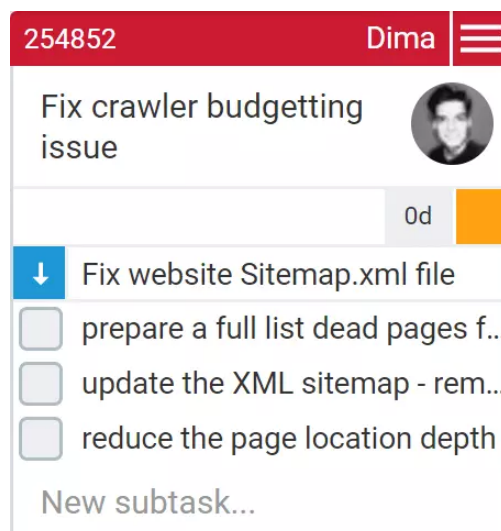
W celu ułatwienia wyszukiwania informacji, należy zawsze korzystać z tego samego, zdefiniowanego sposobu wizualizacji. Zawartość karty, należy nieustannie ulepszać i modyfikować.

### Klasy usług

Dzieli zadania na różne rodzaje. Klasyfikują je na podstawie ich charakterystyki i wymagań biznesowych. Definiują one jak traktować dane zadanie, jego priorytet, jak wygląda jego przepływ oraz jak dane zadanie wpływa na zasady stworzonego systemu. W celu ułatwienia identyfikacji, często są wizualizowane za pomocą odrębnych kolorów. Do powszechnych klas usług zdefiniowanych przez Davida Andersona, należą:

1. przyspieszone - największy priorytet
2. nienamagalne - bez dostrzegalnej wartości biznesowej
3. dtandardowe - typowe codzienne zadania
4. z datą - ustalony termin dostarczenia [6]

Podobnie jak z kartami Kanban, to zespół dopasowuje je do swoich potrzeb i określa jakie klasy usług są potrzebne i czym się one wyróżniają. Klasy usług ułatwiają samoorganizację zespołów w priorytetyzacji wybieranych zadań. Dzięki temu eliminowane są konflikty oraz minimalizowane są koszty zarządzania i koordynacji.



**Rysunek 1.12.** Elektroniczna karta Kanban

Źródło: [businessmap.io/kanban-resources/getting-started/what-is-kanban-card](https://businessmap.io/kanban-resources/getting-started/what-is-kanban-card)

### 1.3.2. Praca w toku

Ważnym elementem w metodyce Kanban jest zwiększenie wydajności i szybkości dostarczania wartości biznesowej poprzez ograniczenie pracy w toku. Jest to podyktowane nieefektywnością wielozadaniowości. Potwierdza to badanie Geraldna Weinberga, które dowiodło, że na przełączanie kontekstu między projektami programistycznymi, tracimy aż do 10% czasu [34].

Ograniczanie pracy w toku, powoduje, że zaczynamy pracować w systemie "pull". Polega on na tym, że karta zadania jest "pobierana" do kolejnego etapu tylko w momencie, w którym dostępne jest miejsce w następnej fazie. Zastosowanie tego podejścia, uwidacznia wąskie gardła w procesie.

#### Praca w toku

Inaczej: praca cząstkowa. (ang. work in progress - WIP) Jest to cała niezakończona praca w projekcie. Obejmuje ona wszystkie zadania, które nie zbudowały jeszcze wymaganej wartości biznesowej.

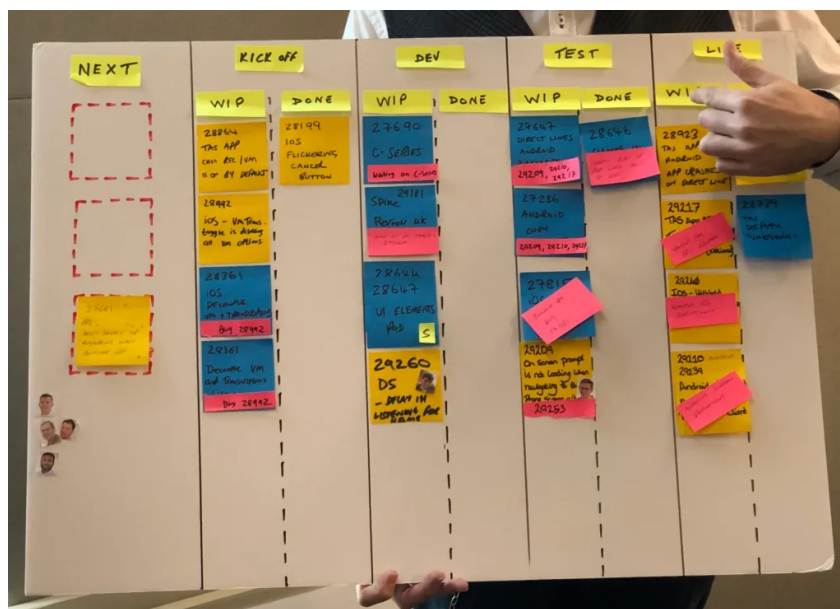
Duża liczba WIP może spowodować wystąpienie problemów związanych z przełączaniem kontekstu oraz z wydłużeniem czasu uzyskania informacji zwrotnej. Wpływa to na zwiększenie narzutu pracy, ryzyka oraz długu technicznego. Zwiększona ilość pracy w toku wymaga większego nakładu na koordynację i zarządzanie zadaniami. Aby uniknąć wymienionych negatywnych skutków, metodyka Kanban wprowadza limity pracy w toku.



### Limit WIP

Ograniczenie pracy w toku, pozwala na kontrolowanie przepływu pracy i pomaga w uniknięciu przeciążenia. Zwiększa wydajność, przyspiesza dostarczania wartości biznesowej i otrzymywanie informacji zwrotnych, które są kluczową częścią wszystkich metodyk Agile. Nie istnieje idealna wartość WIP. Należy wybrać konkretną liczbę a następnie ją modyfikować w celu poprawienia przepływu. W ramach Kanban, w zależności od decyzji zespołu, limity WIP są definiowane na poziomie, kolumn, klas usług, całej tablicy lub zadań na osobę. Wprowadzanie limitów, pozwala uwidocznic wąskie gardła. Najczęściej największe problemy pojawiają się na tablicy Kanban jako pierwsze.

Zgodnie z *prawem Johna Little'a*, możliwe jest zwiększenie efektywności wykonywania zadań poprzez minimalizację ilości pracy w toku. Relacja między WIP a czasem dostarczania ma swoje źródła w *teorii kolejowania*. Prawo to zostało sformułowane w 1961 roku. Mówi ono, że im więcej zadań wykonujemy jednocześnie, tym dłużej zajmie nam ich zakończenie. Częstym sloganem w metodyce Kanban w kontekście limitów WIP jest: "Przestań zaczynać, zacznij kończyć". Wiąże się to z 5. główną praktyką Kanban. Zespół powinien dążyć do jak najszybszego zakończenia zadania i dostarczenia klientowi efektów swojej pracy. Tworzymy w ten sposób krótką pętlę opinii. Pozwala to na szybkie wykrycie błędów i zwiększenie jakości dostarczanych funkcjonalności.



**Rysunek 1.13.** Fizyczna tablica Kanban

Źródło: [www.solutioneers.co.uk/what-is-a-kanban-board/](http://www.solutioneers.co.uk/what-is-a-kanban-board/)

### Prawo Little'a

"Średnia liczba rzeczy w systemie jest równa iloczynowi średniego czasu przebywania w systemie oraz średniego tempa ich przybywania. Prawo Little'a jest wyrażone wzorem:

$$L = \lambda \cdot W$$

Gdzie:

$L$  - średnia liczba rzeczy w systemie

$\lambda$  - średnie tempo przybywania rzeczy do systemu

$W$  - średni czas przebywania rzeczy w systemie [83]

### Przykład wykorzystania prawa Little'a

Po odpowiednim przekształceniach wyżej wymienionego wzoru, otrzymujemy:

$$\frac{L}{\lambda} = W$$

Dzięki temu, możemy wyliczyć średni czas wykonywania zadania. Zakładając, że zespół złożony jest z 4 osób, a średnia liczba zadań w systemie wynosi 8, to średni czas wykonywania zadania wynosi 2 jednostki czasu. ( $8/4 = 2$ )

Natomiast jeśli zmniejszymy liczbę zadań w systemie do 2 to średni czas wykonywania zadania wyniesie 0.5 jednostki czasu. ( $2/4 = 0.5$ ) Widać więc, że im mniej zadań w systemie, tym szybciej są one wykonywane. Zmniejsza się długość pętli informacji zwrotnej. Nasz system staje się bardziej elastyczny, a nasze dostarczane funkcjonalności są bardziej dopasowane do potrzeb klienta biznesowego.

### **1.3.3. Zarządzanie procesem**

Przepustowość procesu jest ograniczana przez wiele czynników. Spowalniają one cały system. Wpływ większości z nich jesteśmy w stanie naszym działaniem minimalizować. Każde wprowadzone usprawnienie dotyczące przepływu pracy, jest udoskonaleniem całego systemu.

### Przepływ i jego nieustanna poprawa

Stan, w jakim zadanie przechodzi przez proces. Składają się na niego przewidywalność, płynność oraz przepustowość.

Osiągnięcie efektywnego przepływu jest jednym z głównych celów Kanban. Jest to ciągle wyzwanie, którego należy się podejmować przez cały czas trwania projektu. Wymaga ono nieustannego monitorowania i optymalizacji. Jest to cel

każdego zespołu Kanban. Polega on na zarządzaniu przepływem w taki sposób, aby zadania zwinnie i bez przeszkód ( *płynnie*) przechodziły między etapami. Należy jednak uważać na nadmiernym skupieniu się na szybkości, gdyż może to skutkować znacznym obniżeniem jakości i poważnym zwiększeniem nakładu pracy w przyszłości. Powszechnymi praktykami do usprawnienia procesu są:

1. usuwanie blokad
2. zarządzanie wąskimi gardłami (manipulacja WIP)
3. dbanie o jakość
4. zmniejszenie wielkości zadań (do porównywalnych rozmiarów)
5. pomoc innym członkom zespołu w zakończeniu zadań
6. tworzenie zespołu wielofunkcyjnego (integracja unikalnego zestawu wiedzy i umiejętności w celu zwiększenia elastyczności i wydajności)

Powodem do wprowadzenia zmian mogą być sygnały od członków zespołu i problemy widoczne na tablicy Kanban. Ten temat jest często poruszany na spotkaniach członków zespołu. Dobrą praktyką jest podejmowanie decyzji opartych na danych i obserwacjach. Wymaga to zdefiniowania metryk i wskaźników wydajności oraz konsekwentne ich monitorowanie.

#### Trzymanie się wyznaczonych norm i zasad

Kanban jest bardzo elastyczną metodyką. Możemy ją dostosować do swoich potrzeb. Oznacza to, że każdy projekt może rządzić się innymi zasadami i normami. Szczególnie ważnym elementem dobrego zarządzania jest ich jasne określenie. Każdy członek zespołu powinien je znać i przestrzegać. W przeciwnym wypadku, może to doprowadzić do chaosu i nieefektywności.

#### Codzienny stand-up

Rzeczowe i krótkie spotkania (do 15 minut) na stojąco, odbywające się codziennie. Służą do omówienia aktualnego stanu tablicy Kanban. Dobrą praktyką jest przeprowadzanie ich regularnie w tym samym miejscu i czasie. W trakcie spotkania, członkowie zespołu skupiają się na zadaniach i przepływie. Zebranie zazwyczaj zaczyna się od omówienia zadań zablokowanych lub tych niezakończonych, znajdujących się jak najbliżej prawej części tablicy. Jest to wynikiem celu, którym jest jak najszybszego dostarczenie wartości biznesowej. Codzienny stand-up może również posłużyć jako forum do koordynacji zadań, przeanalizowania problemów i konfliktów oraz ich rozwiązywania.



**Rysunek 1.14.** Codzienny stand-up

Źródło: [jlzych.com/case-studies/optimizely-discovery-kanban/](http://jlzych.com/case-studies/optimizely-discovery-kanban/)

Zachowanie procesu i przepływu w dobrym stanie, wymaga planowania, podejmowania decyzji i szacowania. Powinny one odbyć się w optymalnym momencie, idealnie na czas (Just-In-Time). Przewidywanie z dużym wyprzedzeniem jest nieefektywne. Wiąże się z nadmierną pracą i ryzykiem wystąpienia licznych zmian. Zespół samoorganizujący się, wszakże musi znać aktualne priorytety oraz mieć kolejne, jasno określone zadania. Planowanie przyszłej pracy może być oparte na zdarzeniach. To one wskazują, kiedy rozpocząć konkretne działania. Taką technikę sygnalizującą można bezproblemowo wdrożyć za pomocą ustalenia pewnych zasad.

### **Rejest produktu (backlog)**

Backlog produktu to zgodnie z definicją przyjętą przez firmę *Atlassian* to "uporządkowana pod względem priorytetów, wynikająca z harmonogramu oraz wymagań lista prac dla zespołu. Najważniejsze elementy są widoczne na szczycie backlogu produktu, aby zespół wiedział, co powinien dostarczyć w pierwszej kolejności" [18]. Zdefiniowanie ich porządku, powinno być oparte na analizie szacowanej wielkości, ryzyka oraz celów biznesowych. Rejestr produktu, nie musi należeć do tablicy Kanban. Może być osobnym bytem z którego wybierane są do kolejki następne zadania do realizacji.

#### Punkt zamówienia

Określony punkt lub moment, który wywołuje konkretne wydarzenie w projekcie. Często stosowany do pobrania kolejnych zadań z *backlog'u*. (w momencie, gdy ilość prac w kolejce spadnie do określonego przez zespół poziomu, należy rozpocząć planowanie kolejnych zadań.) Może również skutkować organizacją specjalnych spotkań, na przykład w przypadku wprowadzenia nowej wersji systemu lub w rezultacie napotkanych trudności – na potrzeby retrospekcji.

#### Szacowanie wielkości zadań

Estymacja ilości czasu i wysiłku potrzebnego do zrealizowania zadania. W metodyce Kanban, szacowanie jest opcjonalne. Przedstawiane jest jako liczba punktów lub rozmiar. Miary te są relatywnymi szacunkami. Wynikają z porównywania zadań. Często są wykorzystywane liczby z ciągu Fibonacciego (1, 2, 3, 5, 8, 13, 21) lub rozmiary koszulek. (XS, S, M, L, XL)

### **1.3.4. Ewolucja procesu**

Kanban jest bardzo elastyczną metodyką. To zespół definiuje znaczną część zasad i praktyk. Ustalone reguły powinny się jednak zmieniać. Standard musi być poddawany ciągłej dyskusji. Kanban wizualizuje powody do ulepszeń. Należy to wykorzystać. Okazją do rozmowy na ten temat, są specjalne spotkania zwane retrospektywami. W celu podejmowania rzetelnych i przemyślanych decyzji związanych z ewolucją procesu, należy dokonywać ich na podstawie zebranych danych i wskaźników. Zbiór skompletowanych informacji warto poddać wizualizacji i utworzyć z nich wykres, co ułatwi ich analizę.

#### Retrospektywa

Stanowi szóstą główną praktykę Kanban. Polega na kolektywnym usprawnieniu procesu i sposobu pracy. Dokonuje się to za pomocą ewolucji eksperymentalnej. Retrospektywa jest specjalnym spotkaniem zespołu Kanban, mającym na celu doskonalenie metodyki, poprzez przeanalizowanie procesu i udzielenie konstruktywnej krytyki na jego temat. Znajdywane na nim są sfery, które wymagają poprawy. Ustalane są usprawnienia, które zostaną wdrożone, aby przynieść korzyść całemu systemowi. Mają one postać eksperymentalną. Należy wprowadzić obiektywną metodę oceny wprowadzonych zmian. Ułatwia to analizę i dalsze podejmowanie decyzji. Retrospektywa może być organizowane w wyniku zajścia jakiegoś wydarzenia (np.: wydania nowej wersji) lub w stałych interwałach czasowych.

### Wskaźniki wydajności

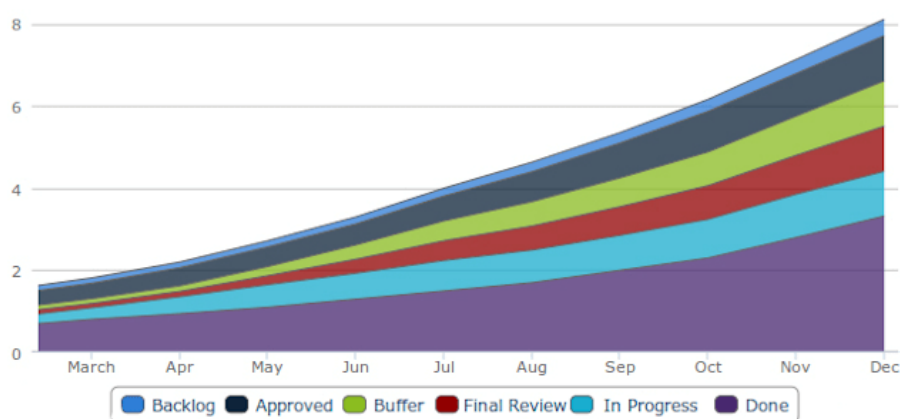
Pozwalają na ocenę efektywności i wydajności procesu. Można je obliczyć na podstawie zebranych danych. W metodyce Kanban często można się spotkać z takimi wskaźnikami jak:

1. czas cyklu - obejmuje tylko rzeczywisty czas pracy w określonych fazach
2. czas dostarczania - czas potrzebny do przejścia całego procesu
3. przepustowość - liczba zakończonych zadań w określonym czasie
4. liczba blokad

### Wykres przepływu skumulowanego

Składa się z wielu cennych danych ułatwiających analizę przepływu pracy. Jest prosty w interpretacji. Na wykresie, nagłe skoki odzwierciedlają obecność wąskich gardeł, a płaskie odcinki oznaczają brak aktywności w procesie. Z kolei płynny, równomierny wzrost, wskazuje na stabilny i zrównoważony postęp prac. Utworzenie go wymaga rzetelnego i systematycznego zbierania codziennie danych na temat liczby WIP i statusu każdego pojedynczego zadania. Odpowiadające sobie wartości są sumowane (kumulowane) i umieszczane na diagramie. Można z niego wyczytać relacje pomiędzy określonymi wskaźnikami oraz informacje takie jak:

1. rozmiar rejestru produktu
2. czas dostarczania
3. czas cyklu
4. całkowity poziom WIP



**Rysunek 1.15.** Diagram przepływu skumulowanego

Źródło: [kanbantool.com/kanban-guide/kanban-fundamentals/manage-and-measure-flow](http://kanbantool.com/kanban-guide/kanban-fundamentals/manage-and-measure-flow)

## 1.4. ZASTOSOWANIE METODYKI KANBAN W PROJEKCIE BRAINBOOST

### 1.4.1. Wybór metodyki

Po dokładnej analizie i dyskusji, zespół *Brainboost* postanowił przyjąć metodykę Kanban jako główną strategię pracy. Rozważany był również Scrum, jednakże podjęta decyzja wynika z kilku kluczowych powodów. Pierwszym z nich, jest preferencja każdego z członków zespołu do ciągłego przepływu pracy, w przeciwieństwie do cykli iteracyjnych, które są charakterystyczne dla Scrum'a. Dodatkowo, zespołowi zależało, aby uniknąć wprowadzania zbyt wielu formalności i zasad związanych ze Scrum'em, które mogłyby znacząco zabierać czas na ich realizację i potencjalnie ograniczać elastyczność projektu. Głównym postawionym celem była maksymalizacja czasu poświęconego na dostarczenie wartości biznesowej, dlatego też zostało wybrane mniej rygorystyczne podejście, jakie oferuje Kanban.

### 1.4.2. Organizacja pracy zespołu projektowego

Od początku aż do końca projektu, skład pozostał niezmienny. Zespół wyszedł z założenia, że prace programistyczne nad daną funkcjonalnością, będą wykonywane przez każdego holistycznie. Dlatego, wszyscy postanowili programować zarówno backend jak i frontend aplikacji. Pomimo, że Kanban nie wymaga wprowadzania ról w zespole, postanowiono, że każdy z członków będzie miał swoją własną powinność. Każda z nich cechowała się konkretnym obszarem odpowiedzialności. Zgodnie z pełnioną funkcją, każdy opiekował się innym aspektem projektu:

#### 1. Dawid Gostyński - DevOps

- automatyzacja procesów
- konfiguracja środowisk i deploymentów (infrastruktura + aplikacje)
- zarządzanie repozytorium

#### 2. Michał Nieruchalski - główny developer

- wspieranie zespołu w kwestiach programistycznych
- wybór języków i architektury

#### 3. Krystian Osiński - manager projektu

- zarządzanie systemem Kanban
- kontakt z klientem
- materiały projektowe: dokumentacje, prezentacje, prototypy graficzne

Klient zespołu *Brainboost* (przedstawiciel firmy *Skyblue Education*<sup>2</sup>) został wdrożony w metodykę Kanban. Krótkie szkolenie odbyło się na platformie *Discord*<sup>3</sup>. Zostało ono zorganizowane w celu zapoznania klienta z charakterystyką pracy zespołu. Dzięki temu, została nawiązana owocna współpraca na znanych wszystkich warunkach. Na początku pierwszego semestru prac, spotkania mające na celu zbieranie wymagań odbywały się 2 razy na tydzień. Miały one formę wideokonferencji. W pozostałym czasie trwania projektu, spotkania były przeprowadzane co 2 tygodnie. W ich trakcie prezentowane były dotychczasowe postępy prac oraz były uszczegóławiane dalsze wymagania jak i wymogi ich zmian. Okazyjnie, w wyniku aktualnego zapotrzebowania, spotkania były realizowane częściej, co stanowiło realizację zasady *Just-In-Time*.

### 1.4.3. Wizualizacja procesu

Wynikiem rozproszenia członków zespołu oraz konieczności efektywnej komunikacji, była decyzja o zastosowaniu tablicy elektronicznej. Na początku projektu zespół wybrał narzędzie *Trello*<sup>4</sup>, jednak w miarę rozwoju systemu i analizy potrzeb związanych z wymaganiami oceniania projektu, potrzebowano bardziej rozbudowanego narzędzia. Po wspólnej dyskusji, postanowiono przenieść całą dotychczasową zawartość do *Jiry*<sup>5</sup>.

Decyzja ta pozwoliła na lepsze dostosowanie procesu do potrzeb zespołu w wyniku rozbudowanych możliwości nowego narzędzia. Istotnym aspektem, który przyczynił się do tej zmiany, była zdolność do automatycznego generowania wykresów przepływu skumulowanego. Diagram ten miał kluczowe znaczenie zarówno dla monitorowania postępu projektu, jak i skutecznego zarządzania pracą, a także był wymagany w ramach przedmiotu studiów.

#### Tablica Kanban

Został zdefiniowany proces, który następnie został zmapowany na tablicę Kanban. Została ona podzielona na 5 kolumn:

1. To-Do - kolejka zadań
2. In Progress - zadania w trakcie realizacji
3. Code Review - zadania wykonane, oczekujące oceny kodu
4. Completed - zakończone zadania
5. Discontinuing - zadania porzucone

---

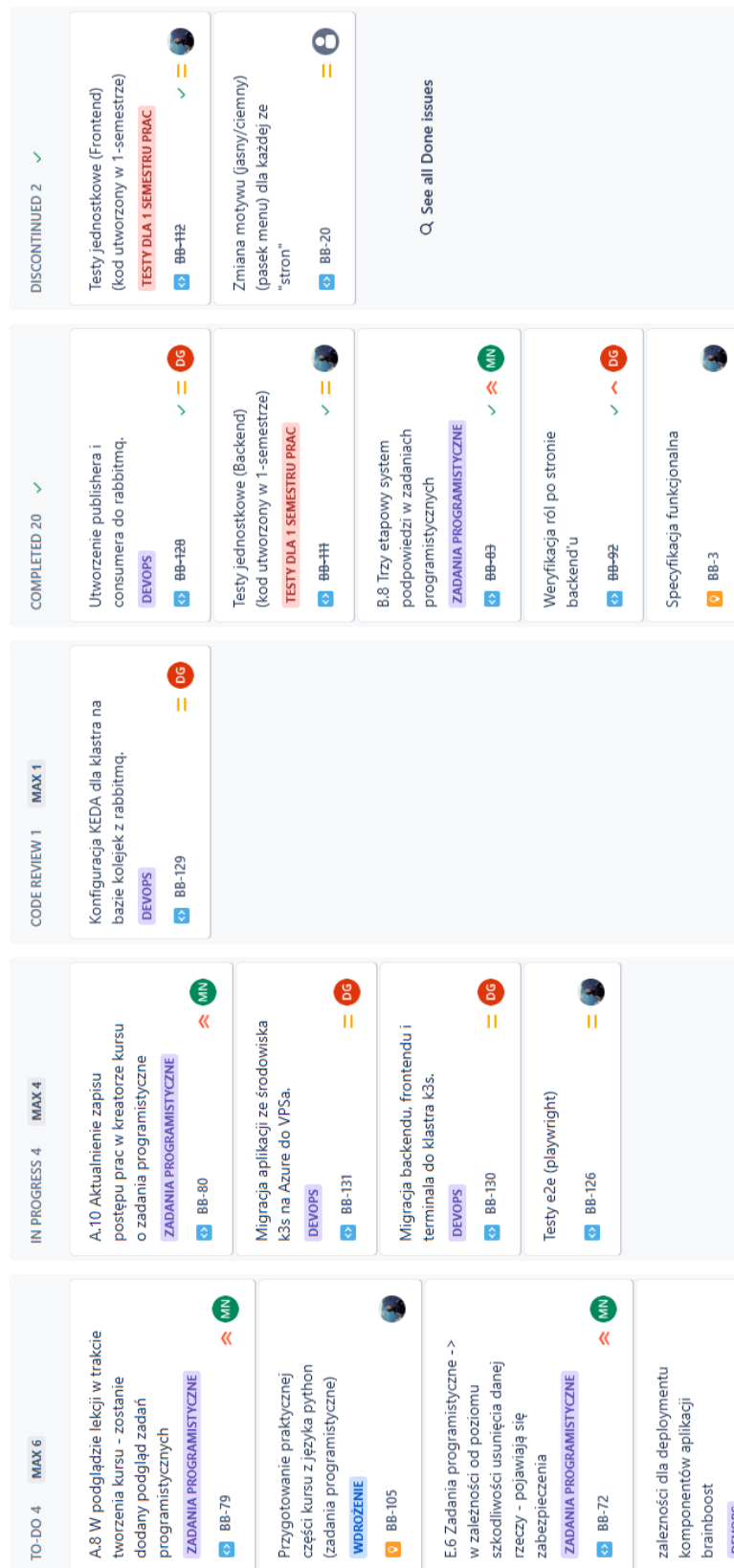
<sup>2</sup> [www.skyblue.education](http://www.skyblue.education)

<sup>3</sup> [www.discord.com](http://www.discord.com)

<sup>4</sup> [www.trello.com](http://www.trello.com)

<sup>5</sup> [www.atlassian.com/pl/software/jira](http://www.atlassian.com/pl/software/jira)



Rysunek 1.16. Tablica Kanban projektu *Brainboost*

▼ Backlog (31 issues)

0310

BB-72 E.6 Zadania programistyczne -> w zależności od poziomu szkodliwości usunięcia danej rzeczy - pojawiają się zabezpieczenia	ZADANIA PROGRAMIS...	TO-DO ▼	👤
BB-61 A.3 Podobny styl zadań programistycznych dla danego typu kategorii kursu	ZADANIA PROGRAMIS...	TO-DO ▼	👤
BB-85 F.2 Zapobiegnięcie awarii/spowolnienia systemu gdy uczeń wprowadzi szkodliwy kod do zadania programistycznego	ZADANIA PROGRAMIS...	TO-DO ▼	👤
BB-81 B.2 Przechodzenie pomiędzy zadaniami programistycznymi w formie zakładek	ZADANIA PROGRAMIS...	TO-DO ▼	👤
BB-37 B.6 Manualne oznaczenie lekcji jako ukończonej (przez ucznia)	DODATKI DO LEKCJI	TO-DO ▼	👤
BB-36 B.5 Pasek postępu na pasku menu w trakcie przechodzenia kursu / graficzne ukazanie ukończonych lekcji w menu nawigacyjnym	DODATKI DO LEKCJI	TO-DO ▼	👤
BB-35 B.4 Po kliknięciu przez ucznia przycisku - kontynuuj kursu - zostanie on automatycznie przekierowany do ostatniej nie przerobionej ...	DODATKI DO LEKCJI	TO-DO ▼	👤
BB-69 E.1 Strona umożliwia dokonanie płatności za kurs lub model subskrypcyjny	PŁATNOŚCI	TO-DO ▼	👤
BB-70 E.2 Automatyzacja płatności, tworzenia konta i udostępniania go użytkownikowi	PŁATNOŚCI	TO-DO ▼	👤
BB-60 B.9 Możliwość Dodawania wątków/komentarzy do lekcji przez uczniów i nauczycieli	KOMENTARZE	TO-DO ▼	👤
BB-95 B.10 Przed wstawieniem komentarza następuje jego automatyczna weryfikacja	KOMENTARZE	TO-DO ▼	👤
BB-62 B.2 Dodanie do paska nawigacji działającego przechodzenia do sekcji komentarzy w widoku lekcji	KOMENTARZE	TO-DO ▼	👤
BB-96 D.1 Każdy pracownik w pasku menu ma ukazaną ilość nowych komentarzy.	KOMENTARZE	TO-DO ▼	👤

Rysunek 1.17. Backlog produktu

Zespół postanowił, że limit WIP zostanie określony tylko dla poszczególnych kolumn w tablicy Kanban. Ograniczone zostały pierwsze trzy kolumny. Kolumna *To-Do* otrzymała limit 6. Przyjęta liczba pozwalała na samoorganizację członków zespołu oraz dawała wizję bliskiej przyszłości projektu. Przenoszone były do niej zadania z *backlog'u* w momencie osiągnięcia *punktu zamówienia*, którym była liczba: 3. Rejestr produktu był posortowany zgodnie z priorytetem zadań. Wybór i omawianie pracy następowały w trakcie cyklicznych spotkań członków zespołu.

Kolejna faza: *In Progress*, dostała limit 4. To ograniczenie było podyktowane ilością członków zespołu z uwzględnieniem, że niektóre zadania mogą zostać zablokowane lub mogą wymagać podjęcia dodatkowej pracy w ramach innego. Znajdowały się w niej zadania, wobec których została podjęta jakakolwiek praca.

Następny etap: *Code Review*, miał limit 1. Ustanowienie go miało na celu przyspieszenie dostarczenia wartości biznesowej i zmniejszenie długości pętli informacji zwrotnej. Tak małe ograniczenie wyniknęło z trafnych spostrzeżeń w trakcie trwania projektu.

W pierwszym semestrze, istniały również kolumny: "ready for code review" oraz "rejected". W wyniku ewolucji procesu zostały usunięte.

### Przepływ zadań

W narzędziu elektronicznym Jira nie zostały zdefiniowane reguły dotyczące przenoszenia zadań pomiędzy kolumnami tablicy. Zespół jednak jasno zdefiniował sprecyzowane zasady. Przepływ zadań był uwarunkowany klasą usług zadania. Zostały wyróżnione:

1. project management
2. task (zadania programistyczne / DevOps)
3. bug (naprawa defektów w oprogramowaniu)

 Task

 Bug

 Project Management

**Rysunek 1.18.** Ikony reprezentujące klasy usług

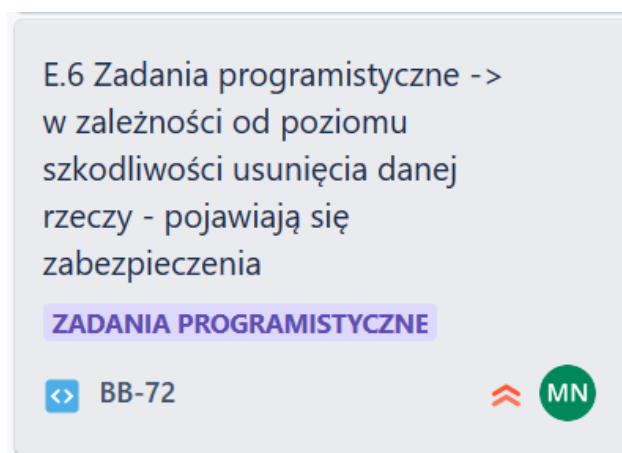
Klasy "task" i "bug" były przeznaczone dla wszystkich zadań, które wymagały oceny kodu. Taka praca przepływała przez tablicę kolumna po kolumnie, aż do etapu "completed". Typ "project management" był swobodnym określeniem pracy pokroju zbierania wymagań, tworzenia dokumentacji, prezentacji czy zarządzania procesem. Takie zadania pomijały etap "code review".

Wszystkie zadania mogły z każdego etapu trafić do fazy: "discontiuend". (z wyłączeniem zadań ukończonych). Każda z tych klas nie miała wpływu na priorytet zadania.

### Karty zadań

Dzięki korzystaniu z elektronicznego narzędzia do zarządzania projektami, pojedyncza karta Kanban zawierała w sobie wiele kluczowych informacji. Podstawowy widok obejmował tylko najważniejsze informacje. W projekcie *Brainboost* należą do nich:

1. tytuł karty
2. identyfikatory referencyjne
3. ogólna kategoria zadania
4. klasa usług
5. priorytet zadania
6. awatar osoby zajmującej się zadaniem



**Rysunek 1.19.** Karta Kanban - widok podstawowy

Priorytet karty był wyznaczany na podstawie analizy ryzyka oraz głównych celów klienta. Został przedstawiany jako ikona składająca się z linii skierowanych zgodnie z "poziomem ważności". Funkcjonalności wymagane oraz obszary krytyczne dla oceny projektu otrzymały najwyższy priorytet (strzałki w górę

w kolorze pomarańczowym). Zadania, które przybliżały zespół do spełnienia większej ilości kryteriów przedmiotu dostawały średni priorytet (żółte poziome linie), a dodatkowe funkcjonalności, niski. (niebieskie strzałki skierowane w dół)

Jeśli tytuł zadania pokrywał się z kluczowymi i wymaganymi funkcjonalnościami, to otrzymywał on identyfikator referencyjny, który odwoływał się do konkretnego numeru w dokumencie wymagań funkcjonalnych. (E.6)

Dodatkowy identyfikator referencyjny automatycznie nadawany przez narzędzie Jira (np.: BB-72) był wykorzystywany do mapowania zadania w systemie kontroli wersji *Git*. Numer ten służył do nazywania poszczególnych rozgałęzień w repozytorium. Dzięki temu, można było łatwo odnaleźć kod źródłowy, który był odpowiedzialny za daną funkcjonalność.

Zgodnie z przekonaniem ruchu "*#NoEstimates*" [23], zespół nie szacował rozmiaru pracy czy czasu potrzebnego do wykonania zadania.

The screenshot shows a Jira issue page for 'E.6 Zadania programistyczne -> w zależności od poziomu szkodliwości usunięcia danej rzeczy - pojawiają się zabezpieczenia'. The issue is assigned to Michał Nieruchalski (MN) and has a 'Highest' priority. The description includes a link to 'Wymaganie E.6 link'. The activity section shows a comment from Krystian Osiński dated 8 November 2023 at 17:57, stating: 'Zgodnie z decyzją klienta: należy wybrać "średnie" zabezpieczenie.' The page also features a 'To-Do' dropdown, 'Actions' dropdown, and a 'Details' sidebar with 'Assignee' and 'Created' information.

Rysunek 1.20. Karta Kanban - widok rozszerzony

Istnieje możliwość uzyskania szczegółów karty. Po jej kliknięciu, pojawia się widok z rozszerzonym zakresem informacji. Przydatnymi i często wykorzystywanymi polami był opis zadania oraz komentarze. Umieszczano w nich dodatkowe dane, zdjęcia i linki, pomocne w zrealizowaniu danego zadania. Dzięki temu ograniczono czas poświęcony na tłumaczenie wymagań. Umożliwiło to samoorganizację zespołu. Umieszczone linki kierowały zainteresowanych do odpowiednich miejsc w dokumencie wymagań projektowych. To właśnie w nim znajdowały się szczegółowe informacje na temat funkcjonalności. Były one umieszczane zgodnie z przyjętym przez zespół szablonem.

Numer wymagania	<b>ID wymagania np: X.1</b>
Tytuł wymagania	<b>Tytuł wymagania użytkownika</b>
<b>Kolor obwodu - informuje o priorytecie wymagania</b>	
Kamień milowy	(I) / (II) / (III) / (IV) / (V) - sposób zapisu ułatwia wyszukiwanie wymagań o podanym kamieniu milowym w dokumencie
Priorytet implementacji	<b>Niski</b> (oczekiwane) / <b>Średni</b> lub <b>Wysoki</b> (Wymagane)
Szczegóły	Szczegółowy opis - wymagania systemowe.  Jeśli dane wymaganie użytkownika będzie wymagało pracy w więcej niż jednym kamieniu milowym to, wymagania szczegółowe wewnątrz tej komórki tabeli będą rozdzielone na podstawie kamienia milowego w którym mają zostać zaimplementowane.
Materiały pomocnicze	powiązane wymagania / linki / grafiki / modele /diagramy UML

Rysunek 1.21. Szablon wymagania funkcjonalnego

Numer wymagania	<b>E.6</b>
Tytuł wymagania	<b>W zależności od poziomu szkodliwości usunięcia danej rzeczy – powinny pojawić się coraz to mocniejsze zabezpieczenia.</b>
Kamień milowy	(II) / (III) / (IV)
Priorytet implementacji	<b>Wysoki</b>
Ogólna koncepcja	Aplikacja powinna chronić użytkownika przed przypadkowym usunięciem danego elementu. Utrata niektórych rzeczy, może nieść za sobą duże konsekwencje - z racji tego wymagany jest zróżnicowany system zabezpieczeń. Taki system który nie będzie utrudniał pracy - dlatego klient nie wymaga zabezpieczeń przed usunięciem mało kluczowych elementów.
Szczegóły	<p>1. W zależności od poziomu szkodliwości usunięcia/zmiany danej rzeczy - mają być coraz to mocniejsze zabezpieczenia. Wyróżnić można:</p> <ul style="list-style-type: none"> <li>a) <b>Brak lub niewielka szkodliwość</b> - brak zabezpieczeń przed usunięciem. np.: usunięcie pustej lekcji, pustego zadania</li> <li>b) <b>Średnia szkodliwość</b> - małe zabezpieczenie np.: Usunięcie konta pracownika, usunięcie komentarza, usunięcie zadania (Można użyć okna dialogowego typu confirm)</li> <li>c) <b>Duża szkodliwość</b> - mocne zabezpieczenie np.: usunięcie modułu, usunięcie kursu. ( Można użyć do tego np.: okna dialogowego typu prompt (wymagane wpisanie jakiegoś randomowo wygenerowanego przykładu tekstu)</li> </ul>
Materiały pomocnicze	BRAK

Rysunek 1.22. Wymaganie E.6 w dokumencie wymagań funkcjonalnych

## 1.5. ZARZĄDZANIE PROCESEM I JEGO EWOLUCJA

### 1.5.1. Cykliczne spotkania członków zespołu

W wyniku ograniczonej ilości czasu przeznaczanego na realizację projektu, zespół postanowił, że spotkania będą odbywać się w formie wideokonferencji. Wykorzystywanym do tego narzędziem była aplikacja *Microsoft Teams*. Zostały ustalone konkretne dni i godziny, w których zespół miał odbyć 15 minutową rozmowę na temat aktualnego stanu projektu. W przypadku rzadkich sytuacji kryzysowych była ona przedłużana do 30 minut. Spotkania te z założenia odbywały się w poniedziałki i czwartki o godzinie 20:00.

Ustalona została zasada, że w wyniku wydarzeń losowych (jeśli chociaż jedna osoba nie będzie mogła wziąć udziału w danym spotkaniu) – spotkanie zostanie przełożone na najbliższy możliwy termin odpowiadający całemu zespołowi projektowemu.

Na spotkaniach odbywał się tak zwany "obchód" tablicy Kanban. Rozmawiano na temat zadań, które znajdują się najbliżej ukończenia procesu. Analizowano aktualne trudności i zadania zablokowane. Jeśli w kolumnie liczba zadań osiągnęła punkt graniczny, omawiane były kolejne zadania z *backlog'u* które następnie trafiały do kolejki zadań (kolumna *To-Do*). W trakcie spotkań, zespół dyskutował również na temat ewentualnych zmian w procesie.


### 1.5.2. Retrospektywa

Miały miejsce łącznie dwa spotkania retrospektywne, które podobnie do tych cyklicznych, odbywały się w formie wideokonferencji. Polegały one na wypisaniu elementów które się podobały, których brakowało oraz tych które nam nie odpowiadały. Zostały one przeprowadzane na samym początku drugiego semestru oraz w jego połowie. Podczas tych spotkań członkowie zespołu skupiali się na przeglądzie procesu projektowego. Wspólnie identyfikowali problemy, które pojawiły się w trakcie realizacji projektu i poszukiwali rozwiązań, mających na celu jego usprawnienie. Wszystkie wnioski i ustalenia były starannie dokumentowane, aby móc wykorzystać je w przyszłości i wdrożyć w celu ewolucji procesu.



Co się podobało	Czego brakowało	Co się nie podobało
Podobała mi się nasza praca nad dostarczeniem produktu pod presją czasu (przed kluczowymi datami w projekcie)	Brakowało mi częstszej komunikacji kto jest zablokowany "przez życie" (tzn. nie może pracować nad projektem przez np.: spotkania rodzinne)	Wyolbrzymiony stres przed prezentacją projektu
Sprawne reagowanie na zmiany: podobało mi się jak Dawid zrobił deployment apki o północy przed prezentacją przed komisją.	Brakuje nam cały czas deploymentu, miejsca gdzie nasza apka będzie cały czas wystawiona - nie mamy też na czym stawiać tej apki.	nie podobał mi się przestój jaki był z code review i mergowaniem zmian (wąskie gardło!)
Podobały mi się klarownie zdefiniowane role w projekcie. Wiadomo kto jest za co odpowiedzialny i kto jest głową danego sektora. Do kogo się zwrócić w wyniku jakiś problemów czy niejasności	frustrację z powodu niskiej liczby punktów przyznawanych za działania związane z DevOps w porównaniu do zadań programistycznych.	Nie podoba mi się że nie wiem po kolei które ficzery dowozić i na czym się skupić, tj. brakuje mi prioryteyzacji zadań (choć zawsze wystarczy napisać do Krystiana)
dobra organizacja, wszystko na czas robione, duża zasługa Krystiana	Szybszych odpowiedzi na zadane pytanie na messengerze. (Wyświetlenie - brak odpisania)	Nie podobało mi się pisanie o projekcie na messengerze zamiast poczekać na wyznaczony czas na daily.
Michał programuje jak szalony, wszystko ogarnia, jest komfort psychiczny że jak coś się sypie, to Michał za 2 minuty poprawi; fajnie też że można do niego podbijać jak się czegoś nie wie		Mam wrażenie że mamy za długie spotkania - musimy ustalić se limit czasu na każde spotkanie
Pomaganie sobie w przypadku napotkania problemów. np.: poprzez Pair programming.		pytania o inżynierkę w niedzielę po 17:00

Rysunek 1.23. Informacje zwrotne na temat procesu

Czynność 	Odpowiedzialni za zmianę	Czynności do wykonania
Zakupienie domeny internetowej oraz usługi hostingowej.	Michał Nieruchalski	1. Zakup odpowiedniej oferty spełniającej wymagania projektu + wyznaczenie równej składki
Komunikacja na temat projektu w czasie do tego wyznaczonym.	Cały zespół	Komunikacja w sprawie projektu tylko w czasie spotkania Kanban (z wyłączeniem pilnych, krytycznych sytuacji)
Rozszerzyć projekt o praktyki DevOps	Dawid Gostyński	1. Wdrożenie DevOps i CI/CD do projektu zgodnie z wymaganiami przedmiotu
Code Review i mergowanie	Michał Nieruchalski, Dawid Gostyński	1. Zmniejszenie limitu WIP na kolumnach "ready for code review" i "in code review" z 4 na 1. (Krystian Osiński) 2. Działanie w systemie pull! Dążymy do jak najszybszego dostarczania!
Zmniejszenie pracy nad dokumentacją	Krystian Osiński	Tworzenie dobrej ale minimalistycznej dokumentacji, zgodnie z wym. przedmiotu
Trzymanie się wyznaczonych ram czasowych w kontekście spotkań członków zespołu.	Krystian Osiński	Kanban Meeting - max: 15 min.   Problemy - max: 30 min.   Retrospektywa - max: 30 min.
Jawne przedstawianie braku możliwości prac w danym czasie nad projektem	Cały zespół	Zwiększenie transparentności pracy i procesu poprzez jawną komunikację
Poprawa Backlog'u	Krystian Osiński	Backlog powinien być zgodny ze sztuką. Nadanie odpowiednich priorytetów zadaniom. Posortowanie zadań zgodnie z ich priorytetem

Rysunek 1.24. Wnioski z retrospektywy

Wprowadzenie retrospektywy okazało się dobrym pomysłem. Zespół zauważył wiele irytujących i utrudniających pracę problemów. Wymienił się również pozytywnymi spostrzeżeniami na temat pracy poszczególnych członków. Wnioski posłużyły jako początek do wprowadzenia konkretnych zmian. Każda z nich się powiodła i przyniosła wymierne korzyści. Zespół zaczął czerpać większą przyjemność z pracy i zauważył znaczący wzrost wydajności.

Wprowadzone zmiany dotyczące procesu Kanban:

#### 1. Zmiany tablicy Kanban

- Zauważono, że kolumny: *"ready for code review"* i *"in code review"* (Obie miały limit 4) były wąskimi gardłami procesu. Zadania, które do nich trafiły zatrzymywały się na dłuższy czas. Powodowało to opóźnienia w dostarczaniu wartości biznesowej. Zwiększała się pętla informacji zwrotnej, a to prowadziło do zwiększenia przyszłego nakładu pracy związanego ze zmianą wymagań lub poprawą błędów. Zdecydowano się na usunięcie jednej z nich oraz na nadanie drugiej ograniczenia WIP równego 1. Została ona przemianowana na *"code review"*.
- W celu zmniejszenia niepotrzebnych informacji została usunięta kolumna *"rejected"*. Została wykorzystana tylko dwa razy w trakcie całego projektu. Zdecydowano, że zadanie, które nie zostało zaakceptowane podczas recenzji kodu, pozostanie dalej w tej fazie.

#### 2. Zmiany w spotkaniach i komunikacji

- Została wprowadzona większa dyscyplina na wspólnych spotkaniach w celu skrócenia ich czasu trwania. Zdecydowano się na wprowadzenie limitu czasowego na każde z nich.
- Rozmowy na temat projektu które nie były krytyczne czekały na cykliczne spotkanie. Zaprzeszono nadmiarowej komunikacji na czacie. Wprowadzono *"work life balance"*.

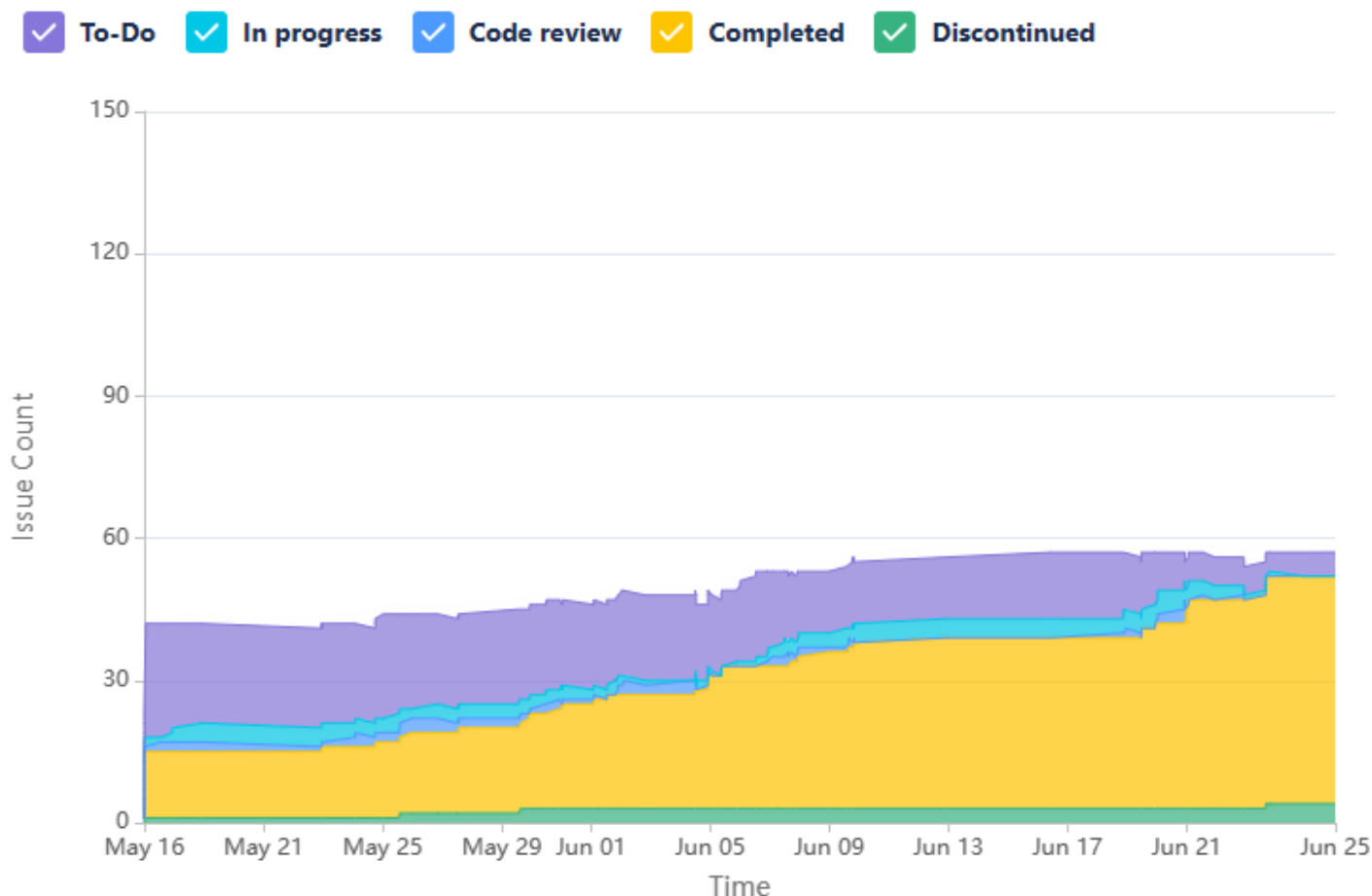
#### 3. Zmiana w kartach Kanban i backlog'u

- Zauważono problem z samoorganizacją zespołu. Wynikał on z nieposortowanych zadań w *backlog'u*. Zdecydowano się na wprowadzenie ikon reprezentujących priorytety zadań. Następnie zgodnie z nimi zostały uporządkowane.
- Karty Kanban otrzymały również pole opisujące moduł aplikacji, dla której wartość biznesową dostarczało dane zadanie. Pomogło to w szybszej identyfikacji oraz w zarządzaniu projektem.

### 1.5.3. Przepływ a praca zespołu projektowego

W celu zrealizowania wymagań przedmiotu, dobraliśmy się na podstawie naszych zdolności i dobrej znajomości. Stworzyliśmy zgrany i zintegrowany zespół wielofunkcyjny. Każdy członek mógł liczyć na pomoc reszty grupy. W trakcie wytwarzania Oprogramowywania nasze umiejętności spowodowały, że czas poświęcony na zgłębienie nowych informacji (w naszym obszarze odpowiedzialności) był minimalny. Pozwoliło to na szybkie i efektywne rozwiązywanie wyzwań i docelowo ograniczyło związane z nimi potencjalne przestoje.

Przepływ procesu utrzymywał się na względnie stałym poziomie przez cały okres trwania projektu. Zdarzały się jednak momenty wzmożonej pracy jak i chwile przestoju. Wynikały one z występowania krytycznych momentów w trakcie semestru, którymi były oddawanie wyznaczonych przyrostów (kamieni milowych) oraz zaliczenia innych przedmiotów. Zwiększona ilość pracy była wynikiem chęci dostarczenia wszystkich wymaganych funkcjonalności na czas. W pierwszym semestrze spowolnienie procesu było wynikiem nauki na egzamin z przedmiotu *Statystyka*. (około 10 - 19 czerwca) W drugim semestrze, zespół musiał poświęcić równolegle czas na pisanie pracy dyplomowej. Jeśli było to możliwe, całościowa praca zespołu była tak zarządzana, aby minimum jedna osoba w danym czasie rozwijała projekt. Mimo tych trudnień, zespół zdołał dostarczyć wszystkie wymagane funkcjonalności w wyznaczonym terminie.

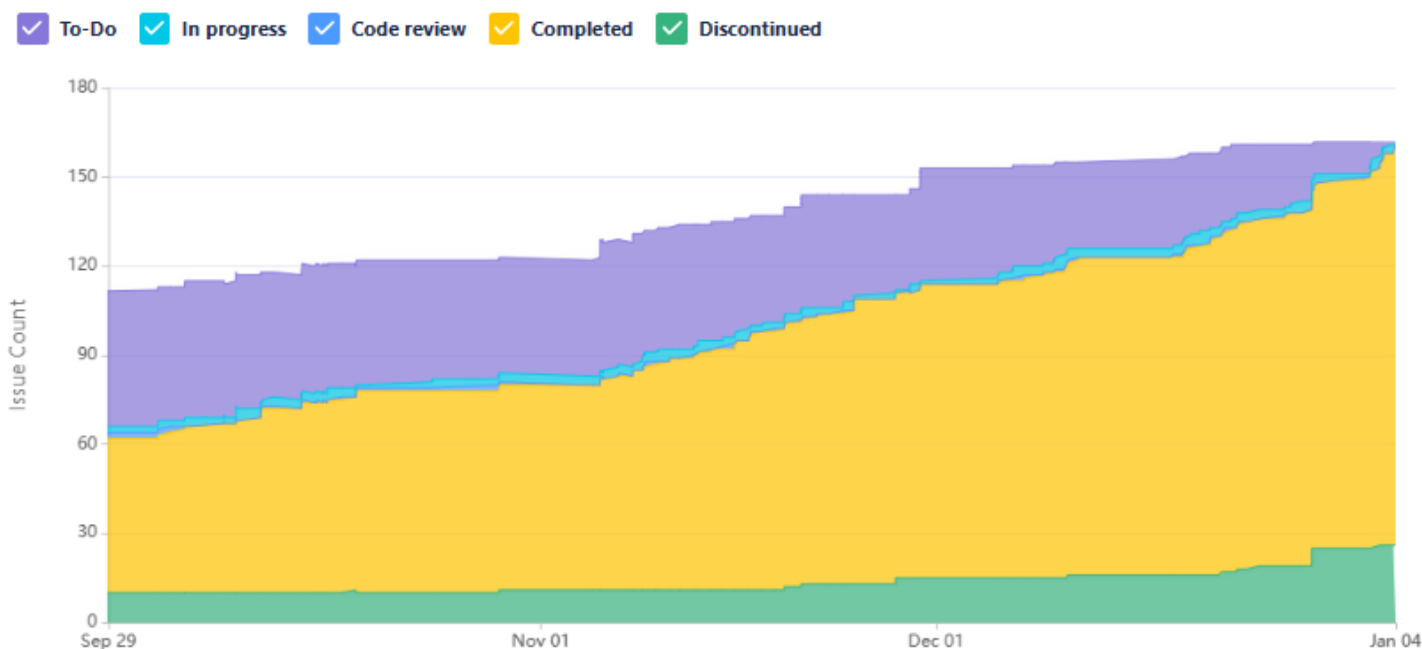


Rysunek 1.25. Diagram przepływu skumulowanego - 1 semestr

W wyniku podjętej decyzji we wstępnej fazie projektu, dotyczącej holistycznego podejścia do zadań programistycznych oraz dobrej koordynacji pracy, żaden członek zespołu nie był zablokowany przez cały okres trwania projektu. Praca była pobierana w taki sposób, aby każdy miał nad czym pracować. W przypadku czekania na doprecyzowanie wymagań czy innych mniejszych wyzwań, dobrze dobrany WIP pozwalał takiej osobie podjąć się innej czynności. Dzięki temu, zespół był w stanie utrzymać stały przyrost pracy. Na koniec pierwszego semestru został obliczony współczynnik przepustowości procesu. Wyniósł on 0.6 (50 zadań w około 83 dni)

W drugim semestrze, proces zbierania głównych wymagań rozpoczął się już pod koniec sierpnia. Było to podyktowane prośbą klienta, dla którego okres jesieni w związku z rozpoczęciem w tym czasie większości oferowanych przez jego firmę zajęć, jest wyjątkowo wymagający. Faktyczna praca zespołu projektowego oraz doprecyzowywanie wymagań rozpoczęły się na początku października. Właśnie wtedy na wykresie przepływu skumulowanego widać zakończenie wakacyjnego przestoju.

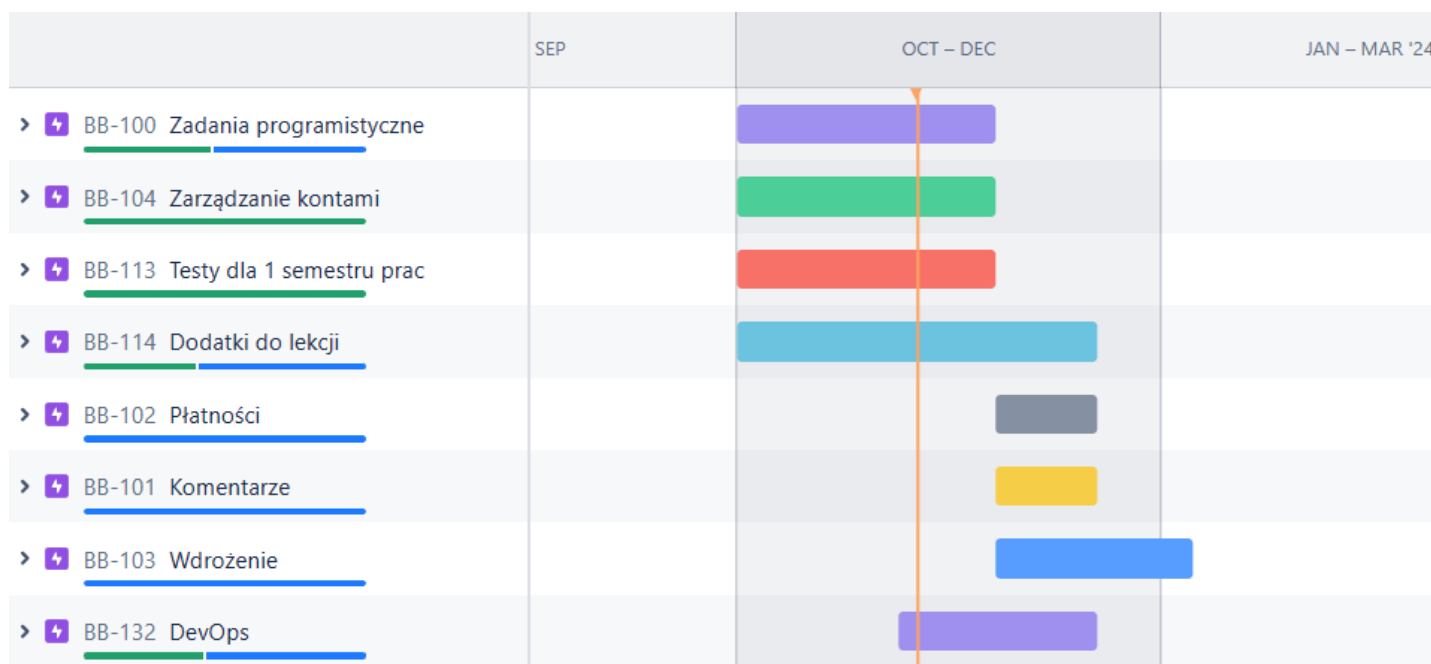
Współczynnik przepustowości na zakończenie drugiego semestru wynosił 0.81 (96 zadań w około 118 dni). Wzrost wydajności był wynikiem rozbijania większych zadań na mniejsze podzadania. Skróciło to pętlę informacji zwrotnej.



Rysunek 1.26. Diagram przepływu skumulowanego - 2 semestr

### Zarządzanie czasem

W zarządzaniu projektem pomogło wyznaczenie konkretnych terminów i celów projektowych. Manager projektu nadzorował postępy prac. Dbał o wykonywanie zadań o najwyższym priorytecie. Monitorował pozostały czas do końca każdego kamienia milowego. W drugim semestrze zostało wykorzystane dodatkowe narzędzie, oferowane przez Jirę. Był nim diagram Gantt'a, na którym był zwizualizowany zaplanowany czas oddania głównych modułów. Wykres ten oferował również monitorowanie ilości pozostałych zadań, wymaganych do ukończenia całej funkcjonalności.



Rysunek 1.27. Plan dostarczania funkcjonalności

### 1.5.4. Obszary do udoskonalenia

Osobiście dostrzegam dwa główne elementy do poprawy w zarządzaniu procesem projektu Brainboost:

Pierwszym z nich jest subiektywność podejmowanych działań. Zespół nie wykorzystał w pełni potencjału metodyki Kanban. Zarządzanie projektem oraz jego ewolucja były oparte na osobistej obserwacji i doświadczeniu każdego członka zespołu. Liczne decyzje nie zostały poparte danymi. Jedynym obserwowanym elementem był postęp, zwizualizowany na wykresie przepływu skumulowanego. Pilnowany był jednostajny przyrost wartości, co przedstawiało

się w umiarkowanym wzroście diagramu. Optycznie były dostrzegane przestoje i czas zmorzonej pracy. Przejście na bardziej zorganizowany sposób pracy pozwoliłoby na dokładniejsze śledzenie postępu projektu i formułowanie bardziej obiektywnych wniosków. Przez cały okres trwania projektu, automatycznie zbierane dane dotyczące wydajności procesu nie były dogłębnie analizowane. Wskaźniki czasu cyklu oraz czasu dostarczania nie były sprawdzane. Podobny problem dotyczy się obliczania przepustowości, która została wyliczona łącznie tylko 2 razy, na zakończenie każdego semestru. Brak monitorowania wynikał z następnego problemu.

Drugim zagadnieniem jest zróżnicowana wielkość zadań w projekcie. Zespół zdecydował, że nie będzie estymował ich wielkości. Wprowadził jedynie priorytety, które były nadawane przez menedżera projektu. Zmienny rozmiar pracy powodował, że przypuszczany czas dostarczania był problematyczny do określenia. Dlatego nie prowadzono obliczeń dotyczących wskaźników stanu przepływu pracy. (czas dostarczania, czas cyklu, przepustowość). Hipotetycznym rozwiązaniem tego problemu byłoby jednak wprowadzenie estymacji zadań. Pozwoliłoby to na lepsze planowanie i rozkładanie pracy w czasie. Zespół byłby w stanie przewidzieć, ile czasu zajmie wykonanie zadania, które zostało oszacowane na  $x$  punktów. Innym pomysłem byłoby rozbięcie wszystkich zadań tak, aby miały szacunkowo podobną wielkość.

Należy pamiętać, że zaproponowane powyższe rozwiązania nie są idealne. Wprowadzenie ich mogłoby być jednoznaczne z poświęceniem sporej ilości czasu na zarządzanie nimi. Podejmowanie się estymacji każdego zadania, rozbijanie ich na mniejsze części oraz dokładne analizy wskaźników, wiązały się z poświęcaniem czasu, w którym zespół mógł daną pracę wykonywać i dostarczać realną wartość. Przyniesione dzięki potencjalnym zmianom korzyści mogłyby być niewspółmierne do nakładu pracy. Dlatego też nie uważam, że projekt był prowadzony nierzetelnie. Uważam jednak, że stopniowe i ewolucyjne wprowadzenie powyższych zmian mogłoby z czasem przynieść wymierne korzyści. (wraz z nabywanym doświadczeniem przez zespół w szacowaniu zadań i liczenia wskaźników)

## 1.6. PODSUMOWANIE

### 1.6.1. Wynik projektu

Końcowym rezultatem projektu Brainboost jest w pełni funkcjonalna aplikacja internetowa, która została pomyślnie wdrożona. Oprogramowanie zostało zrealizowane zgodnie z założonym harmonogramem. Wszystkie wymagane kryteria zostały dostarczone na czas. Projekt zakończył się sukcesem, osiągając zamierzone cele i dostarczył gotowy produkt zgodny z oczekiwaniami klienta biznesowego.

### 1.6.2. Wnioski

Wdrożenie metodyki Kanban w projekcie Brainboost okazało się decyzją, która przyniosła pozytywne rezultaty. Dzięki wizualizacji i jasno sformułowanym zasadom, zespół był w stanie utrzymać relatywnie stały przepływ pracy, co przełożyło się na terminowe dostarczenie wszystkich funkcjonalności zgodnie z założonym planem projektowym.

Dobrze dobrany limit pracy w toku oraz reagowanie na niepokojące sygnały pozwoliło na zmniejszenie negatywnego wpływu wąskich gardeł na proces. Pozbycie się zbędnych i nieużywanych kolumn w tablicy Kanban, oczyściło ją ze zbędnych informacji i pozwoliło na skupienie się na szybszym dostarczaniu wartości biznesowej.

Ustanowiony proces Kanban przyczynił się do zmniejszenia pętli informacji zwrotnej. To z kolei skutkowało w dostarczaniu lepiej dopasowanych i zdefiniowanych funkcjonalności. Dodatkowo ograniczyło to pracę związaną ze zmianami wymagań lub poprawami błędów w późniejszych fazach projektu.

Uważam, że ewolucja procesu wskazuje na rozwój całego zespołu. Rozpoczęto pracę z podejściem: "zaczynij tam, gdzie jesteś". Następnie wraz z nabywanym doświadczeniem i wiedzą zaczęto wprowadzać kolejne zmiany. Większość wprowadzonych usprawnień miała miejsce dopiero w drugim semestrze. Wyniknęło to z faktu, że w przerwie międzysemestralnej rozpocząłem zgłębianie tematu metodyki Kanban. Wiedza, którą zdobyłem w tym czasie, pozwoliła mi na zidentyfikowanie obszarów do poprawy. Uważam, że znacząca ewolucja procesu mogła i powinna nastąpić dużo wcześniej. Nie zmienia to faktu, że transformacja procesu została przeprowadzona pomyślnie i docelowo doprowadziła do sukcesu projektu.

Mimo że nie wykorzystano w pełni potencjału metodyki Kanban i nie wprowadzono zasad dotyczących mierzenia i analizowania wskaźników stanu prze-



pływu pracy, zespół był świadomy swoich działań i podejmował decyzje w oparciu o swoje doświadczenie i subiektywne opinie. Wszelkie spostrzeżenia, które zostały przedstawione w czasie retrospektyw były trafne i poskutkowały ewolucją procesu. Zmiany przez nie wprowadzone nie były obiektywnie mierzalne, ale przyniosły dostrzegalne korzyści.

Biorąc pod uwagę sukces projektu oraz wszystkie wymienione wyżej pozytywne aspekty, można stwierdzić, że metodyka Kanban była odpowiednim wyborem dla projektu jak i dla jego zespołu. Zdobyta wiedza, jak i doświadczenie wynikające z zarządzania projektem Brainboost i jego ewolucji, pozwoliły mi na lepsze zrozumienie metodyki Kanban. Jestem przekonany, że w przyszłości będę w stanie jeszcze efektywniej wykorzystać tę metodykę, co przyniesie jeszcze lepsze rezultaty.

## ZAKOŃCZENIE

W powyższej pracy opisaliśmy wykorzystane przez nas narzędzia i technologie, które umożliwiły sprawne zaprojektowanie i wdrożenie aplikacji Brainboost. Wykorzystanie zwinnych metod rozwoju oprogramowania zapewniło zarówno dostosowanie do potrzeb klienta jak i lepszą reakcję na nieustannie zmieniające się wymagania oraz pojawiające się uwagi. Użycie nowoczesnego frameworka Angular pozwoliło na zaprojektowanie intuicyjnego interfejsu użytkownika bogatego w interaktywne elementy. Zastosowanie podejścia DevOps w stopniu znacznym ułatwiło i zautomatyzowało pracę związaną z wprowadzaniem kolejnych wersji oprogramowania na środowisko produkcyjne.

Pozytywny odbiór aplikacji przez klienta potwierdza, że wytworzony produkt jest bardzo dobrym narzędziem dydaktycznym z dużym potencjałem na rozwój.

## Bibliografia

- [1] State of Agile. *Badanie popularności metodyk zwinnych*. URL: [info.digital.ai/rs/981-LQX-968/images/SOA15.pdf](https://info.digital.ai/rs/981-LQX-968/images/SOA15.pdf) (term. wiz. 09.11.2023).
- [2] Bueno Natale Vinto and Alex Soto. *GitOps Cookbook: Kubernetes Automation in Practice*. O'Reilly, 2023.
- [3] Todd Ekenstam and Alexander Matyushentsev. *GitOps and Kubernetes: Continuous Deployment with Argo CD, Jenkins X, and Flux*. Manning, 2021.
- [4] Agile Alliance. *Historia manifestu Agile*. URL: [agilemanifesto.org/history.html](https://agilemanifesto.org/history.html) (term. wiz. 10.11.2023).
- [5] Agile Alliance. *Manifest Agile*. URL: [agilemanifesto.org/history.html](https://agilemanifesto.org/history.html) (term. wiz. 10.11.2023).
- [6] Dawid Anderson. *Kanban, Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.
- [7] Dawid Anderson. *School of Management*. URL: [djaa.com/david-j-anderson/](https://djaa.com/david-j-anderson/) (term. wiz. 11.11.2023).
- [8] Dawid Andreson i Andy Cermichael. *Kieszonkowy przewodnik po Kanbanie*. Kanban University Press. 2016. URL: [resources.kanban.university](https://resources.kanban.university) (term. wiz. 08.11.2023).
- [9] Angular. *Component Lifecycle*. URL: [angular.dev/guide/components/lifecycle](https://angular.dev/guide/components/lifecycle) (term. wiz. 09.12.2023).
- [10] Angular. *Dependency injection*. URL: [angular.dev/guide/di](https://angular.dev/guide/di) (term. wiz. 17.12.2023).
- [11] Angular. *Dependency injection in Angular*. URL: [angular.io/guide/dependency-injection-overview](https://angular.io/guide/dependency-injection-overview) (term. wiz. 17.12.2023).
- [12] Angular. *Directives*. URL: [angular.dev/guide/directives](https://angular.dev/guide/directives) (term. wiz. 10.12.2023).
- [13] Angular. *Essentials / Components*. URL: [angular.dev/essentials/components](https://angular.dev/essentials/components) (term. wiz. 09.12.2023).

- [14] Angular. *Hierarchical injectors*. URL: [angular.dev/guide/di/hierarchical-dependency-injection](https://angular.dev/guide/di/hierarchical-dependency-injection) (term. wiz. 16.12.2023).
- [15] Angular. *Pipes*. URL: [angular.dev/guide/pipes](https://angular.dev/guide/pipes) (term. wiz. 10.12.2023).
- [16] Angular. *What is Angular*. URL: [angular.io/guide/what-is-angular](https://angular.io/guide/what-is-angular) (term. wiz. 06.12.2023).
- [17] AngularJS. URL: [docs.angularjs.org](https://docs.angularjs.org) (term. wiz. 06.12.2023).
- [18] Atlassian. *Backlog*. URL: [www.atlassian.com/agile/scrum/backlogs](https://www.atlassian.com/agile/scrum/backlogs).
- [19] Gigi Sayfan and Bilgin Ibrayam. *Mastering Kubernetes: Dive into Kubernetes and learn how to create and operate world-class cloud-native systems*. Packt Publishing, 2023.
- [20] Blurify. *Kanban for everyone*. URL: [blurify.pl/blog/metoda-kanban-optymalizacja-procesow-biznesowych/](https://blurify.pl/blog/metoda-kanban-optymalizacja-procesow-biznesowych/) (term. wiz. 09.11.2023).
- [21] Barry Boehm. *A spiral model of software development and enhancement*. ACM SIGSOFT Software engineering notes, 1986.
- [22] Christopher Bradford. *A Case for Databases on Kubernetes from a Former Skeptic*. URL: <https://thenewstack.io/a-case-for-databases-on-kubernetes-from-a-former-skeptic> (term. wiz. 13.01.2024).
- [23] Bulldogjob. *noestimates*. URL: [bulldogjob.pl/readme/ruch-noestimates-czyli-koniec-estymat](https://bulldogjob.pl/readme/ruch-noestimates-czyli-koniec-estymat) (term. wiz. 13.11.2023).
- [24] Businessmap. *Podstawy Kanban*. URL: [businessmap.io/pl/zasoby-kanban/pierwsze-kroki/kanban](https://businessmap.io/pl/zasoby-kanban/pierwsze-kroki/kanban) (term. wiz. 07.11.2023).
- [25] CertManager. URL: <https://github.com/jetstack/cert-manager/releases/download/v1.7.1/cert-manager.yaml> (term. wiz. 13.01.2024).
- [26] The Learning Channel. *Kubernetes - Networking Series*. URL: <https://www.youtube.com/watch?v=B6FsWNUnRo0&list=PLSAko72nKb8QWsfPpBlsw-kOdMBD7sra-> (term. wiz. 13.01.2024).
- [27] Patrick Debois i in. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.
- [28] Amarachi Iheanacho and Divine Odazie. *A Brief History of DevOps and Its Impact on Software Development*. URL: <https://everythingdevops.dev/a-brief-history-of-devops-and-its-impact-on-software-development/#the-rise-of-the-agile-methodology> (term. wiz. 13.01.2024).
- [29] Alin Dobra. *History of DevOps – When Did DevOps Become a Thing?* URL: [History%20of%20DevOps%20%E2%80%93%20When%20Did%20DevOps%20Become%20a%20Thing?](https://www.youtube.com/watch?v=3-9fnjzmXWA) (term. wiz. 13.01.2024).
- [30] Brendan Eich. *A Brief History of JavaScript by the Creator of JavaScript*. URL: [youtube.com/watch?v=3-9fnjzmXWA](https://www.youtube.com/watch?v=3-9fnjzmXWA) (term. wiz. 09.12.2023).

- [31] Chris Milsted and Gabriele Bartolini. *Data On Kubernetes, Deploying And Running PostgreSQL*. URL: <https://www.youtube.com/watch?v=99uSJXkKpe1> (term. wiz. 13.01.2024).
- [32] Agarwal Gaurav. *Modern DevOps Practices: Implement and secure DevOps in the public cloud with cutting-edge tools, tips, tricks, and techniques*. Packt Publishing, 2021.
- [33] Geeksforgeeks. *Waterfall model*. URL: [www.geeksforgeeks.org/waterfall-model/](http://www.geeksforgeeks.org/waterfall-model/) (term. wiz. 09.11.2023).
- [34] Weiberg Gerald. *Quality Software Management: Systems Thinking*. Dorset House, 1992.
- [35] Paul Hammant. *Trunk Based Development*. URL: <https://trunkbaseddevelopment.com> (term. wiz. 13.01.2024).
- [36] Marcus hammarberg i Joakim Sunden. *Kanban, zobacz jak skutecznie zarządzać pracą!* Gliwice: Helion, 2015.
- [37] HoneyPot. *Kubernetes - The Documentary*. URL: <https://www.youtube.com/watch?v=BE77h7dmoQU> (term. wiz. 13.01.2024).
- [38] Sommerville Ian. *Inżynieria oprogramowania*. Warszawa: PWN, 2020.
- [39] Brink Nielsen and Jacob Andresen Santosh Yadav. *Accelerating Angular Development with Ivy. A practical guide to building faster and more testable Angular apps with the new Ivy engine*. Apress, 2021.
- [40] *javascript kubernetes client*. URL: <https://github.com/kubernetes-client/javascript> (term. wiz. 13.01.2024).
- [41] Nicole Forsgren and Jez Humble Gene Kim. *Przyspieszenie. Lean i DevOps w rozwoju firm technologicznych*. Helion, 2019.
- [42] *jQuery*. URL: [jquery.com](http://jquery.com) (term. wiz. 06.12.2023).
- [43] John Arundel and Justin Domingus. *Kubernetes - rozwiązania chmurowe w świecie DevOps. Tworzenie, wdrażanie i skalowanie nowoczesnych aplikacji chmurowych*. Helion, 2021.
- [44] Łukasz Kałużny. *Przegląd architektury w Azure typu LLD*. URL: <https://kaluzny.io/przeglad-architektury-w-azure-typu-lld-czesc-pierwsza> (term. wiz. 13.01.2024).
- [45] Mikael Krief. *Learning DevOps. A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins*. Packt Publishing, 2022.
- [46] Mohamed Labouardy. *Pipeline as Code. Continuous Delivery with Jenkins, Kubernetes, and Terraform*. Manning, 2021.
- [47] Brent Laster. *Learning GitHub Actions*. O'Reilly, 2023.
- [48] linkedin. *Dawid Anderson*. URL: [www.linkedin.com/in/agilemanagement/](http://www.linkedin.com/in/agilemanagement/) (term. wiz. 11.11.2023).

- [49] Scott Surovich and Marc Boorshtein. *Kubernetes i Docker w środowisku produkcyjnym przedsiębiorstwa. Konteneryzacja i skalowanie aplikacji oraz jej integracja z systemami korporacyjnymi*. Helion, 2022.
- [50] Russ McKendrick. *Infrastructure as Code for Beginners: Deploy and manage your cloud-based services with Terraform and Ansible*. Packt Publishing, 2023.
- [51] MDN. *JavaScript*. URL: [developer.mozilla.org/en-US/docs/Web/JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript) (term. wiz. 06.12.2023).
- [52] MDN. *Resize Observer*. URL: [developer.mozilla.org/en-US/docs/Web/API/ResizeObserver](https://developer.mozilla.org/en-US/docs/Web/API/ResizeObserver) (term. wiz. 10.12.2023).
- [53] Henry van Merode. *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress, 2023.
- [54] MetalLB. *MetalLB repository*. URL: <https://raw.githubusercontent.com/metallb/metallb/v0.13.12/config/manifests/metallb-native.yaml> (term. wiz. 13.01.2024).
- [55] Piotr Minkowski. *Which JDK to Choose on Kubernetes*. URL: <https://piotrminkowski.com/2023/02/17/which-jdk-to-choose-on-kubernetes> (term. wiz. 13.01.2024).
- [56] MIT. *Przetwarzanie informacji w mózgu*. URL: [news.mit.edu/2014/in-the-blink-of-an-eye-0116](https://news.mit.edu/2014/in-the-blink-of-an-eye-0116) (term. wiz. 11.11.2023).
- [57] Kief Morris. *Infrastructure as Code: Dynamic Systems for the Cloud Age*. O'Reilly, 2021.
- [58] Jeff Carpenter and Patrick McFadin. *Managing Cloud Native Data on Kubernetes: Architecting Cloud Native Data Services Using Open Source Technology*. O'Reilly, 2023.
- [59] Nigel Poulton. *Docker Deep Dive*. Nielson Book Services, 2023.
- [60] Liz Rice. *Kontenery. Bezpieczne wdrożenia. Podstawowe koncepcje i technologie*. Helion, 2021.
- [61] Winston Royce. *Managing the Development of Large Software Systems*. TRW, 1970.
- [62] Rohit Salecha. *Practical GitOps: Infrastructure Management Using Terraform, AWS, and GitHub Actions*. Apress, 2022.
- [63] Gabriel N. Schenker. *The Ultimate Docker Container Book. Build, test, ship, and run containers with Docker and Kubernetes*. Packt Publishing, 2023.
- [64] Fabio Spampinato. *Cash*. URL: [github.com/fabiospampinato/cash](https://github.com/fabiospampinato/cash) (term. wiz. 06.01.2024).
- [65] Mateusz Stefańczyk. *Angular Standalone API*. URL: [angular.io/blog/2022/12/28/standalone-api](https://angular.io/blog/2022/12/28/standalone-api) (term. wiz. 17.12.2023).

- [66] International Institute For Asian Studies. *Kanban signs*. URL: [www.iias.asia/the-review/kanban-traditional-shop-signs-japan](http://www.iias.asia/the-review/kanban-traditional-shop-signs-japan) (term. wiz. 01.11.2023).
- [67] Kanban Tool. *Kanban History*. URL: [kanbantool.com/kanban-guide/kanban-history](http://kanbantool.com/kanban-guide/kanban-history) (term. wiz. 30.10.2023).
- [68] Toyota. *Andon*. URL: [mag.toyota.co.uk/andon-toyota-production-system/](http://mag.toyota.co.uk/andon-toyota-production-system/) (term. wiz. 06.11.2023).
- [69] Toyota. *Historia Toyota*. URL: [www.toyota-global.com/company/history\\_of\\_toyota/75years/text/](http://www.toyota-global.com/company/history_of_toyota/75years/text/) (term. wiz. 05.11.2023).
- [70] Toyota. *Historia TPS*. URL: [global.toyota/en/company/vision-and-philosophy/production-system/](http://global.toyota/en/company/vision-and-philosophy/production-system/) (term. wiz. 06.11.2023).
- [71] Toyota. *TPS*. URL: [www.toyotapl.com/world-of-toyota/system-produkcji](http://www.toyotapl.com/world-of-toyota/system-produkcji) (term. wiz. 03.11.2023).
- [72] *TypeScript*. URL: [typescriptlang.org](http://typescriptlang.org) (term. wiz. 06.12.2023).
- [73] Wikipedia. *Angular*. URL: [en.wikipedia.org/wiki/Angular\\_\(web\\_framework\)](http://en.wikipedia.org/wiki/Angular_(web_framework)) (term. wiz. 06.12.2023).
- [74] Wikipedia. *AngularJS*. URL: [en.wikipedia.org/wiki/AngularJS](http://en.wikipedia.org/wiki/AngularJS) (term. wiz. 06.12.2023).
- [75] Wikipedia. *ECMAScript*. URL: [en.wikipedia.org/wiki/ECMAScript](http://en.wikipedia.org/wiki/ECMAScript) (term. wiz. 06.12.2023).
- [76] Wikipedia. *Edo period*. URL: [en.wikipedia.org/wiki/Edo\\_period](http://en.wikipedia.org/wiki/Edo_period) (term. wiz. 29.10.2023).
- [77] Wikipedia. *History of Toyota*. URL: [en.wikipedia.org/wiki/History\\_of\\_Toyota](http://en.wikipedia.org/wiki/History_of_Toyota) (term. wiz. 04.11.2023).
- [78] Wikipedia. *JavaScript*. URL: [en.wikipedia.org/wiki/JavaScript](http://en.wikipedia.org/wiki/JavaScript) (term. wiz. 06.12.2023).
- [79] Wikipedia. *jQuery*. URL: [en.wikipedia.org/wiki/JQuery](http://en.wikipedia.org/wiki/JQuery) (term. wiz. 06.12.2023).
- [80] Wikipedia. *Kanban*. URL: [en.wikipedia.org/wiki/Kanban\\_\(development\)](http://en.wikipedia.org/wiki/Kanban_(development)) (term. wiz. 10.11.2023).
- [81] Wikipedia. *Muda*. URL: [en.wikipedia.org/wiki/Muda\\_\(Japanese\\_term\)](http://en.wikipedia.org/wiki/Muda_(Japanese_term)) (term. wiz. 04.11.2023).
- [82] Wikipedia. *Node.js*. URL: [en.wikipedia.org/wiki/Node.js](http://en.wikipedia.org/wiki/Node.js) (term. wiz. 06.12.2023).
- [83] Wikipedia. *Prawo Little'a*. URL: [pl.wikipedia.org/wiki/Prawo\\_Little%E2%80%99a](http://pl.wikipedia.org/wiki/Prawo_Little%E2%80%99a) (term. wiz. 12.11.2023).
- [84] Wikipedia. *Pure function*. URL: [en.wikipedia.org/wiki/Pure\\_function](http://en.wikipedia.org/wiki/Pure_function) (term. wiz. 10.12.2023).

- [85] Wikipedia. *Rup Model*. URL: [pl.wikipedia.org/wiki/Rational\\_Unified\\_Process](https://pl.wikipedia.org/wiki/Rational_Unified_Process) (term. wiz. 01.12.2023).
- [86] Wikipedia. *Spiral model*. URL: [pl.wikipedia.org/wiki/Model\\_spiralny](https://pl.wikipedia.org/wiki/Model_spiralny) (term. wiz. 01.12.2023).
- [87] Wikipedia. *Taiichi Ohno*. URL: [pl.wikipedia.org/wiki/Taiichi\\_%C5%8Cno](https://pl.wikipedia.org/wiki/Taiichi_%C5%8Cno) (term. wiz. 02.11.2023).
- [88] Wikipedia. *Taiichi Ohno*. URL: [en.wikipedia.org/wiki/Taiichi\\_Ohno](https://en.wikipedia.org/wiki/Taiichi_Ohno) (term. wiz. 02.11.2023).
- [89] Wikipedia. *TypeScript*. URL: [en.wikipedia.org/wiki/TypeScript](https://en.wikipedia.org/wiki/TypeScript) (term. wiz. 06.12.2023).
- [90] Wikipedia. *Waterfall model*. URL: [en.wikipedia.org/wiki/Waterfall%5C\\_model](https://en.wikipedia.org/wiki/Waterfall%5C_model) (term. wiz. 08.11.2023).
- [91] Timmy Willison. *jQuery 3.1.1 Released!* URL: [blog.jquery.com/2016/09/22/jquery-3-1-1-released](https://blog.jquery.com/2016/09/22/jquery-3-1-1-released) (term. wiz. 06.01.2024).
- [92] Henning Wolf. *Zwinne projekty w klasycznej organizacji Scrum, Kanban, XP*. Gliwice: Helion, 2014.
- [93] Natan Yellin. *For the Love of God, Stop Using CPU Limits on Kubernetes*. URL: <https://home.robusta.dev/blog/stop-using-cpu-limits> (term. wiz. 13.01.2024).
- [94] Zepto.js. *Zepto.js*. URL: [zeptojs.com](https://zeptojs.com) (term. wiz. 06.01.2024).



## BRAINBOOST - WIZJA I WYMAGANIA PROJEKTOWE

### Wizja i wykorzystanie systemu

Naszym klientem jest firma, która zajmuje się tworzeniem różnego rodzaju kursów edukacyjnych zdalnych oraz stacjonarnych m.in. nauka języków, robotyki, programowania. Wszystkie z kursów prowadzone są w formie grupowej, pod ciągłym nadzorem nauczyciela. Nasz system powstał w wyniku zainteresowania firmy w organizowaniu asynchronicznych kursów online w formie indywidualnej, bez ciągłego nadzoru nauczyciela. W takim podejściu kursant, może uczyć się wtedy gdy ma chwilę wolnego czasu. W trakcie przerabiania kursu, kursant nie zostanie jednak pozostawiony w osamotnieniu. Może bowiem poprosić autora kursu/innych kursantów o pomoc, lub zadać pytanie na które odpowiedź dostanie po pewnym czasie (gdy pracownik zauważy rozpoczętą dyskusję). W założeniu, nauczyciel nie musi poświęcać na prowadzenie kursu tyle czasu ile poświęca podczas synchronicznych kursów grupowych, dzięki czemu firma, może obsłużyć większą ilość kursantów w krótkim czasie. Jednocześnie w pełni elastyczne godziny kursu są atrakcyjnym rozwiązaniem dla osób, które dysponują ograniczonym czasem.

Aktualnie system traktowany jest jako atrakcyjny dodatek dla obecnych uczniów szkoły. Jest on również wykorzystywany w trakcie lekcji, warsztatów i pólkoloni. Klient przewiduje ewolucję systemu. Planuje wprowadzić możliwość subskrypcyjnego dostępu do platformy dydaktycznej, dla osób nieposiadających wykupionych kursów synchronicznych.

Głównym produktem naszego projektu jest aplikacja webowa do przeprowadzania kursów online w formie asynchronicznej, z zapewnieniem wsparcia ze strony nauczyciela. Naszym głównym celem było dostarczenie naszemu klientowi platformy, dzięki której będzie w stanie w wygodny i efektywny sposób tworzyć, udostępniać i przeprowadzać kursy online. W ramach kreatora, pracownicy korzystają z edytowalnych komponentów: filmu, tekstu obsługującego język markdown oraz zadań programistycznych. Nasza aplikacja umożliwia uruchamianie kodu napisanego przez kursanta. Obsługujemy język python, javascript oraz HTML i CSS. Pracownicy mogą wprowadzać oraz usuwać konta

uczniów do systemu za pomocą pojedynczego pliku .xlsx w wykorzystywanym przez firmę od lat formacie.

## **Wymagania нефunkcjonalne**

### **A. Tworzenie kursu:**

1. Podczas procesu tworzenia kursu istnieje możliwość wyboru odpowiedniego tytułu, podtytułu oraz zdjęcia.
2. Tworząc kurs wybieramy jego kategorię (jakiego typu treści będziemy nauczać).
3. Każdy kurs powinien mieć podobny styl.
4. Istnieje możliwość dodawania modułów i lekcji.
5. Widoczny pasek nawigacji (moduły i lekcje).
6. W trakcie tworzenia lekcji nauczyciel wykorzystuje komponenty, które zostaną przez niego uzupełnione konkretnymi treściami, linkami i zadaniami. Składają się na nie następująco:
  - a. film
  - b. tekst (markdown)
  - c. Zadania programistyczne (w zależności od kategorii kursu: interaktywnych lub nieinteraktywnych, czyli takich bez możliwości kompilacji na stronie: polecenia będą umieszczane jako komponent b.).
7. W trakcie tworzenia kursu istnieje możliwość edycji i usunięcia każdej z jego części.
8. Możliwość włączenia i wyłączenia podglądu widoku kursu z perspektywy ucznia.
9. Możliwość dodawania materiałów dydaktycznych dla każdej z lekcji (osobna sekcja).
10. Zapisywanie postępów pracy w kreatorze.
11. Możliwość tworzenia, edycji i udostępnienia kursów przez każdego pracownika.
12. Usunąć kurs może tylko administrator i super-administrator.

### **B. Przechodzenie kursu:**

1. Widoczny pasek nawigacji między modułami i lekcjami. Możliwość związania i rozwijania lekcji oraz modułów.
2. Uczeń w widoku pojedynczej lekcji będzie mógł się przemieszczać pomiędzy treścią, materiałami do pobrania i komentarzami za pomocą zakładek.
3. Możliwość pobrania materiałów pomocniczych i skorzystania z linków.

4. Po ponownym zalogowaniu, użytkownik będzie kontynuować naukę od pierwszej nieukończonej lekcji. (na podstawie manualnie odhaczonych lekcji przez ucznia)
5. Widoczny pasek postępu na pasku menu (postęp mierzony ukończonymi lekcjami) oraz graficzne ukazanie skończonych lekcji na pasku nawigacyjnym w trakcie przechodzenia lekcji.
6. W każdej lekcji będzie możliwość zaznaczenia jej jako ukończoną (oraz możliwość wykonania operacji odwrotnej)
7. Zadanie programistyczne/webowe będzie składać się z kilku okien (zadania webowe będą mieć tylko opcje b i c (treść jako markdown):
  - a. treść zadania
  - b. okna do wpisywania kodu przez ucznia
  - c. okna ukazującego wynik uruchomienia programu (konsola/wygląd strony internetowej).
8. W zadaniu programistycznym w ramach treści będzie dostęp do systemu odpowiedzi.
9. Możliwość interakcji z innymi uczestnikami kursu i mentorami firmy Sky Blue Education poprzez dodawanie wątków i odpowiedzi do nich.
10. Przed wstawieniem komentarza następuje jego weryfikacja czy nie jest wulgarny.
11. Przykładowy kod zapisany w języku markdown będzie kolorowy zgodnie ze składnią danego języka.

### **C. Administrowanie kontami pracowników:**

1. Możliwość dodania użytkowników wraz z określeniem ich roli (uczeń, administrator, nauczyciel)
2. Możliwość usunięcia konta określonego pracownika (zabezpieczenie: administrator nie może usunąć innego administratora (może to zrobić tylko superadmin)
3. Stworzenie wyszukiwarki uczniów i pracowników na podstawie ich emaila.

### **D. zarządzanie komentarzami:**

1. Każdy pracownik w pasku menu ma ukazaną ilość nowych komentarzy
2. Specjalny widok dla pracownika, w którym to ma ukazane wszystkie "nowe" komentarze i może na nie zareagować.

**E. Role, logowanie oraz menu:**

1. Każdy pracownik powinien mieć dostęp do podglądu stanu kursu ucznia i jego postępów.
2. Każdy pracownik może stworzyć wiele kont uczniów na podstawie dokumentu przechowującego ich dane. (xlsx)
3. Wspólny ekran logowania do aplikacji dla każdej z ról.
4. Menu główne dla ucznia to widok wszystkich dostępnych dla niego kursów. Widok pracowników natomiast rozszerzony jest o dostęp do kursów będących w trakcie tworzenia.
5. Będą istnieć Tylko 4 role na stronie: uczeń, nauczyciel, administrator, super-administrator.
6. W zależności od poziomu szkodliwości usunięcia danej rzeczy, powinny pojawić się coraz to mocniejsze zabezpieczenia (np.: usunięcie kursu, elementu kursu, komentarza, pracownika itp.)
7. Pasek nawigacji oraz pasek menu w aplikacji będą sticky, co oznacza, że będą pozostawać widoczne dla użytkownika nawet podczas przewijania strony.
8. W pasku menu znajdują się: ikona reprezentująca użytkownika, oraz logo Sky Blue.
9. Po kliknięciu w ikonę pojawi się możliwość wylogowania lub przejścia do ustawień.
10. Możliwość zmiany avatara przez uczniów.
11. Po wprowadzeniu błędnej adresy strony, użytkownik zostanie przekierowany do strony informującej o błędnym adresie - 404.

**Wymagania niefunkcjonalne**

1. Każdy loguje się za pomocą hasła, które musi składać się z min. 8 znaków: w tym min. Jedna duża i mała litera, min. jedna cyfra i min. 1 znak specjalny.
2. Cały system nie ulegnie awarii i spowolnieniu, jeśli użytkownik wprowadzi szkodliwy kod do zadania programistycznego. Np.: wieczną pętlę, wykonującą obliczenia i zapisującą kolejne elementy do listy.
3. Przygotowanie materiałów o platformie edukacyjnej dla pracowników firmy w formie dokumentu .pdf i prezentacji .pptx.
4. W ramach materiałów dydaktycznych na temat platformy mają również zostać wstawione minimum 3 linki do stron umożliwiających przejście do materiałów i filmów oferujących informacje na temat języka markdown oraz

- minimum jednego linku do strony umożliwiającej ćwiczenie tego języka online.
5. Kolory przewodnie aplikacji muszą być w kolorach firmy (odcienie niebieskiego, bieli, czerni i szarości - z wyłączeniem koloru zielonego, pomarańczowego i czerwonego dla odpowiednich przycisków / ostrzeżeń)
  6. Strona musi się ładować w mniej niż 4 sekundy.

### **Mierzalne wskaźniki wdrożeniowe**

1. Aplikacja będzie udostępniona pod wskazaną przez klienta nazwą w domenie np.: <https://skyblue.brain-boost.pl>
2. Przewiduje się możliwość 1000 zarejestrowanych użytkowników.
3. Aplikacja na początku będzie oferowała dostęp do 1 wstępnego kursu programowania w języku Python. Kurs zostanie stworzony na podstawie już istniejącego kursu - oferowanego przez firmę klienta (kurs ten składa się z 8 lekcji. Kurs brainboost ma zostać skrócony do 2 modułów (podstawy, warunki oraz listy).
4. Aplikacja będzie udostępniać materiały szkoleniowe dla pracowników firmy w postaci dokumentu PDF i prezentacji .pptx. W ramach tych materiałów zostaną umieszczone minimum 3 linki do filmów na YT lub stron internetowych, uczących jak stosować język markdown. W dodatku zostanie umieszczony minimum 1 link do strony, która pozwala na naukę tego języka
5. Zgodność z RODO i z innymi aspektami prawnymi.