Zeszyty Naukowe WWSI, No 18, Vol. 12, 2018, s. 7-44
DOI: 10.26348/znwwsi.18.7

Wykorzystanie metodyki Scrum przy implementacji przykładowego systemu bankowości internetowej

Anna Kisielińska-Ptasznik[*], Ewa Figielska[**]

Warszawska Wyższa Szkoła Informatyki

Abstrakt

W artykule przedstawiony został proces tworzenia przykładowego systemu bankowości internetowej z wykorzystaniem jednej ze zwinnych metodyk zarządzania projektami, a mianowicie metodyki Scrum. Przeprowadzona została analiza potrzeb klienta, na podstawie której określono Wizję produktu, opracowany został Rejestr Produktu oraz zaplanowane Sprinty. Pokazane zostały efekty wykonania prac w kolejnych Sprintach. Wskazano różnice między przyrostowym a kaskadowym podejściem do wytwarzania oprogramowania w kontekście tworzonego systemu.

Słowa kluczowe **–** Scrum, podejście przyrostowe, bankowość internetowa

1. Wprowadzenie

Systemy informatyczne mogą być zarządzane przy zastosowaniu różnych metodyk. Dawniej najpopularniejszym podejściem do zarządzania był model kaskadowy, według którego tworzenie systemu odbywało się w następujących etapach: analiza

[*] E-mail: a_kisielinska@poczta.wwsi.edu.pl
[**] E-mail: efigielska@poczta.wwsi.edu.pl

Zgłoszono do druku 28 czerwca 2018 r.

wymagań biznesowych, projekt, implementacja, testowanie, akceptacja klienta i wydanie, przy czym rozpoczęcie kolejnego etapu było możliwe po zakończeniu etapu poprzedniego. Podejście to może być stosowane nawet dzisiaj, przy założeniu, że mamy bardzo dobrze zdefiniowane wymagania oraz gwarancję, że nie będą one w trakcie realizacji projektu zmieniane. Wiadomo, że takie warunki są mało prawdopodobne i w rzeczywistych przedsięwzięciach trudno je zagwarantować. Zazwyczaj klient początkowo nie ma kompletnej wizji systemu. Często krystalizuje się ona dopiero w momencie zetknięcia się z produktem. W przypadku stosowania modelu kaskadowego, pierwsze zetknięcie się klienta z produktem następuje w chwili jego odbioru. Wówczas, wprowadzenie zmian do produktu staje się bardzo kosztowne (np. wymaga powrotu do wstępnych faz jego realizacji) lub wręcz niemożliwe. Jeżeli klient ma wizję systemu i dobrze ją zdefiniuje, to niekoniecznie sprawdzi się ona w rzeczywistości. W takim przypadku projekt będzie zrealizowany poprawnie, ale użytkownicy systemu nie będą z niego zadowoleni. Wiele projektów należy do grupy projektów innowacyjnych, które z założenia nie będą na początku dobrze zdefiniowane. Ponadto częste zmiany w świecie biznesu, prawa oraz trendów, powodują zmianę wymagań niezależnie od klienta.

Odpowiedzią na powyższe problemy są podejścia zwinne. Zasadniczą cechą podejścia zwinnego jest przyrostowe wytwarzanie produktu w stosunkowo krótkich iteracjach. Klient w niedługim czasie dostaje pierwszą wersję systemu z najistotniejszymi funkcjami. Może się do nich odnieść, zaproponować zmianę lub dodać inną funkcjonalność. Zmiany te są uwzględniane w kolejnych iteracjach. W ten sposób komunikacja między klientem a wykonawcą jest efektywniejsza, a zmiany wprowadzane i zatwierdzane są na bieżąco. Ponadto, podejście zwinne pozwala utrzymać wysokie zaangażowanie zespołu, ponieważ krótkie iteracje i możliwość śledzenia postępów pracy eliminują w pewnym stopniu „wypalenie" programistów.

W artykule przedstawiony został proces tworzenia przykładowego systemu bankowości internetowej, który będzie nazywany „eBankiem", z wykorzystaniem jednej ze zwinnych metodyk zarządzania projektami, a mianowicie metodyki Scrum. W rozdziale 2 artykułu przedstawione zostały podstawowe pojęcia związane z metodyką Scrum. W rozdziale 3 przeprowadzona została analiza potrzeb klienta, na podstawie której określono listę cech systemu. W rozdziale 4 opisane zostały

poszczególne iteracje procesu implementacji systemu. Podsumowanie artykułu znajduje się w rozdziale 5.

Przy realizacji systemu „eBank" zastosowano następujące oprogramowanie: środowisko Microsoft Visual Studio 2017 wersja Community, język programowania c# i technologię ASP.NET MVC (implementacja systemu), MS SQL Server 2017 (zarządzanie warstwą danych), Team Foundation Server (TFS) 2017 (wspomaganie procesu zarządzania i planowania), Enterprise Architect 13 firmy SPARX – wersja próbna (wykonanie diagramów i modeli).

2. Metodyka Scrum

Metodyka Scrum w zastosowaniu do tworzenia oprogramowania została opracowana w latach dziewięćdziesiątych ubiegłego wieku przez Sutherlanda i Schwabera [1], którzy inspirację zaczerpnęli z pracy Nonaka i Takeuschi [2], przedstawiającej nowe, w stosunku do modelu kaskadowego, koncepcje zarządzania procesem wytwarzania produktów.

W podejściu Scrum [1, 3, 4, 5, 6] produkt jest tworzony przyrostowo w kolejnych iteracjach, nazywanych Sprintami, przy czym efektem poszczególnych Sprintów (również tych początkowych) musi być działająca część produktu.

Prace rozpoczynają się od ustalenia Wizji produktu. Nakreśla ona zasadnicze cechy, którymi powinien charakteryzować się produkt, aby spełnić oczekiwania klienta. Lista cech produktu nazywana jest Rejestrem Produktu (ang. *Product Backlog*). Lista ta jest uporządkowana według priorytetów nadawanych cechom na podstawie wartości, jaką przedstawiają dla klienta, oraz związanego z ich realizacją ryzyka. Cechy te mogą być prezentowane w postaci opowiadań (historii) użytkownika (ang. *User Stories*).

W kolejnych Sprintach realizowana jest pewna liczba cech o najwyższym priorytecie wybranych spośród cech jeszcze nie wykonanych. Wybrane do wykonania w danym Sprincie cechy muszą być w tym Sprincie zrealizowane w całości. Rejestr Produktu nie jest katalogiem zamkniętym. Jeżeli zachodzi taka potrzeba, jest on modyfikowany lub rozszerzany przed wykonaniem kolejnego Sprintu.

Przy realizacji projektu bierze udział przynajmniej jeden zespół, nazywany Zespołem Scrumowym (ang. *Scrum Team*), w którym można wyróżnić 3 role: Właściciel Produktu, Scrum Master oraz członek Zespołu Deweloperskiego.

Zespół Deweloperski (ang. *Development Team*) nie ma narzuconej wewnętrznej struktury, a jego członkowie nie mają przypisanych ról – wszyscy są deweloperami, chociaż zajmują się realizacją różnych zadań. Zespół organizuje się samodzielnie i musi dbać o to, aby jego członkowie wspólnie posiadali niezbędne w realizacji projektu umiejętności. Zazwyczaj wielkość zespołu to od 5 do 9 osób.

Scrum Master przewodzi procesowi Scrum, dba o właściwe rozumienie i stosowanie Scruma przez Zespół Deweloperski. Zadanie to wypełnia obserwując sytuację, pomagając w rozwiązywaniu problemów, wspomagając samoorganizację zespołu i usuwając ewentualne przeszkody.

Właściciel Produktu (ang. *Product Owner*) ma wiedzę w danej dziedzinie, rozumie przypadki biznesowe, rynek i klienta. Do jego zadań należy określenie cech produktu i ich priorytetów w Rejestrze Produktu, przeprowadzanie aktualizacji Rejestru Produktu oraz akceptacja bądź odrzucenie efektów prac.

Jak wspomniano wcześniej, produkt jest wytwarzany iteracyjnie w tzw. Sprintach. Pojedynczy Sprint trwa zazwyczaj od 1 do 4 tygodni. Rozpoczęcie danego Sprintu rozpoczyna się od jego planowania (ang. *Sprint Planning*) podczas spotkania, w którym bierze udział Zespół Scrumowy. To wtedy ustalany jest Cel Sprintu (ang. *Sprint Goal*) oraz wybierane są cechy z Rejestru Produktu o najwyższym priorytecie, których realizacja przełoży się na osiągnięcie tego celu. Cechy zostają następnie rozbite na bardziej szczegółowe zadania – w ten sposób powstaje Rejestr Sprintu (ang. *Sprint Backlog*).

Po zaplanowaniu Sprintu Zespół Deweloperski przystępuje do realizacji poszczególnych zadań, o których przydziale i kolejności wykonania decydują sami członkowie zespołu. Postępy w realizacji zadań są monitorowane przy pomocy codziennych krótkich spotkań (ang. *daily stand-ups*), podczas których każdy omawia, co zdołał wykonać od ostatniego spotkania, co zamierza wykonać do następnego spotkania oraz czy według niego istnieją jakieś przeszkody, które mogą utrudnić lub uniemożliwić osiągnięcie celu. W codziennych spotkaniach biorą udział wszyscy członkowie Zespołu Deweloperskiego i Scrum Master.

Po zakończeniu prac deweloperskich odbywa się Przegląd Sprintu (ang. *Sprint Review*), którego uczestnikami są zarówno członkowie Zespołu Scrumowego jak i inne zainteresowane osoby: klienci, sponsorzy, członkowie innych zespołów. Podczas tego spotkania omawiany jest aktualny stan produktu i prezentowane jest jego

działanie. Ponadto, Właściciel Produktu omawia stan Rejestru Produktu oraz przewidywania dotyczące dalszego postępu prac. Kolejnym kluczowym elementem jest dyskusja na temat dalszych losów projektu, w wyniku której może nastąpić modyfikacja Rejestru Produktu poprzez dodanie nowych cech czy też zmianę priorytetów.

Retrospektywa Sprintu (ang. *Sprint Retrospection*) jest ostatnią aktywnością przeprowadzaną w ramach Sprintu – odbywa się tuż po przeglądzie, w gronie członków Zespołu Scrumowego. Jej celem jest analiza pracy zespołu w ramach tego Sprintu i możliwości jej usprawnienia. Omawiane są również problemy, jeśli miały miejsce, oraz działania naprawcze, które mogą zostać podjęte w podobnych sytuacjach w czasie następnego Sprintu.

3. Wizja tworzonego systemu „eBank”

Wizja produktu w metodyce Scrum wyrażona jest w postaci opowiadań użytkownika określających ogólne wyobrażenie klienta o systemie. Projektowany system „eBank” opisano w następujący sposób:

- Klient ma mieć możliwość przeglądania przez Internet historii wszystkich wykonanych przez niego operacji;
- Klient ma mieć możliwość przeglądania danych dotyczących posiadanych przez niego rachunków;
- Klient ma mieć możliwość zarządzania przelewami przez Internet;
- Klient ma mieć możliwość zarządzania lokatami terminowymi przez Internet;
- Osoba niebędąca klientem banku lub klient, który nie potwierdził swojej tożsamości, nie ma dostępu do funkcji serwisu za wyjątkiem funkcji dostępnych dla gości.

Powyższe wymagania zostały określone bardzo ogólnie i dlatego nie mogłyby stanowić podstawy do rozpoczęcia prac nad systemem w kaskadowym modelu wytwarzania oprogramowania. Taka sytuacja generowałaby ryzyko swobodnej interpretacji wymagań przez zespół projektowy, a to mogłoby być powodem nieporozumień w trakcie odbioru i skazać przedsięwzięcie na niepowodzenie.

Podejście zwinne, jakim jest metodyka Scrum, daje możliwość rozpoczęcia realizacji projektu, mimo braku szczegółowej specyfikacji wymagań całego produktu. Jest to możliwe dzięki temu, że poszczególne cechy produktu są dokładnie określane

tuż przed ich realizacją, czyli w Sprintach. Niemniej jednak, już na tym etapie należy poznać główne procesy biznesowe, zrozumieć je, poznać wąskie gardła i zastanowić się, jak można je usprawnić lub całkowicie przebudować, aby osiągnąć zamierzone cele realizacji projektu.

3.1. Analiza głównych procesów biznesowych

Omówienie z odbiorcą wszystkich implementowanych procesów biznesowych jest kluczowe w procesie analizy. Aby móc zdefiniować cechy nowego systemu i jego funkcje, należy dokładnie zapoznać się z procesami funkcjonującymi w istniejącym systemie informacyjnym banku, wyodrębnić z nich aktywności czy podprocesy, które da się usprawnić lub przebudować, eliminując nieefektywne aktywności i zastępując je nowymi.

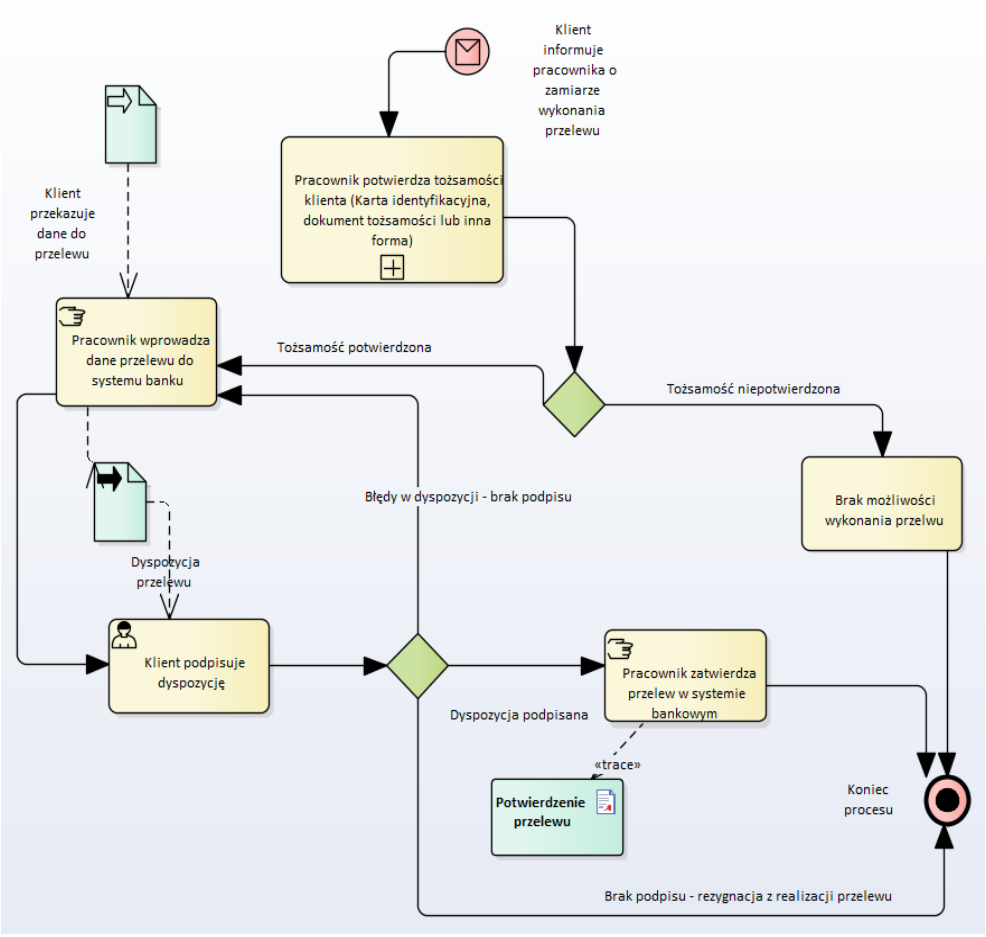
W kolejnych podrozdziałach przeprowadzona zostanie analiza dwóch kluczowych dla tworzonego systemu procesów: przelewu bankowego i lokaty terminowej.

3.1.1. *Przelew bankowy*

Do realizacji przelewu bankowego w oddziale banku klient musi stawić się osobiście, potwierdzić swoją tożsamość i dostarczyć niezbędne dane: numer rachunku nadawcy, numer rachunku odbiorcy, nazwę i adres odbiorcy, kwotę, walutę i tytuł przelewu. Pracownik oddziału wprowadza i zatwierdza przelew w systemie banku, po czym odbywa się wykonanie operacji międzybankowej. Informacja o realizacji przelewu zapisywana jest w centralnej bazie danych banku, a klient dostaje potwierdzenie wykonania przelewu w formie wydruku. Szczegółowo proces realizacji przelewu przedstawia diagram na rysunku 1.

Wąskim gardłem omawianego procesu jest jednoczesna obsługa przez pracownika oddziału tylko jednego klienta. Cały proces może trwać od 5 do 10 minut (średnio 7,5 minuty), z czego wynika, że przy założeniu ciągłej siedmiogodzinnej pracy, jeden pracownik może dziennie obsłużyć: $7 \cdot 60/7{,}5 = 56$ klientów. Zakładając, że w oddziale banku są 4 takie stanowiska i obsada wynosi 100%, to dziennie oddział może zrealizować ok. 200 przelewów. Oczywiście stanowisko w oddziale musi obsługiwać inne procesy (wpłaty, wypłaty, sprawdzenie salda, wydruk historii

rachunku, zakładanie i zrywanie lokat), przez co wynik 200 przelewów dziennie należy zmniejszyć relatywnie w odniesieniu do proporcji innych zadań.



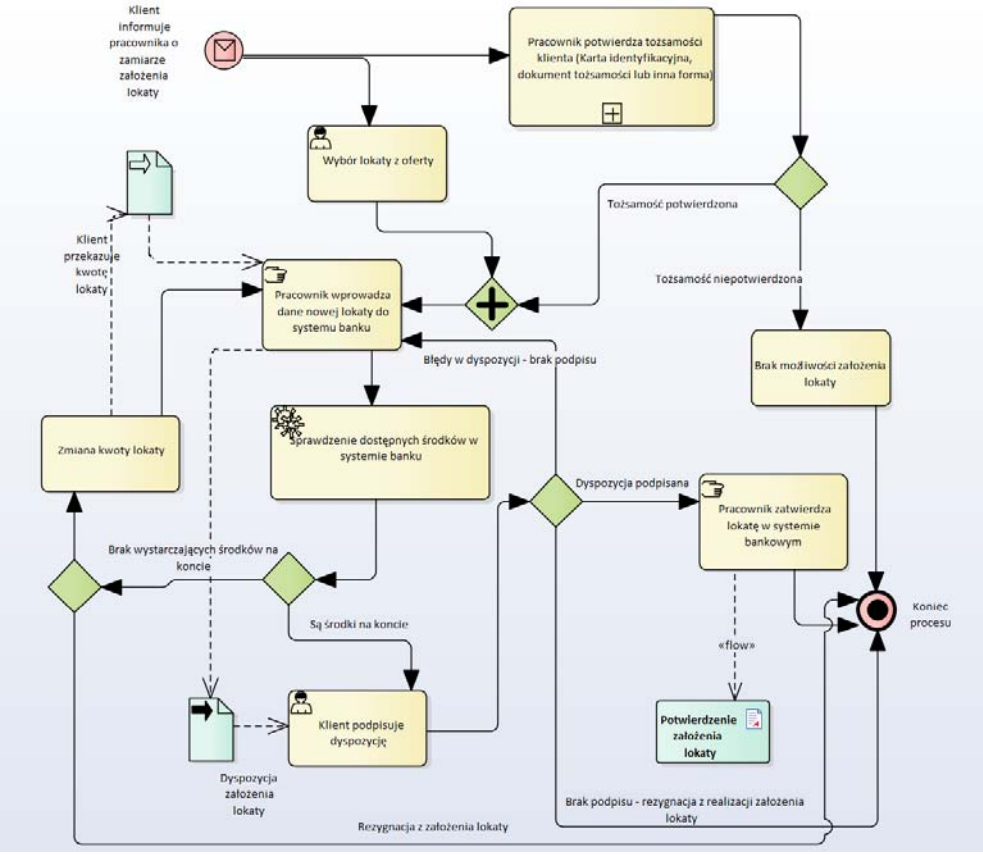
Rysunek 1. Szczegóły procesu realizacji przelewu

W systemie bankowości internetowej można zastąpić synchroniczny proces obsługi przelewu (jeden pracownik – jeden klient) nowym podprocesem, w którym klient bez udziału pracownika banku wprowadza dane przelewu, uwierzytelnia się i zatwierdza przelew. Poprzez platformę internetową, tylko w kontekście realizacji przelewów, bank będzie mógł obsłużyć nieporównywalnie więcej klientów. Sam czas

obsługi może skrócić się nawet do kilkudziesięciu sekund. Można to osiągnąć w przypadku wprowadzenia szablonu przelewów, czyli przelewów zdefiniowanych, gdzie dane do przelewu oprócz kwoty klient przygotowuje zawczasu i zapisuje w systemie. Następnie klient w dowolnym czasie wybiera z listy przelew zdefiniowany, wprowadza kwotę i zatwierdza. Zakładając, że realizacja takiego przelewu będzie zajmować klientowi jedną minutę, to w ciągu 7 godzin można wykonać $7 \cdot 60 = 420$ przelewów. To nie jest wszystko – z założenia system ma być dostępny 24 h/dobę, zatem $24 \cdot 60$ daje 1440 przelewów. Ten wynik nie uwzględnia możliwości równoległego wprowadzania danych i wykonywania przelewów. Przyjmując, że współczesne systemy serwerowe są w stanie przetwarzać tysiące żądań jednocześnie, śmiało można uznać, że osiągnięcie wyniku rzędu miliona przelewów na dobę nie stanowiłoby problemu. Praktycznie eliminuje to omawiane wąskie gardło procesu i można założyć, że system obsłuży dowolną liczbę przelewów w ciągu doby.

3.1.2. ***Lokata terminowa***

Aby założyć lokatę terminową, klient musi również osobiście stawić się w oddziale banku, potwierdzić swoją tożsamość i dostarczyć niezbędne dane pracownikowi oddziału: numer rachunku, z którego mają być pobrane środki na lokatę, kwotę lokaty, rodzaj lokaty, datę uruchomienia lokaty, informacje dodatkowe (np. czy lokata będzie odnawialna). Dyspozycja założenia lokaty powinna zostać zatwierdzona przez klienta własnoręcznym podpisem. Pracownik oddziału wprowadza i zatwierdza założenie lokaty w systemie banku, po czym wykonywany jest, zewnętrzny w stosunku do projektowanego systemu, proces realizacji operacji wewnątrzbankowej. Informacja o założeniu lokaty jest zapisywana w centralnej bazie danych banku, a klient dostaje potwierdzenie założenia lokaty w formie wydruku. Szczegółowo proces zakładania lokaty przestawia rysunek 2.



Rysunek 2. Szczegóły procesu zakładania lokaty

3.2. Rejestr Produktu

Na podstawie Wizji produktu (przedstawionej w poprzednim rozdziale), po jej uszczegółowieniu, utworzony został Rejestr Produktu, zawierający wykaz wszystkich oczekiwanych cech końcowego produktu. Cechy te zostały podzielone na następujące grupy: operacje ogólne, bezpieczeństwo, przelewy, lokaty, rachunki i historia operacji. Rejestr Produktu wraz z opisem cech przedstawiony jest w tabeli 1.

Tabela 1. Rejestr Produktu

Cecha	Opis
Operacje ogólne	
Strona startowa	Wyświetla stronę startową banku
Strona informacji o banku	Wyświetla stronę z informacjami o banku
Strona informacji kontaktowych	Wyświetla stronę z danymi kontaktowymi i adresowymi banku
Kursy walut	Wyświetla aktualne średnie kursy walut NBP
Bezpieczeństwo	
Zmiana hasła użytkownika	Pozwala zmienić użytkownikowi hasło
Odtworzenie hasła użytkownika	Operacja umożliwi odzyskanie dostępu do konta w przypadku, gdy klient zapomni hasła
Zmiana adresu e-mail	Pozwala zmienić klientowi adres e-mail służący do kontaktów z bankiem
Uwierzytelnianie użytkownika	Pozwala na identyfikację i uwierzytelnienie użytkownika uzyskującego dostęp do serwisu, użytkownik nieuwierzytelniony traktowany jest jako gość
Rachunki	
Lista rachunków	Wyświetla listę rachunków klienta
Szczegóły rachunku	Wyświetla szczegółowe informacje o rachunku klienta
Przelewy jednorazowe	
Nowy przelew jednorazowy	Wyświetla formularz z danymi przelewu jednorazowego i pozwala na wprowadzenie danych oraz wykonanie przelewu
Lista złożonych przelewów	Wyświetla listę złożonych przez użytkownika przelewów
Historia Operacji	
Lista operacji	Wyświetla listę operacji historycznych
Szczegóły operacji	Wyświetla szczegóły wybranej operacji
Filtrowanie wykonanych operacji	Pozwala przefiltrować listę operacji według konta, na którym operacja została wykonana
Lokaty	
Lista lokat	Wyświetla listę założonych i aktywnych lokat, pozwala na przejście do szczegółów lokaty
Szczegóły lokaty	Wyświetla szczegóły wybranej lokaty

Założenie lokaty	Wyświetla ofertę lokat i pozwala wybrać typ lokaty oraz konto, z którego mają być pobrane środki, wyświetla formularz z danymi lokaty i pozwala na wprowadzenie danych oraz zatwierdzenie założenia lokaty.
Zerwanie lokaty	Pozwala zerwać dowolną lokatę, jeżeli warunki określone w opisie lokaty na to pozwalają
Przelewy zdefiniowane	
Lista przelewów zdefiniowanych	Wyświetla listę przelewów zdefiniowanych klienta i pozwala na zarządzanie przelewem
Nowy przelew zdefiniowany	Pozwala dodać przelew zdefiniowany
Edycja przelewu zdefiniowanego	Wyświetla formularz z danymi przelewu zdefiniowanego i pozwala na jego modyfikację
Usuwanie przelewu zdefiniowanego	Wyświetla dane przelewu zdefiniowanego oraz monit o potwierdzenie usunięcia przelewu
Wykonanie przelewu zdefiniowanego	Pozwala wybrać przelew zdefiniowany, wyświetla formularz z danymi przelewu i pozwala na wprowadzenie danych oraz wykonanie przelewu

Grupa operacji ogólnych zawiera stronę startową, na której klient może zapoznać się ze sposobem korzystania z serwisu, strony informacyjne oraz średnie kursy walut Narodowego Banku Polskiego (NBP). Następną grupę wymagań stanowią funkcje związane z szeroko rozumianym bezpieczeństwem: uwierzytelnianie użytkownika, autoryzacja oraz operacje dotyczące zarządzania kontem użytkownika. Grupa funkcjonalności o nazwie „Rachunki" ma umożliwić klientowi podgląd listy jego rachunków oraz przeglądanie szczegółowych informacji o rachunku. Grupa przelewów została podzielona na dwie podgrupy „Przelewy jednorazowe" i „Przelewy zdefiniowane" ze względu na to, że stanowią odrębne pojęcia i procesy biznesowe. Podział jest również istotny ze względu na priorytety realizacji tych dwóch grup cech. Ważniejszym elementem są przelewy jednorazowe natomiast funkcjonalności przelewów zdefiniowanych mogą być zrealizowane później. Grupa funkcjonalności „Lokaty" pozwala klientowi na zakładanie i zrywanie lokat terminowych. Ponadto, umożliwia ona podgląd lokat założonych zarówno za pomocą serwisu, jak również w oddziale oraz na zapoznanie się z bieżącą ofertą banku w zakresie warunków i oprocentowania. Funkcje z grupy „Historia operacji" pozwalają klientowi na przeglądanie i wyszukiwanie wszystkich zrealizowanych na kontach klienta operacji,

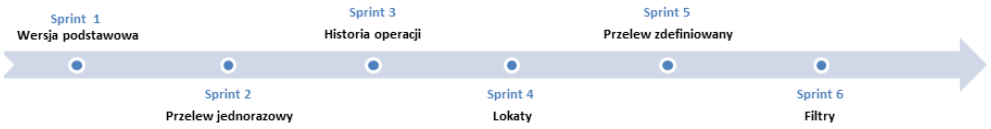
takich jak: przelewy, wpłaty i wypłaty oddziałowe oraz bankomatowe, jak również transakcje bezgotówkowe zrealizowane w punktach sprzedaży.

4. Sprinty

Proces implementacji cech systemu „eBank" został podzielony wstępnie na 6 Sprintów, tak jak pokazano na rysunku 3. Sprinty zostały nazwane zgodnie z cechami, które będą w nich realizowane.

W niniejszym artykule w sposób szczegółowy została opisana realizacja trzech pierwszych Sprintów. Dla Sprintów 4-6 przedstawiono tylko wybrane elementy ich wykonania (szczegóły dotyczące tych Sprintów można znaleźć w pracy [7]).

W każdym Sprincie powstaje działająca w określonym zakresie nowa wersja aplikacji, która może zostać wdrożona. Oczywiście należałoby uzgodnić z zamawiającym, jaki jest minimalny zakres funkcjonalny, na podstawie którego można wydać i wdrożyć system produkcyjnie. Biorąc pod uwagę istotę bankowości internetowej, czyli możliwość wykonywania podstawowych operacji finansowych (przelewów) oraz możliwość przeglądania historii operacji, produkt powstały w pierwszych trzech Sprintach mógłby stanowić pierwsze produkcyjne wydanie aplikacji.



Rysunek 3. Podział procesu wytwarzania na Sprinty

4.1. Sprint 1 – Wersja podstawowa

Celem pierwszego Sprintu jest wytworzenie wersji podstawowej systemu „eBank" zawierającej kluczowe mechanizmy i niezbędne elementy architektury aplikacji, które pozwolą w kolejnych Sprintach rozszerzać funkcjonalność systemu.

Do Rejestru Sprintu 1 zostały wydzielone elementy Rejestru Produktu, których wynikiem jest baza danych klientów banku i ich kont, jak również szkielet aplikacji

wraz z implementacją strony startowej, informacyjnej i kontaktowej, listy i szczegóły kont oraz mechanizm logowania. Rejestr Sprintu 1 przedstawiony został w tabeli 2.

Tabela 2. Rejestr Sprintu 1

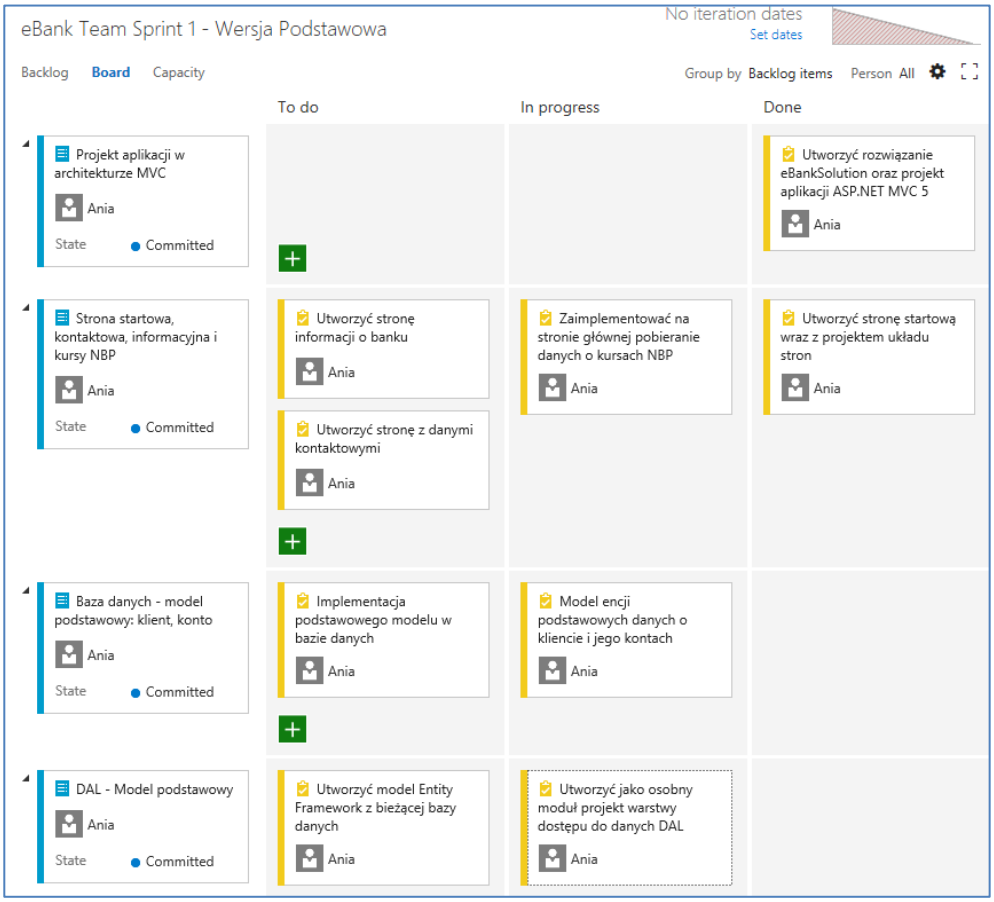
Nr	Element Rejestru Sprintu
1	Projekt aplikacji w architekturze MVC
2	Strona startowa, kontaktowa, informacyjna, kursy walut NBP
3	Baza danych – model podstawowy: klient, konto
4	DAL – model podstawowy
5	Ekran – logowanie klienta
6	Ekran – lista rachunków
7	Ekran – szczegóły rachunku

Elementy Rejestru Sprintu 1 zostały zdekomponowane na konkretne zadania. Rysunek 4 (wykonany z użyciem TFS 2017) przedstawia dekompozycję czterech pierwszych elementów rejestru. Widzimy na nim, w pierwszej kolumnie, elementy Rejestru Sprintu. Dla każdego elementu określone zostały zadania. Dla każdego zadania widoczny jest jego status: do zrobienia (*To d*o), w trakcie wykonywania (*In progress*) i wykonane (*Done*). Dzięki takiej prezentacji każdy członek Zespołu Scrumowego może zapoznać się z postępem prac i według niego planować wykonanie własnych zadań.

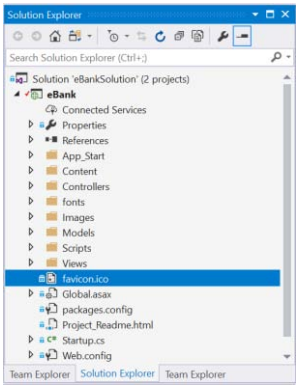
Pierwszym elementem Rejestru Sprintu jest projekt aplikacji www. Został on utworzony jako projekt aplikacji ASP.Net MVC. Szablon aplikacji MVC zawiera szkielet układu strony oraz przykładowy kontroler z trzema widokami, dzięki czemu programista nie musi się skupiać na budowaniu aplikacji od podstaw, tylko może zająć się pisaniem poszczególnych kontrolerów i widoków. Do rozwiązania (eBankSolution) został dodany projekt ASP.NET MVC o nazwie „eBank".

Na rysunku 5 przedstawiona została struktura katalogowa projektu. Odpowiada ona wzorcowi MVC. Folder „Controllers" zawiera klasy kontrolerów, folder „Views" zawiera podkatalogi, a w nich pliki (cshtml) poszczególnych widoków, natomiast w folderze „Models" znajdują się klasy warstwy modelu wygenerowane z szablonu. W folderze tym nie ma klas modelu danych aplikacji „eBank", ponieważ

będą one dołączone jako odrębny moduł (biblioteka) stanowiący warstwę dostępu do danych (ang. *Data Access Layer* – DAL), co znajduje swoje uzasadnienie w fakcie, iż warstwa ta powinna stanowić odrębny komponent systemu, aby ułatwić późniejsze wprowadzanie zmian, szczególnie, że przyjęta metodyka realizacji projektu zakłada z definicji przyrostowe wprowadzanie nowych elementów, co pociąga za sobą zmiany.

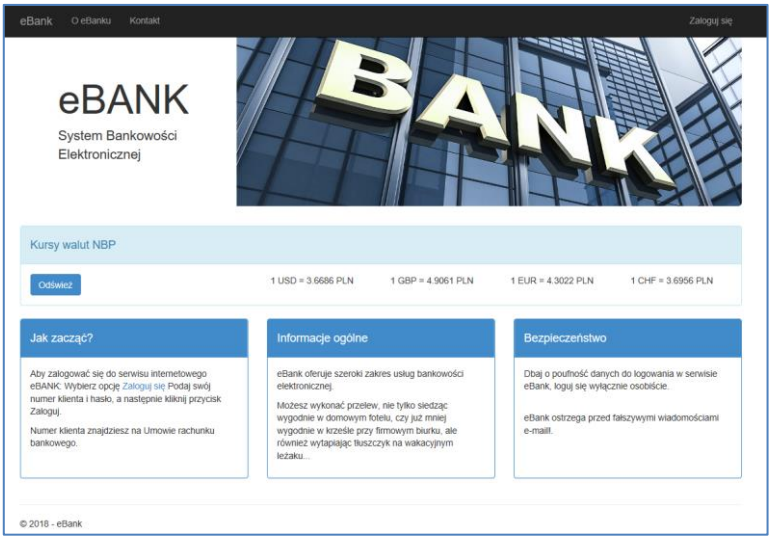


Rysunek 4. Dekompozycja elementów Rejestru Sprintu 1 na zadania



Rysunek 5. Projekt „eBank" w przeglądarce rozwiązania

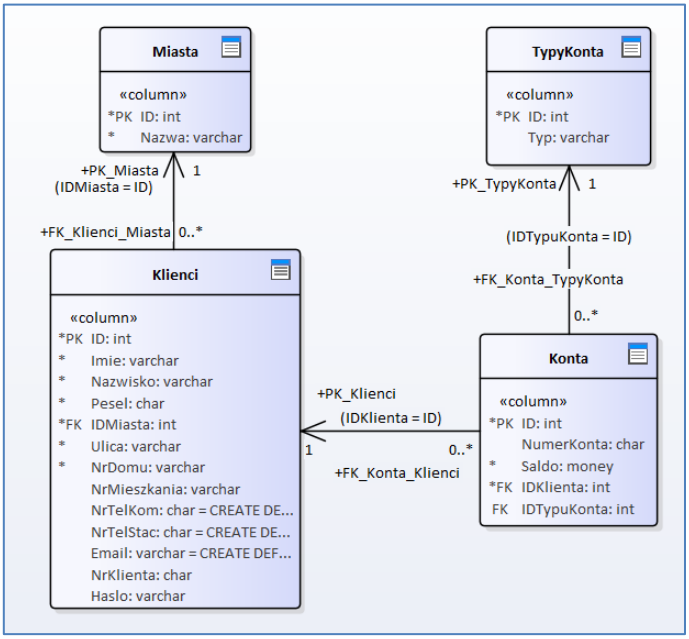
Po utworzeniu projektu aplikacji „eBank" zaplanowano wykonanie strony startowej, w skład której wchodzą trzy podstrony informacyjne i mechanizm pobierania aktualnych średnich kursów walut z serwisu NBP. Prototyp okna strony startowej pokazany jest na rysunku 6.



Rysunek 6. Prototyp okna strony startowej

4.1.1. ***Baza danych***

Niezbędnym elementem, który będzie fundamentem do rozbudowy systemu i implementacji poszczególnych cech systemu, jest początkowa wersja bazy danych. Zadanie realizacji modelu podstawowego zostało podzielone na dwa podzadania: wykonanie modelu danych i implementacja wykonanego modelu na serwerze bazy danych. Wszystkie operacje bankowe związane są z klientem banku i jego rachunkami, w związku z czym podstawowy model danych opiera się o dwie główne tabele Klienci i Konta oraz tabele słownikowe TypyKonta i Miasta. Model danych przedstawiony jest na rysunku 7.

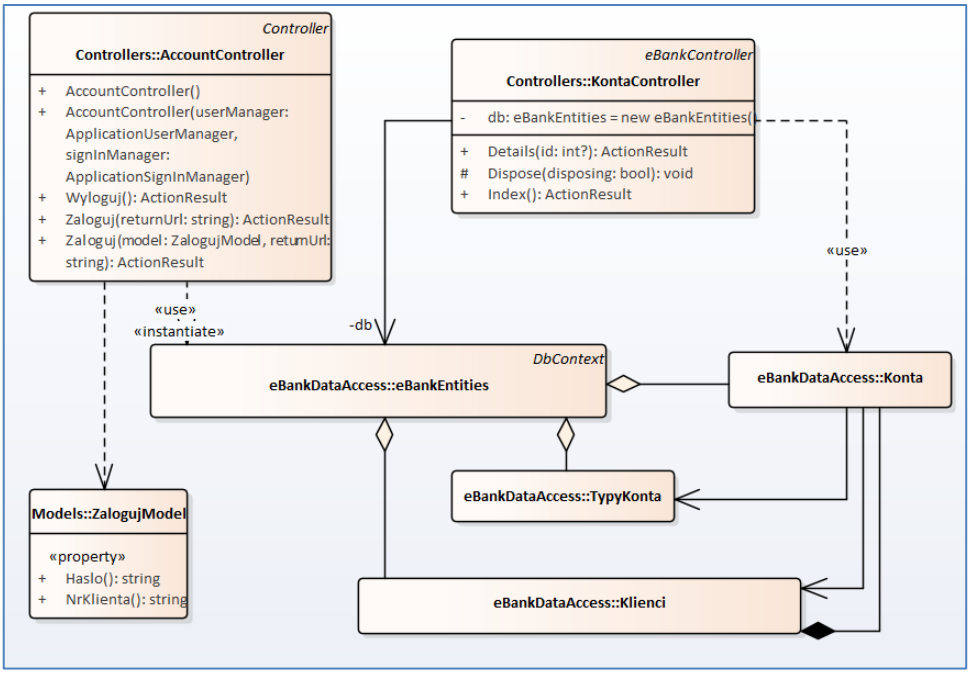


Rysunek 7. Diagram bazy danych – wersja podstawowa

Kolejne zadanie polega na zaprojektowaniu i zaimplementowaniu warstwy dostępu do danych, która odseparuje warstwę prezentacji, czyli aplikację WWW, od warstwy danych zrealizowanej na serwerze SQL. W projekcie wykorzystany został silnik Entity Framework (EF), który pozwolił na utworzenie klas modelu danych w podejściu Database First z zaimplementowanej bazy danych. Wykorzystanie biblioteki EF

pozwala na sprawne modyfikowanie warstwy DAL w przypadku wprowadzania zmian w bazie danych. Po zmianach, jakie będą realizowane w bazie danych w kolejnych Sprintach, wystarczy zaktualizować model EF. W związku z rozdzieleniem architektury aplikacji na warstwę prezentacji i dostępu do danych, operacja ta może być wykonywana niezależnie od prac realizowanych w warstwie prezentacji. Ma to szczególne znaczenie w pracy zespołowej przy realizacji rzeczywistych projektów.

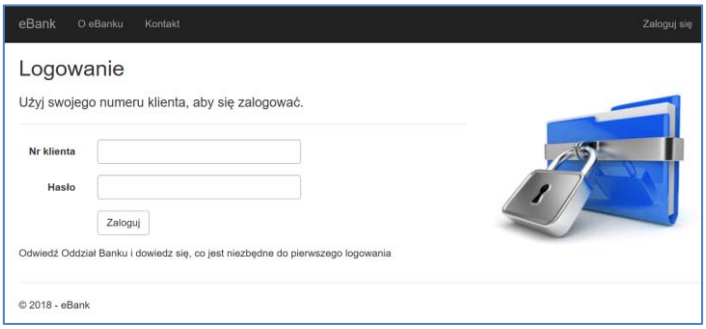
Zaimplementowanie niezbędnych elementów wzorca MVC ułatwia mechanizm generowania kontrolerów i widoków w aplikacji ASP.NET MVC. Pozwala on szybko utworzyć szablon gotowych klas kontrolera i jego metod CRUD (ang. Create, Read, Update, Delete) oraz plików *.cshtml widoków. Kod kontrolerów i widoków zawartych w szablonie należy zmodyfikować według własnych potrzeb. Po skompilowaniu warstwy DAL wygenerowano cztery widoki do podstawowych operacji CRUD oraz kontroler z operacjami na koncie klienta. Zależności pomiędzy klasami kontrolerów oraz klasami wcześniej utworzonego modelu w warstwie DAL przedstawia rysunek 8.



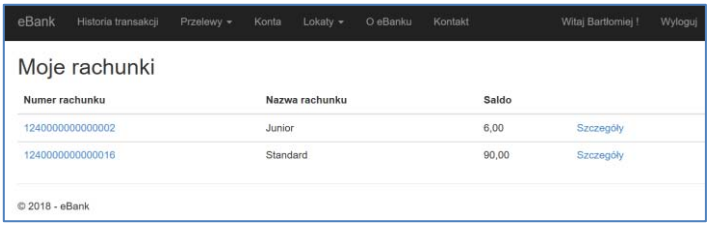
Rysunek 8. Diagram klas kontrolerów i modelu – Sprint 1

Przebudowane zostały metody kontrolerów KontaController i AccountController oraz dostosowano pliki widoków tak, aby uwzględnić wymagania dotyczące interfejsu użytkownika. Na rysunku 9 przedstawiono widoki poszczególnych ekranów implementowanych w Sprincie 1.

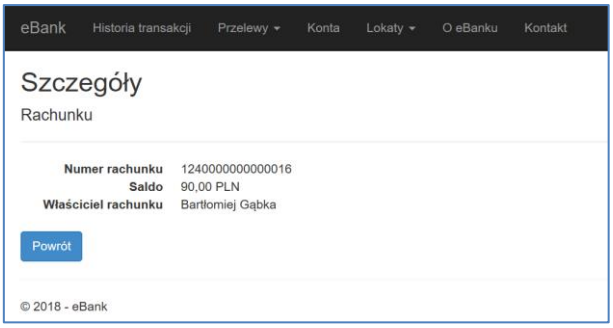
Ekran logowania



Ekran zawierający listę rachunków



Ekran ze szczegółami rachunku



Rysunek 9. Ekrany widoków implementowanych w Sprincie 1

4.1.2. ***Testy funkcjonalne***

Ideą podejść zwinnych jest nieustanne testowanie wykonywanych modułów. Testy dzielą się na wiele poziomów: testy jednostkowe, funkcjonalne, akceptacyjne,

integracyjne, regresji. Są one kluczowym elementem w procesie wytwarzania wysokiej jakości oprogramowania. Na rysunku 10 przedstawiono projekt testu badającego poprawność procesu logowania.

PT001: Logowanie do systemu		
Dane wejściowe	Login: 10000001 Hasło: qaz123	
Oczekiwany rezultat	Test kończy się poprawnym zalogowaniem i wyświetleniem strony startowej z menu rozszerzonym o przelewy, konta, lokaty.	
Warunki początkowe	Użytkownik jest niezalogowany	
Projekt scenariusza testowego		
Scenariusz testowy		**Uwagi**
1. Użytkownik uruchamia stronę startową banku		http://localhost:55419/
2. System wyświetla stronę startową		
3. Użytkownik uruchamia funkcję menu „Zaloguj się”		Link „Zaloguj się” po prawej stronie menu głównego
4. System wyświetla stronę logowania		
5. Użytkownik wprowadza dane logowania i zatwierdza przyciskiem „Zaloguj”		Patrz wiersz „dane wejściowe”
6. System uruchamia stronę startową i wyświetla menu rozszerzone o funkcje użytkownika zalogowanego		
Realizacja		
Wynik (pozytywny, negatywny, z uwagami)	Pozytywny	
Uwagi		

Rysunek 10. Przypadek testowy 001 – Logowanie do systemu

Test logowania zrealizowany został pomyślnie, natomiast system po kilkunastu minutach bezczynności wylogował użytkownika, co spowodowało błąd odczytu Id-Klienta z obiektu sesji (Session) przy próbie wykonania jakiejkolwiek operacji. Ta sytuacja wymusiła w kolejnym Sprincie wykonanie mechanizmu przekierowującego żądanie do okna logowania w przypadku wykrycia zakończenia sesji.

4.2. Sprint 2 – Przelew jednorazowy

Celem drugiego Sprintu było zaimplementowanie kluczowej funkcji serwisu „eBank”, jaką jest realizacja przelewów jednorazowych. Do Rejestru Sprintu 2 przydzielone

zostały elementy Rejestru Produktu z grupy „Przelewy jednorazowe" zapewniające dodanie do struktury bazy danych oraz warstwy DAL niezbędnych tabel i klas, które pozwolą na ewidencję i realizację przelewów, oraz elementy zakładające realizację interfejsu użytkownika w tym zakresie. Rejestr Sprintu 2 jest przedstawiony w tabeli 3.

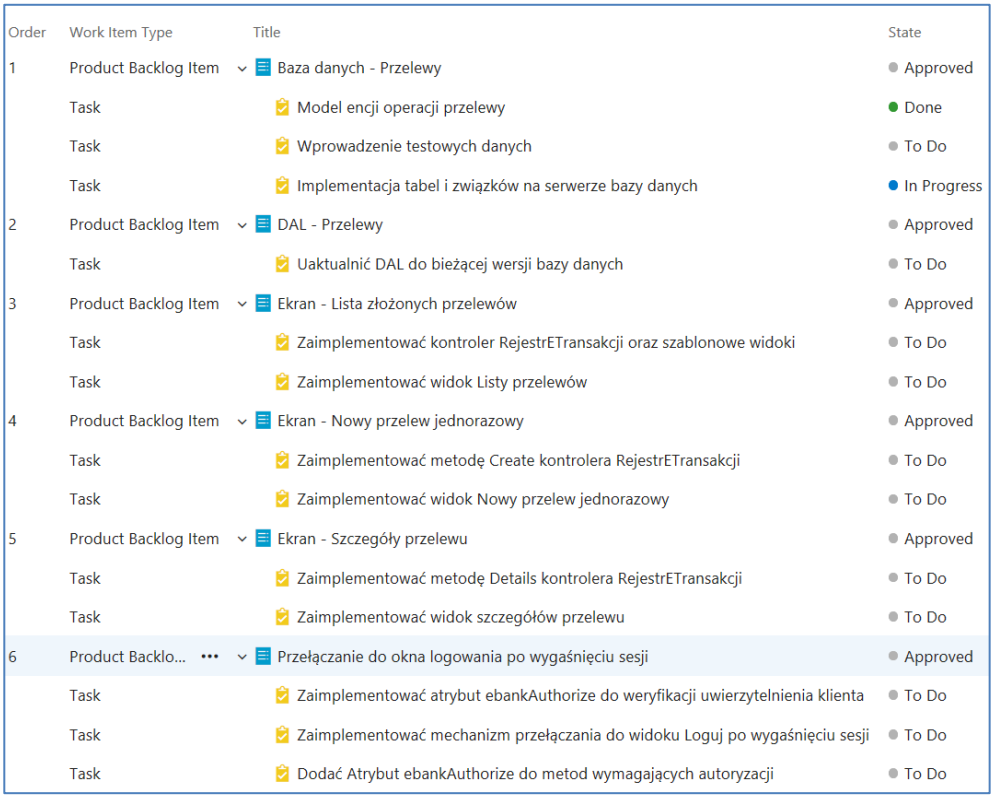
Tabela 3. Rejestr Sprintu 2

Nr	Element Rejestru Sprintu
1	Baza danych – Przelewy
2	DAL – Przelewy
3	Ekran – Lista złożonych przelewów
4	Ekran – Nowy przelew jednorazowy
5	Ekran – Szczegóły przelewu
6	Przełączanie do okna logowania po wygaśnięciu sesji

Aby zaimplementować funkcjonalności cechy „Nowy przelew jednorazowy" oraz „Lista złożonych przelewów", należy ustalić z Właścicielem Produktu dokładny przebieg realizacji przelewu oraz jakie informacje powinny znaleźć się na liście złożonych przelewów.

W celu wykonania przelewu, użytkownik musi być zalogowany do aplikacji, musi wybrać opcję przelewy z menu na stronie startowej, a następnie dostarczyć niezbędne dane omówione na etapie analizy biznesowej procesu. Po wypełnieniu danych, operacja powinna zostać zatwierdzona przez użytkownika. Na tym etapie mogą zostać zastosowane dodatkowe zabezpieczenia w postaci jednorazowego kodu przekazanego użytkownikowi w postaci karty z kodami przesłanej pocztą tradycyjną lub pojedynczego kodu wygenerowanego w trakcie przeprowadzania operacji i przesłanego użytkownikowi w postaci wiadomości tekstowej – sms. Elementy autoryzacji pojedynczej operacji nie zostały na tym etapie jeszcze zdefiniowane, lecz należy je uwzględnić w architekturze i implementacji kodu. Po wykonaniu przelewu użytkownik powinien mieć możliwość uzyskania potwierdzenia przeprowadzonej operacji z wyszczególnieniem wszystkich niezbędnych danych.

Zadania wyodrębnione dla elementów Rejestru Sprintu 2 zaprezentowane zostały na rysunku 11 (zrzut z TFS 2017).



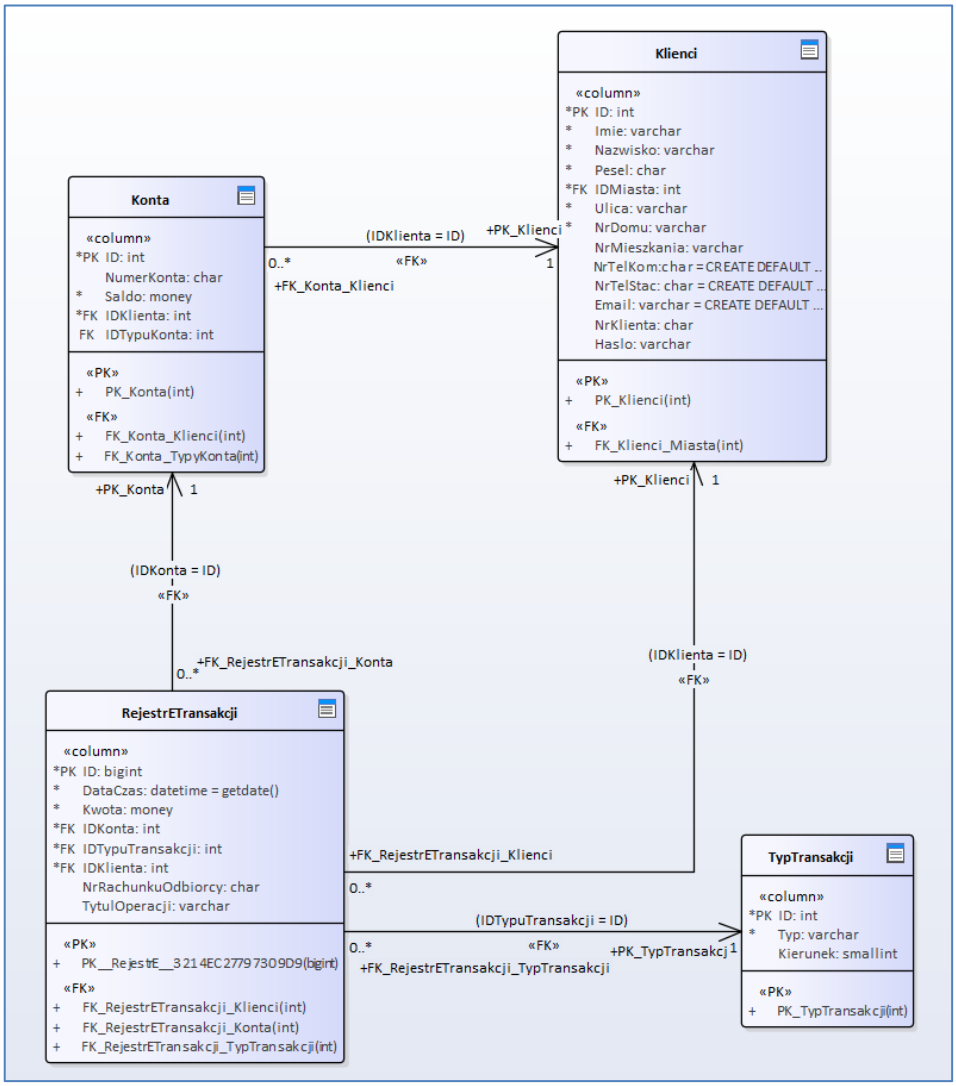
Order	Work Item Type	Title	State
1	Product Backlog Item	Baza danych - Przelewy	Approved
	Task	Model encji operacji przelewy	Done
	Task	Wprowadzenie testowych danych	To Do
	Task	Implementacja tabel i związków na serwerze bazy danych	In Progress
2	Product Backlog Item	DAL - Przelewy	Approved
	Task	Uaktualnić DAL do bieżącej wersji bazy danych	To Do
3	Product Backlog Item	Ekran - Lista złożonych przelewów	Approved
	Task	Zaimplementować kontroler RejestrETransakcji oraz szablonowe widoki	To Do
	Task	Zaimplementować widok Listy przelewów	To Do
4	Product Backlog Item	Ekran - Nowy przelew jednorazowy	Approved
	Task	Zaimplementować metodę Create kontrolera RejestrETransakcji	To Do
	Task	Zaimplementować widok Nowy przelew jednorazowy	To Do
5	Product Backlog Item	Ekran - Szczegóły przelewu	Approved
	Task	Zaimplementować metodę Details kontrolera RejestrETransakcji	To Do
	Task	Zaimplementować widok szczegółów przelewu	To Do
6	Product Backlo...	Przełączanie do okna logowania po wygaśnięciu sesji	Approved
	Task	Zaimplementować atrybut ebankAuthorize do weryfikacji uwierzytelnienia klienta	To Do
	Task	Zaimplementować mechanizm przełączania do widoku Loguj po wygaśnięciu sesji	To Do
	Task	Dodać Atrybut ebankAuthorize do metod wymagających autoryzacji	To Do

Rysunek 11. Dekompozycja elementów Rejestru Sprintu 2 na zadania

Działania w Sprintcie 2 polegały na zamodelowaniu niezbędnych, w stosunku do zaimplementowanego modelu podstawowego, encji i związków potrzebnych do realizacji przelewów oraz zaimplementowaniu powstałego modelu w bazie danych, uaktualnieniu warstwy DAL oraz realizacji logiki i odpowiednich widoków. Ostatnim elementem Rejestru Sprintu 2 było utworzenie mechanizmu obsługującego sesję użytkownika i przekierowanie do okna logowania.

4.2.1. *Baza danych*

Powstała w Sprincie 1 baza danych została rozszerzona o tabelę RejestrETransakcji, która będzie ewidencjonowała wszystkie wykonane przelewy, co zostało pokazane na rysunku 12.

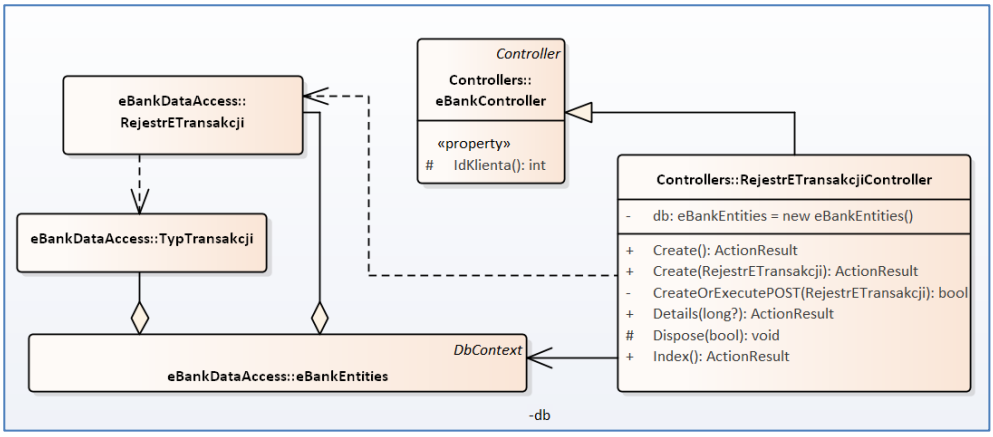


Rysunek 12. Diagram bazy danych – przelewy

RejestrETransakcji jest w relacji z wcześniej utworzonymi tabelami Konta i Klienci oraz nową encją słownikową TypTransakcji. Zaimplementowany model bazy pozwoli na ewidencję przelewów realizowanych za pośrednictwem systemu „eBank" oraz na realizację kolejnego elementu systemu, czyli „Listy zrealizowanych przelewów".

4.2.2. ***Ekrany***

Realizację elementu „Ekran – Lista złożonych przelewów” rozpoczęto od wygenerowania kontrolera i widoków z szablonu oraz wprowadzenia niezbędnych zmian w kodzie metody Index() i w widoku Index.cshtml. Na rysunku 13 przedstawiono diagram nowo utworzonych klas. Element „Ekran – Nowy przelew jednorazowy” wymaga realizacji dwóch zadań: zaimplementowania metod Create() kontrolera RejestrETransakcjiController i modyfikacji widoku Create.cshtml zgodnie z założeniami. Element „Ekran – Szczegóły przelewu” wymaga modyfikacji metody Details() kontrolera i dostosowania pliku widoku Details.cshtml.



Rysunek 13. Diagram klas kontrolerów i modelu – Sprint 2

Klasy kontrolerów domyślnie dziedziczą po klasie bazowej Controller z biblioteki EF. W przypadku projektu „eBank” pojawił się pewien problem, polegający na konieczności wykonywania niemal wszelkich operacji w kontekście zalogowanego klienta. Należało zapewnić ciągły dostęp do klucza głównego ID klienta w ramach jego sesji. Zapamiętanie klucza głównego klienta banku zostało zrealizowane za pomocą obiektu Session klasy Controller. Spowodowało to wykonywanie tego samego kodu, czyli odczytu ID z sesji i przekierowania do strony logowania w przypadku zakończenia sesji użytkownika w pozostałych metodach kontrolera. Można się było spodziewać, że pozostałe funkcje implementowane w późniejszym czasie,

również będą wymagały takiego mechanizmu. Aby nie powtarzać tego samego kodu w wielu miejscach, co byłoby niezgodne z ideą programowania obiektowego, zastosowano pośrednią klasę bazową dla tworzonych kontrolerów dziedziczącą po klasie Controller, która implementowała omawianą funkcjonalność jako publiczną właściwość IdKlienta (rysunek 14).

```
public class eBankController: Controller
{
    protected int IdKlienta
    {
        get
        {
            return (int)ViewBag.IdKlienta;
        }
    }
}
```

Rysunek 14. Kod klasy eBankController

4.2.3. *Przełączenie do okna logowania po wygaśnięciu sesji*

Większość funkcji dostępnych dla klienta banku wykonuje się w kontekście zalogowanego użytkownika. Niemalże każda metoda kontrolera musi pobrać ID zalogowanego klienta, które przechowywane jest w obiekcie Session. Problemem było wygaśnięcie sesji podczas wykonywania operacji. Wówczas kontroler nie odnajdował ID klienta w sesji, co generowało wyjątek i brak możliwości odświeżenia ekranu.

W celu rozwiązania tego problemu, aby kompleksowo zapewnić autoryzację klienta oraz przekierowanie do okna logowania w przypadku wygaśnięcia sesji, zastosowano mechanizm filtrów ASP.NET MVC, który pozwala na dodawanie pewnych funkcjonalności przy pomocy atrybutów klasy lub metody.

Mechanizm ten zapamiętuje adres URL, z którego przekierowanie nastąpiło, co pozwala wrócić do wcześniej wykonywanej operacji po poprawnym zalogowaniu. Atrybut będzie dodawany przed każdą metodą kontrolerów, której wykonanie wymaga zalogowanego użytkownika, co zapewni uproszczenie implementacji kontrolerów i kompleksowe rozwiązanie problemu. Implementację atrybutu przedstawia rysunek 15.

```
class ebankAuthorizeAttribute: ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        int id=0;
        try
        {
            id = (filterContext.HttpContext.Session["Klient"] as Logowanie_Result).ID;

            base.OnActionExecuting(filterContext);
        }
        catch
        {
            filterContext.Result= new RedirectToRouteResult(
                new RouteValueDictionary {
                    { "controller", "Account" },
                    { "action", "Zaloguj" },
                    { "returnUrl", filterContext.HttpContext.Request.RawUrl} });
        }
        filterContext.Controller.ViewBag.IdKlienta = id;
    }
}
```

Rysunek 15. Klasa filtra autoryzacji ebankAuthorizeAttribute

4.2.4. *Testy funkcjonalne*

Rysunek 16 przedstawia projekt przypadku testowego 002, którego celem jest zweryfikowanie funkcji realizacji przelewu zdefiniowanego.

PT002: Realizacja przelewu jednorazowego		
Dane wejściowe	Login: 10000001 Hasło: qaz123 Kwota:150 Konto: 1240000000000001 Numer Rach. Odbiorcy: 1340000000435213 Tytułem: „Opłata za wycieczkę”	
Oczekiwany rezultat	Test kończy się poprawnym zrealizowaniem przelewu i wyświetleniem strony Lista przelewów. Saldo konta jest pomniejszone o kwotę przelewu.	
Warunki początkowe	Użytkownik jest zalogowany z uruchomioną stroną startową	
Projekt scenariusza testowego		
Scenariusz testowy		**Uwagi**
1.	Użytkownik uruchamia łącze „Przelewy->Nowy przelew” z menu aplikacji	
2.	System wyświetla stronę nowy przelew	
3.	Użytkownik uzupełnia dane przelewu i wciska przycisk „Wykonaj”	Patrz wiersz „dane wejściowe”
4.	System rejestruje przelew w bazie danych obciążając wybrane konto klienta kwotą przelewu	
5.	System wyświetla okno „Moje przelewy” gdzie widnieje wiersz z wykonanym przelewem.	
6.	Użytkownik sprawdza saldo konta wykorzystanego do realizacji przelewu wybierając z menu opcję „Konta”	Przy koncie widoczne jest bieżące saldo po obciążeniu kwotą przelewu.
Realizacja		
Wynik (pozytywny, negatywny, z uwagami)	pozytywny	
Uwagi		

Rysunek 16. Przypadek testowy 002 – Realizacja przelewu jednorazowego

4.3. Sprint 3 – Historia operacji

Celem trzeciego Sprintu było zaimplementowanie kolejnej kluczowej funkcji serwisu „eBank", jaką jest historia operacji.

Do Rejestru Sprintu przydzielone zostały elementy zapewniające dodanie do struktury bazy danych oraz warstwy DAL niezbędnych tabel i klas, które pozwolą na ewidencję wszystkich pozostałych operacji w stosunku do operacji wykonywanych online (oddziałowych oraz kartowych), oraz elementy zakładające realizację interfejsu użytkownika w tym zakresie. Elementy Rejestru Sprintu 3 są przedstawione w tabeli 4.

Tabela 4. Rejestr Sprintu 3

Nr	Element Rejestru Sprintu
1	Baza danych – Historia operacji
2	DAL – Historia operacji
3	Ekran – Lista historii operacji
4	Ekran – Szczegóły operacji

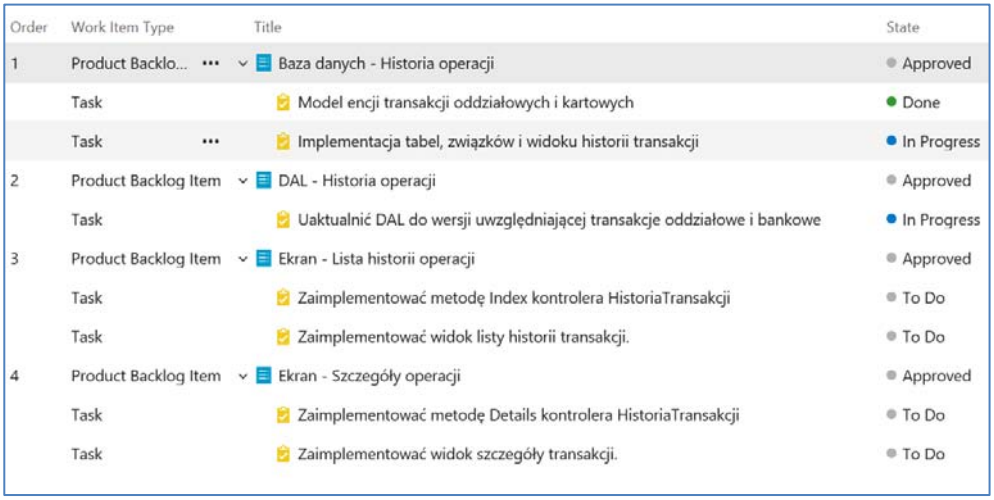
Aby można było zaimplementować funkcjonalność cechy „Historia operacji", należy ustalić z właścicielem produktu, jakie informacje tego typu raport powinien zawierać.

W skład operacji prezentowanych w historii operacji wchodzą wszystkie typy transakcji dostępne w banku: transakcje przeprowadzane online (przelewy), transakcje oddziałowe (wpłaty, wypłaty, przelewy) oraz transakcje kartowe (wypłaty i wpłaty bankomatowe, transakcje w punktach sprzedaży POS). Zbiorczy raport będący historią transakcji, powinien zawierać następujące pola: kwota, typ transakcji, data operacji, numer rachunku odbiorcy, tytuł operacji. Ważne jest, aby w historii transakcji były widoczne wszystkie transakcje przeprowadzane ze wszystkich kont przypisanych do danego klienta. Warto byłoby też dołączyć funkcjonalność dającą możliwość wyświetlenia szczegółów każdej transakcji – takie szczegółowe informacje pojawiałyby się po przejściu na kolejny ekran i obejmowałyby, dodatkowo w stosunku

do informacji wyświetlonych na liście transakcji, numer rachunku odbiorcy i datę wzbogaconą o dokładny czas transakcji.

Elementy Rejestru Sprintu 3 zostały podzielone na zadania, jak przedstawiono na rysunku 17 (zrzut z TFS 2017).

Realizacja wybranych elementów polegała na zamodelowaniu niezbędnych, w stosunku do zaimplementowanego w poprzednich Sprintach modelu, encji i związków potrzebnych do realizacji historii operacji oraz na zaimplementowaniu powstałego modelu w bazie danych, uaktualnieniu warstwy DAL oraz realizacji logiki i odpowiednich widoków.

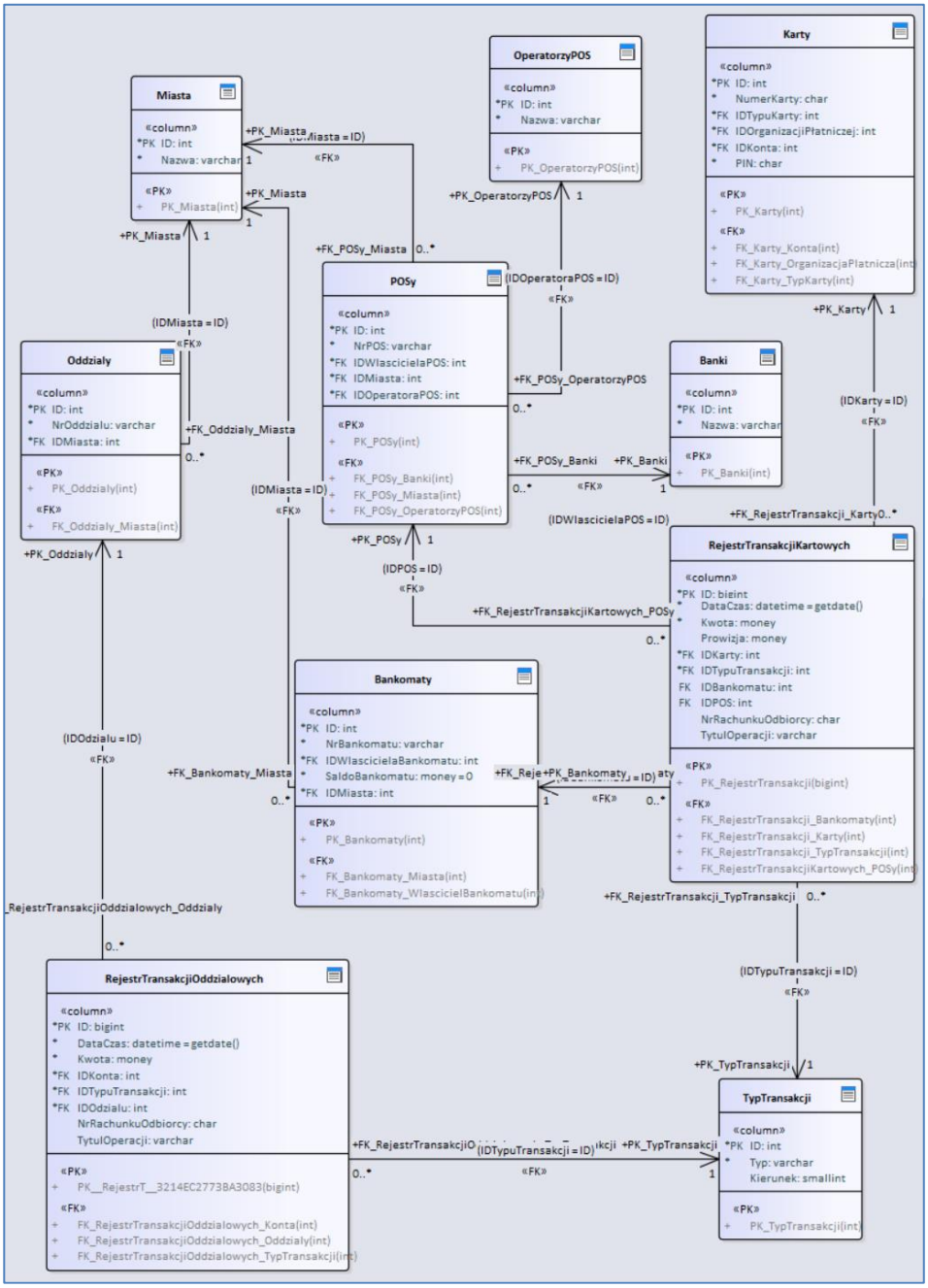


Order	Work Item Type	Title	State
1	Product Backlo...	Baza danych - Historia operacji	Approved
	Task	Model encji transakcji oddziałowych i kartowych	Done
	Task	Implementacja tabel, związków i widoku historii transakcji	In Progress
2	Product Backlog Item	DAL - Historia operacji	Approved
	Task	Uaktualnić DAL do wersji uwzględniającej transakcje oddziałowe i bankowe	In Progress
3	Product Backlog Item	Ekran - Lista historii operacji	Approved
	Task	Zaimplementować metodę Index kontrolera HistoriaTransakcji	To Do
	Task	Zaimplementować widok listy historii transakcji.	To Do
4	Product Backlog Item	Ekran - Szczegóły operacji	Approved
	Task	Zaimplementować metodę Details kontrolera HistoriaTransakcji	To Do
	Task	Zaimplementować widok szczegóły transakcji.	To Do

Rysunek 17. Dekompozycja elementów Rejestru Sprintu 3 na zadania

4.3.1. ***Baza danych***

Część bazy danych powstała w Sprincie 3 jest przedstawiona na rysunku 18. Są tam widoczne 2 główne tabele: RejestrTransakcjiKartowych oraz RejestrTransakcjiOddzialowych, które będą ewidencjonowały wszystkie transakcje wykonane w oddziałach banku, takie jak wpłaty, wypłaty, przelewy oraz transakcje wykonane przy pomocy kart płatniczych, czyli wpłaty i wypłaty bankomatowe, jak również transakcje w punktach sprzedaży POS. Uwzględnione są także tabele słownikowe: Banki, Bankomaty, Karty, Oddzialy, OperatorzyPOS i POSy.



Rysunek 18. Diagram bazy danych – historia transakcji

Ze względu na czytelność diagramu pominięto mniej istotne tabele (również dodane do bazy w Sprincie 3): NiebankowiOperatorzyATM, OrganizacjaPlatnicza, WlascicielBankomatu, Prowizje i TypKarty. Ponadto, na diagramie zostały pokazane związki nowo tworzonych tabel z tabelami utworzonymi w trakcie poprzednich sprintów, czyli Miasta i TypTransakcji.

Szczególnym elementem obsługi historii transakcji było zaprojektowanie i implementacja mechanizmu agregującego wszystkie typy transakcji: eTransakcje, transakcje kartowe i oddziałowe. Zadanie zrealizowano za pomocą widoku (perspektywy) bazy danych, w którym zastosowano operację UNION języka SQL ze względu na agregację danych z różnych tabel (rysunek 19).

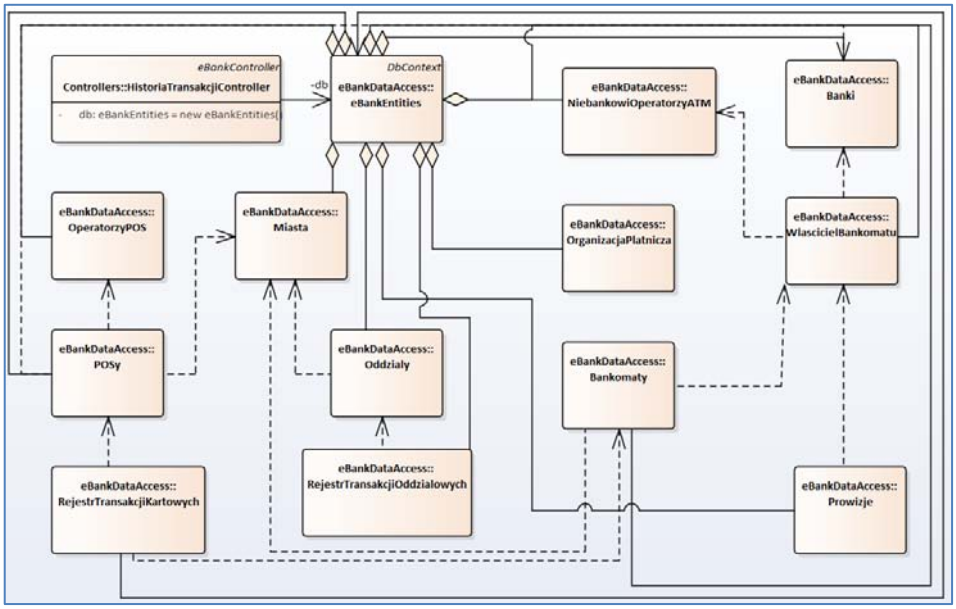
```
CREATE VIEW [dbo].[HistoriaTransakcji]
AS
SELECT        ret.ID + 100000 AS ID, ret.DataCzas, ret.Kwota * tt.Kierunek AS Kwota,
              tt.Typ, tt.Kierunek, ret.NrRachunkuOdbiorcy, ret.TytulOperacji,
              ret.IDKonta, ret.IDKlienta
FROM          dbo.RejestrETransakcji ret INNER JOIN
              dbo.TypTransakcji tt ON ret.IDTypuTransakcji = tt.ID
UNION

    SELECT        rtk.ID + 200000 AS ID, rtk.DataCzas, rtk.Kwota * tt.Kierunek AS Kwota,
                  tt.Typ, tt.Kierunek, rtk.NrRachunkuOdbiorcy,rtk.TytulOperacji,
                  k.ID AS IDKonta, ko.IDKlienta
    FROM          dbo.RejestrTransakcjiKartowych rtk INNER JOIN
                              dbo.Karty k ON k.ID = rtk.IDKarty INNER JOIN
                              Konta ko ON ko.ID = k.IDKonta INNER JOIN
                              dbo.TypTransakcji tt ON rtk.IDTypuTransakcji = tt.ID
    UNION

        SELECT        rto.ID + 300000 AS ID, rto.DataCzas, rto.Kwota * tt.Kierunek AS Kwota,
                      tt.Typ, tt.Kierunek,  rto.NrRachunkuOdbiorcy,rto.TytulOperacji,
                      k.ID AS IDKonta, k.IDKlienta
        FROM            dbo.RejestrTransakcjiOddzialowych rto INNER JOIN
                                  dbo.Konta k ON k.ID = rto.IDKonta INNER JOIN
                                  dbo.TypTransakcji tt ON rto.IDTypuTransakcji = tt.ID
```

Rysunek 19. Kod SQL widoku HistoriaTransakcji

Nowe klasy modelu danych w relacji z kontekstem bazy danych eBankEntities przedstawia rysunek 20.

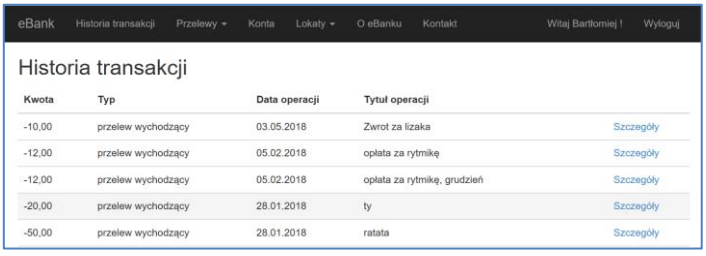


Rysunek 20. Diagram klas modelu warstwy DAL – Sprint 3

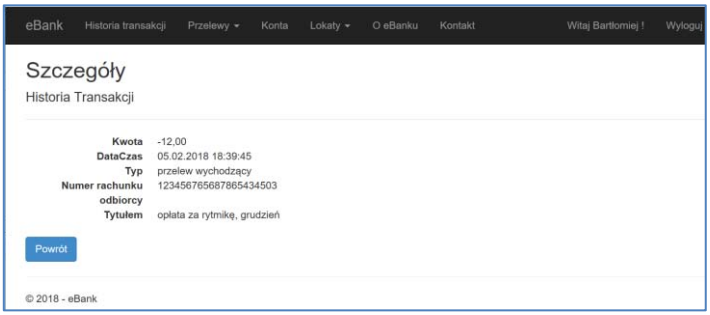
4.3.2. ***Ekrany***

Podobnie jak we wcześniejszych sprintach nowe dane modelu wymagały utworzenia odpowiadających im kontrolerów. W tym przypadku do implementacji historii i szczegółów transakcji utworzony został jedynie kontroler HistoriaTransakcjiController wraz z szablonowymi widokami. Zarówno metody kontrolera, jak i pliki widoku zostały zmodyfikowane, aby spełnić wymagania wyżej wymienionych funkcjonalności. Zaimplementowane ekrany „Historia transakcji" oraz „Szczegóły transakcji" zilustrowano na rysunku 21.

Ekran historii transakcji



Ekran szczegółów historii transakcji



Rysunek 21. Ekrany dodane w Sprincie 3.

Ekran historii transakcji wyświetla listę wykonanych transakcji w formie tabelarycznej oraz umożliwia przejście do szczegółów transakcji. Ekran szczegółów transakcji wyświetla wszystkie dane dotyczące wybranej transakcji i pozwala powrócić do listy historii transakcji.

4.3.3. *Testy funkcjonalne*

Rysunek 22 przedstawia projekt testów funkcjonalności „Lista historii transakcji". Test przebiegł pozytywnie, uwaga dotycząca filtrowania po koncie klienta zostanie uwzględniona i zrealizowana w jednym z kolejnych sprintów.

PT003: Lista historii transakcji		
Dane wejściowe	Login: 10000001 Hasło: qaz123	
Oczekiwany rezultat	Test kończy się wyświetleniem wszystkich transakcji danego klienta.	
Warunki początkowe	Użytkownik jest zalogowany z uruchomioną stroną startową	
Projekt scenariusza testowego		
Scenariusz testowy		**Uwagi**
7. Użytkownik uruchamia łącze „Historia transakcji" z menu aplikacji		
8. System wyświetla stronę z listą transakcji klienta		
Realizacja		
Wynik (pozytywny, negatywny, z uwagami)	Pozytywny, z uwagami	
Uwagi	Przydałoby się filtrowanie po numerze konta klienta, ponieważ wyświetlane są transakcje z wszystkich kont klienta jednocześnie.	

Rysunek 22. Przypadek testowy 003 – Lista historii transakcji

4.4. Sprint 4 – Lokaty

Celem czwartego Sprintu było zaimplementowanie kolejnej funkcji serwisu „eBank", – zakładania lokat (jak już wspomniano wcześniej, w niniejszym artykule Sprinty 4, 5 i 6 zostaną tylko zarysowane, natomiast ich szczegóły można znaleźć w [7]).

Do Rejestru Sprintu 4 (tabela 5) przydzielone zostały elementy zapewniające dodanie do struktury bazy danych oraz warstwy DAL niezbędnych tabel i klas, które pozwolą na ewidencję i zakładanie lokat oraz elementy zakładające realizację interfejsu użytkownika w tym zakresie.

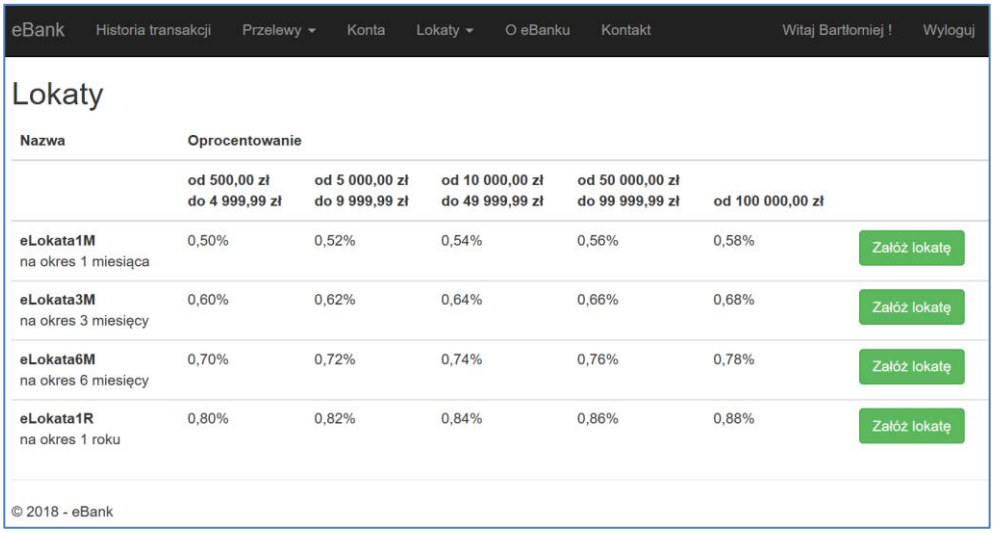
Tabela 5. Rejestr Sprintu 4

nr	Element Rejestru Sprintu
1	Baza danych – Lokaty
2	DAL – Lokaty
3	Ekran – Nowa lokata terminowa
4	Ekran – Szczegóły lokaty
5	Ekran – Lista lokat terminowych
6	Ekran – Zerwij lokatę

Jak wynika z Rejestru Sprintu, znaczna część pracy skupia się wokół interfejsu użytkownika. Wynika to z faktu, że klient wykonuje aż 4 operacje związane z lokatami.

Realizacja elementów polega na zamodelowaniu niezbędnych, w stosunku do zaimplementowanego w poprzednich Sprintach modelu, encji i związków potrzebnych do realizacji zakładania lokat, a także zaimplementowaniu powstałego modelu w bazie danych, uaktualnieniu warstwy DAL oraz realizacji logiki i odpowiednich widoków.

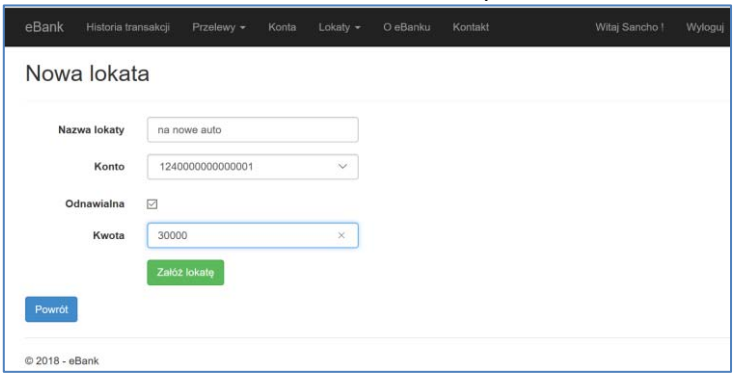
Najciekawszym zagadnieniem realizacji Sprintu 4, było wygenerowanie widoku oferty lokat (rysunek 23). Algorytm zastosowany w widoku oferty lokaty generuje odpowiednio kolumny odpowiadające zakresom kwot lokaty oraz wiersze odpowiadające zdefiniowanym w bazie okresom lokat.



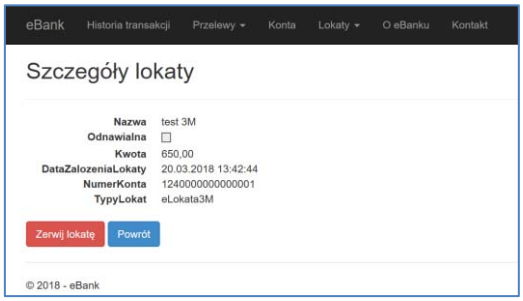
Rysunek 23. Ekran oferty lokat w operacji Nowa lokata

Po wybraniu oferty klient przechodzi do okna nowej lokaty. Ekran „Szczegóły lokaty" wyświetla szczegółowe dane lokaty oraz pozwala zerwać lokatę za pomocą przycisku. Ekran „Moje lokaty" wyświetla w postaci tabelarycznej założone przez klienta lokaty. Z tego poziomu można przejść do szczegółów lokaty albo ją zerwać. Ekran „Zerwanie lokaty" pełni funkcję dodatkowego zabezpieczenia, gdzie klient musi potwierdzić chęć zerwania lokaty. Widoki powyższych okien zaprezentowano na rysunku 24.

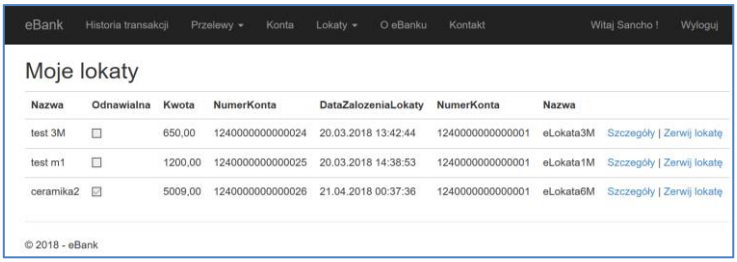
Ekran zakładania lokaty



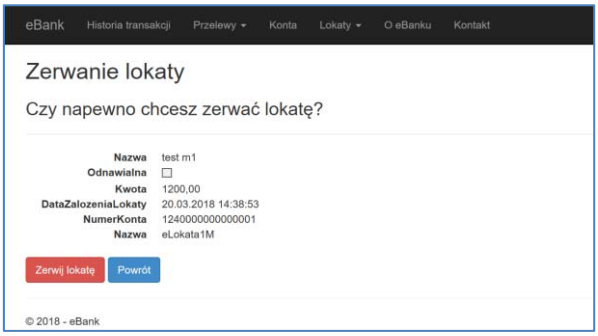
Ekran wyświetlający szczegóły lokaty



Ekran prezentujący lokaty klienta



Ekran zerwania lokaty



Rysunek 24. Ekrany obsługi lokat utworzone w Sprincie 4

4.5. Sprint 5 – Przelew zdefiniowany

Celem piątego Sprintu było zaimplementowanie kolejnej funkcji serwisu „eBank", jaką są przelewy zdefiniowane. Do Rejestru Sprintu wydzielone zostały elementy zapewniające dodanie do struktury bazy danych oraz warstwy DAL niezbędnych

tabel i klas, które pozwolą na utworzenie oraz wykonanie przelewu zdefiniowanego, a także elementy zakładające realizację interfejsu użytkownika w tym zakresie.

Tabela 6. Rejestr Sprintu 5

Nr	Element Rejestru Sprintu
1	Baza danych – Przelew zdefiniowany
2	DAL – Przelew zdefiniowany
3	Ekran – Lista przelewów zdefiniowanych
4	Ekran – Nowy przelew zdefiniowany
5	Bug – „Nowy przelew" lista wyboru konta zawiera niepotrzebnie numery kont lokat
6	Ekran – Edytuj przelew zdefiniowany
7	Ekran – Usuń przelew zdefiniowany
8	Ekran – Wykonaj przelew zdefiniowany

Jak przedstawiono w tabeli 6, w Sprincie 5 uwzględniono błąd (ang. *Bug*), który polegał na tym, że na listach kont wyświetlały się robocze rachunki lokat obok standardowych rachunków bankowych, co mogłoby spowodować nieoczekiwane zachowanie systemu np. założenie lokaty przy wykorzystaniu środków z konta innej lokaty.

Realizacja wybranych elementów polegała na zamodelowaniu niezbędnych, w stosunku do zaimplementowanego w poprzednich Sprintach modelu, encji i związków potrzebnych do utworzenia i realizacji przelewów zdefiniowanych oraz zaimplementowaniu powstałego modelu w bazie danych, uaktualnieniu warstwy DAL oraz realizacji logiki i odpowiednich widoków.

4.6. Sprint 6 – Filtry

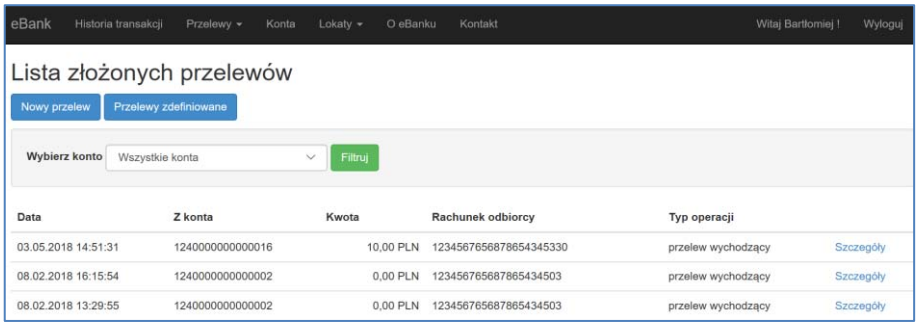
Ostatni Sprint zawiera planowanie i implementację kosmetycznych zmian w już wykonanych modułach systemu. Do Sprintu 6 (tabela 7) przydzielone zostały tylko dwa elementy związane z filtrowaniem danych, w ekranach „Historia operacji" i „Listy złożonych przelewów".

Tabela 7. Rejestr Sprintu 6

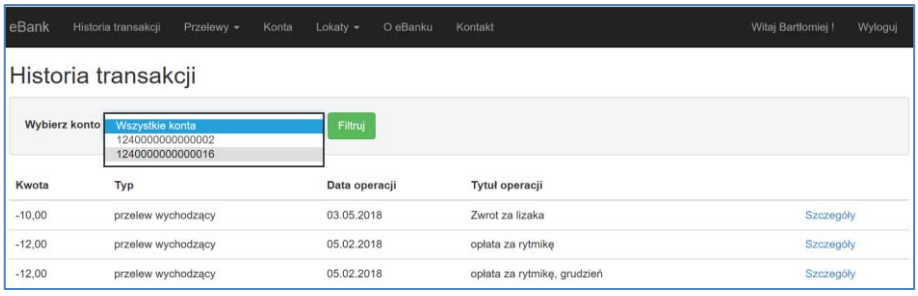
Nr	Element Rejestru Sprintu
1	Ekran – Filtrowanie wykonanych operacji
2	Ekran – Filtrowanie przelewów po koncie

Wykonanie obydwu filtrów polegało na tym, że do widoków listy przelewów i operacji został dodany formularz z listą wyboru konta oraz zmodyfikowane zostały kontrolery. Na rysunku 25 przedstawione zostały obydwa widoki. W szarej ramce nad tabelą danych znajduje się pole wyświetlające listę kont klienta oraz przycisk „Filtruj" przeładowujący stronę z uwzględnieniem wybranego konta.

Filtr konta w oknie „Lista złożonych przelewów"



Filtr konta w oknie „Historia transakcji"



Rysunek 25. Ekrany utworzone w Sprincie 6

Ciekawym rozwiązaniem jest dodanie do listy kont pozycji „Wszystkie konta", która pozwoliła na wyświetlenie transakcji i przelewów ze wszystkich kont. Implementacja polegała na dodaniu na pierwszej pozycji ręcznie utworzonego obiektu Konto o polach ID = 0 i NumerKonta = „Wszystkie konta" do listy kont klienta pobranej z bazy danych. Metoda generująca listę transakcji sprawdza, czy id wybranego konta jest równe 0. Jeżeli tak, to pobierane są wszystkie transakcje klienta, w przeciwnym wypadku, przy pobieraniu wyników z bazy danych, uwzględniane jest wybrane konto.

5. Podsumowanie

W artykule przedstawiony został proces tworzenia przykładowego systemu bankowości internetowej z wykorzystaniem metodyki Scrum. Przeprowadzona została analiza potrzeb klienta, na podstawie której określono Wizję produktu, opracowany został Rejestr Produktu oraz zaplanowane Sprinty. Pokazane zostały efekty wykonania prac w kolejnych Sprintach.

Jak większość współczesnych systemów informatycznych, system bankowości internetowej podlega ciągłym zmianom. Z jednej strony jest to wprowadzanie nowych funkcjonalności oczekiwanych przez biznes lub użytkowników, dostosowywanie się do warunków zewnętrznych systemu (zmiany prawne, technologiczne i społeczne), a z drugiej strony ciągłe doskonalenie produktu, usuwanie błędów i poprawianie wydajności.

Przedstawiony w niniejszej pracy system „eBank", dzięki zastosowanemu zwinnemu podejściu Scrum, mógłby zostać przy niewielkim nakładzie pracy rozwinięty o nowe funkcje, na przykład umożliwienie realizacji przelewów w różnych walutach, obsługę przelewów cyklicznych lub zapisanie przelewu zdefiniowanego jako przelewu cyklicznego.

Literatura

[1] J. Sutherland, *Scrum. A revolutionary approach to building teams, beating deadlines and boosting productivity*, London: Random House Business Books, 2014.

[2] I. Nonaka, H. Takeuchi, *The new product development game*. "Harvard business review" 64 (1), 1986.
[3] K.S. Rubin, *Scrum. Praktyczny przewodnik po najpopularniejszej metodyce Agile*, Gliwice: Wydawnictwo HELION, 2014.
[4] K. Kaczor, *Scrum i nie tylko. Teoria i praktyka w metodach Agile*, Warszawa: Wydawnictwo Naukowe PWN, 2016.
[5] *Niezbędnik Juniora. Scrum*, [Online] https://kobietydokodu.pl/niezbednik-juniora-scrum/ [Data dostępu: 01.05.2018].
[6] *Scrum*, [Online] http://scrum.org.pl. [Data uzyskania: 01.05.2018].
[7] A. Kisielińska-Ptasznik, *Projekt i implementacja systemu bankowości internetowej*, praca inżynierska, Warszawa: Warszawska Wyższa Szkoła Informatyki, 2018.

Using Scrum framework in the implementation of a sample Internet banking system

Abstract

The paper presents the process of implementing a sample Internet banking system with the usage of one of the agile frameworks, namely Scrum. The Product vision is determined on the basis of the analysis of system requirements, the Product Backlog is created and Sprints are planned. The results of successive Sprints are shown. The differences between the incremental and the waterfall approaches to software development are indicated in the context of the created system.

Keywords – Scrum, incremental approach, Internet banking