Projekt zespołowy

Aplikacja z kuponami

Jakub Brodziński Arkadiusz Ziobrowski Paweł Drozdowski

Przemysław Wypchał Szymon Świderski

Rozdział 1

Wstęp

1.1 Wstęp

Celem projektu było stworzenie aplikacji webowej umożliwiającej tworzenie ankiet i nagradzanie użytkowników kuponami na produkty i usługi w zamian za ich wypełnienie.

Aplikacja pozwala firmom na tworzenie ankiet i zbieranie statystyk na temat udzielonych odpowiedzi, natomiast niezarejestrowani jako firma użytkownicy mają możliwość wypełnienia ankiety przy użyciu aplikacji webowej, bądź aplikacji mobilnej, w zamian za co otrzymają interesujący ich kupon. Każdy użytkownik ma gwarancję otrzymania nagrody w zamian za wypełnienie ankiety dzięki zastosowaniu rezerwacji kuponów. Statystyki pozwalają firmie określić najpopularniejsze odpowiedzi i wyznaczyć trendy wśród badanych grup wiekowych i narodowości.

Dokumentacja składa sie z czterech rozdziałów. W dalszej części rozdziału pierwszego scharakteryzowano założenia funkcjonalne i niefunkcjonalne systemu. W rozdziale drugim przedstawiono analizę problemu. Omówiono w nim problemy związane z tworzeniem systemu zapewniającego dane funkcjonalności, a także przedstawiono rozwiązania użyte w innych systemach o podobnych założeniach funkcjonalnych i niefunkcjonalnych. W rozdziale trzecim przedstawiono szczegółowy projekt systemu wraz z diagramami w notacji UML, pomagającymi zrozumienie zależności między komponentami systemu i wspomagającymi opis procesu działania aplikacji. W tym rozdziale przedstawiono również technologie zastosowane do stworzenia systemu, dzięki którym mógł on być napisany w sposób bardziej nowoczesny i efektywny. Opisane zostały również możliwości rozwoju i modyfikacji systemu. W rozdziale czwartym omówiono implementację systemu wraz z opisem kodu. Został tam szczegółowo opisany proces działania systemu z naciskiem na spójność między poszczególnymi komponentami.

1.2 Specyfikacja wymagań

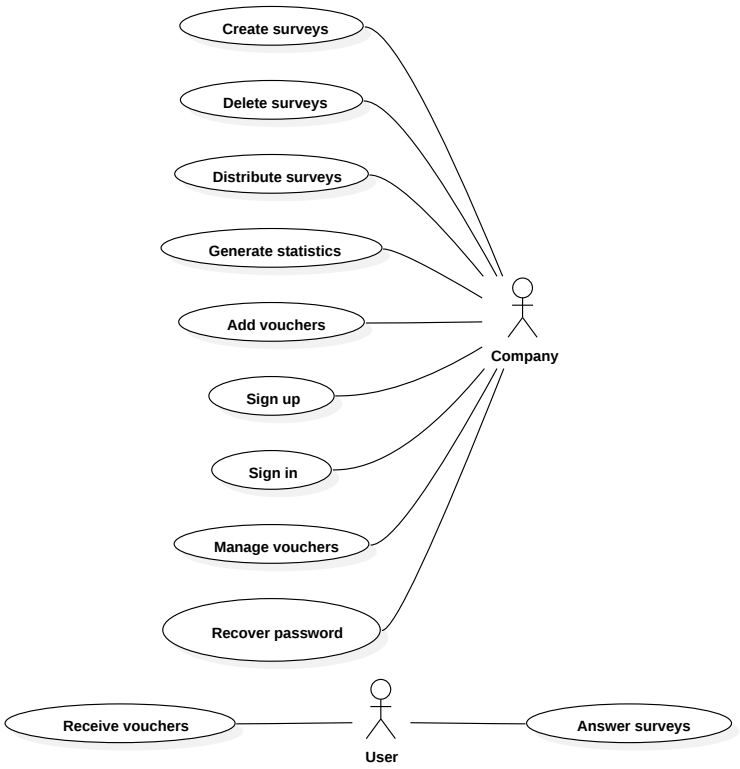
1.2.1 Założenia funkcjonalne

System pozwala firmie na założenie konta, którym będzie reprezentowana w serwisie. Możliwe jest również odzyskiwanie hasła przy użyciu adresu e-mail podanego w czasie rejestracji. Firma może utworzyć w systemie ankietę oraz powiązać ją z kuponem na wybraną przez siebie usługę lub produkt. Po powiązaniu kuponu z ankietą firma może dodać do kuponu dowolną ilość kodów promocyjnych, które będą dystrybuowane w zamian za wypełnienie ankiety. Firma może również przeglądać statystyki dotyczące wypełnionych ankiet, które zapewniają informacje o procentowym rozkładzie odpowiedzi na dane pytanie oraz o wieku oraz narodowości wypełniającego. Ponadto firma ma możliwość przeglądania poszczególnych wypełnionych ankiet. Firma ma także możliwość przeglądania wszystkich swoich ankiet w systemie oraz usuwania poszczególnych ankiet oraz edycji, dodawania oraz usuwania kuponów. Firma może również trwale usunąć konto z systemu. Wypełniający ankiety może wypełnić ankietę za pomocą aplikacji webowej, bądź aplikacji mobilnej i w prosty sposób otrzymać kupon drogą mailową lub bezpośrednio na ekranie urządzenia po przesłaniu odpowiedzi.

1.2.2 Założenia niefunkcjonalne

Proces logowania, rejestracji i odzyskiwania hasła jest bezpieczny dla firmy i nie grozi utratą jakichkolwiek wrażliwych danych. Ankiety mogą zawierać różne rodzaje pytań i w trakcie ich tworzenia firma powinna mieć łatwość ich bieżącej edycji. Interfejs graficzny aplikacji webowej i mobilnej powinien być lekki i nowoczesny, o spójnej szacie kolorystycznej. Firma powinna mieć łatwość przeglądania swoich ankiet w systemie i pozwiązanych z nimi ankiet. Użytkownicy powinni być anonimowi w systemie, a dane przekazywane firmie powinny uniemożliwiać identyfikację poszczególnych osób, udzielających odpowiedzi na ankiety. Użytkownik powinien mieć możliwość łatwego wyboru ankiety z jasno określoną nagrodą w zamian za jej wypełnienie. Użytkownik powinien mieć również gwarancję otrzymania nagrody nawet w przypadku ograniczonej ilości kodów promocyjnych związanych z danym kuponem.

Funkcjonalności ze względu na użytkowników można również przedstawić w formie diagramu UML przypadków użycia.



Rysunek 1.1: Diagram przypadków użycia dla systemu.

W dalszej części dokumentacji będą używane terminy, których znaczenie znajduje swój odpowiednik w modelu systemu. Terminy te zostaną teraz opisane dla zwiększenia czytelności i jasności dokumentacji:

- **Ankieta** - zbiór pytań otwartych, ze skalą lub jednokrotnego/wielokrotnego wyboru.
- **Kupon** - zniżka na konkretny produkt lub usługę.
- **Kod promocyjny** - kod powiązany z kuponem, uprawniający do zniżki na konkretny produkt lub usługę.

Rozdział 2

Analiza problemu

Podobną funkcjonalność do opisywanego w dokumentacji systemu ma aplikacja webowa *surveybee.net*. Oferuje ona nagrody pieniężne za wypełnienie ankiety, jednak minimalna stawka zarobionych pieniędzy, pozwalająca na wypłatę sprawia, że wypełniający musiałby poświęcić dużo czasu, aby otrzymać swoją nagrodę. Nasz system pozwala na natychmiastowe otrzymanie nagrody w postaci kuponu, bezpośrednio po wypełnieniu ankiety.

Innym systemem, znacznie bardziej popularnym i posiadającym większą reputację jest *surveymonkey.com*. System ten pozwala firmom na tworzenie ankiet, które następnie mogą być wypełniane przez użytkowników. Aplikacja ta nie pozwala jednak na wprowadzenie gratyfikacji w zamian za wypełnienie ankiety.

Nasz system posiada zatem założenia funkcjonalne, którym różni się od produktów dostępnych na rynku. Implementacja takich założeń wiązała się z rozwiązaniem kilku problemów, które umożliwiałyby efektywne i bezpieczne wprowadzenie wspomnianych w rodziale pierwszym funkcjonalności.

2.0.1 Problem liczby kuponów

Jednym z najważniejszych założeń funkcjonalnych naszego systemu jest przekazywanie użytkownikom kuponów w zamian za wypełnienie ankiety. Problem ten jest trudny w sytuacji, gdy ilość kodów promocyjnych powiązanych z kuponami jest ograniczona. W przypadku naiwnego rozwiązania, polegającego na przydzielaniu kuponu wypełniającemu w momencie kliknięcia przycisku `Wyślij`, mogłoby dojść do sytuacji, w której ostatni kod zostałby przydzielony osobie, która szybciej rozwiązała ankietę. Osoba, która rozwiązywałaby ankietę wolniej po kliknięciu przycisku `Wyślij` zostałaby poinformowana o braku kodu promocyjnego. Jej czas przeznaczony na wypełnienie ankiety nie zostałby więc nagrodzony, a sam wypełniający poczułby się oszukany.

Aby uniknąć takiej sytuacji wprowadzone zostało blokowanie kuponów, które jest połączone z sesją użytkownika. Użytkownik po wybraniu interesującej go ankiety ma wiązany swój identyfikator sesji z kodem promocyjnym. W przypadku braku kodów promocyjnych dla danego kuponu, ankieta nie będzie widoczna dla użytkowników w systemie do momentu dodania większej ilości kodów promocyjnych. Dzięki takiemu rozwiązaniu użytkownik ma gwarancję otrzymania nagrody już na początku wybrania ankiety. Takie rozwiązanie wiąże się również z możliwościami paraliżowania systemu przez złośliwych użytkowników. W negatywnym przypadku złośliwy użytkownik mógłby zablokować wszystkie kody promocyjne dla siebie, przez co ankieta stałaby się niewidoczna dla innych użytkowników. Aby zapobiec takiej sytuacji wprowadzone zostały stosowne zabezpieczenia. Użytkownik systemu może w danym momencie mieć tylko jeden kod promocyjny powiązany z identyfikatorem sesji. Powiązanie te jest aktywne w systemie tylko przez 15 minut. Ponadto zamknięcie ankiety poprzez naciśnięcie przycisku `Anuluj` spowoduje zwrócenie kodu promocyjnego do puli. Dokładny opis procesu blokowania kuponu jest opisany w rodziale czwartym.

2.0.2 Problem anonimowości

W trosce o rzetelność odpowiedzi system zapewnia anonimowość odpowiadających. W bazie danych nie są przechowywane żadne dane osobowe, pozwalające na jednoznaczne zidentyfikowanie wypełniającego ankietę. Wszystkie dane osobowe, które mogą jednoznacznie wiązać użytkownika z konkretną osobą nie są przechowywane w bazie i są całkowicie opcjonalne. Przykładem takich danych jest adres e-mail, na który może zostać wysłany kupon po wypełnieniu ankiety. Jest on podawany opcjonalnie po wypełnieniu ankiety i nie jest nigdzie przechowywany. Innymi danymi zbieranymi od użytkownika są wiek oraz kraj pochodzenia, jednak informacje takie nie pozwalają wnioskować jednoznacznie o tożsamości wypełniającego.

2.0.3 Problem bezpieczeństwa

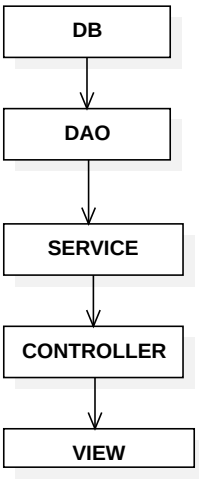
W systemie, który jako główne założenie funkcjonalne oferuje wartość materialną w zamian za wypełnienie ankiety, ważne jest zapewnienie bezpieczeństwa. Pierwszym problemem była walidacja odpowiedzi na ankiety. System należało zabezpieczyć przed atakami XSS zarówno przeprowadzanymi ze strony wypełniającego, jak i firmy. Aby ich uniknąć, wszystkie przesyłane pytania i odpowiedzi są walidowane przez serwer. Jako dodatkową walidację, zostało wprowadzone sprawdzanie tworzonych ankiet od strony front-endu. Nie jest to jednak zabezpieczenie wystarczające, aby uniemożliwić takie ataki, gdyż rozwiązania front-endowe mogą być modyfikowane od strony klienta. System jest również zabezpieczony przed atakami typu CSRF poprzez użycie tokenów CSRF. Szczegóły tego rozwiązania zostały opisane w rodziale trzecim. Zabezpieczone zostały również komponenty systemu, do których dostęp powinny mieć tylko zalogowane firmy, dzięki zastosowaniu podziału na role i autoryzacji w systemie. To rozwiązanie również zostało opisane dokładnie w rodziale trzecim. Dane firmy przechowywane są w bazie danych w sposób minimalizujący straty w razie ewentualnego wycieku danych. Hasła firm trzymane są w postaci hasha z solą, dzięki czemu adwersarz porównujący hasła nie jest w stanie stwierdzić identyczności między hasłami poszczególnych firm. Również proces odzyskiwania hasła został zaprojektowany w sposób bezpieczny, który uniemożliwia przejęcie kontroli nad kontem firmowym osobie trzeciej.

Rozdział 3

Projekt

3.0.1 Architektura projektu

Cieżko jest mówić o naszej aplikacji jak o jednym bycie, ponieważ można ją podzielić na dwa podsystemu, gdzie w obu przypadkach archiktektura systemu jest wielowarstwowa. Jednym z podsystemów jest aplikacja webowa, która została zaprojektowana w oparciu o wzorzec **MVC** (ang. Model-View-Controller), który to narzucił wielowarstwową architektura. Oddzielając logikę biznesową od modelu (tj. danych) oraz interfejsu użytkownika aplikacji została zaprojektowana zgodnie z zasadami **GRASP** (ang. General responsibility assignment software patterns). Taka a nie inna architektura aplikacji webowej oprócz znacznego zwiększenia czytelności kodu pozwoliła na wielowarstwowe zabezpieczenia, które zostały nałożone na każdą z trzech głównych warstw naszej aplikacji. Samą część łącząca wartswe modelu oraz controller'a możemy podzielić na dwie podwarstwy: **DAO** (ang. Data Access Object) oraz **Service**.



Rysunek 3.1: System podzielony na podwarstwy.

Warstwa **DAO** odpowiedzialna jest bezpośrednio na komunikacje z wartswą modelu, tj. bazą danych. Natomiast to w warstwie serwisowej, która korzysta z warstwy **DAO** została zaimplementowana cała logika biznesowa i to właśnie z warstwy serwisowej korzystamy w kontrolerach.

Drugim podsystemem naszego projektu jest aplikacja webowa, która również jest systemem rozproszonym komunikującym się z serwerem (który jest częścią pierwszego podsystemu) wykorzystując **REST** (ang. Representational State Transfer) oferując użytkownikowi jedynie część funkcjonalności aplikacji webowej.

W projekcie poza stosowaniem zasad **GRASP** zostały wykorzystane takie wzorce projektowe jak **Proxy**, **Template Method**, **Adapter**, **Factory** oraz **Decorator**, **Dependency Injection**, **Aspect**

Oriented Programing, **Singleton**, które w sposób naturalny współgrały z użytymi przez nas technologiami.

3.0.2 Przypadki użycia i scenariusze

UC1	Add Survey
Actors	Company
Preconditions	Company is signed in and is in the dashboard page.
Postconditions	Company added survey.
Main Success Scenario	1.Company hits on 'Add Survey' button.
	2.Company hits proper 'Add Question' button depending on what type of question company wants.
	3.Company fills up question's content.
	4.Company repeats steps from point 2. until completes survey.
	5.Company hits 'Add Voucher' button in order to compose survey with concere voucher.
	6.Company hits 'Confirm Survey' button.
Alternate flows	2a.Company hits 'Confirm Survey' button without adding any questions:
	1.System signals error and doesn't proceed.
	5a.Company hits 'Confirm Survey' button without adding voucher:
	1.System signals error and doesn't proceed.

Powyższy przypadek użycia opisuje proces dodawania ankiety so systemu. Szczegóły dodawania ankiety zostały opisane w rozdziale czwartym.

UC2	Sign in
Actors	Company
Preconditions	Company is registered in the System
Postconditions	Company is signed in.
Main Success Scenario	1.Company enters the website and is on landing page
	2.Company hits the Sign In button.
	3.System moves the Company to the Sign In page.
	4.Company enters the login.
	5.Company enters the password.
	6.Company hits the Sign In/Enter button.
	7.System moven the Company to the dashboard page.
Alternate flows	2a.Company hits the Sign Up button:
	1.System moves the Company to the Sign Up page.
	6a.Company hits the Sign Up button or "Doesn't hvae an account? Create one here now!"text.
	1.System moves the Company to the Sign Up page.
	6b.Company hits "Forgot the password"text.
	1.Company is guided through password recovery process.
	7a.Invalid login:
	1.System signals error and doesn't proceed/stays on the Sign In page.
	7b.Invalid password:
	1.System signals error and doesn't proceed/stays on the Sign In page.

Powyższy przypadek użycia opisuje proces logowania do systemu. Szczegółowy proces został opisany w rozdziale czwartym.

UC3	Sign up
Actors	Company
Preconditions	Company is not already registered in the System.
Postconditions	Company is registered in the System. Company is able to access its dashboard.
Main Success Scenario	1.Company enters the website and is on landing page.
	2.Company hits the Sign Up button.
	3.System moves the Company to the Sign Up page.
	4.Company enters login, password, name and address.
	5.Company hits Sign Up/Enter button.
	6.System notifies Company about successful registration.
	7.System moves Company to the landing page.
	8. System can also demand entering a verification token from e-mail after the registration.
Alternate flows	2a.Company hits the Sign In button:
	1.System moves the Company to the Sign In page.
	5a.Company hits Cancel/Back button:
	1.System moves the Company to the landing page.
	6a.Login/e-mail already taken:
	1.System signals error and doesn't proceed.
	6b.Login/Name/Address contains forbidden symbols:
	1.System signals error, suggests the proper symbol range and doesn't proceed.
	6c.Any of registration fields is empty:
	1.System signals error, suggests to fill the empty field and doesn't proceed.

Powyższy przypadek użycia opisuje proces rejestracji w systemie. Szczegółowy proces został opisany w rozdziale czwartym.

UC4	Filling up the survey
Actors	User
Preconditions	None or the mobile app installed required , if user want's to make it that way
Postconditions	User accomplished fulfilling the survey. User gain a voucher code.
Main Success Scenario	1.User chose a company, which voucher might be interesting for him (either in mobile app or website).
	2.System moves user to website where the survey is shown.
	3.User fulfilss whole survey step by step.
	4.User hits 'Done' button.
	5.System validates survey and user moves forward.
	6.System notifies user that the survey is accepted.
	7.System asks user for e-mail Adress
	8.User can aceppt his e-mail Adress by pressing the button.
	9.System sends voucher to user.
	10.System notifies user that the voucher has been sent.
Alternate flows	5a.Survey rejected, validation didn't pass:
	1.Special informations are displayed.
	2.User is asked to fix his answers.
Alternate flows	9a.User passed e-mail that don't match e-mail address' pattern:
	1.Special informations are displayed.
	2.User is asked to pass proper e-mail address.

Powyższy przypadek użycia opisuje proces wypełniania ankiety. Szczegółowy proces został opisany w rozdziale czwartym.

UC5	Add Voucher
Actors	Company
Preconditions	Company is signed in and is in dashboard page.
Postconditions	Company added voucher.
Main Success Scenario	1.Company hits „Add voucher" button.
	2.Company fills up voucher details(some details in plain text,type of discount, discount rate etc.)
	3.Company hits „Add voucher" button to add voucher to right survey from dropdown list.
	4.Company confirms voucher by „Confirm voucher" button.
	5.Voucher is added.
Alternate flows	5a. Voucher already exist:
	1.Proper error message is displayed.
	2.Filled for is displayed, company is asked to pass diffrent voucher.
	5b.Company didn't pick any survey:
	1.Proper error message is displayed.
	2.Filled for is displayed, company is asked to pass diffrent voucher.

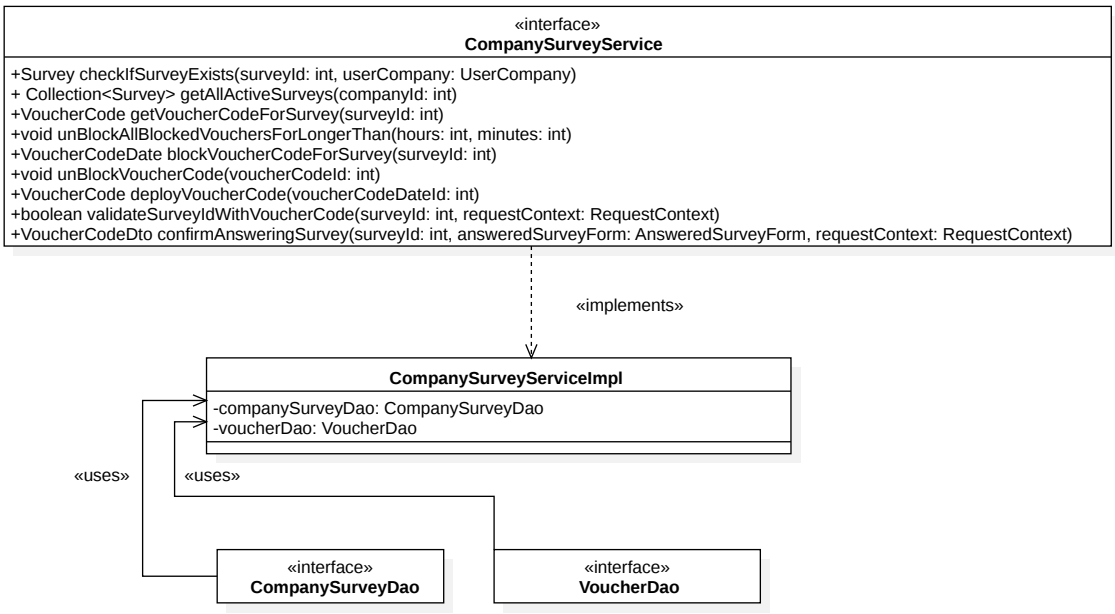
Powyższy przypadek użycia opisuje proces dodawania kuponu. Szczegółowy proces został opisany w rozdziale czwartym.

UC6	Remove account
Actors	Company
Preconditions	Company is registered in system.
Postconditions	Company doesn't have an account.
Main Success Scenario	1.Company hits „Delete account" button
	2.Confirmation window appears asking for confirmation.
	3.Company confirms the action by entering the password to the site.
	4.All information about the company, surveys and coupons are deleted from database.
	5.Company sees statement „Your account has been deleted.".

Powyższy przypadek użyca opisuje proces usuwania konta z systemu. Wraz z usunięciem konta z systemu, z bazy danych są usuwane kaskadowo wszystkie krotki, które były powiązane z danym kontem firmowym.

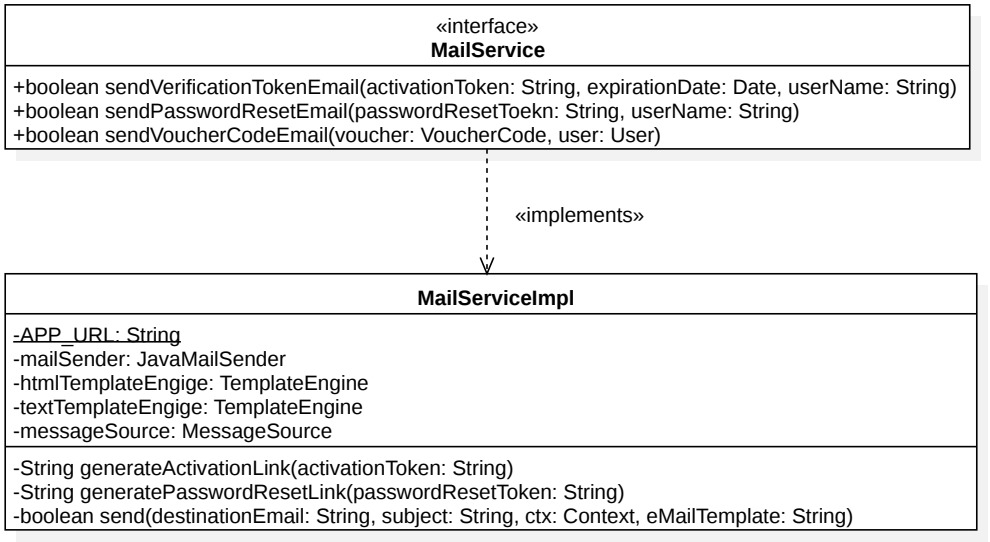
3.0.3 Diagramy klas

Diagramy przedstawione poniżej przedstawiają klasy serwisowe, w których to właśnie została zaimplementowana cała logika biznesowa naszego systemu i to one reprezentują funkcjonalność i możliwości naszej aplikacji webowej. W celu zwiększenia czytelności diagramów z ich większości zostały usunięte trywialne metody (nie zawierające logiki biznesowej), których zadaniem było dodanie, edycja, usunięcie przekazywanego obiektu w bazie danych.



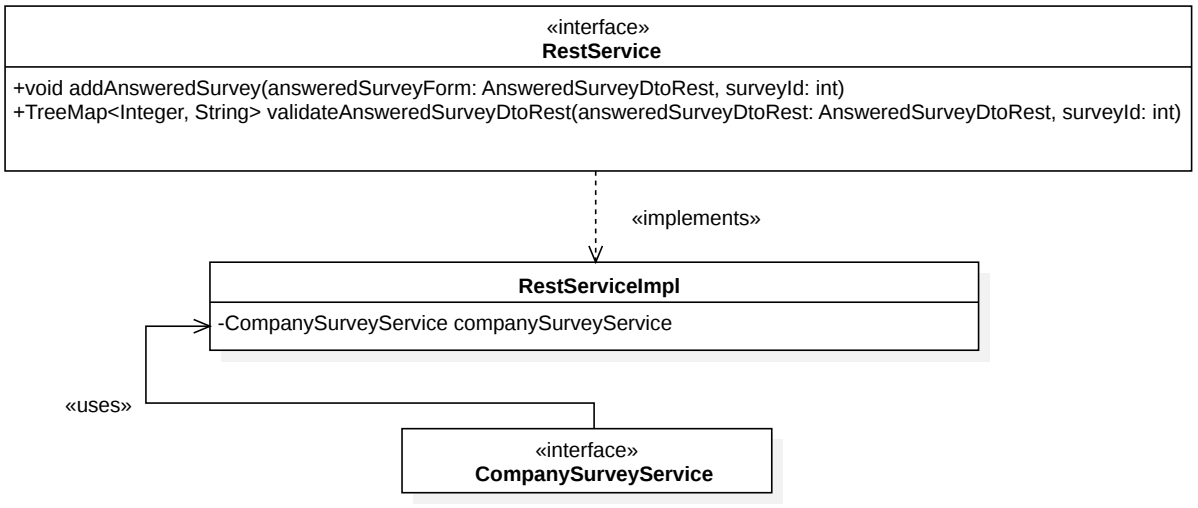
Rysunek 3.2: Diagram klas dla interfejsu serwisowego *CompanySurveyService*

Powyższy diagram przedstawia diagram klas dla serwisu, który przede wszystskim obsługuje funkcjonalność związana pobieraniem aktywnym, przeglądaniem oraz wypełnianiem ankiet jak również walidacją tych czynności. Ponadto odpowiedzialny jest za podjęcie decyzji czy dany kupon powininen zostać wydany czy też nie.



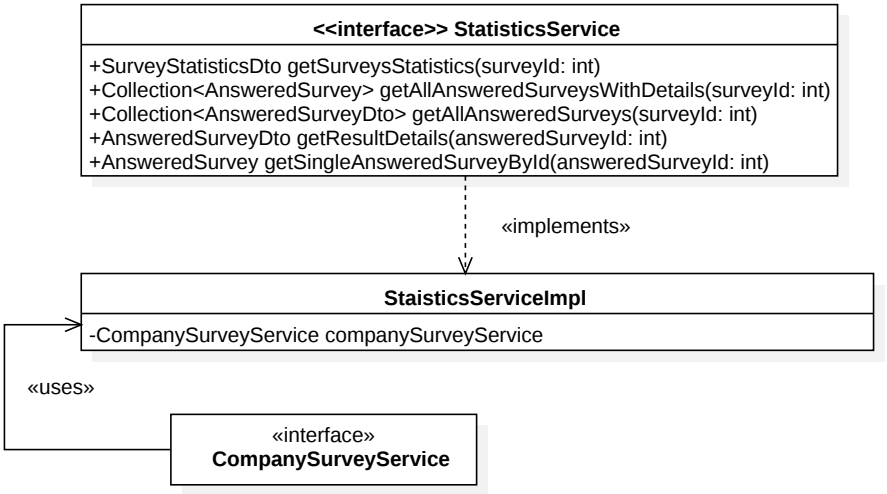
Rysunek 3.3: Diagram klas dla interfejsu serwisowego *MailService*

W **MailService** każda metoda odpowiada innemu typu wiadomości, która zostanie wysłąna.



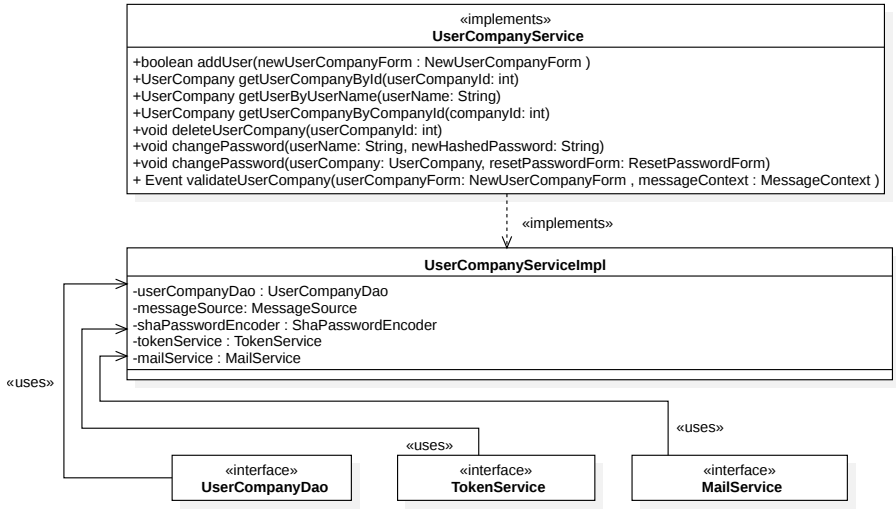
Rysunek 3.4: Diagram klas dla interfejsu serwisowego *RestService*

Powyższy diagram klas przedstawia diagram klas serwisowych obsługujących **REST Api**, które jest wykorzystywane przez aplikacje mobilna.



Rysunek 3.5: Diagram klas dla interfejsu serwisowego *StatisticsService*

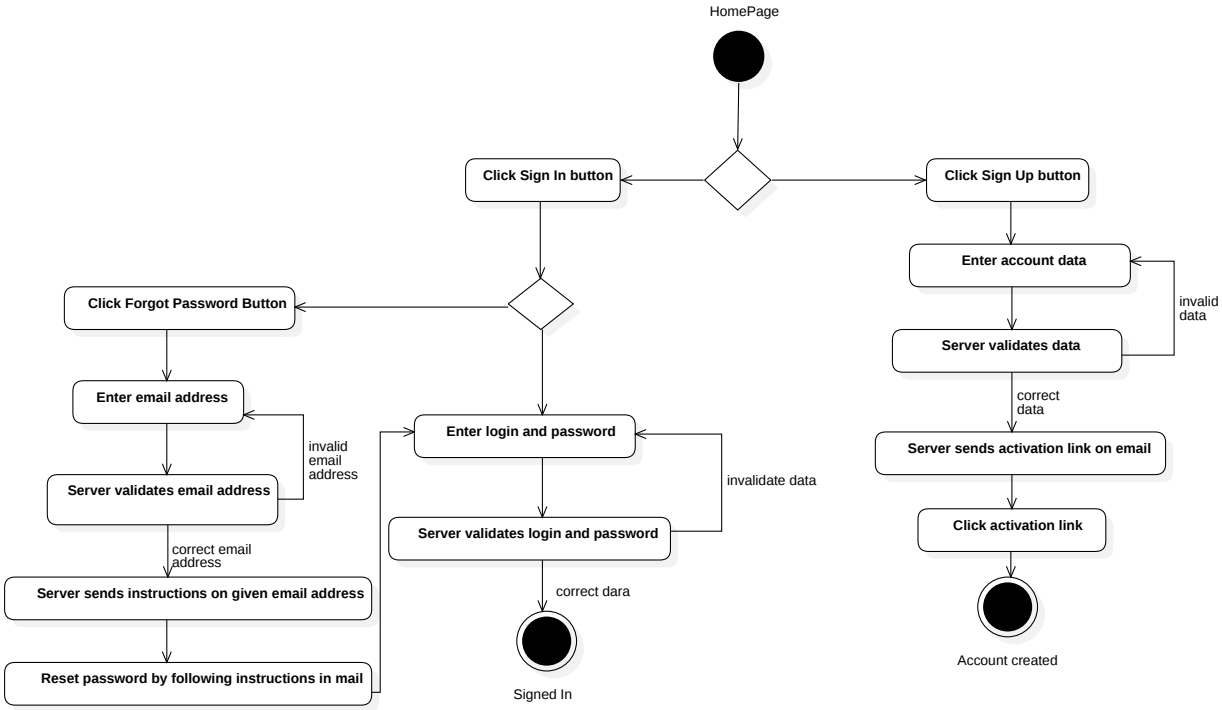
W powyżej przedstawionych klasach serwisowych obliczane są statystyki, które każdy użytkownik ze stworzonymi ankietami może przeglądać.



Rysunek 3.6: Diagram klas dla interfejsu serwisowego *UserCompanyService*.

Klasy serwisowe w diagramie powyżej odpowiadają za część funckjonalności związanej z kontem firmy, tj. edycja danych, rejestracja, logowanie, zmiana hasła itd.

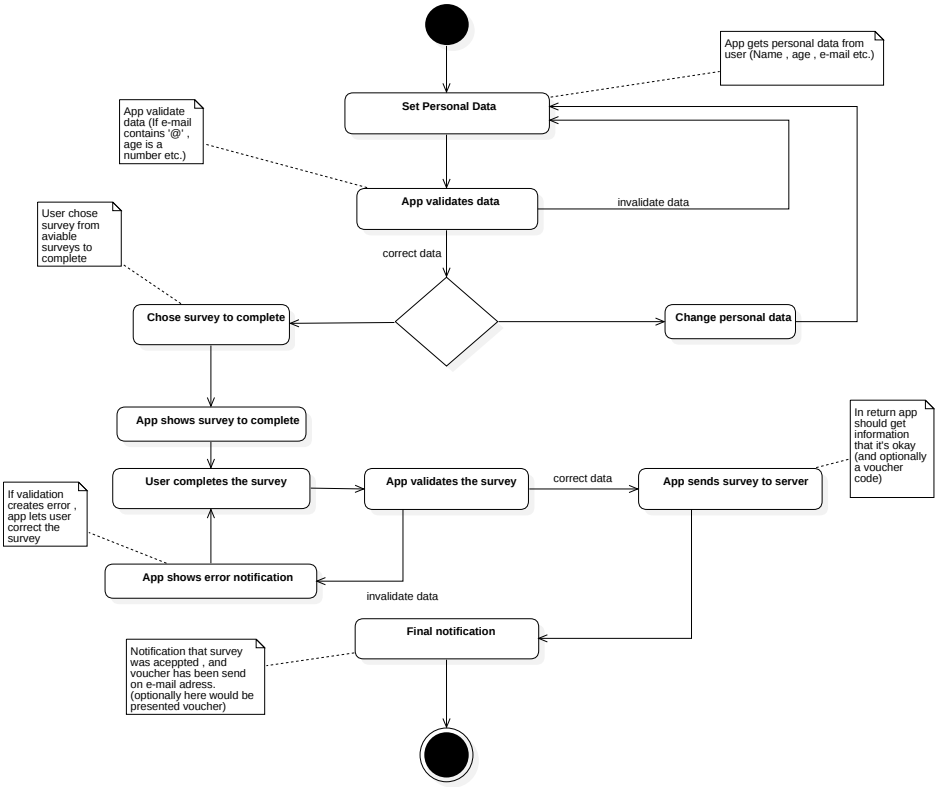
3.0.4 Diagramy aktywności



Rysunek 3.7: Diagram aktywności dla logowania i rejestracji.

Powyższy diagram aktywności opisuje proces logowania i rejestracji w aplikacji webowej. Szczegółowy opis został przedstawiony w rodziale czwartym.

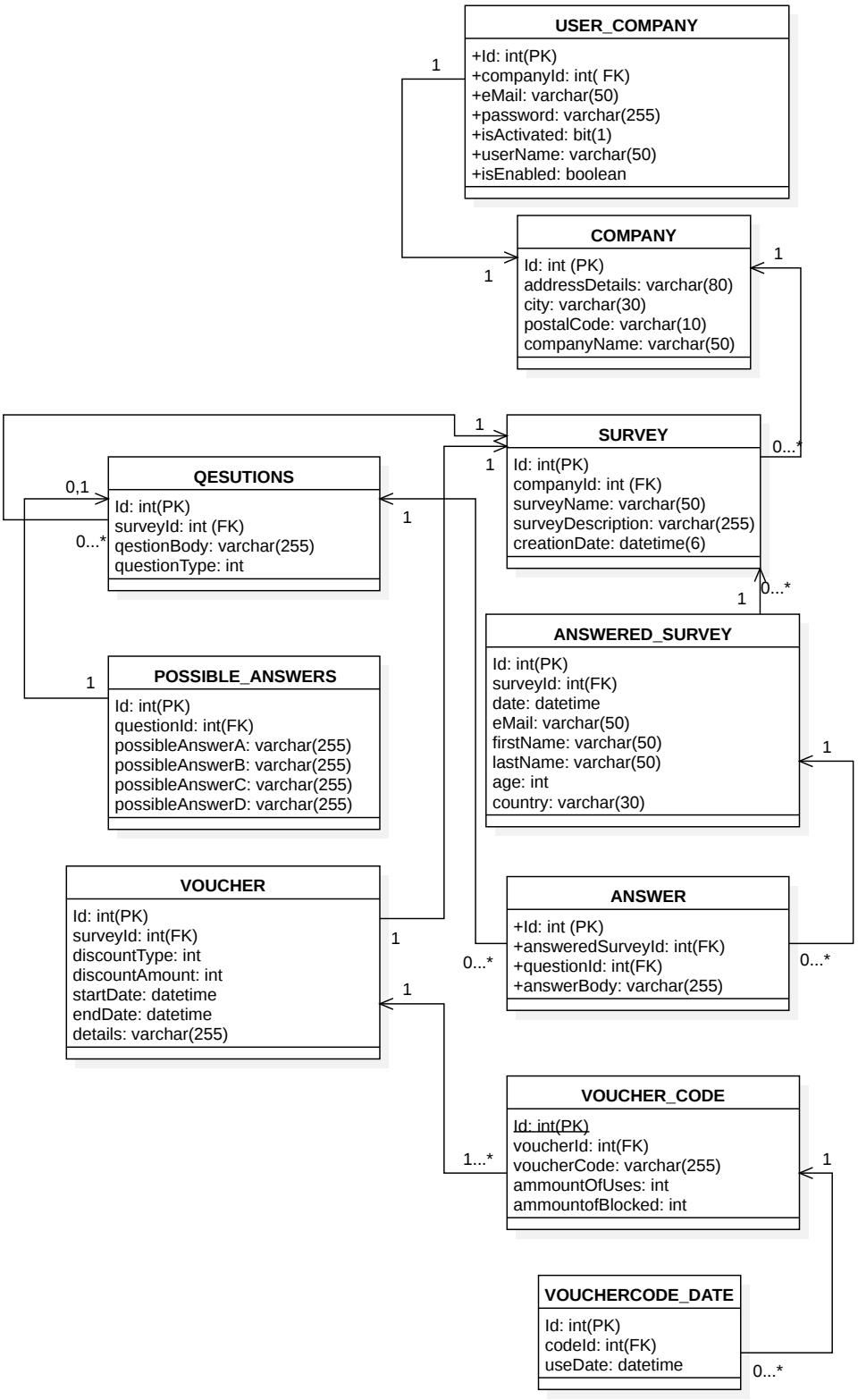
Głównym celem aplikacji mobilnej dla naszego serwisu, było umożliwienie użytkownikom w łatwy i prosty sposób uzupełnianie ankiet, oraz otrzymywanie kuponów w taki sposób, aby np. móc je okazać



Rysunek 3.8: Diagram aktywności dla aplikacji mobilnej.

w sklepie na urządzeniu mobilnym. Przy pierwszym uruchomieniu aplikacji, użytkownik jest proszony o wprowadzenie danych dla celów statystyki. Jest to komfortowe rozwiązanie dla tych użytkowników, którzy planują korzystać z naszej aplikacji więcej niż raz. Aplikacja następnie wyświetla listę dostępnych firm oraz ankiet. Dzięki korzystaniu z aplikacji, użytkownik ma pod ręką możliwość wypełnienia dowolnej z możliwych ankiet w każdym momencie oraz dostęp do swoich wszystkich otrzymanych i niewykorzystanych kuponów.

3.0.5 Projekt bazy danych



Rysunek 3.9: Diagram klas przedstawiający baze danych systemu

W celu odpowiedniego zrozumienia struktury naszej aplikacji koniecznie jest zaznajomienie się kilkoma kluczowymi założeniami naszego systemu, którego miały bezpośredni wpływ na sposób w jaki zaprojektowaliśmy baze danych. Najważniejszym założeniem było przypisanie tylko i wyłącznie jednego typu kupony do jednej ankiety. Żadna ankieta może oferować tylko wyłacznie jeden typ kupony jako nagrodę za wypełnienie ankiety. Każdy voucher (który reprezentuje typ nagrody) może już miec przypisany więcej niż jeden kupon, każdy z nich może byc wielokrotnego użytku (wtedy należy zaznaczyć jak wiele osób z niego może skorzystać) lub też jednorazowego użytku.

Równie ważnym założeniem było stworzenie tablicy **VoucherCode_Date**, który odpowiada za "blokowanie" kupony na czas wypełnienia ankiet.

3.0.6 Opis protokołów

Ruch po naszej aplikacji ma miejsce przy użyciu protokołu **TLS**. Co ważne dostęp do każdej podusługi lub też podstrony naszego systemu wymaga połączenia zabezpieczonego, a użytkwonik chcący korzystać z naszej aplikacji przy użyciu protokołu **HTML** jest automatycznie przekierowywany na odpowiednią strone, która wykorzystuje już **TLS**. Podobno sytuacja ma miejsce z aplikacją mobilną, której połączenie z serwerem ma miejscu przy użyciu **TLS**.

Część aplikacji webowej dbająca o uwierzytelnianie oraz autoryzacje (ang. authentication and authorization) przychodzących połączeń są filtry, które możemy uznać za protokół. Są one częścią **Servlet'ów**, która ma za zadanie jak sama nazwa wskazuje filtrować przychodzące zapytania, jak również w zależności od samego zapytania w odpowiedni sposób reagować. Informacje zawarte w zapytaniach oraz odpowiedziach serwera, które zostają dynamicznie przechowycone przez filtry są wykorzystywane między innymi do tego aby zidentyfikować autora zapytania (autoryzacja) i jeżeli będzie taka konieczność nadać mu odpowiednie uprawnienia (uwierzytelnienie). Jednym z zaimplementowanych przez nas zabezpieczeniem jest autoryzacja na poziomie metod serwisowych, jak również nadawanie ról użytkownikom w zależności od typu uprawnienień, które posiadają. Każdej osboie korzystającej z naszej aplikacji zostaje nadany unikatowy indetyfikator sesji, którym zostaje powiązany on i uprawnienia, które posiada. Właśnie te informacje wykorzystuje **The Security Filter Chain**, który jest częścią jednego z używanych przez nas narzędzi. Kiedy użytkownik legitymuje się swoim identyfikatorem sesji **The Security Filter Chain** (od tego momentu nazywany filtrem w celu zwiększenia czytelności tekstu) sprwadza w bazie danych czy istnieje już powiązane z tym identyfikatorem połączenie, w przypadku gdy zostanie to potwierdzone filter sprawdza jakie uprawnienia posiada to połączenie, skutkuje to powiązaniem uwierzytelnieniu zapytania przychodzącego do serwera, co następnie umożliwia wykorzystanie autoryzacji na poziomie metod, która sprawdza czy nasze uprawnienia są wystarczające.

Każdy użytkownik aplikacji webowej posiada odgórnie pewne uprawnienia, a w przypadku zalogowania się następuje dodatkowe uwierzytelnienie, gdzie jego identyfikator sesji wiązany z kontem do którego jest aktualnie zalogowany. To właśnie dzięki temu filter bezpieczeństwa potrafi jednoznacznie potwierdzić czy przychodzące zapytanie legitymujące się jedynie identyfikatorem sesji ma wystarczające uprawnienia do danej podusługi.

Ze względu na taki a nie inny sposób uwierzytelniania oraz autoryzacji konieczne również było użycie dodatkowego zabezpieczenia w postaci losowych tokenów chroniących przed atakami **CSRF** (ang. Cross-site request forgery), które w trakcie generowania zostają powiązane z identyfikatorem sesji, na którego życzenie zostały wygenerowane. Każdy użytkownik naszego serwisu podczas zapytań typu **POST** (sytuacja taka ma najczęściej miejsce podczas zatwierdzaniu różnego typu danych do serwera) musi "wylegitymować się" wygenerowanym wcześniej dla niego **CSRF tokenem** jak również identyfikatorem sesji, dopiero wtedy filter bezpieczeństwa zatwierdza takie zapytanie i przekazuje je do warstwy kontrolerów.

Częścią naszego systemu jest serwis mailowy, przez który jesteśmy w stanie komunikować się z klientami. Korzysta on z protkołu **STMP**.

3.0.7 Możliwości rozwoju systemu

Wdrożenie systemu Po zbudowaniu aplikacji i dodaniu artefaktu w `IDE` generujemy plik `JAR` lub `WAR`, który można wdrożyć w chmurze, np. *Amazon Web Services*, które po skonfigurowaniu `Load Balancera` pozwoli na korzystanie z wersji *live* aplikacji webowej.

Aby wdrożyć aplikację mobilną należy utworzyć konto deweloperskie w serwisie *Google Play*, a następnie wygenerować klucze dla aplikacji mobilnej, powiązać je z aplikacją mobilną i przesłać pliki aplikacji do serwisu *Google'a*.

System gwarantuje całkowite pokrycie wszystkich założeń funkcjonalnych i niefunkcjonalnych. Możliwe jest jednak rozwój systemu o kolejne funkcjonalności. Bezpośrednimi funkcjonalnościami, które w łatwy sposób mogą zostać zbudowane na bazie istiejących rozwiązań są między innymi powiązanie kodów promocyjnych kuponu z kodami QR. Aby wprowadzić taką funkcjonalność będzie należało dodać kolejne pole w klasie odpowiadającej w modelu za kupon, które będzie przechowywać kod QR. Kod ten może być w łatwy sposób zwracany do użytkownika i wyświetlany na ekranie wraz z pozostałymi szczegółami kuponu. Kolejną funkcjonalnością, o którą może zostać rozbudowany system jest wprowadzenie algorytmu decydującego o zdobywaniu kuponu. Kupony mogą być wydawane co piąte wypełnienie ankiety, z dowolnym prawdopodobieństwem lub w ogóle. System jest napisany w sposób, który umożliwia modyfikacje serwisu bez konieczności przeprojektowania pozostałej logiki. System charakteryzuje się wysoką spójnością i niskim sprzężeniem.

Rozdział 4

Dokumentacja kodu

4.1 Użyte technologie

Całość systemu została zaimplementowana w jęzku obiektowym **Java 8** przy użyciu narzędzi dostępnych w **Java EE** (ang. enterprise edition). Do automatyzacji budowania projektu w przypadku aplikajci webowej został wykorzystany **Maven**, natomiast jeśli chodzi o aplikacje webową był to **Gradle**. Framework w oparciu którego została napisana aplikacja mobilna jest **Retrofit**. Ułatwia on w znaczny sposób pracę z różnego typu HTML'owych API (ang. application programming interface), m.in REST API. Narzędzie w znaczny sposób ułatwiło nam mapowanie obiket na JSON'a oraz JSON'a na obiekty, jak również samo wysyłanie zapytań oraz odbieranie i interpretacja odpowiedzi serwera, pamietając o konieczności legitymacji się identyfikatorem sesji.

Jeśli chodzi o część systemu odpowiedzialną za aplikacje webową technologiami, które zostały wykorzystane są narzędzia z rodziny **Spring**. Program został zaprojektowany zgodnie z zasadami **GRASP**,a podczas implementacji z zachowaniem tych zasadł pomógł nam **Spring Core**, ułatwiał on wykorzystanie wzorca **DI** (ang. Dependency Injection), co skutkowało zachowaniem niskiego sprzężenia oraz wyskokiej spójności klas. Znaczna większość obiektów, które były wstrzykiwane poprzez **DI** były przedstawicielami wzorca projektowego **Singleton**, co skutowało zmniejszeniem zapotrzebania na zasoby komputera naszego serwera. Singletonami są wszystkie klasy serwisowe, kontrolery oraz klasy konfiguracyjne. Poza klasami konfiguracyjnymi wszystkie klasy wstrzykiwane przy użyciu **Spring Core** były implementacjami interfesjów, co pozwoliło **Springowi** w bardzo lekki i łatwy sposób przy użyciu refleksji budowanie **Proxy** tych obiektów w trakcie uruchomienia aplikacji i załadowania kontekstu,a nie jak miałoby by to miejsce w przypadki gdyby obiekty wstrzykiwane rozszerzały klase (lub też nie rozserzały żadnej kalsy abstrakcyjnej) modyfikując kod bitowy, który został wcześniej skompilowany. W różnych częsiach projektu mieliśmy doczynienia z programowaniem aspektowym, które również było możliwe, dzięki **Spring Core** oraz wcześniej stworzonym **Proxy** do wstrzykiwanych klas. Framework, z którym najwięcej pracowaliśmy to **Spring MVC**. Jest to narzędzie, które umożliwia z jasny sposób budować aplikacje webowe w opraciu o wzorzec projektowy **MVC**. Dzięki niemu w łatwy sposób mogliśmy zbudować most łączący widok z modelem, czyli kontrolery. **Spring MVC** rownież ułatwił nam walidacje dancyh otrzymanych przez użytkowników poprzez pare na adnotacjach dostępnych w pakietach **javax.validation.api**. Wraz z **Spring MVC** wykorzystywany był **Jackson**, czyli biblioteka odpowiedzialana za mapowanie instancje klas na obiekty JSON, oraz w drugą stronę.

W zabezpieczaniu naszego systemu pomogły nam rownież **Spring Security** oraz **Spring Session**. Drugie z nich dawało nam większą kontrole nad sesjami i jej atrybutami (tak jak **CSRF tokeny** oraz blokowane kupony, które rownież wiązane były z sesją) jak również umożliwiało nam to przechowywanie identyfiaktorów sesji w bazie danych. Jeśli chodzi o **Spring Security** miał on wpływ na implementacja autoryzacji oraz uwierzytelniania w naszym serwisie. To narzędzie sprawdzało podczas otrzymania zapytania przez serwer, czy dany użytkownik legitymujący się identyfikatorem sesji ma odpowiednie uprawnienia do przeglądania danej podstrony jak również wywoływania metod zarówna w kontrolerze oraz w klasach serwisowych. Umożliwił w łatwy sposób również implementacje różnego typu handler'ów

dla przypadków, gdy użytkownikowi nie udało się prawdiłowo zalogować/wylogować lub gdy właśnie probował przejść do strony, do której nie ma dostępu. Narzędzie to również automatycznie hashuje hasła podane przez użytkowników (po wcześniejszym zadeklarowaniu wybranego przez nas algorytmu haszującego), co jest znaczące jeśli chodzi o przechowywanie wrażliwych dla użytkoników danych.

Kolejnym narzędziem jest **Thymeleaf**, który został wykorzystanu do front-end'owej częsci projektu, czyli silnik umożliwiający tworzenie szablonów HTML, który jest w łatwy sposób jest integrowany ze **Spring MVC**. Dzięki możliwości korzystania m.in. z pętl, branch'ów oraz fragmentów kod HTML'owy stworzony przy użyciu Thymeleafa jest spójny oraz czytelny, gdyż miejscami przypomina kod jakiegoś standardowego języka programowania. Podczas prac nad modelem zostały wykorzystane również : **JavaScript**,**jQuery**,**CSS**.

Przydatnym narzędziem okazały się również **Spring WebFlow** oraz **Spring Mail**, które oferują wysokopoziomowe interfejsy dla wielostopniowych formularzy (ang. wizards) oraz wysyłce maili o zadanych wcześniej wyglądach (szablonach HTML). Pełna integracja z kodem Javowym w narzędziu odpowiedzialnym za formularze jest nie do opisania, ponieważ dzięki temu w formularzach jesteśmy wstanie zastosować branche, pracować na wprowadzonych przez użytkownika danych, jak również je przetwarzać na każdym etapie wykonywania formualrza.

Narzędziami przez nas użytymi były rownież frameworki ORM (ang. Object-Relational Mapping), a chodzi tu o **Hibernate**. Poprzez właśnie to narzędzie następowała komunikacja z naszą bazą danych **MySQL**, czyli wprowadzanie,edycja oraz usuwanie danych. **Hibernate** odpowiedzialny był za mapowanie krotek bazodanowych na instacje klas Modelu oraz instacji klas modelu na zapytania SQL'owe. Wykorzystany został rownież do wygenerowania gotowej bazy danych, na której później pracowaliśmy.

Używamy również frameworka `caffeine`, który służy do cachowania. W naszym systemie są cachowane statystyki, które są elementem najbardziej obciążającym system. Cache ulega dezaktualizacji po 20 minutach i musi zostać wtedy odświeżony. Dodanie do systemu ankiety o danym *id* powoduje również dezaktualizację cachu. Nie jest więc wykonywany niepotrzebny narzut pracy serwera, związany z dostarczaniem funkcjonalności statystyk w serwerze.

4.2 Rejestracja

Do implementacji mechanizmu rejestracji zostało wykorzystane narzędzie o nazwie **Spring WebFlow**, które jest wykorzystywane do wielostopniowego formularza (ang. wizard). W pliku XML-owym możemy odwoływac się do metod serwisowych, implementować na podstawie tego branch'e oraz decydowac na jakim etapie formularza jaka grupa walidacji będzie walidowana.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <flow xmlns="http://www.springframework.org/schema/webflow"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4        xsi:schemaLocation="http://www.springframework.org/schema/webflow http://
             www.springframework.org/schema/webflow/spring-webflow-2.4.xsd">
5
6      <var name="company" class="pwr.groupproject.vouchers.bean.form.
           NewUserCompanyForm"/>
7
8      <view-state id="step1" view="/signup/signup1.html" model="company"
9                  validation-hints="'validationGroup1'" >
10         <transition on="nextStep" to="step2">
11             <evaluate expression="userCompanyServiceImpl.validateUserCompany(
                   company,messageContext)" />
12         </transition>
13         <transition on="cancel" to="cancel" validate="false" bind="false" />
14     </view-state>
15
```

```
16      <view-state id="step2" view="/signup/signup2.html" model="company"
17                  validation-hints="'validationGroup2'">
18          <transition on="nextStep" to="success">
19              <evaluate expression="userCompanyServiceImpl.addUser(company)" />
20          </transition>
21          <transition on="cancel" to="cancel" validate="false" />
22          <transition on="previousStep" to="step1" validate="false" />
23      </view-state>
24
25      <end-state id="success" view="externalRedirect:/?acc=1" />
26      <end-state id="cancel" view="externalRedirect:/" />
27  </flow>
```

Listing 4.1: Listing kodu odpowiedzialnego za rejestrację.

Dla pierwszego etapu rejestracji do modelu (który jest ten sam dla całego trwania procesu) dodawany jest pusty formularz, czyli obiekt w którym będziemy przechowywać wszystkie dane wprowadzone przez użytkownika. Następnie po zatwierdzeniu pierwszego etapu przez użytkownika , sprwadzamy czy wprowadzone dane są zgodne z wymogami przedstawionymi przez serwer (np. czy podany przez użytkownika adres e-mail nie znajduje się już w naszej bazie danych) poprzez $< evaluateexpression =$ "$userCompanyServiceImpl.validateUserCompany(company, messageContext)$"/ >, w której wywołujemy klase serwisową naszego systemu. W przypadku, gdy dane nie spełnią wymogów wyświetlamy odpowiednie komunikaty błędów, w których jasno informujemy klienta co należy zrobić, aby owe wymogi spełnić. W przypadku gdy dane zostaną zatwierdzone przez serwer przechodzimy do drugiego etapu formularza $< view - stateid = śtep2$"... $< /view - state >$. Tam użytkownik uzupełnia formularz kolejną serią danych (np. hasłem oraz jego powtórzeniem) po zatwierdzeniu których serwer poraz kolejny waliduje dane wejściowe, tym razem już cały formularz. W przypadku przejścia wszystkich wymogów walidacji zostaje wykonana instrukcja $< transitionon = ńextStep$"$to = śuccess$»... $< /transition >$, co skutkuje wywołanem funkcji serwisowej $addUser(obj : Company)$. Po pomyślnym dodaniu użytkownika do bazy danych proces trafia do linijki 25 w powyższym listingu i zostaje przekierowany na strone startową, na której dzięki parametrowi "acc" wyświetli się odpowiedni komunikat informujący o konieczności aktywacji konta przed zalogowaniem się.

4.3 Resetowanie hasła dla konta

```
1   @RequestMapping(value = "forgot_password", method = RequestMethod.GET)
2   public String forgottenPassword(Model model) {
3     model.addAttribute("form", new ForgotPasswordForm());
4     return "auth/forgot_password.html";
5   }
6
7   @RequestMapping(value = "forgot_password", method = RequestMethod.POST)
8   public String forgottenPassword(@ModelAttribute(name = "form") @Validated
        ForgotPasswordForm forgotPasswordForm, BindingResult bindingResult) {
9     if (bindingResult.hasErrors())
10      return "auth/forgot_password.html";
11
12    UserCompany userCompany = userCompanyService.getUserByUserName(
          forgotPasswordForm.getUserName());
13    if (userCompany == null) {
14      bindingResult.rejectValue("userName", "email.dont.exist", messageSource.
            getMessage("message.wrong.email", null, LocaleContextHolder.getLocale()));
15      return "auth/forgot_password.html";
16    } else if (!userCompany.isEnabled()) {
17      bindingResult.rejectValue("userName", "account.not.activated", messageSource.
            getMessage("messages.account.not.activated", null, LocaleContextHolder.
            getLocale()));
```

```
18      return "auth/forgot_password.html";
19    } else {
20      PasswordResetToken passwordResetToken = tokenService.
            generateNewPasswordResetToken(userCompany);
21      mailService.sendPasswordResetEmail(passwordResetToken.getToken(), userCompany
            .getUsername());
22    }
23
24    return "redirect:/?acc=5";
25  }
```

Listing 4.2: Listing kodu obsługującego odzyskiwanie hasła.

Metoda oznaczona adnotacją $@RequestMapping(value = "/reset_password", method = RequestMethod.GET)$ przygotowywuje pusty formularz który ma zawierać jedynie adres e-mail konta dla którego użytkownik chce zresetować hasło. W przypadku, gdy użytkownik poda adres e-mail, który faktycznie znajduje się w naszej bazie danych zostaje generowany jednorazowy token (co zostało przedstawione w listingu poniżej), który zostaje wysłany w mail'u na wcześniej wskazany adres e-mail.

```
1   @Override
2   public PasswordResetToken generateNewPasswordResetToken(UserCompany userCompany)
        {
3     tokenDao.deleteUsersResetTokens(userCompany.getUsername());
4     PasswordResetToken passwordResetToken = new PasswordResetToken();
5     passwordResetToken.setUserCompany(userCompany);
6     passwordResetToken.setToken(RandomString.make(TOKEN_LENGTH));
7     tokenDao.addPasswordResetToken(passwordResetToken);
8     return passwordResetToken;
9   }
10
11  @Override
12  public PasswordResetToken validatePasswordResetToken(String passwordResetToken)
        throws WrongTokenException {
13    try {
14      return tokenDao.getPasswordResetTokenByToken(passwordResetToken);
15    } catch (NoResultException ex) {
16      throw new WrongTokenException();
17    }
18  }
```

Listing 4.3: Listing kodu generującego token do odzyskiwania hasła.

Tylko i wyłącznie jeden token odpowiadający za resetowanie hasła może być pisany do konta, tak więc przed dodaniem nowego "PasswordResetToken" ewentualny obiekt, który mógł istnieć wcześniej jest kasowany.

```
1   @RequestMapping(value = "/reset_password", method = RequestMethod.GET)
2   public String resetPassword(@RequestParam("t") String token, Model model) {
3     try {
4       tokenService.validatePasswordResetToken(token);
5       ResetPasswordForm form = new ResetPasswordForm();
6       form.setResetPasswordToken(token);
7       model.addAttribute("resetPasswordForm", form);
8       model.addAttribute("tokenStatus", TokenStatus.OK);
9     } catch (WrongTokenException ex) {
10      model.addAttribute("tokenStatus", TokenStatus.WRONG);
11    }
12    return "token/reset_password.html";
13  }
```

```
14
15 @RequestMapping(value = "/reset_password", method = RequestMethod.POST)
16 public String resetPassword(@ModelAttribute @Validated ResetPasswordForm
       resetPasswordForm, BindingResult bindingResult, Model model) {
17   if (bindingResult.hasErrors()) {
18     try {
19       tokenService.validatePasswordResetToken(resetPasswordForm.
             getResetPasswordToken());
20       model.addAttribute("tokenStatus", TokenStatus.OK);
21     } catch (WrongTokenException ex) {
22       model.addAttribute("tokenStatus", TokenStatus.WRONG);
23     }
24     return "token/reset_password.html";
25   }
26   userCompanyService.changePassword(resetPasswordForm);
27   return "token/reset_password_success.html";
28 }
```

Listing 4.4: Listing kodu resetującego hasło.

Użytkownik klikający w link w mailu wysłanym przez nasz system przekazujemy w postaci parametru zapytani *GET* wcześniej wygenerowany token. Po wcześniejszej walidacji tokenu użytkwonik jest przeniesiony do strony zawierający formularz, w którym podaje dwa razy to samo nowe hasło, którym od momentu potwierdzenia formularza będzie się posługiwał.

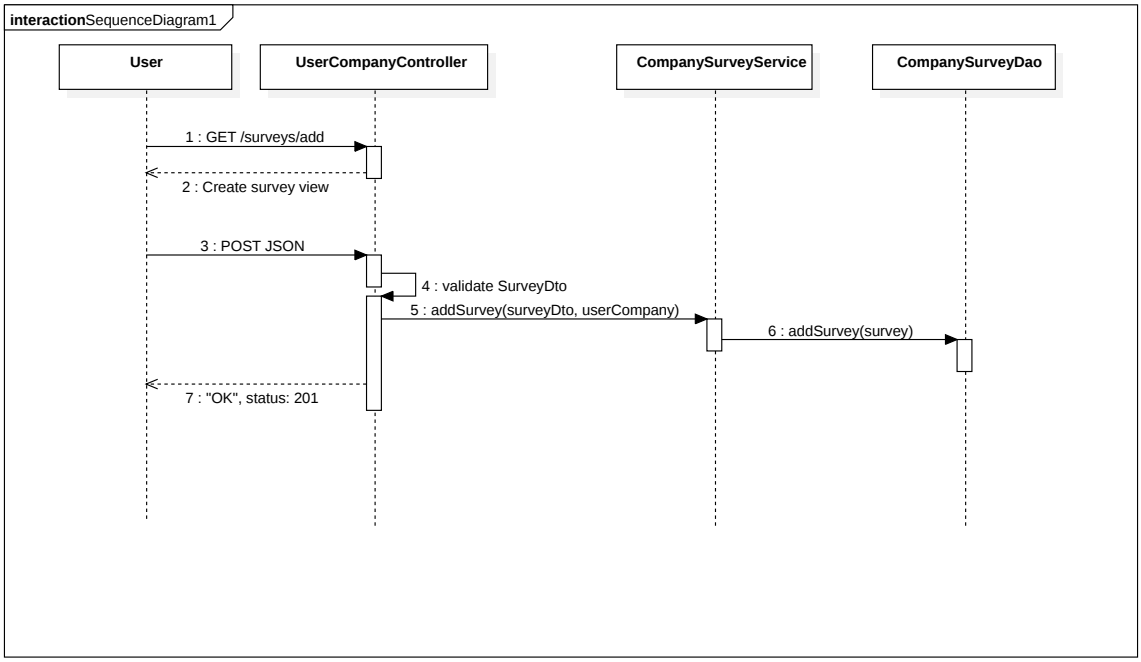
4.4 Tworzenie ankiety

Proces tworzenia ankiety zaczyna się w momencie wysłania żądania `GET` przez zalogowaną firmę pod adres `/surveys/add`. Po stronie klienta jest wczytywany panel do tworzenia ankiety pozwalający na bieżącą edycję ankiety w trakcie jej tworzenia. Panel pozwala na dowolne zmiany nazwy, opisu ankiety oraz kolejności pytań. Jest możliwe również usuwanie poszczególnych pytań z ankiety. Możliwy jest również podgląd ankiety w czasie rzeczywistym. Do pól odpowiadających za nazwę oraz opis ankiety są podłączone słuchacze zdarzeń, które po zmianie wartości wspomnianych pól wywołują *callback*, wprowadzający zmiany w podglądzie ankiety. Pytania w czasie tworzenia ankiety są przechowywane w tablicy obiektów w języku `Javascript`. Każde dodane pytanie ma podmienianą treść przy użyciu wyrażenia regularnego, dzięki czemu od strony front-endu pytania zabezpieczane są przed próbą wstrzyknięcia złośliwego skryptu. Dla znaków `"`, `'`, `/`, `<` oraz `>` są stosowane sekwencje ucieczkowe.

Po nacisnięciu przycisku `Dodaj ankietę`, do serwera wysyłane jest żądanie `POST` zawierające w ciele `JSON`, który jest automatycznie mapowany na obiekt z języka `Java`, dzięki zastosowaniu parsera `Jackson`. Obiekt klasy `SurveyDto` jest walidowany pod kątem zawartości niedozwolonych znaków oraz braku pól pustych. Jeżeli walidacja się powiedzie, do klienta jest zwracana odpowiedź ze statusem `HttpServletResponse.SC_CREATED` i po stronie klienta następuje przekierowanie do panelu zarządzania ankietami. W przypadku, gdy walidacja się nie powiedzie, do klienta zwracana jest odpowiedź ze statusem `HttpServletResponse.SC_NOT_ACCEPTABLE` i następuje powiadomienie o niepowodzeniu dodania ankiety.

Po udanej walidacji obiektu klasy `SurveyDto` wywoływana jest metoda `addSurvey` z serwisu `CompanySurveyService`, która dodaje utworzoną ankietę do bazy danych. Przed dodaniem ankiety do bazy danych, na podstawie obiektu `SurveyDto` jest tworzony obiekt klasy `Survey`, który jest integralną częścią modelu aplikacji. Dzięki zastosowaniu obiektów klasy `SurveyDto` do komunikacji z klientem, możliwe jest oszczędność na przesyle danych, gdyż do komunikacji nie są wykorzystywane niepotrzebne z punktu widzenia klienta dane, używane jednakże w bazie. Szybsza komunikacja została uzyskana kosztem narzutu pracy serwera, którą trzeba przeznaczyć na mapowanie obiektu klasy `SurveyDto` na obiekt klasy `Survey`.

Zmapowany obiekt klasy `Survey` jest w kolejnym kroku zapisywany w bazie danych, przy użyciu frameworka `Hibernate`. Framework ten pozwala na relacyjno-obiektowe mapowanie, dzięki czemu zobowiązanie do zapisania obiektu do bazy danych może być przesunięte na ten framework. Więcej o działaniu frameworka w projekcie można przeczytać w rodziale trzecim.



Rysunek 4.1: Diagram sekwencji dla poprawnego dodania nowej ankiety.

Na powyższym diagramie widoczne są połączenia między komponentami systemu dla dodawania nowej ankiety. Klasa `UserCompanyController` służy do komunikacji między użytkownikiem a resztą systemu. Walidacja zaznaczona na diagramie jako odwołanie `UserCompanyController` jest w rzeczywistości oddelegowana do walidatora, który dzięki zastosowanym w kodzie adnotacjom zna reguły walidacji pól w klasach. Oddelegowanie walidacji do walidatora jest oznaczone przy życiu adnotacji `@Validated` w argumencie kontrolera, który jest wiązany z ciałem żądania HTTP.

4.5 Dodawanie kuponu

Proces dodawania kuponu zaczyna się w momencie wysłania żądania GET przez zalogowaną firmę pod adres `/surveys/id/addVoucher`, gdzie id jest id wcześniej utworzonej ankiety. Po stronie klienta jest wczytywany panel do dodawania kuponu do wybranej ankiety. Panel pozwala na ustawienie rodzaju zniżki, jej wartości, opisu kuponu, a także czasu ważności kuponu.

Przy dodawaniu nowego kuponu tworzymy dane przypisywane do danego żądania i umieszczamy je jako atrybuty w `Modelu`. Dane te po wysłaniu odpowiedzi lub kolejnego żądania są wiązane z odpowiednimi obiektami w `Javie`. Dzięki takiemu rozwiązaniu można łatwo kontrolować walidację formularzy i łatwo sczytywać z nich dane, które następnie są procesowane w systemie.

```
1 @RequestMapping(value = "/surveys/{id}/addVoucher", method = RequestMethod.GET)
2     public String voucher(@AuthenticationPrincipal UserCompany userCompany,
          @PathVariable("id") int surveyId, Model model) {
3         if(!checkForSurveyExistence(surveyId,userCompany)){
4             return "error.html";
5         }
6         model.addAttribute("surveyId", surveyId);
```

```
7           model.addAttribute("discountType", DiscountType.values());
8           model.addAttribute("voucherForm",new VoucherForm());
9
10          return "my_account/vouchers/add_voucher";
11      }
```

Listing 4.5: Listing kodu wywoływanego przy otrzymaniu żądania GET dla dodawania kuponu.

W linijce pierwszej został określony mapping żądania `GET`. W linijkach 6-8 dodawane są atrybuty do modelu, które następnie będą zapełniane danymi, które dostarczy klient. W linijce 10 jako odpowiedź na żądanie jest zwracany HTML z widokiem panelu dodawania kuponu.

Po naciśnięciu przycisku `Zatwierdź`, do serwera wysyłane jest żądanie POST z obiektem `VoucherForm`. Obiekt klasy `VoucherForm` jest walidowany pod kątem zawartości niedozwolonych znaków oraz braku pól pustych. Jeżeli walidacja się powiedzie, do klienta zwracane są odpowiedzi analogicznie jak w przypadku procesu tworzenia ankiety.

Po udanej walidacji obiektu klasy `VoucherForm` wywoływana jest metoda `addVouchersForm` z serwisu `CompanySurveyService`, która dodaje utworzony kupon do bazy danych, łącząc go z odpowiednią ankietą. Zmapowany obiekt klasy `Voucher` jest w kolejnym kroku zapisywany w bazie danych, przy użyciu frameworka Hibernate. Proces dodawania kuponu jest zatem analogiczny, jak proces dodawania nowej ankiety.

4.6 Wypełnianie ankiety

Proces wypełniania ankiety zaczyna się od kliknięcia przez użytkownika przycisku `Wypełnij ankietę` na pasku nawigacyjnym. Użytkownik wybiera interesującą go firmę, a następnie ankietę. Ankiety są przedstawione w sposób jasno informujący o nagrodzie obowiązującej w zamian za jej wypełnienie. Wybranie ankiety blokuje jeden z kodów promocyjnych dla danego kuponu i wiąże go z identyfikatorem sesji użytkownika. Blokada kuponu dla użytkownika trwa przez 20 minut, a po upłynięciu tego czasu sesja wygaśnie i ankieta będzie musiała być wypełniona kolejny raz.

```
1  @Override
2      public void doFilter(ServletRequest request, ServletResponse response,
           FilterChain chain) throws IOException, ServletException {
3          HttpServletRequest httpServletRequest = (HttpServletRequest) request;
4          HttpSession session = httpServletRequest.getSession(true);
5          Integer vDateId = (Integer) session.getAttribute("vCode");
6          if (vDateId != null) {
7              try {
8                  VoucherCodeDate voucherCodeDate = companySurveyService.
                       getVoucherCodeDateById(vDateId);
9                  LocalDateTime localDateTime = LocalDateTime.now().minusMinutes
                       (20);
10                 if (Date.from(localDateTime.atZone(ZoneId.systemDefault()).
                       toInstant()).compareTo(voucherCodeDate.getUseDate()) > 0) {
11                     session.removeAttribute("vCode");
12                     companySurveyService.deleteVoucherCodeDate(vDateId);
13                 }
14             } catch (Exception e) {
15                 if(!(e instanceof NullPointerException))
16                     e.printStackTrace();
17                 session.removeAttribute("vCode");
18             }
19         }
20         chain.doFilter(request, response);
21     }
```

Listing 4.6: Listing kodu zwalniającego kod kuponu po wygaśnięciu sesji

Powyższy listing prezentuje metodę z klasy `VoucherCodeFilter`. Filtr ten dodany jest do łańcucha filtrów, który jest rekurencyjnie przechodzony za każdym razem, gdy do serwera trafia żądanie. Po przejściu przez filtry otrzymujemy zmodyfikowane żądanie, które następnie jest procesowane przez `dispatcherServlet`, ustalający co to za żądanie i do jakiego adresu się odnosi. Filtry i `dispatcherServlet` są częścią frameworka `Spring Boot`. Po trafieniu żądania do filtra `VoucherCodeFilter` następuje sprawdzenie, czy w atrybutach sesji powiązanej z danym żądaniem znajduje się zablokowany kod kuponu. Jeżeli z identyfikatorem sesji związany jest jakiś kupon to w linijkach 8-10 sprawdzane jest czy nie minął czas przeznaczony na wypełnienie ankiety. Jeżeli czas został przekroczony to w linijkach 11-12 usuwane jest powiązanie sesji z kodem kuponu i kod wraca do puli.

Za proces wypełnienia ankiety i dostarczenia użytkownikowi kuponu odpowiada `Spring WebFlow`. W rozdziale trzecim zostało opisane działanie tego komponentu frameworku `Spring`. Służy on do tworzenia wieloetapowych formularzy, tzw. *wizardów*. Dzięki zastosowaniu etapowego podejścia nie jest możliwe odwołanie się do adresu, który przekazuje kupon użytkownikowi, bez wcześniejszego wypełnienia ankiety. `WebFlow` umożliwia również kontrolę odblokowywania kuponów w przypadku, gdy użytkownik zrezygnuje z wysyłania ankiety. Po kliknięciu przycisku `Anuluj` w ankiecie wysyłane jest żądanie do serwera, które usuwa powiązanie kuponu z identyfikatorem sesji użytkownika. Gdy użytkownik zamknie okno przeglądarki kupon zostanie zwolniony dopiero po wygaśnięciu sesji.

Po każdym kroku formularza wypełniania ankiety, dane są walidowane. Nie jest możliwe przejście do kolejnego kroku, bez pozytywnej walidacji danych z poprzedniego etapu formularza. Do walidacji wypełnionej ankiety używana jest metoda `validateAnsweredSurveyForm` z serwisu `CompanySurveyService`.

Drugim krokiem formularza wieloetapowego jest podanie przez użytkownika danych statystycznych, takich jak wiek oraz kraj pochodzenia. Dane te również są walidowane. W tym kroku możliwe jest również podanie adresu e-mail, na który zostanie przesłany kupon. Jest to jednak pole opcjonalne, gdyż jednym z założeń niefunkcjonalnych aplikacji było zapewnienie użytkownikom anonimowości. W

przypadku podania przez użytkownika adresu e-mail, wywoływana jest metoda `sendVoucherCodeEmail` z serwisu `mailService`, która przesyła kod na podany adres e-mail.

```
1  private boolean send(String destinationEmail, String subject, Context ctx,
2          String eMailTemplate) {
3          MimeMessage eMailMessage = mailSender.createMimeMessage();
4          MimeMessageHelper eMailMessageHelper;
5          try {
6              eMailMessageHelper = new MimeMessageHelper(eMailMessage, true);
7              eMailMessageHelper.setTo(destinationEmail);
8              eMailMessageHelper.setSubject(subject);
9
10             final String htmlContent = this.htmlTemplateEngine.
11                     process(eMailTemplate, ctx);
12             eMailMessageHelper.setText(htmlContent, true);
13             mailSender.send(eMailMessage);
14             return true;
15         } catch (MessagingException e) {
16             e.printStackTrace();
17             return false;
18         }
19     }
```

Listing 4.7: Listing kodu wysyłającego e-mail.

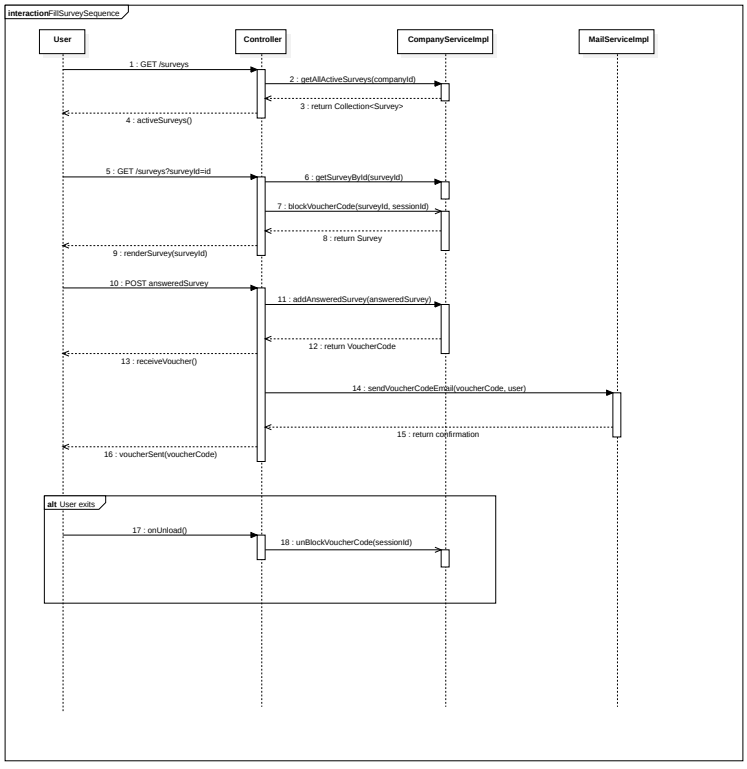
Do wysyłania wiadomości e-mail przez system służy klasa `JavaMailSender`, która do komunikacji wykorzystuje protokół SMTP. Wszystkie wiadomości tworzone przez system, nie tylko te zawierające w sobie kupony promocyjne, są budowane w standardzie MIME. Framework `Spring` pozwala na łatwe tworzenie szablonów wiadomości e-mail, które można dynamicznie zapełniać zawartością w trakcie działania systemu. W powyższym listingu została przedstawiona metoda odpowiadająca za wysyłanie wiadomości e-mail w serwisie `MailService`. Na początku tworzony jest obiekt wiadomości w standardzie MIME, który jest następnie wypełniany wartościami takimi, jak adresat i nadawca. Zawartość wiadomości jest generowany przy użyciu silnika do szablonów pochodzącego z frameworka `Spring`. Przekazując do silnika szablonów kontekst wiadomości, zawierający niezbędne wartości, wykorzystywane przy dynamicznym tworzeniu treści maila, można w łatwy sposób uzyskać kod HTML, który następnie zostanie przesłany jako wiadomość do nadawcy.

Bezpośrednio przed wysłaniem do użytkownika kuponu wiadomością e-mail, bądź przed przekierowaniem użytkownika na stronę z kodem promocyjnym, następuje dodanie ankiety do bazy danych i usunięcie powiązania kuponu z identyfikatorem sesji. Dodanie wypełnionej ankiety do bazy danych jest przeprowadzane analogicznie, jak dodawanie kuponu lub ankiety. Obiekt jest zapisywany w bazie danych przy użyciu frameworka `Hibernate`, który zapewnia mapowanie relacyjno-obiektowe. Za usunięcie powiązania kuponu z identyfikatorem sesji odpowiada metoda `confirmAnsweringSurvey` z serwisu `CompanySurveyService`.

```
1   @CacheEvict(value = {"surveyStat","ansList"},key = "#surveyId")
2     @Override
3     public VoucherCodeDto confirmAnsweringSurvey(Integer surveyId,
4       AnsweredSurveyForm answeredSurveyForm, RequestContext requestContext) {
5         HttpSession httpSession = ((HttpServletRequest) requestContext.
              getExternalContext().getNativeRequest()).getSession(true);
6         Integer vCodeId = (Integer) httpSession.getAttribute("vCode");
7         httpSession.removeAttribute("vCode");
8         VoucherCode voucherCode = deployVoucherCode(vCodeId);
9         String email = answeredSurveyForm.getEmail();
10        if(email != null && !email.equals("")) {
11            mailService.sendVoucherCodeEmail(voucherCode, email);
12        }
13        return new VoucherCodeDto(voucherCode);
14    }
```

Listing 4.8: Listing kodu blokującego kod promocyjny.

Powyższy listing pokazuje działanie metody odpowiedzialnej za dostarczenie kuponu do użytkownika i zwolnienie jego powiązania z identyfikatorem sesji. W linijkach 5-8 usuwane jest powiązanie kodu kuponu z identyfikatorem sesji, natomiast w linijkach 9-12 jest wysyłana wiadomość e-mail, o ile użytkownik podał opcjonalny adres e-mail. Metoda zwraca `Data Transfer Object VoucherCodeDto`, który następnie jest wyświetlany użytkownikowi po przekierowaniu. Warto również zwrócić uwagę na dezaktualizację cache'a po dodaniu nowej ankiety. Cache jest używany w systemie do przechowywania statystyk. Więcej o cache'u oraz statstykach można przeczytać w podrozdziale 4.6.



Rysunek 4.2: Diagram sekwencji dla wypełniania ankiety.

Powyższy diagram sekwencji przedstawia proces wypełniania ankiety w aplikacji webowej.

4.7 Wypełnianie ankiety przez aplikację mobilną

Aby móc wypełnić ankietę w aplikacji mobilnej, w tle wykonuje się wiele procesów zarówno aby dostarczyć odpowiednią ankietę do aplikacji mobilnej, jak i odpowiedzi udzielone przez użytkownika do serwera. Najpierw aplikacja mobilna, po wybraniu przez użytkownika firmy oraz ankiety, przesyła za pomocą frameworka retrofit2 metodą post informacje na temat interesującej użytkownika ankiety. Serwer w momencie kiedy odbierze informacje, tworzy sesje dla naszej aplikacji, rezerwuje jeden kupon dla danego tokena sesji, oraz przesyła jako odpowiedź wyżej wymieniony token, wraz z pytaniami i typami pytań do konkretnie wybranej przez nas ankiety. Aplikacja mobilna odbierając plik JSON, zapisuje sobie plik cookie (który zapewnia nam utrzymanie sesji), oraz załącza pytania do listview w naszej nowo utworzonej aktywności. W trakcie dołączania pytań, aplikacja weryfikuje dla każdego z pytań jakiego jest ono rodzaju, i na tej podstawie tworzy kolejne elementy listview aby wyglądem pasowały do typu pytania. Na samym dole jest tworzony przycisk, służący do zaakceptowania naszej ankiety. Aplikacja w momencie gdy ankieta jest gotowa do wypełniania, umożliwia wysłanie ankiety dopiero wtedy, gdy zweryfikuje że odpowiedzi do ankiet są „niepuste". Gdy użytkownik naciśnie przycisk akceptujący ankietę, i aplikacja dokona poprawnej walidacji ankiety następuje pobranie odpowiedzi z poszczególnych pól ankiety, umieszczenie ich w pliku JSON, oraz przesłanie ich metodą POST do serwera. Do naszej metody załączany jest plik cookies w którym przechowywany jest token sesji. W następnym kroku serwer przetwarza informacje przekazane przez aplikacje, zaś aplikacja czeka na odpowiedź z serwera. Jeżeli serwer nie był w stanie przetworzyć danych, (albo akurat w momencie gdy wysyłaliśmy nasza ankietę nie było zasięgu) nasza aplikacja mobilna informuje użytkownika o wystąpieniu błędu i cofa nas do ankiety. W przeciwnym przypadku serwer powinien zwrócić do aplikacji mobilnej plik JSON wraz z kodem promocyjnym oraz pozostałymi informacjami odnośnie otrzymanego vouchera. Gdy aplikacja odbierze wyżej wymieniony kod promocyjny, z automatu przekieruje nas do zakładki z posiadanymi kuponami. W aplikacji mobilnej w zależności od tego czy podamy adres e-mail, możemy otrzymać kupon dodatkowo na konto e-mail. Warto dodatkowo zwrócić uwagę na fakt, że aplikacja została zaopatrzona w obsługę rezygnacji z wypełniania ankiety. Oznacza to, że w momencie gdy będziemy chcieli w aplikacji wrócić do menu głównego, albo zakończymy działanie aplikacji, to nasz program za pomocą metody POST wyśle informacje do serwera wraz z tokenem sesji o odblokowaniu vouchera który był dla nas wcześniej zarezerwowany.

4.8 Statystyki

```
1  @Cacheable("surveyStat")
2  @Override
3  public SurveyStatisticsDto getSurveysStatistics(int surveyId) {
4    Collection<AnsweredSurvey> answeredSurveys = getAllAnsweredSurveysWithDetails(
         surveyId);
5    Survey survey = companySurveyService.getSurveyByIdWithQuestion(surveyId);
6    SurveyStatisticsDto surveyStatisticsDto = new SurveyStatisticsDto();
7    surveyStatisticsDto.setAmmount(answeredSurveys.size());
8    surveyStatisticsDto.setSurveyName(survey.getSurveyName());
9
10   //average age
11   double averageAge = answeredSurveys.stream().mapToInt(a -> a.getUser().getAge()
         ).average().orElse(0.0);
12   surveyStatisticsDto.setAge(averageAge);
13
14   //average country
15   List<String> countries = answeredSurveys.stream().map(a -> a.getUser().
         getCountry()).collect(Collectors.groupingBy(Function.identity(), Collectors.
         counting())).entrySet().stream().sorted(Comparator.comparingLong(Map.Entry::
         getValue)).limit(3).map(Map.Entry::getKey).collect(Collectors.toList());
16   while (countries.size() != 3)
17     countries.add("N/A");
18   surveyStatisticsDto.setCountry(countries.toArray(new String[3]));
19
```

```
20    if (answeredSurveys.size() == 0)
21      return surveyStatisticsDto;
22
23    //initialaizing iterators
24    Iterator<AnsweredSurvey> answeredSurveyIterator = answeredSurveys.iterator();
25    Iterator<Question> qIterator = survey.getQuestions().iterator();
26    Iterator<Answer>[] aIteratorArray = new Iterator[answeredSurveys.size()];
27    IntStream.range(0, aIteratorArray.length).forEach(i -> aIteratorArray[i] =
          answeredSurveyIterator.next().getAnswersList().iterator());
28
29    int answersSize = answeredSurveys.size();
30    List<QuestionStatisticsDto> questionStatisticsDtoList = surveyStatisticsDto.
          getQuestionWithAnswersList();
31    while (qIterator.hasNext()) {
32      Question q = qIterator.next();
33      QuestionType qType = q.getQuestionType();
34      QuestionStatisticsDto questionStatisticsDto = new QuestionStatisticsDto();
35      questionStatisticsDto.setQuestionBody(q.getQuestionBody());
36      questionStatisticsDto.setQuestionType(qType);
37
38      switch (qType) {
39        case OPEN:
40          Arrays.stream(aIteratorArray).forEach(Iterator::next);
41          questionStatisticsDto.setAnswers(null);
42          break;
43        case RANGED:
44          double average = 0;
45          for (Iterator<Answer> anAIteratorArray : aIteratorArray) {
46            String temp = anAIteratorArray.next().getAnswer();
47            average += Double.parseDouble(temp);
48          }
49          questionStatisticsDto.getAnswers()[0].setAnswersStat(Double.toString(
                average / answersSize));
50          break;
51        default:
52          double[] apperances = new double[4];
53          for (Iterator<Answer> anAIteratorArray : aIteratorArray) {
54            Answer a = anAIteratorArray.next();
55            String[] splited = a.getAnswer().split(",");
56            for (String s : splited) {
57              switch (s) {
58                case "A":
59                  apperances[0]++;
60                  break;
61                case "B":
62                  apperances[1]++;
63                  break;
64                case "C":
65                  apperances[2]++;
66                  break;
67                case "D":
68                  apperances[3]++;
69                    }
70            }
71          }
72          IntStream.range(0, apperances.length).forEach(a -> questionStatisticsDto.
                getAnswers()[a].setAnswersStat(Double.toString(100 * apperances[a] /
                answersSize)));
73          questionStatisticsDto.setPossibleAnswers(q.getPossibleAnswers());
74          break;
75      }
```

```
76
77          questionStatisticsDtoList.add(questionStatisticsDto);
78      }
79      return surveyStatisticsDto;
80  }
```

Listing 4.9: Listing kodu pobierającego statystyki dla danej ankiety.

Na powyższym listingu warto zauważyć, że metoda jest cache'owana tj. zapmietywane jest przez serwer wartość, która zostanie zwrócona dla danego argumentu. Przy dużej liczbie rozwiązanych ankiet obliczanie statystyk, może być stosunkow czasochłonne, a sam cache zapewnia nam, że nie statystyki dla tych samych danych nie będa liczone kilkukrotnie.

Pierwszym etapem jest pobranie z bazy danych listy wypełnionych ankiet, a następnie policzenie średniego wieku przy użyciu **Stream API**, **Lamba Expressions** oraz klasy generycznej **Optional<T>**. Przy użyiu tych samych "narzędzi" dostępnych w Javie zostaje obliczona posortowana lista krajów, względem częstości wypełniania danej ankiety w tym kraju. Lista ta jest ograniczona do 3 najlepszych wyników, a w przypadku gdy zawiera ona mniej niż 3 kraje zostaje wypełniona wyrażeniem "N/A". W przypadku gdy ankieta nie została rozwiązana zwracany jest pusty obiad, w innej sytuacji w pętli iterujemy po pytaniach w kolejnych odpowiedziach, tzn. na początku dla każdej odpowiedzi na pytanie numer 1 liczymy statystyki, następnie robimy to dla każdej istniejącej odpowiedzi na pytanie numer 2 itd. Jeżeli pytanie jest pytaniem typu "ranged" (tj. w skali od 0 do 10) liczona jest średnia arytmetyczna wyniku. Jeżeli jest to pytanie typu zamkniętego lub zamkniętego wieloktronego wyboru przy pomocy **Stream API**, **Lamba Expressions** oraz klasy generycznej **Optional<T>** liczona jest częstość występowania danej odpowiedzi (która należy do zbioru $\{A, B, C, D\}$). Następnie przy pomocy set'erów i get'erów klasy **SurveyStatisticsDto** obliczone wcześniej dane zostają "wrzucone" do obiektu wyjściowego, który jest ostatecznie zwracany przez metode.