

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Systemów Informatycznych

System wspierający zarządzanie i podział obowiązków domowych

Zuzanna Santorowska

Numer albumu 300262

promotor
mgr inż. Agnieszka Malanowska

WARSZAWA 2023

System wspierający zarządzanie i podział obowiązków domowych

Streszczenie.

Wiele domów zmaga się z problemem nierównorzędnej pracy na rzecz gospodarstwa domowego. Część osób pracuje dużo więcej niż inne, co często jest spowodowane brakiem ustalonego kryterium oceny, ile kto robi. W takim wypadku dobrym rozwiązaniem jest wspólne stworzenie wymagań i używanie systemu, który mógłby zrewidować, czy ustalenia są przestrzegane. Między innymi z tego powodu celem niniejszej pracy jest stworzenie aplikacji internetowej CommOn służącej do zarządzania obowiązkami domowymi. W celu zmotywowania użytkowników do wykonywania wszystkich (także tych nielubianych) obowiązków domowych, CommOn wprowadza dynamicznie zmieniającą się punktację zadań, tak aby te, które były dłużej niewykonywane, miały wyższy priorytet. CommOn jest aplikacją internetową o architekturze trójwarstwowej. Warstwa prezentacji powstawała w języku TypeScript, między innymi z użyciem szkieletu Angular. Warstwa logiki biznesowej została napisana za pomocą Javy oraz Springa. Warstwa danych składa się z bazy danych MySQL. W pracy przedstawiono stawiane aplikacji wymagania, szczegółowy opis jej budowy, testy, jakim była poddawana oraz zaprezentowano przykładowy scenariusz jej działania.

Słowa kluczowe: obowiązki domowe, aplikacja internetowa, architektura trójwarstwowa, MySQL, Spring, Java, Angular, TypeScript, grywalizacja

A system supporting the management and division of household chores

Abstract.

Many homes struggle with the problem of unequal work for the household. Some people work much more than others, which is often caused by the lack of an established criterion for assessing how much someone does. In this case, a good solution is to create requirements together and use a system that could review whether the arrangements are being followed. For this reason, the purpose of this thesis is to create a web application, named CommOn, for managing household chores. To motivate users to perform all household chores, including those they dislike, CommOn introduces a dynamically changing task score, so that those that have been uncompleted for a long time have a higher priority. CommOn is a web application with a three-tier architecture. The presentation layer was created in TypeScript, using the Angular framework. The business logic layer was written using Java and Spring. The data layer consists of a MySQL database. The thesis presents the requirements for the application, a detailed description of its construction, testing scenarios, and basic use cases.

Keywords: household chores, web application, three-tier architecture, MySQL, Spring, Java, Angular, TypeScript, gamification



.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanego z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp	9
2. Problematyka	10
2.1. Proponowane rozwiązanie	10
2.2. Grywalizacja	11
2.3. Przedłużenie moratorium psycho-społecznego	12
3. Przegląd dziedziny	13
3.1. Taksonomia	13
3.2. Charakterystyka istniejących aplikacji	14
3.3. Porównanie cech analizowanych aplikacji	17
4. Analiza wymagań	19
4.1. Wymagania funkcjonalne i przypadki użycia	19
4.2. Wymagania niefunkcjonalne	23
5. Zastosowane technologie	25
5.1. Kryteria wyboru narzędzi	25
5.2. Warstwa prezentacji	26
5.2.1. Narzędzia	26
5.2.2. Testowanie	27
5.3. Warstwa logiki biznesowej	27
5.3.1. Narzędzia	28
5.3.2. Testowanie	30
5.4. Pozostałe narzędzia	31
6. Projekt i implementacja	33
6.1. Architektura	33
6.1.1. Diagram ER bazy danych	34
6.1.2. Architektura warstwy logiki biznesowej	34
6.1.3. Architektura warstwy prezentacji	41
6.2. Implementacja	46
6.2.1. Przetwarzanie żądań użytkownika	46
6.2.2. Model relacyjny bazy danych	52
6.2.3. Wejście i wyjście	54
7. Testy	61
7.1. Testy automatyczne	61
7.1.1. Warstwa logiki biznesowej	61
7.1.2. Warstwa prezentacji	68
7.2. Testy manualne	71
8. Przykład użycia	72

9. Podsumowanie	84
Bibliografia	87
Wykaz symboli i skrótów	92
Spis rysunków	93
Spis tabel	94

1. Wstęp

Wiele gospodarstw domowych zmaga się z problemem polegającym na nierównym wkładzie pracy poszczególnych domowników w wykonywanie codziennych obowiązków. Niektórzy mieszkańcy pracują dużo więcej niż pozostali, a przyczyną takiego stanu rzeczy często jest brak jakiegokolwiek kryterium oceny tego, ile kto robi. Dobrym rozwiązaniem mogłoby być w takiej sytuacji wspólne stworzenie wymagań i zastosowanie systemu, który pomagałby w pilnowaniu, czy ustalenia są przestrzegane.

W ramach niniejszej pracy autorka zdecydowała się na próbę rozwiązania zaznaczonych problemów, ponieważ dostrzegła potrzebę takiego rozwiązania w swoim otoczeniu. Szczegółowy opis problemów związanych z brakiem usystematyzowanego sposobu podziału obowiązków domowych i ich przyczyn przedstawiono w rozdziale 2.

Rozwiązaniem, które zaimplementowano, jest aplikacja internetowa CommOn służąca do zarządzania obowiązkami domowymi. Umożliwia ona łączenie się użytkowników w gospodarstwa domowe i sprawdzenie stanu wykonania zadań przez współmieszkańców. Jej istotnym aspektem jest system punktowania obowiązków domowych. Z jednej strony, na koniec każdego tygodnia z konta każdego użytkownika odejmowana jest pewna liczba punktów, co ma symbolizować, że każdy partycypuje w utrzymaniu domu na miarę swoich możliwości. Z drugiej strony, CommOn wprowadza mechanizm dynamicznego zmieniania cen zadań, tak aby wyższy priorytet miało zadanie niewykonane przez dłuższy czas.

W ramach niniejszej pracy wykonano przegląd dziedziny, aby zorientować się w rynku istniejących aplikacji i zbadać zasadność tworzenia nowej. Jego wyniki opisano w rozdziale 3. Z przeglądu wynika, że istnieje luka w dziedzinie aplikacji wspomagających zarządzanie obowiązkami domowymi, szczególnie jeśli chodzi o aplikacje internetowe. Z tego powodu zdecydowano się na implementację tego rodzaju aplikacji. Przeprowadzono analizę wymagań stawianych tworzonemu narzędziu, która została opisana w rozdziale 4. Aby aplikacja stworzona była za pomocą nowych i wartościowych technologii, przeanalizowano dostępne opcje, ich wynik jest udokumentowany w rozdziale 5.

Efektem powyższych decyzji jest realizacja CommOn w językach Java [1] i TypeScript [2], z wykorzystaniem szkieletów Spring [3] i Angular [4]. Jako bazę danych wykorzystano MySQL [5]. W rozdziale 6 opisano projekt oraz implementację aplikacji CommOn. Następnie opisano rodzaje testów, jakie przeprowadzano na aplikacji, aby zwiększyć prawdopodobieństwo jej poprawnego działania, wyniki znajdują się w rozdziale 7. Zaprezentowano również przykład użycia obrazujący działanie oraz wygląd interfejsu graficznego (rozdział 8). Mocne i słabe strony stworzonej aplikacji, a także możliwości jej dalszego rozwoju, zostały omówione w rozdziale 9.

2. Problematyka

Z doświadczeń wielu znanych autorów wynika, że różnice w oczekiwaniach dotyczących wykonywania obowiązków domowych stanowią poważny problem wśród młodych ludzi wyjeżdżających na studia, którzy są zmuszeni współdzielić mieszkanie z innymi, często niezbyt dobrze znanymi, ludźmi. W takich wypadkach często osoba, której bardziej zależy na czystości, sprząta więcej niż pozostali. W takiej osobie z czasem buduje się poczucie niesprawiedliwości, które może powodować konflikty i napiętą atmosferę.

Aby rozwiązać ten problem, początkowo zastosowano najprostszą strategię. Polega ona na spisaniu wszystkich zadań i przypisaniu ich do poszczególnych osób. Po określonym czasie następuje rotacja zadań, aby ostatecznie każdy wykonał każde zadanie. Jeśli zadania nie mają określonej wartości, to nie da się ich porównywać. Z tego powodu tylko strategia, w której każdy wykonuje każde zadanie, jest sprawiedliwa. Niestety jest to rozwiązanie dalekie od idealnego. Każda osoba posiada własne preferencje, co powoduje, że częściej wykonuje pewne obowiązki od innych. W pierwotnym rozwiązyaniu każdy musiał wykonać każde zadanie. Bez uaktualniania systemu może się okazać, że pewne zadania są bardziej pracochłonne od innych, a ich wykonanie zajmuje więcej czasu, niż początkowo zakładano. Powoduje to notoryczne zaniedbywanie tych obowiązków. Wszystkie wymienione doświadczenia skłoniły autorkę do podjęcia próby rozwiązania tego problemu w ramach niniejszej pracy.

2.1. Proponowane rozwiązanie

Celem pracy jest wyprodukowanie systemu służącego zarządzaniu obowiązkami domowymi i ułatwiającego ich podział między domowników. Aplikacja będzie umożliwiać łączenie się użytkowników w gospodarstwa domowe oraz aktualizację informacji o zadaniach wykonanych przez innych. Nazwa realizowanej aplikacji brzmi CommOn, co nawiązuje do angielskiego słowa *common* (wspólny). W wymowie brzmi jak *come on!* (chodź!), czyli zwrot często używany w ramach zachęty do zrobienia czegoś.

Przewidziany jest system punktowania obowiązków domowych. Na koniec każdego tygodnia z konta każdego użytkownika będzie odejmowana odpowiednia liczba punktów. Takie rozwiązanie ma sugerować domownikom, że każdy na miarę swoich możliwości partytuje w utrzymaniu gospodarstwa domowego. Liczba pobieranych punktów może się różnić pomiędzy użytkownikami. Umożliwia to korzystanie z aplikacji rodzinom, które pragną podzielić się częścią obowiązków ze swoimi dziećmi. Motywacją rodziców może być korzystny wpływ obowiązków domowych na wychowanie dzieci [6]. Niejednorodny podział wymaganego wkładu pracy umożliwia dopasowanie aplikacji do indywidualnych potrzeb dziecka, na przykład jego wieku, ilości czasu wolnego albo czasu spędzonego

w szkole. Punkty przydzielane za poszczególne zadania będą się zmieniać zależnie od częstotliwości ich wykonywania. Jeśli jakieś zadanie długo nie było wykonane, to będzie miało wyższą wartość, aby zmotywować domowników do jego realizacji. Mechanizm ten spowoduje, że zadania dawno niewykonywane, a więc z założenia bardziej priorytetowe, będą bardziej atrakcyjne.

Grupy docelowe, które mogłyby najbardziej korzystać z CommOn, to studenci mieszkający we wspólnym mieszkaniu, rodziny z dziećmi, rodzice mieszkający ze swoimi dorosłymi dziećmi. Autorka jest studentką i cały pomysł pisania aplikacji wywodzi się z chęci rozwiązymania problemów organizacyjnych wewnątrz współdzielonych mieszkań. Dlatego właśnie jedną z głównych grup docelowych są studenci. Kolejną grupą są rodziny z dziećmi. Wielu rodziców chce wychować swoje dzieci na obowiązkowych dorosłych. Badania wskazują, że nałożenie na dzieci obowiązków domowych jest skorelowane z ich dobrobytem [6]. Z tego względu wielu rodziców zleca swoim pociechom pewne obowiązki w domu. Aplikacja mogłaby w łatwy i przyjemny sposób zachęcić dziecko do wykonywania jego obowiązków. Kolejną grupą, która mogłaby skorzystać na używaniu aplikacji, są rodzice mieszkający z dorosłymi dziećmi. Coraz więcej młodych osób w Polsce zwleka z wejściem w tradycyjnie rozumianą dorosłość [7]. Przyczyny tego zjawiska zostaną omówione w dalszej części tego rozdziału. W związku z tym sporo młodych osób, zarówno w trakcie studiów, jak i po ich zakończeniu, mieszka z rodzicami. Taka sytuacja często powoduje powstawanie konfliktów między wykonującymi większość obowiązków domowych rodzicami a dziećmi, które pozostają mało samodzielne. CommOn mogłoby pomóc wyznaczyć oficjalne wymagania w stosunku do dzieci. Pomoże to im stać się bardziej samodzielnymi. Aplikacja nakreśli również rodzicom, że ich potomstwo jest już dorosłe i należy od niego wymagać tyle, co od innych dorosłych osób. To, że młodzi nie będą traktowani ulgowo, może ich też zmotywować do opuszczenia domowego gniazda.

2.2. Grywalizacja

Grywalizacja (ang. *gamification*) [8] to wykorzystanie elementów gier i technik projektowania gier w kontekście niezwiązany z grami, mające na celu angażowanie ludzi, motywowanie do działania, pobudzanie do nauki i rozwiązywania problemów przy osiąganiu przy tym pożądanych zachowań lub innych założonych celów [8]. Ze względu na to, że jednym z problemów, które stara się rozwiązać CommOn, jest brak motywacji do wykonywania zadań, postanowiono skorzystać z doświadczenia projektantów gier, którzy opanowali sztukę angażowania ludzi do perfekcji. W przygotowanej aplikacji użyto następujących elementów obecnych w grach: zadania, punkty, pasek postępu pokazujący, ile jest jeszcze do zrobienia w danym tygodniu, tabela wyników. Wyświetlenie tabli

2. Problematyka

wyników to sposób na wzmacnianie rywalizacji, która potrafi mocno motywować do dalszej pracy.

2.3. Przedłużenie moratorium psycho-społecznego

Przedłużanie moratorium psycho-społecznego oznacza proces odraczania dorosłości przez młodych ludzi [7]. W tym kontekście dorosłość mierzona jest tradycyjnie rozumianymi kryteriami, takimi jak: zawarcie związku małżeńskiego lub zbudowanie stałego związku partnerskiego, rodzicielstwo, zakończenie edukacji formalnej, podjęcie stałej i realizowanej w pełnym wymiarze czasu aktywności zawodowej. Jak wskazuje Marianowska w [7], początkowo powodem opóźniania momentu wejścia w dorosłość były chęci osiągnięcia poziomu życia wyższego niż rodzice. Zmiana ustroju umożliwiła zarobienie większej ilości pieniędzy oraz wydanie ich w bardziej dowolny sposób niż to było możliwe w czasach komunizmu. Z tego względu młodzi ludzie często wybierali najpierw karierę i zdobycie pracy w nowym systemie, a dopiero potem decydowali się na założenie rodziny. Szybko zapechniający się rynek pracy powodował konieczność zdobywania kolejnych dyplomów studiów wyższych, przez co młodzi wydłużali okres edukacji formalnej, aby mieć szansę na coraz to bardziej wymagającym rynku pracy. Przychodzące z krajów zachodnich wzorce, takie jak kult sportowej sylwetki czy konsumpcjonizm powodowały też coraz większe wydatki, przez co wymagania finansowe uległy inflacji. Powodowało to coraz większe skupianie się na zarabianiu pieniędzy i w konsekwencji inne cele, takie jak rodzicielstwo, zeszły na dalszy plan. Kolejne pokolenia otrzymywały coraz więcej uwagi ze strony rodziców, co spowodowało wzrost liczby postaw roszczeniowych i wyuczoną bezradność. Opóźnienie momentu wyprowadzki było spowodowane przyzwoleniem nadopiekuńczych rodziców oraz kryzysem na rynku mieszkaniowym po 1989 roku. Liczba budowanych nowych mieszkań znaczaco zmalała, co przełożyło się na ich wyzsze ceny [7]. Wszystkie te czynniki doprowadziły do wydłużenia się okresu wkraczania w dorosłość i narastania związanych z tym problemów. Aplikacja CommOn może pomóc rozwiązać część problemów związanych z brakiem motywacji młodych ludzi do wyprowadzki, która jest spowodowana wygodą przebywania w domu, w którym często nie mają oni prawie żadnych obowiązków.

3. Przegląd dziedziny

W celu wykazania zasadności tworzenia nowej aplikacji do zarządzania obowiązkami domowymi przeprowadzono przegląd konkurencyjnych programów obecnie dostępnych na rynku. Z założenia poszukiwano rozwiązań dostępnych za darmo, dlatego wszelkie płatne aplikacje nie zostały uwzględnione. Poszukiwania prowadzono za pomocą wyszukiwarki Google [9] oraz sklepu Google Play [10] na komórkę, czyli w sposób, w który prawdopodobnie robią to potencjalni klienci CommOn. Sprawdzono 7 najbardziej popularnych aplikacji. Znaleziono aplikacje, które mają podobny cel do CommOn, jednak żadna z nich nie spełnia kluczowego założenia, jakim jest dynamiczne zwiększanie się ceny zadania w czasie. Oprócz oczywistych plusów zorientowania się w rynku dodatkowymi korzyściami, wynikającymi z przeprowadzonego przeglądu, były inspiracja dobrymi rozwiązaniami oraz identyfikacja błędów, które popełnili twórcy aplikacji. Aby porównywać ze sobą dwa obiekty, należy najpierw ustalić słownik pojęć. Z tego względu w tym rozdziale najpierw zdefiniowano słownik pojęć, następnie przeanalizowano wszystkie sprawdzane aplikacje, aby na końcu przejść do porównania ich ze sobą.

3.1. Taksonomia

Taksonomia jest to systematyczny sposób opisu lub klasyfikacji obiektów danej kategorii. W celu porównania tworzonej aplikacji z konkurencją rynkową należy ustalić sposób opisu różnic i podobieństw. Innymi słowy, istnieje konieczność zdefiniowania taksonomii, która będzie określać wymiary, w których aplikacje mogą realizować różne funkcjonalności. Definicje przyjęte na potrzeby niniejszej pracy umieszczone w tabeli 3.1.

Konkurencyjne aplikacje dostępne na rynku mogą posiadać wiele przydatnych funkcji, które warto wziąć pod uwagę podczas analizy. Lista zidentyfikowanych istotnych funkcji znajduje się poniżej.

1. Tworzenie zadań
2. Punktowanie zadań
3. Zainicjowanie systemu przykładowymi zadaniami
4. Okresowe powtarzanie zadań
5. Zmiana punktacji zadań w zależności od ostatniego czasu wykonania
6. Okresowe pobieranie punktów
7. Przypisanie zadania do konkretnego użytkownika
8. Możliwość cofnięcia oznaczenia zadania jako wykonane przez użytkownika innego niż wykonawca zadania w ramach jednego domu
9. Utworzenie rankingu domowników
10. Zdobywanie osiągnięć
11. Zdobywanie nagród

3. Przegląd dziedziny

12. Osiąganie kolejnych poziomów
13. Podział domu na pomieszczenia
14. Automatyczna ponowna ocena wartości zadań, które notorycznie nie są wykonywane na czas
15. Łączenie kont użytkowników w jeden dom

W kolejnych rozdziałach przedstawiono krótką charakterystykę każdej z przeglądanych aplikacji. W tabeli 3.2 znajduje się porównanie tych narzędzi pod kątem wyżej wymienionych właściwości.

Tabela 3.1. Definicje pojęć stosowanych w aplikacji

Pojęcie	Definicja
Użytkownik, mieszkaniec	Osoba fizyczna korzystająca z aplikacji
Dom, gospodarstwo domowe	Obiekt reprezentujący zbiór użytkowników, którzy w rzeczywistości mieszkają w jednym lokalu. Wszystkie osoby wewnątrz domu mają wspólną pulę zadań.
Pokój, pomieszczenie	Pomieszczenie, w którym jest wykonywane zadanie. Pokoje dzielą zadania na kategorie ułatwiające użytkownikom korzystanie z aplikacji.
Zadanie, czynność, obowiązek	Czynność, którą należy wykonać w domu. Użytkownicy mogą zaliczać zadania, co oznacza wykonanie określonej czynności w świecie rzeczywistym.
Punkt	Jednostka reprezentująca ilość pracy i czasu koniecznych do realizacji zadania. Zadanie jest tym więcej warte, im więcej pracy lub czasu należy na nie poświęcić.
Nagroda	Przedmiot lub czynność, którą użytkownicy mogą zdobyć za wyznaczane przez siebie nawzajem cele.
Osiągnięcie	Odznaczenie, które użytkownicy mogą zdobyć za cele wyznaczone przez projektanta aplikacji.

3.2. Charakterystyka istniejących aplikacji

Tody [11] to zadowalająca aplikacja mobilna pozwalająca na tworzenie pomieszczeń i zadań wewnętrz nich. Posiada pasek postępu obrazujący, jak dużo obowiązków w danym pokoju jest jeszcze do zrobienia. Podczas tworzenia zadań podpowiada swoje własne propozycje. Nie posiada satysfakcjonującego przycisku służącego do oznaczenia zadania jako zaliczone. Aby tego dokonać, należy przejść do szczegółów czynności, wtedy dopiero pojawia się możliwość zaznaczenia, że wykonano zadanie. Aplikacja pozwala łączyć konta użytkowników w jedno gospodarstwo domowe. Jest prosta, o dość intuicyjnej obsłudze.

Clean My House [12] to oferująca mało funkcji aplikacja mobilna z bardzo nieintuicyjną obsługą. Wyświetla wiele informacji, ale są one sformatowane niejasno. Użytkownik musi wczytać się, zamiast móc od razu zorientować się, czy dana czynność została już wykonana. Oznaczanie zadania jako zrobione również nie jest intuicyjne, należy kliknąć zadanie i wybrać środkową opcję w liście pod tytułem zadania. Dodawanie zadań także powoduje problemy. Podzielenie domu na pomieszczenia jest możliwe, nawet są zdefiniowane wstępne kategorie, jednak jest ich niezwykle dużo, co zaciemnia obraz. Co więcej, jeśli użytkownik nie ma ochoty ich używać, trzeba je usuwać pojedynczo. Podsumowując, aplikacja posiada dużo przydatnych cech, jednak jest bardzo źle sformatowana.

Cleaning Checklist [13] to aplikacja mobilna niewielkich rozmiarów, która zdecydowanie różni się od pozostałych reprezentantów gatunku. Zadania muszą być zapisywane w konkretnych kategoriach, więc dodanie nowego zadania jest stosunkowo skomplikowane. Jeśli zadanie nie należy do konkretnej kategorii, to i tak należy dla tego zadania kategorię stworzyć. Nie da się łączyć użytkowników w domostwa. Aplikacja to trochę bardziej zaawansowana lista czynności do wykonania. Posiada jednak całkiem intuicyjny system zaliczania zadań. Wstępne zainicjowanie systemu jest proponowane, jednak nie ma takiej możliwości podczas tworzenia zadań. Rozkład przycisków jest nieprzemyślany, można przez przypadek kliknąć przycisk proponujący całkowite zrestartowanie domu i wszystkich zadań. Przycisk znajduje się na głównym ekranie aplikacji, więc bardzo prawdopodobne jest naciśnięcie go przez przypadek. Podczas korzystania niezwykle często pokazywane są reklamy, co znacznie pogarsza doświadczenie korzystania z Cleaning Checklist.

Chores App [14] to nieintuicyjna aplikacja mobilna ze skomplikowaną obsługą. Aby dodać zadanie, należy przewinąć ekran do połowy aplikacji i kliknąć przycisk planowania. Wtedy użytkownik przenoszony jest do specjalnego ekranu, w którym znajduje się spis wszystkich zadań, gdzie dopiero można znaleźć przycisk dodający nowy obowiązek. Aplikacja posiada system punktowy ze skalą od 1 do 6, przy czym progi punktowe nie są w żaden sposób opisane słownie jako np. bardzo łatwy, łatwy czy trudny, więc interpretacja tej dyskretnej skali jest zostawiona użytkownikowi. Do minusów aplikacji należy również zaliczyć stronę wizualną. Aplikacja ma jednak również pewne zalety, jak choćby możliwość dodania użytkownika, który nie może samodzielnie założyć konta, czyli na przykład dziecka bądź osoby starszej. Plusem może być również ranking domowników. Niestety, nie każda rodzina chce rywalizować ze sobą w ten sposób, więc taki ranking powinien być opcjonalny.

Flatify [15] to aplikacja mobilna oferująca oryginalny interfejs użytkownika. Jest mało zorientowana wokół zadań do wykonania w porównaniu z innymi aplikacjami – pierwszą rzeczą, jaka jest wyświetlana, jest możliwość wysłania wiadomości zamiast listy zadań do

3. Przegląd dziedziny

zrobienia. Może z tego względu stworzeniu zadania brak intuicyjności. Aplikacja ma możliwość wprowadzenia danych płatniczych, tworzenia spotkań oraz wysyłania wiadomości, co może sugerować, że twórcom przyświecał nieco inny cel niż zarządzanie obowiązkami domowymi. Jest ona raczej skierowana do współlokatorów, ponieważ istnieje obowiązek przypisania zadania do konkretnej osoby oraz jest wbudowany domyślny mechanizm powodujący, że gdy jedna osoba zrobiła dane zadanie, to druga musi je wykonać następnym razem. Posiada sześciopunktową skalę trudności, jednak w przeciwieństwie do Clean My House [12], punkty na skali są opisane. Ma możliwość łączenia kont z użytecznym dodatkiem pokazującym, czy dany użytkownik w danym momencie jest zalogowany do aplikacji. Nie da się jednak dodać użytkownika niezalogowanego, np. dziecka czy starszej osoby nieposiadającej telefonu, jak w Chores App [14].

OurHome [16] to najlepsza z analizowanych aplikacji. OurHome posiada dopasowany do najmłodszych styl graficzny, jest schludnie zaprojektowana i intuicyjna. Oferuje możliwość przypisywania punktów, proponuje domyślne wartości, lecz w łatwy sposób umożliwia ich zmianę na dowolną wartość. Udostępnia funkcję określenia przewidywanego czasu wykonania zadania oraz system oznaczania różnych kategorii zadań. Minusem jest to, że można wybrać tylko jedną kategorię. Użytkownicy mogą się łączyć w gospodarstwa domowe. OurHome umożliwia odejmowanie co ustalony okres odpowiedniej liczby punktów oraz ustalenia oddzielnego celu punktowego dla różnych użytkowników. Oferuje system nagród. Niestety, jak każda aplikacja, OurHome ma również słabe strony. Czasami zdaje się nie działać poprawnie – punkty należne za zaliczone zadanie są przyznawane użytkownikowi, ale samych zadań nie widać na liście wykonanych. Aplikacja nie zmienia dynamicznie ceny zadania w reakcji na opóźnienie jego wykonania. OurHome nie ma funkcji inicjalizacji systemu. Jest to istotne, ponieważ wielu użytkowników zniechęca się, zanim rozpocznie korzystanie z aplikacji. Wynika to z długiego czasu oraz sporego wysiłku potrzebnego do zainicjowania wszystkich zadań i ustalenia ich cen względem pozostałych czynności. Istnieje również internetowa wersja tej aplikacji [17]. Autorce udało się ją znaleźć za pomocą strony [18]. Nie jest tak wygodna, jak jej mobilny odpowiednik. Działa dość wolno, próby jej użycia wskazują, że czas reakcji na każde kolejne kliknięcie użytkownika wynosi średnio około 2s, a czasem nawet około 7s. Posiada ten sam problem z działaniem co wersja mobilna. Układ strony ma bardzo podobny do odpowiednika na komórkę, ale na dużym ekranie jest to mniej intuicyjne oraz, subiektywnie patrząc, mniej estetyczne. Przykładowo mały plus w prawym górnym rogu ekranu służący do dodawania zadań dobrze sprawdza się w wersji mobilnej, ale w wersji internetowej jest zdecydowanie za mały i źle umiejscowiony. OurHome nie posiada również dynamicznego systemu zmiany punktów, który jest kluczowy w niniejszej pracy.

Chores & Allowance Bot [19] to jedna z dwóch zidentyfikowanych w ramach przeglądu aplikacji internetowych, która oferuje funkcje związane z zarządzaniem obowiązkami domowymi. Niestety bezpłatnie użytkowanie aplikacji jest możliwe tylko przez siedem dni, a dalszy dostęp jest płatny i kosztuje 29,99\$ za rok. Co więcej, podczas rejestracji od razu należy podać informacje o karcie płatniczej. Ze względu na ograniczone zaufanie autorki względem podawania takich informacji w internecie nie założyła ona konta. Zakładano poszukiwanie darmowych usług, z tego względu dalej nie rozważano tej aplikacji. Z opisu dostępnego na stronie producenta wynika jednak, że jest to aplikacja głównie skierowana do dzieci, mająca kolorowy i prosty interfejs, więc grupa jej odbiorców różni się od tej przewidywanej dla przygotowywanej w niniejszej pracy aplikacji Common, która ma służyć całym rodzinom. Aplikacja Chores & Allowance Bot bardziej przypomina grę.

3.3. Porównanie cech analizowanych aplikacji

Większość przeanalizowanych aplikacji, oprócz częściowo OurHome [16] oraz Chores & Allowance Bot [19], są aplikacjami mobilnymi. Widać, więc że jest pole, na którym aplikacja CommOn mogłaby z sukcesem pełnić swoje zadanie. Na potrzeby niniejszego przeglądu aplikacje mobilne testowane na telefonie z systemem operacyjnym Android. Cechy wszystkich analizowanych aplikacji porównano w tabeli 3.2. Jak widać, znaczna większość nie udostępnia nawet połowy pożądanych funkcji. Tody oferuje 5 z 15 wymienionych cech, Clean My House zaledwie 3 z 15, Cleaning Checklist posiada 2 z 15, Chores App wypada nieco lepiej z 6 na 15 cech, Flatify posiada połowę, OurHome aż 9 z 15. Tylko dwie aplikacje spełniają minimum połowę z analizowanych kryteriów. Najlepiej wypada OurHome, zaraz po niej Flatify, a na 3 miejscu znajduje się Chores App. Chores & Allowance Bot niestety nie da się rzetelnie porównać na podstawie ograniczonych płatnym dostępem informacji. Przeanalizowanie innych aplikacji, nawet jeśli ich poziom okazał się niewystarczający, doprowadziło do następujących konkluzji. Do dobrych pomysłów można zaliczyć odliczanie liczby punktów innej dla każdego użytkownika oraz podział na pomieszczenia. Porządkuje on zadania i pozwala je umiejscawić w domu. Przykładowym błędem okazał się mało intuicyjny sposób obsługi aplikacji. Najlepszym konkurencyjnym aplikacjom brakuje przede wszystkim automatycznego ponownego wyliczenia punktów przyznawanych za zadanie wraz z upływającym czasem. Jest to istotny mechanizm, ponieważ znacząco wspomaga sprawiedliwy podział obowiązków i wykonywanie ich na czas, co zmniejsza prawdopodobieństwo porzucenia przez rodzinę pomysłu uporządkowania podziału obowiązków [20]. Obecnie na rynku znajduje się mało aplikacji internetowych, a istniejące mają słabą jakość bądź są płatne. Na rynku aplikacji do wspomagania zarządzaniem obowiązkami domowymi jest luka, którą aplikacja taka jak CommOn mogłaby z sukcesem wypełnić.

3. Przegląd dziedziny

Tabela 3.2. Porównanie funkcji oferowanych przez konkurencyjne aplikacje

	Chores & Allowance Bot	Tody	Clean My House	Cleaning Checklist	Chores App	Flatify	OurHome
Łączanie kont	Nie wiadomo	Tak	Nie	Nie	Tak	Tak	Tak
Tworzenie zadań	Nie wiadomo	Tak	Tak	Tak	Tak	Tak	Tak
Podział domu na pomieszczenia	Nie wiadomo	Tak	Tak	Nie	Nie	Nie	Tak
Przypisanie zadania do konkretnego użytkownika	Nie wiadomo	Nie	Nie	Nie	Tak	Tak	Tak
Okresowe powtarzanie zadań	Nie wiadomo	Tak	Tak	Nie	Tak	Tak	Tak
Zdobywanie osiągnięć	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Nie
Zdobywanie nagród	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Tak
Osiaganie kolejnych poziomów	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Nie
Utworzenie rankingu domowników	Nie wiadomo	Nie	Nie	Nie	Tak	Tak	Nie
Możliwość cofnięcia wykonania zadania przez innych	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Tak
Punktowanie zadań	Nie wiadomo	Nie	Nie	Nie	Tak	Tak	Tak
Zmiana punktacji zadań w zależności od ostatniego czasu wykonania	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Nie
Okresowe pobieranie punktów rewaluacja	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Tak
Automatyczna zadań, które notorycznie nie są wykonywane na czas	Nie wiadomo	Nie	Nie	Nie	Nie	Nie	Nie
Zainicjowanie systemu przykładowymi zadaniami	Nie wiadomo	Tak	Nie	Tak	Nie	Tak	Nie
Typ aplikacji	Internetowa	Mobilna	Mobilna	Mobilna	Mobilna	Mobilna	Oba

4. Analiza wymagań

Wymagania funkcjonalne i niefunkcjonalne mają za zadanie zdefiniować funkcje oraz cechy aplikacji. W niniejszym rozdziale przedstawiono wymagania funkcjonalne (rozdział 4.1) i niefunkcjonalne (rozdział 4.2) stawiane aplikacji CommOn oraz zidentyfikowane przypadki użycia.

4.1. Wymagania funkcjonalne i przypadki użycia

Wymagania funkcjonalne określają, jakie funkcje będzie oferować system. Spis wszystkich wymagań funkcjonalnych stawianych aplikacji CommOn znajduje się w tabeli 4.1. Wymagania podzielono na 4 główne podkategorie: zadania, pokoje, użytkownicy oraz punkty. Są to podstawowe obiekty systemu, dlatego zdecydowano się na uporządkowanie wymagań w ten sposób. Można wyróżnić dwa powtarzające się typy wymagań. Pierwszym są z pewnością wymagania dotyczące manipulacji obiektami, czyli tworzenie, modyfikacja i usuwanie obiektów. Kolejnym są akcje CommOn zależne nie od użytkownika, a od czasu – wszelkie czynności wykonywane przez aplikację co określony okres. Najważniejszymi wymaganiami są te dotyczące dynamicznej zmiany ceny zadania oraz ogólniej związane z zarządzaniem punktami.

Tabela 4.1. Wymagania funkcjonalne

Id	Opis
WF 1.	Zadania
WF 1.1	Aplikacja umożliwia tworzenie, modyfikację oraz usuwanie zadań. Zadanie zawiera nazwę, cenę początkową, cenę aktualną, flagę oznaczającą czy jest wykonane, pokój, datę ostatniego wykonania zadania, okres powtarzania zadania, użytkownika, który wykonał to zadanie
WF 1.2	Aplikacja umożliwia obejrzenie wszystkich zadań znajdujących się w domu użytkownika
WF 1.3	Aplikacja umożliwia obejrzenie zadań znajdujących się w domu użytkownika z podziałem na pokoje
WF 1.4	Aplikacja umożliwia zaliczenie zadania
WF 1.5	Aplikacja umożliwia odznaczenie zaliczenia zadania w razie pomyłki
WF 1.6	Aplikacja automatycznie zmienia status zadań na do wykonania po upłynięciu okresu, w jakim mają zostać wykonane od czasu, w jakim ostatnio je wykonywano.
WF 2.	Pokoje
WF 2.1	Aplikacja umożliwia tworzenie, modyfikację oraz usuwanie pokoi
WF 2.2	Aplikacja umożliwia wyświetlenie wszystkich pokoi znajdujących się w domu użytkownika

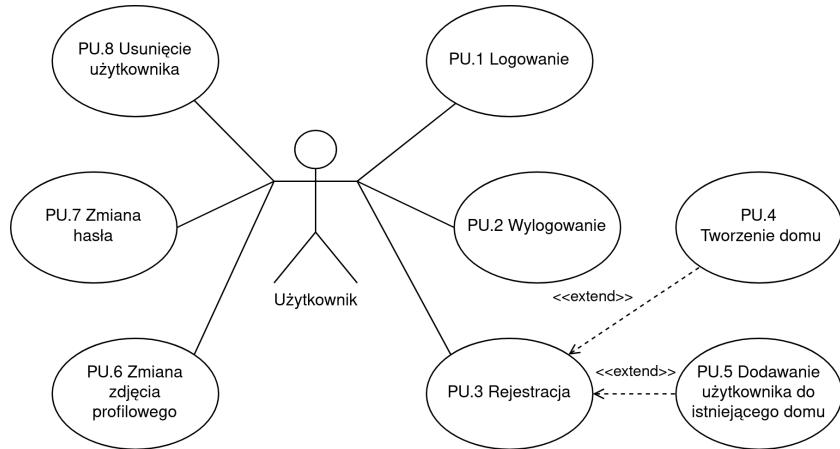
4. Analiza wymagań

Kontynuacja tabeli 4.1

Id	Opis
WF 2.3	Aplikacja umożliwia wybór zdjęcia pokoju podczas tworzenia lub edycji pokoju. Podczas wyboru zdjęcia pokoju wyświetlane są proponowane zdjęcia
WF 2.4	Aplikacja umożliwia wyświetlenie zadań znajdujących się w pokoju
WF 2.5	Aplikacja wyświetla informacje o liczbie zadań pozostałych do zrobienia w pokoju
WF 3.	Użytkownicy
WF 3.7	Aplikacja umożliwia stworzenie oraz usunięcie użytkownika
WF 3.1	Aplikacja umożliwia tworzenie domu dla użytkownika
WF 3.2	Aplikacja umożliwia dodawanie nowego użytkownika do już istniejącego domu za pomocą kodu dołączenia do domu
WF 3.3	Aplikacja umożliwia wyświetlenie kodu dołączenia do domu zalogowanego użytkownika
WF 3.4	Aplikacja umożliwia dodanie zdjęcia profilowego użytkownika. Podczas zmiany zdjęcia profilowego wyświetlane są proponowane zdjęcia
WF 3.5	Aplikacja umożliwia zalogowanie się
WF 3.6	Aplikacja umożliwia wylogowanie się
WF 3.8	Aplikacja umożliwia zmianę hasła
WF 3.9	Aplikacja umożliwia zmianę liczby punktów pobieranych od użytkownika co tydzień
WF 4.	Punkty
WF 4.1	Aplikacja umożliwia zdobywanie punktów przez użytkownika. Punkty naliczane są na poczet użytkownika, który oznaczył zadanie jako wykonane
WF 4.2	Aplikacja umożliwia wyświetlenie stosunku liczby zdobytych punktów do wszystkich wymaganych w danym tygodniu
WF 4.3	Aplikacja umożliwia wyświetlenie liczby zadań wykonanych w danym tygodniu przez użytkownika
WF 4.4	Aplikacja umożliwia wyświetlenie informacji o stanie punktów innych użytkowników z tego samego domu
WF 4.5	Aplikacja raz w tygodniu odejmuje ustalony zakres punktów każdemu użytkownikowi
WF 4.6	Aplikacja umożliwia dynamiczną zmianę ceny w zależności od pierwotnej ceny zadania, okresu, w jakim zadanie ma zostać wykonane oraz daty ostatniego wykonania

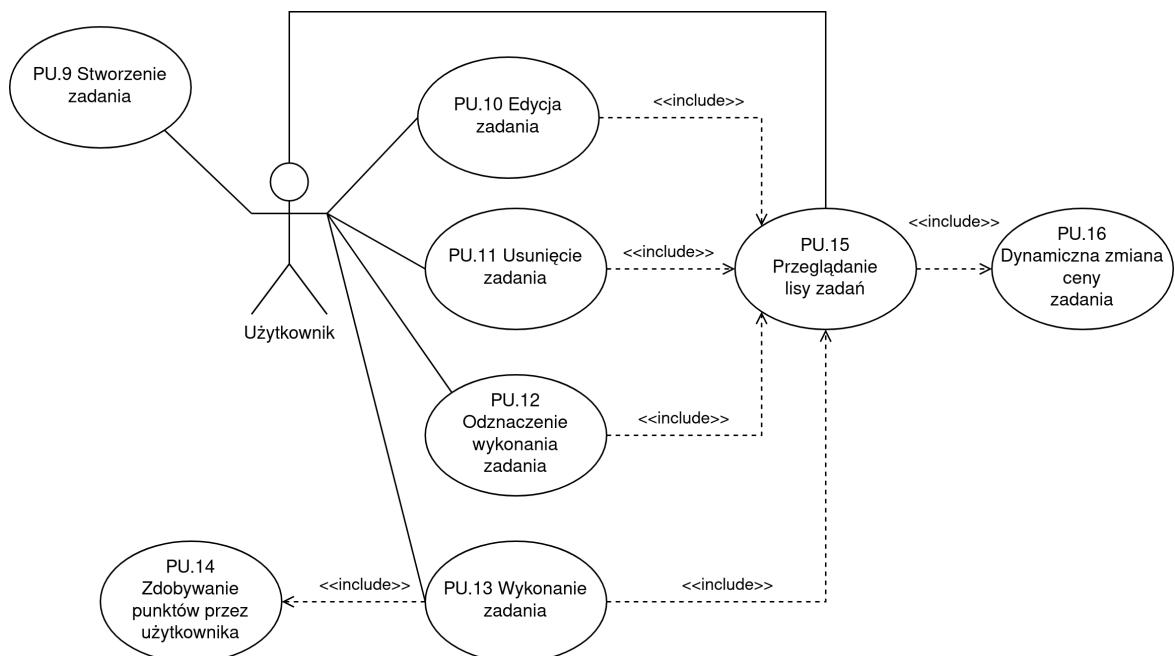
Wymagania funkcjonalne zostały zamodelowane za pomocą realizujących je przypadków użycia. Reprezentują one funkcje systemu dostępne dla jego użytkownika. Drugim aktorem pojawiającym się w modelu przypadków użycia jest czas, który odpowiada za zainicjowanie pewnych funkcji systemu w określonych momentach.

Na pierwszym diagramie (rysunek 4.1) pokazano przypadki użycia powiązane



Rysunek 4.1. Przypadki użycia związane z kontem użytkownika

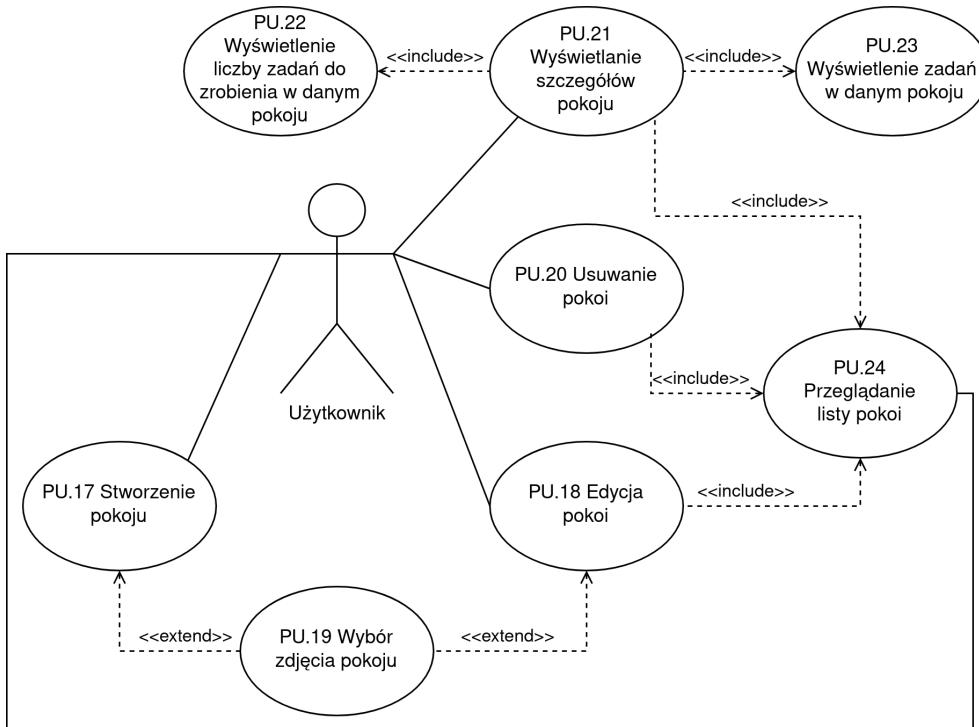
z użytkownikami. Aktorem w tym wypadku jest osoba używająca aplikacji. Przypadek *PU.1* realizuje wymaganie WF 3.5, czyli pozwala na zalogowanie się. *PU.2* symbolizuje wylogowywanie, co realizuje wymaganie WF 3.6. Wymaganie WF 3.7, czyli stworzenie oraz usunięcie konta, jest realizowane przez przypadki *PU.3* oraz *PU.8*. Tworzenie domu dla użytkownika – wymaganie WF 3.1 – jest realizowane przez przypadek *PU.4*, natomiast dodawanie użytkownika do istniejącego domu (wymaganie WF 3.2) przez przypadek *PU.5*. *PU.6*, zmiana zdjęcia profilowego, realizuje WF 3.4 – dodanie zdjęcia profilowego i wyświetlanie proponowanych zdjęć podczas zmiany. Ostatnie wymaganie funkcjonalne dotyczące kont użytkowników (WF 3.8) jest związane ze zmianą hasła, a jego realizacja jest zapewniana przez *PU.7*.



Rysunek 4.2. Przypadki użycia związane z zadaniami

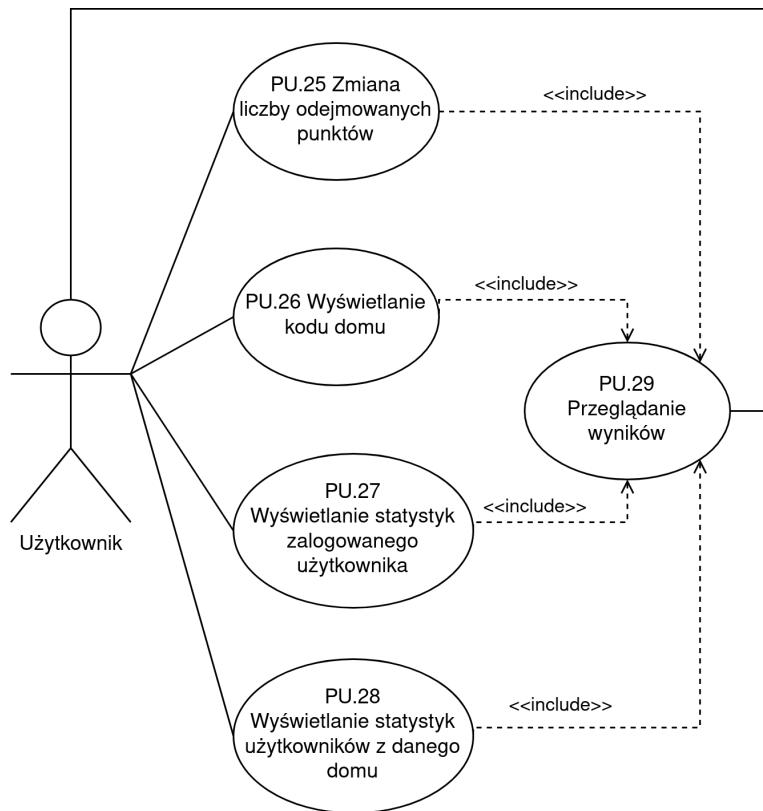
4. Analiza wymagań

Kolejny diagram (rysunek 4.2) jest poświęcony zadaniom. Przypadki *PU.9*, *PU.10*, *PU.11* dotyczą manipulacji zadaniami i dlatego realizują wymaganie WF 1.1. Ważną funkcjonalnością jest też wykonywanie zadań, obrazuje to przypadek *PU.13*, który zapewnia funkcjonalność potrzebną do WF 1.4 (można zaliczać zadania). Przypadek *PU.14* realizuje zaś wymaganie WF 4.1 – zdobywanie punktów przez użytkownika. Odwrotną sytuację, czyli odznaczenie zadania, realizuje przypadek *PU.12*, spełniając tym samym wymaganie WF 1.5. Aby móc oznaczyć zadanie jako wykonane, wymagane jest wyświetlenie listy zadań (*PU.15*). Widok ten zapewnia również osiągnięcie wymagania WF 1.2. Przypadek użycia *PU.16* realizuje wymaganie WF 4.6, czyli dynamiczną zmianę ceny zadania.



Rysunek 4.3. Przypadki użycia związane z pokojami

Diagram 4.3 poświęcony jest pokojom. Przypadki użycia *PU.17*, *PU.18*, *PU.20* realizują wymaganie dotyczące tworzenia, usuwania oraz modyfikacji pokoju (WF 2.1). Wymaganie dotyczące zdjęć pokoju (WF 2.3) jest realizowane za pomocą przypadku *PU.19*. Wyświetlanie szczegółów pokoju jest istotną funkcjonalnością zaprezentowaną w przypadku *PU.21* i realizującą wymaganie WF 1.3. Wewnątrz szczegółów pokoju pokazana jest liczba zadań *do wykonania* w tym pokoju (*PU.22*), która realizuje wymaganie WF 2.5. Wyświetlanie zadań z podziałem na pokoje jest realizowane przez *PU.23*, a spełnia warunek WF 2.4 traktujący o możliwości wyświetlania zadań znajdujących się w konkretnym pokoju. Przeglądanie listy pokoi (*PU.24*) reprezentuje realizację WF 2.2.



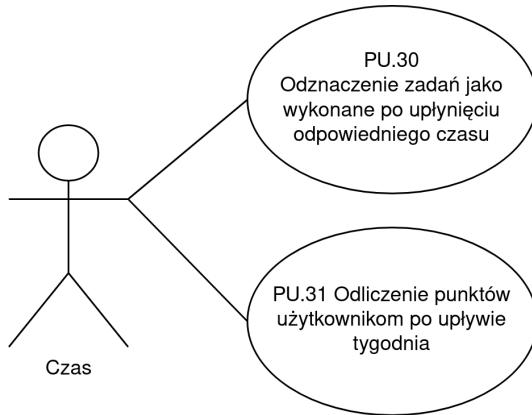
Rysunek 4.4. Przypadki użycia związane z punktami

Przypadki związane z punktami są pokazane na diagramie 4.4. Umożliwienie zmiany liczby odejmowanych punktów *PU.25* realizuje WF 3.9. Przypadek *PU.26*, wyświetlanie kodu domu, zapewnia funkcjonalność niezbędną do realizacji WF 3.3. Podczas oglądania statystyk zalogowanego użytkownika (*PU.27*) spełniane są wymagania WF 4.2 (pokazanie stosunku liczby zdobytych punktów do wszystkich wymaganych w danym tygodniu) oraz WF 4.3 (wyświetlanie liczby zadań zrobionych w danym tygodniu przez zalogowanego użytkownika). Warunek WF 4.4 jest realizowany za pomocą *PU.28* – wyświetlanie statystyk użytkowników danego domu. Przeglądanie wyników (*PU.29*) jest potrzebne podczas realizacji pozostałych przypadków użycia znajdujących się na rysunku 4.4.

Ostatni diagram (rysunek 4.5) jest dość nietypowy, ponieważ aktorem jest czas. Pokazuje on dwa przypadki użycia *PU.30* oraz *PU.31*. Pierwszy z nich odnosi się do zmiany statusu zadania na *do zrobienia* po czasie równym jego okresowi wykonania i realizuje wymaganie WF 1.6. Drugi zaś dotyczy odliczania punktów od puli zebranej przez każdego użytkownika po upływie tygodnia. Z jego pomocą spełnione są warunki wymagane w WF 4.5.

4.2. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają cechy jakościowe aplikacji, takie jak jej niezawodność czy szybkość. Muszą więc precyzyjnie zdefiniować, co te abstrakcyjne



Rysunek 4.5. Przypadki użycia związane z czasem

pojęcia oznaczają w ich kontekście. W przypadku CommOn najważniejszymi wymaganiami niefunkcjonalnymi są intuicyjność i rozszerzalność aplikacji. Intuicyjność jest bardzo istotna ze względu na podstawową grupę odbiorców CommOn, jaką są rodziny. Szczególnie starsze osoby (choć oczywiście nie tylko) potrzebują dobrze zaprojektowanego interfejsu, żeby czerpać radość i satysfakcję z używania systemu. Rozszerzalność jest istotna, ponieważ istnieje jeszcze wiele przydatnych funkcjonalności, które wzbogaciliby tę aplikację, dlatego jakość kodu i dobry projekt to podstawa dalszego rozwoju dla CommOn. Pełna lista wymagań niefunkcjonalnych stawianych aplikacji CommOn znajduje się w tabeli 4.2.

Tabela 4.2. Wymagania niefunkcjonalne

Id	Opis
WNF 1	Wygląd aplikacji jest spójny i atrakcyjny
WNF 2	Aplikacja musi zapewniać intuicyjny interfejs
WNF 3	Opóźnienia ładowania aplikacji nie mogą średnio przekroczyć czasu 2s
WNF 4	Aplikacja przechowuje dane w bezpieczny sposób
WNF 5	Aplikacja udostępnia dane tylko uprawnionym użytkownikom
WNF 6	Aplikacja powinna działać w przeglądarkach Google Chrome [21], Mozilla Firefox [22] oraz Opera [23]
WNF 7	Aplikacja powinna być przygotowana do wprowadzania rozszerzeń funkcjonalności

5. Zastosowane technologie

Rodzaj narzędzi, jakich używa się podczas pisania programu, to niezwykle istotna kwestia. Muszą one być odpowiednio dopasowane do zadania, uwzględniać rozwój oprogramowania w przyszłości oraz proponować wystarczającą jakość dokumentacji, aby używanie ich nie było problemem. Istotną kwestią jest również zaznajomienie programisty z narzędziem, którego będzie używać.

5.1. Kryteria wyboru narzędzi

Wybór technologii zastosowanych w CommOn był wykonywany na podstawie pięciu kryteriów:

1. Czy dana technologia jest popularna?
2. Czy posiada dobrą dokumentację, zarówno oficjalną, jak i zapewnioną ze strony społeczności programistów jej używającej?
3. Czy dana technologia jest stosowana przez firmy programistyczne?
4. Czy dana technologia jest darmowa?
5. Czy jest to technologia, w której autorka chce rozwijać swoje umiejętności?

Fakt, że technologia jest popularna, często wskazuje na intuicyjność i łatwość tworzenia w niej kodu. Popularność podnosi również prawdopodobieństwo występowania dobrej dokumentacji, co znacząco ułatwia naukę nowych technologii. Jeśli spora liczba programistów używa określonego narzędzia, to często firmy na rynku pracy również z niego korzystają. Duża liczba firm używających danej technologii przekłada się na łatwiejsze znalezienie zatrudnienia po zakończeniu studiów. Darmowość rozwiązania, oprócz oczywistych korzyści, często przekłada się na jej popularność.

CommOn jest aplikacją internetową o architekturze trójwarstwowej. Uzasadnienie wyboru tego rodzaju aplikacji oraz dokładny opis jej budowy znajdują się w rozdziale 6. Uwzględniając powyższe kryteria, wytypowano następujące narzędzia programistyczne. Warstwa prezentacji (rozdział 5.2) została napisana z użyciem języka TypeScript [2] oraz szkieletu (*ang. framework*) Angular [4]. Za zdefiniowanie odpowiednich stylów odpowiedzialny jest Bootstrap [24]. Karma [25] oraz Jasmine [26] to narzędzia użyte do tworzenia testów tej warstwy aplikacji. Warstwa logiki biznesowej (rozdział 5.3) tworzonej aplikacji została napisana w Javie [1], z wykorzystaniem szkieletu aplikacji Spring Boot [27], a do komunikacji z bazą danych posłużyło narzędzie Hibernate [28]. Ponadto, powtarzalne metody, takie jak get, set, czy equals, zostały wygenerowane za pomocą narzędzia Lombok [29], a do zarządzania zależnościami projektu posłużył Maven [30]. Testy jednostkowe tej części aplikacji są zrealizowane przy pomocy biblioteki JUnit [31], zaś pokrycie kodu testami jest sprawdzane za pomocą Jacoco [32]. Ostatnią jest warstwa danych służąca przechowywaniu informacji. MySQL [5] posłużył jako rodzaj bazy danych.

5. Zastosowane technologie

Pozostałe narzędzia były użyte głównie w celu poprawienia jakości kodu i pracy oraz umożliwienia wdrożenia. Aby ułatwić wprowadzanie do kodu zmian oraz użyć możliwości związanych z wersjonowaniem, skorzystano z repozytorium kodu GitHub [33]. W ramach usług oferowanych przez stronę GitHub, w postaci GitHub Actions [34], skonstruowano przetwarzanie potokowe wspomagające szybką weryfikację poprawności każdego fragmentu kodu dodawanego do głównej gałęzi repozytorium. Wewnątrz przetwarzania potokowego użyto narzędzia SonarCloud [35] do statycznej analizy kodu. Aby umożliwić wdrożenie, zastosowano Dockera [36]. Technologie zastosowane w poszczególnych warstwach aplikacji CommOn zostały dokładniej opisane w kolejnych rozdziałach.

5.2. Warstwa prezentacji

Warstwa prezentacji pozwala użytkownikowi komunikować się z systemem. Istotnymi cechami tej części aplikacji, są: schludny wygląd, intuicyjna obsługa i szybkość ładowania. Użytkownik za pomocą narzędzi deweloperskich może uzyskać dostęp do kodu, dlatego nie należy w nim przechowywać żadnych poufnych informacji. Językami, które brano pod uwagę przy implementacji warstwy prezentacji, były TypeScript [2] i JavaScript [37]. Ze względu na to, że TypeScript to po prostu język typowany statycznie, komplilujący się do JavaScriptu, zdecydowano się na TypeScript. Statyczne typowanie jest istotne ze względu na późniejsze korzyści, takie jak łatwiejsze utrzymanie kodu oraz potencjalnie mniejsza liczba błędów podczas wykonania programu [38]. TypeScript jest oparty na JavaScriptcie, który jest językiem jednowątkowym. Udostępnia on jednak możliwość wykorzystania przeglądarki do wykonywania zadań asynchronicznych. Przejmuje ona obowiązek oczekiwania na wykonanie zapytania, w tym czasie aplikacja może zająć się przykładowo renderowaniem lub obsługą użytkownika. Asynchroniczne zapytania pozwalają znacznie zwiększyć komfort korzystania z aplikacji, ponieważ umożliwiają reakcję na akcje użytkownika podczas oczekiwania na odpowiedź. Bez tej właściwości użytkownik co chwilę musiałby czekać, oglądając nieresponsywną stronę.

5.2.1. Narzędzia

Najbardziej popularnymi szkieletami wspomagającymi pisanie warstwy graficznej na świecie są w kolejności: React [39], Angular [4] i Vue [40]. Liczba użytkowników GitHuba [33] posiadająca projekt używający danej technologii wynosi 5,7 miliona dla Reacta, 1,7 miliona dla Angulara oraz 167 tysięcy dla Vue [41]. Pomimo że w technologii React programuje więcej osób, Angular posiada dużo większy zbiór dokumentacji oraz jego dogłębne zrozumienie jest łatwiejszym zadaniem [42]. React jest również często stosowany do programowania w paradygmacie funkcyjnym, a nie obiektowym jak Angular. Ze względu na biegłość autorki w Angularze i prostotę pojęcia obiektowego

odrzucono Reacta. Vue pomimo swej prostoty posiada mniejszą społeczność oraz wciąż niewiele dokumentacji. Dodatkowo spora część jego komponentów jest napisana po chińsku, co często utrudnia korzystanie z nich [42]. Ze względu na powyższe argumenty zdecydowano, że w projekcie zostanie użyty Angular.

Ze względu na konieczność eleganckiego wyglądu zewnętrznego tej części aplikacji należy jej nadać odpowiednie style. Samodzielnego tworzenie stylów jest skomplikowane, długotrwałe oraz wymaga odpowiedniego wyczucia, co często stanowi barierę nie do przeskoczenia. Istnieją jednak narzędzia pomagające programistom nadać szykowny wygląd części aplikacji służącej komunikacji z użytkownikiem. Często pojawiającymi się w tym miejscu narzędziami są Bootstrap [24] oraz Angular Material [43]. Ze względu na prostotę użycia oraz subiektywne poczucie estetyczne zdecydowano się na użycie Bootstrapa.

5.2.2. Testowanie

Jasmine [26] to narzędzie służące do testowania kodu napisanego w JavaScriptie. Wspiera ono praktykę rozwoju oprogramowania opartego na zachowaniu (*ang. behavior-driven development*, w skrócie BDD [44]). Jest to praktyka wywodząca się z TDD (*ang. test-driven development* [45]). Więcej na temat TDD można znaleźć w rozdziale 7. BDD skupiająca na tym, żeby testy były napisane w jak najbardziej przyjazny dla człowieka sposób [46]. Osiąga ten cel między innymi poprzez używanie języka zbliżonego do naturalnego (metody describe czy it). Aby uruchomić testy napisane z użyciem Jasmine, należy albo manualnie stworzyć plik HTML [47] oraz załączyć do niego odpowiednie pliki JavaScript oraz CSS [48] i odtworzyć w przeglądarce, albo użyć narzędzia Karma [25]. Zdecydowano się na to drugie rozwiązanie ze względu na to, że praca w terminalu jest dużo szybsza, wygodniejsza oraz możliwa do zautomatyzowania. Oprócz uruchamiania testów w wierszu poleceń, narzędzie to pozwala na sprawdzenie testów w kontekstach różnych przeglądarek internetowych oraz ponownie wykonuje testy w momencie, w którym wykryje zmianę w pliku testowym.

5.3. Warstwa logiki biznesowej

Warstwa logiki biznesowej służy między innymi do komunikacji warstwy prezentacji z bazą danych. Warstwa prezentacji nie powinna mieć dostępu do bazy ze względu na to, że jej kod jest dostępny dla użytkownika i możliwy do modyfikacji za pomocą narzędzi deweloperskich [49]. Powoduje to, iż każde zapytanie wysypane przez użytkownika należy traktować jako potencjalnie zmodyfikowane w szkodliwym celu. Dlatego tak istotna jest walidacja oraz zabezpieczenie dostępu, które zapewnia warstwa logiki biznesowej. Dzięki tej części aplikacji możliwa jest również integracja z innymi systemami tak, aby użytkownik tego nie widział. Część serwerowa umożliwia również wykonanie logiki

5. Zastosowane technologie

aplikacji niezależnej od użytkownika, takiej jak na przykład tworzenie procesów co określony czas.

Najbardziej popularnymi językami służącymi do pisania warstwy logiki biznesowej są Java [1], Python [50] oraz C# [51]. Ze względu na doświadczenie autorki w programowaniu w systemie operacyjnym Ubuntu [52] podjęto decyzję o zrezygnowaniu z Windowsa [53], z którym mocno powiązany jest język C#. Jako że Python jest językiem znanim z dynamicznego typowania, podjęto decyzję o używaniu Javy do rozwoju aplikacji. Statyczne typowanie jest wygodne ze względu na mniejszą liczbę błędów w trakcie działania programu — są one wykrywane w trakcie komplikacji i statycznej analizy kodu. Dużo prościej jest również pracować nad kodem na dłuższą metę, jeśli od razu widać, jakiego typu jest dana zmienna.

5.3.1. Narzędzia

Szkielet aplikacji lub inaczej platforma (*ang. framework*) to abstrakcyjna struktura, w której programista może pisać kod specyficzny dla danej aplikacji. Nie jest on konieczny w rozwoju oprogramowania, jednak zaleca się go stosować ze względu na wartość, którą wnosi. Przykładowo [54]:

1. Pozwala na standaryzację budowy aplikacji, co znaczco ułatwia zrozumienie w przypadku dołączania do już istniejącego projektu. W aplikacji zastosowano jasny podział na typy klas zajmujące się konkretnymi zadaniami, takie jak repozytorium – do komunikacji z bazą danych lub kontroler – wystawiający punkty komunikacji.
2. Pozwala skupić się na dodawanej funkcjonalności zamiast na wykonywaniu pobocznych, koniecznych zadań. Autorka korzystała ze wstrzykiwania zależności, co pozwoliło skupić się na rozwoju oprogramowania zamiast na częstym dodawaniu klas do konstruktorów w razie chęci użycia napisanej funkcjonalności.
3. Automatycznie stosuje wzorce projektowe, co naturalnie przekłada się na wyższą jakość kodu. CommOn jest oparte na wzorcu projektowym Model-Widok-Kontroler [55] (*ang. Model-View-Controller, MVC*), co znacznie upraszcza zarządzanie poszczególnymi warstwami aplikacji.
4. Obniża koszty rozwoju oprogramowania

Najczęściej używanym szkieletem służącym do pisania warstwy serwerowej używającym Javy jest Spring [56] (oraz Spring Boot [27], czyli posiadająca wstępnią konfigurację wersja Springa). Spring [3] pomaga programiście na wiele sposobów, przede wszystkim wyręczając go z obowiązku pisania części funkcjonalności od nowa. Standardowe fragmenty są po prostu automatycznie generowane. Ułatwia to początkującym pracę nad aplikacją, ponieważ znaczco obniża próg wejścia. Spring jest kontrolerem odwróconego sterowania, potrafi zarządzać cyklami życia obiektów w nim zdefiniowanych. Tworzy i wstrzykuje obiekty, których potrzebuje dana klasa oznaczona

jako komponent, by używać ponownie obiektów, które są wolno tworzone, z których można skorzystać wielokrotnie, oraz gdy obiekt już nie jest potrzebny, zaznaczyć go do usunięcia przez odśmieczacz (*ang. garbage collector*), który jest podstawowym narzędziem zarządzającym pamięcią w Javie [1]. Spring wprowadza opakowanie na typową komunikację z bazą za pomocą JDBC [57] (Java Database Connectivity). Znaczaco zmniejsza to ilość kodu potrzebnego do obsłużenia połączenia oraz ułatwia obsługę błędów. Ta warstwa abstrakcji umożliwia również integrację z Hibernate [28]. Spring zapewnia także obsługę transakcji. Aby stworzyć transakcję, wystarczy użyć adnotacji `@Transactional`. Spring definiuje również szereg adnotacji ułatwiających zastosowanie wzorca projektowego Model-Widok-Kontroler. Jest to jedna z jego najczęściej używanych funkcji [56], co tylko dowodzi, jak bardzo jest ona przydatna. Podsumowując, Spring to przydatne narzędzie wspierające programowanie, między innymi, aplikacji internetowych. Znaczaco ułatwia pracę oraz zmniejsza ilość kodu niezbędnego do napisania działającej aplikacji. Jego funkcje zdecydowanie przydały się autorce podczas pisania aplikacji. Zarządzanie wstrzykiwaniem zależności zdecydowanie uprościło pisanie kodu. Oparcie struktury aplikacji na modelu MVC wspieranym przez Spring znacznie uporządkowało strukturę i umożliwiło prostsze zmiany. Użyto również Spring Security [58], czyli szkieletu, który wprowadzał niemalże gotową funkcjonalność logowania oraz uwierzytelniania.

Dwa główne narzędzia służące zarządzaniu zależnościami w Javie to Maven [30] oraz Gradle [59]. Oba narzędzia są darmowe, jednak Maven jest dużo częściej wybierany przez programistów, prawdopodobnie również dlatego, że jest starszy. Aż 67% programistów w Javie przyznaje, że Maven jest ich głównym narzędziem do zarządzania zależnościami, zaledwie 20% mówi to o Gradle'u [60]. O wyborze Mavena zadecydowała nie tylko popularność tej technologii, ale również subiektywne preferencje autorki, która miała zdecydowanie większe doświadczenie w pracy z tym narzędziem.

Maven, poza niezwykle przydatną funkcją zarządzania bibliotekami, zdecydowanie ułatwia również proces tworzenia kodu wykonywalnego. Pozwala na rozbudowę domyślnego sposobu budowania projektu o dodatkowe czynności, takie jak liczenie pokrycia kodu za pomocą Jacoco [32] czy usunięcie folderu, w którym znajdują się artefakty [61]. Najczęściej używanymi przez autorkę komendami Mavena były clean, która czyści repozytorium z wcześniej skompilowanego kodu, oraz install, który kompliuje, testuje oraz pakuje projekt.

Hibernate [28] to narzędzie służące do mapowania obiektowo-relacyjnego (*ang. Object–Relational Mapping, ORM*). Ze względu na różnice w reprezentacji obiektów w realiach relacyjnych (takich jak w relacyjnej bazie danych) oraz językach obiektowych (jak na przykład Java) istnieje konieczność przekształcenia jednej reprezentacji na drugą.

5. Zastosowane technologie

Narzędzia ORM, w tym Hiberante, rozwiązuje problemy pojawiające się przy zmianie sposobu reprezentacji obiektów [62].

Java oferuje wiele przydatnych mechanizmów, ale niektóre fragmenty kodu, takie jak konstruktor przyjmujący jako argumenty wartości wszystkich pól klasy i przypisujący je do atrybutów tej klasy, są powtarzalne i nie wnoszą dodatkowej wartości do programu. Właśnie wtedy przydaje się biblioteka Javy – Lombok [29]. Za pomocą adnotacji zdefiniowanych w tej bibliotece programista jest w stanie wygenerować powtarzalne, lecz niezbędne, metody. Jest to niezwykle przydatna biblioteka ułatwiająca pracę. W CommOn autorka używa jej do generowania metod typu set, get, equals, hashCode oraz konstruktorów bezargumentowych.

5.3.2. Testowanie

JUnit [31] to narzędzie służące do pisania automatycznych testów jednostkowych. Testy jednostkowe służą do sprawdzania poprawności niewielkich fragmentów kodu, na przykład pojedynczych metod klasy. Zaletami testów automatycznych są szybkie działanie oraz często możliwość wychwycenia błędu w krótkim czasie po napisaniu odpowiedzialnego zań fragmentu kodu. Znacznie zawęża to pole poszukiwań wadliwej linijki, co powoduje sprawniejsze rozwiązywanie problemu. Testy ręczne nigdy nie będą tak dokładne ze względu na czynnik ludzki. Ręczne testowanie traci również na powtarzalności wykonywanych akcji, łatwo przeoczyć jakiś fragment. Zastosowanie automatyzacji testów, poprzedzających przykładowo złączenie nowego kodu z ostatnią wersją stabilną, zmusza programistę do stosowania dobrych nawyków. Przydatnym narzędziem, będącym częścią Springa [3], jest również MockMvc [63]. Stosuje się je w celu łatwiejszego testowania warstwy sieciowej. Pozwala na testowanie, czy zapytania kierowane do kontrolerów są odpowiednio obsługiwane. Aby testy kontrolerów były powtarzalne i nie polegały na zapytaniach do bazy danych, które mogą się nie udać, użyto Mockito [64]. Pozwala on na ustalenie odpowiedzi, jaką ma zwrócić metoda używana np. wewnątrz kontrolera do uzyskania obiektów z bazy. Takie rozwiązanie powoduje sprawdzanie tylko fragmentów kodu, które faktycznie nas interesują. Testy jednostkowe, zarówno podstawowej logiki biznesowej, jak i kontrolerów, zostały zrealizowane w ramach aplikacji CommOn i szczegółowo opisane w rozdziale 7.

Jacoco [32] to narzędzie, za pomocą którego można określić pokrycie kodu testami. Sprawdza się je jako jedną z metryk określających jakość pisanych testów. Jak wszystkie metryki tego typu, pokrycie kodu obrazuje tylko jedną ze stron i nie należy jej traktować jako ostatecznego wyznacznika, czy testy są dobrej jakości, czy też nie. Może istnieć kod posiadający bardzo duże pokrycie (np. 90%), jednak nie sprawdzający, czy zawartość obiektów po przejściu testu jest poprawna. Jacoco zostało użyte w ramach automatyzacji

CI/CD wewnątrz narzędzia Sonar Cloud [35]. Więcej informacji na ten temat można znaleźć w rozdziale 5.4.

5.4. Pozostałe narzędzia

Podczas tworzenia aplikacji, oprócz narzędzi wykorzystanych do realizacji warstwy prezentacji i logiki, wykorzystano także szereg innych, które są opisane poniżej. Jednym z podstawowych zadań aplikacji jest przechowywanie danych użytkownika. Jako bazę danych aplikacja posiada kontener dockerowy, na którym uruchomiony jest obraz z bazą danych MySQL [5], dodatkowe informacje dotyczące Dockera [36] znajdują się w dalszej części tego rozdziału. MySQL to najbardziej popularna baza typu otwartoźródłowego na świecie [65]. Mimo sporych możliwości jest również łatwa w użyciu oraz posiada rozległą dokumentację [65]. Rozważono również inne popularne, relacyjne bazy danych, takie jak PostgreSQL [66] czy SQLite [67]. MySQL jest jednak znacznie szybszy od PostgreSQL [68], posiada również bardzo dużą bazę wiedzy oraz jest prostszy do zastosowania. Co prawda PostgreSQL posiada zdecydowanie lepszy system skalowania [69], jednak nie jest to priorytet w tym projekcie. Mimo że SQLite zużywa zdecydowanie mniej pamięci, niestety obsługuje tylko 5 typów danych. MySQL jest lepszy do zastosowania w aplikacji internetowej [70] ze względu na większe możliwości w zakresie bezpieczeństwa i możliwość uwierzytelnienia.

Ciągła integracja oraz ciągłe wdrożenie (*ang. Continuous Integration & Continuous Delivery*, w skrócie CI/CD) to kroki potrzebne do wdrożenia nowego fragmentu kodu. CI/CD, czyli przetwarzanie potokowe, to praktyka polegająca na ulepszeniu procesu dostarczania kodu poprzez automatyzację. Dzięki zautomatyzowaniu tego powtarzanego i często żmudnego procesu rozwijanie oprogramowania jest szybsze, a sam kod lepszej jakości [71]. Narzędziem użyтыm w niniejszej pracy jest GitHub Actions [34], jest to częsty wybór w wypadku użycia repozytorium kodu GitHub [33] ze względu na łatwą integrację. Automatyzacja wdrożenia kodu składa się z pięciu etapów: budowa projektu, wykonanie automatycznych testów, analiza statyczna jakości kodu za pomocą SonarCloud [35], zbudowanie obrazu dockerowego oraz wysłanie obrazu na GitHub Container Registry [72] (GHCR).

Podczas tworzenia CommOn użyto dwóch narzędzi firmy Sonar Source [73]: SonarCloud [35] oraz SonarLint [74]. SonarLint to rozszerzenie środowiskowe, którego używała autorka podczas pracy nad kodem. Pozwala na statyczną analizę kodu wykrywającą między innymi nadmiarowe importy czy zakomentowany kod. SonarCloud to platforma pozwalająca na analizę kodu podczas działania automatyzacji CI/CD. Ma przydatną funkcję pozwalającą określić warunki, które kod powinien spełniać, aby dozwolone było jego zaakceptowanie podczas próby połączenia nowego kodu z już istniejącym. Wewnątrz CI/CD autorka zdefiniowała przykładowo minimalne pokrycie

5. Zastosowane technologie

kodu za pomocą Jacoco [32] oraz brak zapachów (*ang. code smells*). Jeśli SonarCloud nie zaakceptuje kodu, nie da się go połączyć z główną gałęzią zdalnego repozytorium kodu.

Repozytorium kodu to podstawowe narzędzie programisty, które służy zarządzaniu zmianami w kodzie. Umożliwia ono również współpracę wielu osób, jednak nawet jeśli programista pisze sam, jest to niezwykle użyteczne narzędzie. Ze względu na popularność i darmowość wybór padł na GitHub [33]. Poza podstawowymi właściwościami, takimi jak poruszanie się po kodzie w różnych wersjach i ewentualnej możliwości cofnięcia się do wcześniejszych wersji, gdyby obecna okazała się nie działać, repozytorium kodu umożliwia też przechowywanie kodu w internecie na wypadek zniszczenia się lokalnej maszyny. Odpowiednio skonfigurowane repozytorium GitHuba jest proste, co niewątpliwie również można zaliczyć do jego zalet. Posiada wbudowany mechanizm znacznie ułatwiający rozwiązywanie konfliktów w stosunku do klasycznego Gita [75]. Ma możliwość zakazania połączenia swojej zmiany w kodzie pod warunkiem spełnienia określonych wymagań. W CommOn skorzystano z tej możliwości, jeśli próba połączenia nowego kodu z już istniejącym nie spełni warunków przetwarzania potokowego, to nie może zostać połączony, ponieważ GitHub na to nie pozwoli.

Narzędziem użytym w aplikacji jest również Docker [36]. Służy on do tworzenia, wdrażania i uruchomienia aplikacji. Pozwala umieścić program i jego zależności w wirtualnym kontenerze, który można wykonać na innej maszynie. Do tworzenia systemu złożonego z kilku kontenerów służy Docker Compose [76]. Kontenery dockerowe stanowią lżejsze rozwiązanie niż zastosowanie maszyn wirtualnych ze względu na współdzielenie jądra systemu oraz niektórych bibliotek wykorzystywanych podczas startu systemu operacyjnego. Dzięki zastosowaniu kontenerów programista nie musi nic instalować na maszynie, na której ma działać jego aplikacja, wystarczy umieścić odpowiednie instrukcje w pliku konfigurującym obraz aplikacji. Wszystkie potrzebne zasoby zostaną ściągnięte na jego podstawie. Obraz dockerowy stanowi podstawę tworzenia kontenerów dockerowych. Relację między kontenerem i obrazem można porównać do instancji oraz jej klasy, gdzie rolę klasy pełni obraz, a kontener to jego konkretna, działająca instancja. Repozytorium obrazów to miejsce, w którym można umieścić obraz dockerowy, aby móc go w razie potrzeby ściągnąć. Służy podobnym celom co repozytorium kodu — pozwala na zabezpieczenie obrazów przed ewentualną awarią lokalnego sprzętu. W niniejszej pracy użyty jest GitHub Container Registry (GHCR) [72], czyli specjalne repozytorium obrazów dockerowych związane ze środowiskiem GitHuba [33]. Docker automatycznie umieszcza obrazy w swoim repozytorium, Docker Hub [77], dlatego, aby przesłać obraz do GHCR, należy je odpowiednio oznaczyć. Przetwarzanie potokowe wykorzystane w niniejszym projekcie obejmuje tworzenie obrazu dockerowego oraz przesyłanie go do GitHub Container Registry.

6. Projekt i implementacja

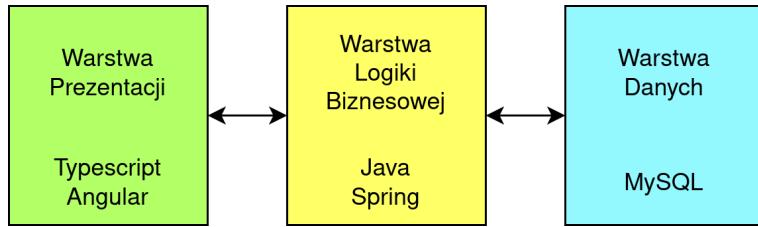
Niniejszy rozdział opisuje podjęte decyzje projektowe, architekturę oraz implementację CommOn. Decyzje dotyczące struktury aplikacji podjęto na podstawie wiedzy, jaką autorka pozyskała ze studiów i pracy zawodowej, książek (np. [55], [78], [79]) oraz stron internetowych zawierających dokumentację użytych narzędzi (np. [80], [81]).

Wpierw należy ustalić, dlaczego podjęto specyficzne decyzje projektowe. Ze względu na grupę docelową użytkowników oraz otwartą niszę na rynku zdecydowano się na stworzenie aplikacji internetowej, a nie mobilnej. Grupą docelową użytkowników CommOn są rodziny z dziećmi. Autorka aplikacji nie chce zachęcać rodziców do kupowania pociechom telefonów w młodym wieku, z tego względu uznano, że bardziej odpowiednim narzędziem, nad którym rodzice mają większą władzę, jest komputer. Kolejnym walorem tego środka przekazu jest rozmiar ekranu, który okazuje się mieć kluczowe znaczenie w przypadku, w którym z aplikacji korzystają również dziadkowie, którzy często mają problemy ze wzrokiem i nie widzą zawartości wyświetlanej na małych ekranach telefonów. Przegląd konkurencyjnych aplikacji, którego wyniki przedstawiono w rozdziale 3, tylko upewnił autorkę w przekonaniu, że na rynku jest otwarta na aplikacje internetowe nisza. Znalezione aplikacje internetowe były płatne lub niskiej jakości.

6.1. Architektura

CommOn jest aplikacją internetową o architekturze trójwarstwowej i składa się z warstw prezentacji, logiki biznesowej oraz danych. Warstwa prezentacji to graficzny interfejs komunikujący się z użytkownikiem końcowym, którego głównymi zadaniami są wyświetlanie danych oraz zbieranie informacji od użytkownika [82]. Warstwa logiki biznesowej ma za zadanie integrować pozostałe warstwy oraz przetwarzanie pozyskane od użytkownika dane [82]. Warstwa danych przechowuje informacje [82]. Podstawową zaletą architektury trójwarstwowej jest separacja funkcji wykonywanych przez warstwy, a w konsekwencji możliwość względnie łatwej wymiany jednego z komponentów bez wpływu na pozostałe [82]. Warstwa prezentacji komunikuje się z warstwą logiki biznesowej, ta natomiast z warstwą danych. Jest to zaprezentowane na rysunku 6.1, który zawiera także informacje o językach i narzędziach użytych do realizacji poszczególnych warstw aplikacji CommOn.

Aby spełnić wymagania niefunkcjonalne dotyczące jakości kodu (WNF 7) autorka pisała kod aplikacji z użyciem wzorców projektowych. W dużej mierze przyczyniło się to do stworzenia jakościowego i rozszerzalnego kodu. Pierwszym z nich jest Model-Widok-Kontroler (*ang. Model-View-Controller, MVC*) [55]. Polega on na separacji logiki manipulującej danymi od sposobu ich wyświetlania. Pozwala to na zwiększenie elastyczności kodu, ponieważ w razie potrzeby da się wymienić jeden z elementów



Rysunek 6.1. Realizacja architektury trójwarstwowej w CommOn

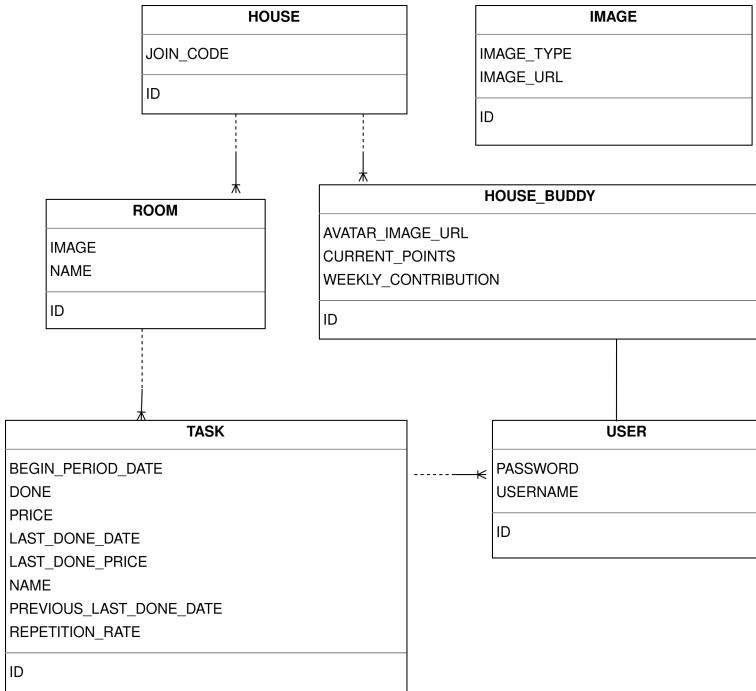
(model, widok lub kontroler) i operacja ta będzie wymagała minimum zmian w pozostałych częściach aplikacji. Drugim wzorcem użyтыm w CommOn jest wzorzec budowniczy (*ang. builder*). Polega on na stworzeniu klasy, która służy tylko do poprawnego konstruowania obiektów innej klasy. Zapewnia to o wiele prostszą i mniej podatną na błędy konstrukcję złożonych obiektów. Więcej na temat klas, które posiadają klasy budujące, można znaleźć w rozdziale 6.1.2.

6.1.1. Diagram ER bazy danych

Baza danych CommOn jest podzielona na 6 encji: TASK – przechowującą informacje związane z zadaniami, ROOM – przechowującą informacje związane z pokojami, HOUSE – przechowującą informacje związane z domami, USER – przechowującą informacje związane z użytkownikami, HOUSE_BUDDY – przechowującą informacje związane z użytkownikami posiadającymi informacje biznesowe, takie jak np. liczba zebranych punktów oraz IMAGE – przechowującą informacje związane z proponowanymi zdjęciami. Encja IMAGE nie posiada relacji z ROOM ani USER, pomimo posiadania przez te encje pola reprezentującego zdjęcie, ze względu na fakt, że w encji IMAGE są przechowywane obrazy proponowane wszystkim użytkownikom. Rozmiar obrazów – reprezentowanych w CommOn za pomocą adresów URL – nie jest duży, dlatego nie ma problemu z przechowywaniem ich jako atrybutu encji. Diagram ER bazy danych zaprojektowanej na potrzeby CommOn przedstawiono na rysunku 6.2

6.1.2. Architektura warstwy logiki biznesowej

Warstwę logiki biznesowej pisano w języku Java [1], korzystając ze szkieletu Spring [3]. Klasy w Springu można podzielić ze względu na funkcję, jaką wykonują, są to kontrolery, serwisy, repozytoria oraz obiekty modelu. Podział ten wynika ze specyfiki szkieletu Spring, a konkretnie – z zestawu dostępnych adnotacji, więcej szczegółów na ten temat można znaleźć w artykule [83]. Kontrolery służą do wymiany informacji z warstwą prezentacji, wystawiają punkty komunikacji (*ang. REST point*) definiujące żądania, jakie rozumie warstwa logiki biznesowej. Serwisy są odpowiedzialne za wykonanie logiki biznesowej, mają dostęp do dalszych części systemu, jakimi są repozytoria. Zabezpieczają repozytoria przed niepoprawnym oraz niepożądanym ich użyciem. Repozytoria formułują oraz zadają zapytania do warstwy danych. Obiekty modelu to klasy, których można używać,



Rysunek 6.2. Diagram ER bazy danych

aby przeprowadzać operacje systemowe, wyodrębniono ich trzy rodzaje: DTO, encje oraz obiekty domenowe. DTO (*ang. Data Transfer Object*) to obiekt, który jest używany do przenoszenia informacji pomiędzy warstwą prezentacji a warstwą logiki biznesowej. Transakcje bazy danych używają encji. Obiekty domenowe są stosowane podczas wykonywania operacji biznesowych. Istnieją jeszcze klasy budujące (*ang. builder*), których zadaniem jest tworzenie obiektów innej klasy (wzorzec projektowy budowniczy [55]) oraz planiści (*ang. scheduler*) zajmujący się uruchamianiem logiki biznesowej o odpowiedniej godzinie.

Warstwę logiki biznesowej podzielono na pakiety zorientowane wokół tematu, jaki obsługują. Wyjątkiem jest pakiet security, w którym trzymane są obiekty konfigurujące Spring Security [58] oraz tylko jemu potrzebne. Stworzono następujące pakiety: tasks (tabela 6.1), rooms (tabela 6.3), users (tabela 6.4), images (tabela 6.7), houses (tabela 6.8), security (tabela 6.9).

Wewnątrz pakietu tasks znajdują się klasy wymienione w tabeli 6.1. Logikę związaną z modelem zadania rozbito na trzy klasy: TaskDTO – używaną do komunikacji z warstwą prezentacji, Task – używaną do celów logiki biznesowej oraz TaskEntity – zapisywaną w bazie danych. Takie rozwiązanie pozwala dostosować używane klasy do celu ich zastosowania, a nie koniecznie do wszystkich tych celów łącznie – jak by to było w przypadku tylko jednej klasy Task odpowiedzialnej za wszystkie trzy zadania. Ze względu na pokaźne rozmiary klas Task, TaskDTO i TaskEntity do ich tworzenia postanowiono użyć klas budujących. Ze względu na konieczność zmieniania statusu

Tabela 6.1. Pakiet klas tasks w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
TaskController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych z zadaniami
TaskService	Serwis	Klasa odpowiedzialna za manipulację zadaniami i zarządzanie nimi. Korzysta z TaskRepository, aby klasy poza nią poprawnie używały danych z bazy
TaskRepository	Repozytorium	Interfejs, za pomocą którego wykonuje się zapytania do bazy danych
TaskEntity	Encja	Klasa reprezentująca encję w bazie danych
Task	Obiekt domenowy	Klasa, na której przeprowadzane są operacje logiki biznesowej
TaskDTO	DTO	Klasa, za pomocą której kontroler komunikuje się z warstwą prezentacji
TaskBuilder	Klasa budująca	Klasa budująca Task
TaskDTOBuilder	Klasa budująca	Klasa budująca TaskDTO
TaskEntityBuilder	Klasa budująca	Klasa budująca TaskEntity
UndoneTasksAfterPeriodScheduler	Planista	Klasa, która zawiera logikę związaną z cofaniem zadań do statusu <i>do zrobienia</i> po określonym czasie

zadań na *do zrobienia* stworzono klasę odpowiedzialną za tę zmianę. W pakiecie tasks znajduje się podpakiet updateAlgorithms w którym przechowywane są klasy obsługujące zmianę ceny zadania (tabela 6.2). Bardziej szczegółowy opis tych klas znajduje się w rozdziale 6.2.

Kolejnym pakietem znajdującym się w implementacji warstwy logiki biznesowej jest pakiet rooms przedstawiony w tabeli 6.3. Zajmuje się on logiką związaną z pokojami. Logikę związaną z modelem pokoju umieszczono w dwóch klasach, w przeciwieństwie do zadań. Taka decyzja projektowa podyktowana była znacznie mniejszą ilością logiki oraz atrybutów znajdujących się wewnątrz tych klas.

Kolejnym pakietem znajdującym się w warstwie logiki biznesowej jest pakiet users

Tabela 6.2. Pakiet klas updateAlgorithms w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
TaskPriceUpdaterService	Serwis	Klasa odpowiedzialna za poprawną zmianę ceny zadań. Podczas tworzenia serwisu można wybrać, jaką implementację algorytmu zmiany będzie stosowana
TaskPriceUpdateAlgorithm	Algorytm	Interfejs, po którym dziedziczyć mają klasy implementujące strategię zmiany ceny zadania
ExponentialTaskPriceUpdateAlgorithm	Algorytm	Klasa implementująca algorytm zmiany ceny zadania

Tabela 6.3. Pakiet klas rooms w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
RoomController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych z pokojami
RoomService	Serwis	Klasa odpowiedzialna za manipulację zadaniami i zarządzanie nimi. Korzysta z RoomRepository, aby klasy pozanią poprawnie używały danych z bazy
RoomRepository	Repozytorium	Interfejs, za pomocą którego wykonuje się zapytania do bazy danych
Room	Encja oraz obiekt domenowy	Klasa reprezentująca encję w bazie danych oraz encję, na której wykonywana jest logika biznesowa
RoomDTO	DTO	Klasa, którą kontroler dostaje z warstwy prezentacji jako dane od użytkownika

(tabela 6.4). Jest on odpowiedzialny za logikę związaną z użytkownikami oraz za logowanie i rejestrację użytkowników. Klasa User posiada tylko identyfikator użytkownika, skrót hasła, nazwę użytkownika oraz relację do obiektu klasy HouseBuddy (zawartej w pakiecie house_buddy i opisanej w tabeli 6.5), która jest odpowiedzialna za przechowywanie informacji potrzebnych w logice biznesowej, takich jak: liczba punktów zgromadzonych przez użytkownika, jego dom, liczba odejmowanych punktów raz w tygodniu. Klasa UserDTO jest wysyłana do warstwy prezentacji, dlatego zawiera potrzebne informacje

6. Projekt i implementacja

zarówno z klasy User, jak i z HouseBuddy. Ze względu na założenie wspólnej pracy nad domem każdy użytkownik powinien co tydzień płacić część z zarobionych punktów symbolizujących trud włożony w utrzymanie domu (wymaganie WF 4.5). Tym zadaniem zajmuje się klasa TaxCollectorScheduler. W pakiecie users znajduje się również pakiet authorization, który zawiera kontrolery służące logowaniu, rejestracji oraz modyfikacji użytkownika (tabela 6.6).

Tabela 6.4. Pakiet klas users w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
UserController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych z użytkownikami, ale niezwiązanych z logowaniem
UserService	Serwis	Klasa odpowiedzialna za manipulację użytkownikami i zarządzanie nimi. Korzysta z UserRepository, aby klasy poza nią poprawnie używały danych z bazy
UserRepository	Repozytorium	Interfejs, za pomocą którego wykonuje się zapytania do bazy danych
User	Encja	Klasa reprezentująca encję w bazie danych, nie posiada informacji związanych z logiką biznesową, za tę część odpowiedzialna jest klasa HouseBuddy, która znajduje się w pakiecie house_buddy w tabeli 6.5
UserDTO	DTO	Klasa, którą kontroler dostaje z warstwy prezentacji jako dane od użytkownika
TaxCollectorScheduler	Planista	Klasa obsługująca cotygodniowe odliczanie części punktów z konta użytkownika

Tabela 6.5. Pakiet klas house_buddy w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
HouseBuddyService	Serwis	Klasa odpowiedzialna za manipulację użytkownikami w sensie logiki biznesowej. Korzysta z HouseBuddyRepository, aby klasy poza nią poprawnie używały danych z bazy
HouseBuddyRepository	Repozytorium	Interfejs, za pomocą którego wykonuje się zapytania do bazy danych
HouseBuddy	Encja	Klasa reprezentująca encję w bazie danych posiadającą informacje związane z logiką biznesową

Autorka wyselekcjonowała grupę proponowanych zdjęć profilowych oraz zdjęć pokoi, których użytkownik może użyć podczas ich tworzenia. Klasy służące do obsługi logiki

Tabela 6.6. Pakiet klas autorization w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
LoginController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych z logowaniem użytkowników
SignUpController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych z rejestracją użytkowników
ModifyUserController	Kontroler	Klasa odpowiedzialna za modyfikację i usuwanie użytkowników

związanej ze zdjęciami znajdują się w pakiecie images (tabela 6.7). Oczywiście użytkownik może również znaleźć w internecie inne zdjęcie i użyć go w miejsce zaproponowanych.

Tabela 6.7. Pakiet klas images w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
ImageController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych proponowanymi zdjęciami
ImageService	Serwis	Klasa odpowiedzialna za manipulację zdjęciami i zarządzanie nimi. Korzysta z ImageRepository, aby klasy poza nią poprawnie używały danych z bazy
ImageRepository	Repozytorium	Interfejs, za pomocą którego wykonuje się zapytania do bazy danych
Image	Encja, DTO i obiekt domenowy	Klasa reprezentująca encję w bazie danych, obiekt wysyłany do warstwy prezentacji oraz obiekt, na którym wykonywane są operacje biznesowe
ImageType	Enum	Klasa reprezentująca typ przechowywanego zdjęcia. Obecnie są to dwa typy: AVATAR, oznaczający proponowane zdjęcia profilowe, oraz ROOM, proponowane zdjęcia pokoi

Kolejnym pakietem znajdującym się w warstwie logiki biznesowej jest pakiet houses widoczny w tabeli 6.8. Oprócz typowych klas kontrolera, serwisu czy repozytorium posiada również klasę JoinCodeGenerator, która jest odpowiedzialna za generowanie kodu dostępu do domu. Pakiet houses zawiera również pakiet exceptions, w którym znajdują się wyjątki.

Tabela 6.8. Pakiet klas houses w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
HouseController	Kontroler	Klasa odpowiedzialna za obsługę żądań HTTP związanych z domami
HouseService	Serwis	Klasa odpowiedzialna za manipulację domami i zarządzanie nimi. Korzysta z HouseRepository, aby klasy poza nią poprawnie używały danych z bazy
HouseRepository	Repozytorium	Interfejs, za pomocą którego wykonuje się zapytania do bazy danych
HouseEntity	Encja i obiekt domenowy	Klasa reprezentująca encję w bazie danych oraz jest używana wewnątrz logiki biznesowej
JoinCodeGenerator	Serwis	Klasa generująca nowy kod dostępu dla domu

Klasa AppSecurityConfig, z pakietu security (tabela 6.9), służy do skonfigurowania Spring Security [58]. Robi to, implementując metodę configure odziedziczoną po klasie WebSecurityConfigurerAdapter [81] oraz definiując zależność zwracającą klasę AuthenticationProvider. W klasie AppSecurityConfig ustawiono komunikację pomiędzy warstwą prezentacji a warstwą logiki biznesowej za pomocą protokołu HTTP (*ang. Hypertext Transfer Protocol*). Ustalono również, które punkty komunikacji nie wymagają zalogowania (punkty obsługujące logowanie oraz rejestrację), a resztę żądań ograniczono tylko do zalogowanych użytkowników. Zdefiniowano również, że hasło zapisywane w bazie danych musi być zakodowane za pomocą funkcji skrótu BCrypt, która jest zaimplementowana w klasie BCryptPasswordEncoder dostarczanej przez Spring Security [58]. UserPrincipal to klasa implementująca klasę springową UserDetails potrzebną podczas używania wewnętrznych mechanizmów Spring Security [58]. UserPrincipal jest nakładką na klasę User, implementuje metody potrzebne do poprawnego działania zabezpieczeń, takie jak getUsername czy isCredentialsNonExpired.

Tabela 6.9. Pakiet klas security w warstwie logiki biznesowej

Nazwa klasy	Rodzaj	Zadanie
AppSecurityConfig	Konfiguracja	Konfiguruje Spring Security
UserPrincipal	Obiekt domenowy	Dostarcza podstawowych informacji na temat użytkownika

6.1.3. Architektura warstwy prezentacji

Warstwa prezentacji została napisana w języku TypeScript [2] z użyciem szkieletu Angular [4]. Ze względu na obiektową naturę języka TypeScript w warstwie prezentacji podstawą są klasy. W aplikacji CommOn dzielą się one na komponenty, modele oraz serwisy, które różnią się znacząco pod względem wykonywanych funkcji. Komponenty to fragmenty aplikacji odpowiedzialne za wyświetlanie swojej zawartości oraz bezpośrednią obsługę użytkownika (obsługa zdarzeń, np. kliknięcie przycisku oraz zbieranie danych za pomocą formularzy). Często formuje się z nich drzewiastą strukturę, w której, aby komponent mógł wyświetlić zawartość innego komponentu, musi zatrzymać jego selektor wewnątrz własnego pliku HTML. Modele to obiekty używane do przechowywania danych przychodzących i wychodzących z warstwy logiki biznesowej. Serwisy natomiast służą komunikacji z warstwą logiki biznesowej oraz obsługą danych.

Warstwę prezentacji podzielono na typowe dla projektów angularowych pakiety powiązane z komponentami. Ze względu na to, że z jednym komponentem często związane są trzy lub nawet cztery pliki (pliki o rozszerzeniach .ts, .html, .css, a czasem również .spec.ts), taki podział jest uzasadniony. Z tego względu w tym rozdziale opisano tylko klasy, jakie znajdują się w pakietach, bez zbędnego dzielenia na pojedyncze pakiety zawierające przeważnie jedną klasę oraz pliki potrzebne do jej wyświetlenia w Angularze [4]. Wszystkie klasy typu komponent znajdują się wewnętrznych pakietach. W poniższych tabelach opisano nazwy pakietów, w których są zawarte. Podstawowymi pakietami znajdującymi się w warstwie prezentacji są: tasks (tabela 6.10), rooms (tabela 6.11), profile (tabela 6.12), houses (tabela 6.13), authentication (tabela 6.14), shared (tabela 6.15), header (tabela 6.16). Ten podział pokierowany jest widokami, na jakie podzielone są dane wewnątrz aplikacji.

W pakiecie tasks (tabela 6.10) znajdują się klasy związane z zadaniami. Klasa Task jest modelem zadania. Klasa TaskService tworzy żądania HTTP dotyczące zadań i wysyła je do warstwy logiki biznesowej. Komponent TasksComponent służy głównie logicznemu podziałowi klas związanych z zadaniami. Wyświetla wszystkie pozostałe komponenty. Komponent TaskListComponent jest pierwszym, który użytkownik widzi po zalogowaniu. Wyświetla listę zadań oraz przycisk umożliwiający dodanie do listy nowego zadania. Jest on również używany podczas wyświetlania szczegółów pokoju, jednak wtedy wyświetla tylko zadania związane z danym pokojem, a nie wszystkie. Komponent EditTaskModalComponent zawiera formularz związany z tworzeniem nowych zadań oraz edycją istniejących zadań.

W pakiecie rooms (tabela 6.11) przechowywane są klasy związane z pokojami. Część elementów pokrywa się co do funkcjonalności z wymienianymi wcześniej klasami z pakietu tasks, dlatego nie opisano ich tutaj. Nowym elementem jest komponent RoomDetailComponent, wyświetlający okno skupiające się na konkretnym pokoju.

Tabela 6.10. Pakiet klas tasks w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
TaskService	Serwis	Klasa odpowiedzialna za wysyłanie żądań HTTP związań z zadaniami oraz za przetrzymywanie danych dotyczących zadań i ich udostępnianie komponentom
Task	Model	Klasa reprezentująca zadanie
TasksComponent	Komponent	Klasa wyświetlająca wszystkie adresy URL związane z zadaniami
TaskListComponent	Komponent	Klasa wyświetlająca listę zadań. Klasa znajduje się w pakiecie task-list
EditTaskModalComponent	Komponent	Klasa wyświetlająca formularz służący do tworzenia nowych zadań oraz edycji już istniejących. Klasa znajduje się w pakiecie edit-task-modal

Wyświetla on listę zadań znajdujących się w pokoju, jego nazwę oraz zdjęcie i podsumowanie mówiące, ile jeszcze zadań do zrobienia pozostaje wewnątrz tego pokoju.

Tabela 6.11. Pakiet klas rooms w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
RoomService	Serwis	Klasa odpowiedzialna za wysyłanie żądań HTTP związań z pokojami oraz za przetrzymywanie danych dotyczących pokoi i ich udostępnianie komponentom
Room	Model	Klasa reprezentująca pokój
RoomsComponent	Komponent	Klasa wyświetlająca wszystkie adresy URL związane z pokojami
RoomListComponent	Komponent	Klasa wyświetlająca listę pokoi. Klasa znajduje się w pakiecie room-list
EditRoomModalComponent	Komponent	Klasa wyświetlająca formularz służący do tworzenia nowych zadań oraz edycji już istniejących. Klasa znajduje się w pakiecie edit-room-modal
RoomDetailComponent	Komponent	Klasa wyświetlająca szczegóły związane z konkretnym pokojem, takie jak lista zadań wewnątrz pokoju czy jego zdjęcie. Klasa znajduje się w pakiecie room-detail

Pakiet profile, opisany w tabeli 6.12, zawiera komponenty związane z profilem użytkownika. Profil umożliwia modyfikację ważnych danych konta użytkownika, takich

jak hasło czy zdjęcie profilowe. Umożliwia również usuwanie użytkownika oraz udostępnia kod domu, który można przekazać innemu użytkownikowi, aby ten mógł dołączyć do wspólnego domu. Komponentem odpowiedzialnym za te wszystkie zadania, ale nie zbierającym informacji potrzebnych do ich realizacji, jest komponent ProfileComponent. Informacje zbierają formularze ChangePasswordModalComponent, dla zmiany hasła, oraz EditPhotoModalComponent, dla zmiany zdjęcia profilowego. ProfileService odpowiada za faktyczne wysłanie żądań HTTP do warstwy logiki biznesowej.

Tabela 6.12. Pakiet klas profile w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
ProfileService	Serwis	Klasa odpowiedzialna za wysyłanie żądań HTTP do warstwy logiki biznesowej związanych z ustawieniami profilu użytkownika oraz za przetrzymywanie danych dotyczących ustawień profilu użytkownika i ich udostępnianie komponentom
ProfileComponent	Komponent	Klasa wyświetlająca widok ustawień profilu użytkownika
EditPhotoModalComponent	Komponent	Klasa wyświetlająca formularz odpowiedzialny za zmianę zdjęcia profilowego użytkownika. Klasa znajduje się w pakiecie edit-photo-modal
ChangePasswordModalComponent	Komponent	Klasa wyświetlająca formularz odpowiedzialny za zmianę hasła. Klasa znajduje się w pakiecie change-password-modal

Pakiet houses zawiera klasy obsługujące wyświetlanie stanu domu, opisującą obecny stan domu dla zalogowanego użytkownika oraz innych użytkowników należących do tego domu. Ze względu na to, że wyświetlana jest lista wszystkich użytkowników, to właśnie tutaj zmieniana jest liczba odejmowanych punktów dla pojedynczych użytkowników, aby od razu można było zobaczyć, jak obecnie ustaliona liczba ma się do pozostałych. Do realizowania tego celu służy klasa EditUserModalComponent. Dodatkowo wyświetlane są informacje dotyczące liczby zrealizowanych zadań od początku danego tygodnia, liczba zdobytych punktów oraz pasek postępu pokazujący, jaką część punktów należnych w danym tygodniu już zrealizowano.

Tabela 6.13. Pakiet klas houses w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
HouseService	Serwis	Klasa odpowiedzialna za wysyłanie żądań HTTP związanych z wynikami dla domu oraz za przechowywanie danych dotyczących wyników dla domu i ich udostępnianie komponentom
HousesComponent	Komponent	Klasa wyświetlająca wszystkie adresy URL związane z wynikami dla domu
HouseComponent	Komponent	Klasa wyświetlająca opis wyników dla domu
EditUserModalComponent	Komponent	Klasa wyświetlająca formularz odpowiedzialny za zmianę punktów odejmowanych użytkownikowi co tydzień. Klasa znajduje się w pakiecie edit-user-modal
HouseBuddy	Model	Klasa reprezentująca użytkownika służącego do obsługi logiki biznesowej, takiej jak np. gromadzenie punktów

Pakiet authentication jest odpowiedzialny za wyświetlanie oraz logikę związaną z logowaniem, oraz rejestracją. Ważną klasą jest AuthInterceptorService, czyli klasa, która przechwytuje odpowiedzi przychodzące z warstwy logiki biznesowej i w razie otrzymania sygnału o braku autoryzacji przekierowuje użytkownika na stronę logowania.

Pakiet shared jest dość nietypowy, ponieważ zamiast zawierać obiekty związane z pojedynczym modelem, zawiera komponenty i serwisy realizujące zadania w różnych miejscach aplikacji. ModalInformationService służy wymianie informacji pomiędzy formularzami a komponentami, które ich używają. Przekazanie informacji pomiędzy komponentami było realizowane w CommOn za pomocą adnotacji @Input-@Output oraz serwisu ModalInformationService. Adnotacje @Input-@Output, zapewniane przez Angulara [4], pozwalają na przekazywanie informacji od nadzędnych komponentów do podrzędnych i vice versa. Robią to na podstawie przywiązywania atrybutów obecnych w dzieciach do rodziców (@Input) oraz emitowania zdarzeń jako reakcji na zdarzenia wewnętrz dzieci (@Output). Niestety za pomocą komunikacji ograniczonej do adnotacji nie da się wywołać obsługi zdarzenia wewnętrz komponentów dzieci. Aby to osiągnąć, wstrzyknięto dzieciom serwis ModalInformationService posiadający emiter zdarzeń, który następnie zasubskrybowano. Ten sam serwis wstrzyknięto do rodziców, którzy mogli użyć zdefiniowanych wewnętrz funkcji, aby wygenerować zdarzenie, na które dzieci zareagują. Klasa ProposedImagesComponent służy wyświetleniu dostarczonych przez nadzędny komponent proponowanych zdjęć. Formularz DeleteModalComponent służy potwierdzaniu, że dany obiekt ma w istocie zostać usunięty. Ze względu na to, że ta

Tabela 6.14. Pakiet klas authentication w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
AuthenticationService	Serwis	Klasa obsługująca żądania HTTP dotyczące logowania i wylogowywania użytkownika
User	Model	Klasa reprezentująca użytkownika do logowania
UserSignup	Model	Klasa reprezentująca użytkownika dołączającego do już istniejącego domu. Różni od klasy User, ponieważ posiada dodatkowo kod domu, do którego rejestrowany użytkownik chce dołączyć
SignUpService	Serwis	Klasa wysyłająca żądania rejestracji
SignUpComponent	Komponent	Klasa wyświetlająca formularz rejestracji. Klasa znajduje się w pakiecie sign-up
LoginComponent	Komponent	Klasa wyświetlająca formularz logowania. Klasa znajduje się w pakiecie login
AuthInterceptorService	Serwis	Klasa przechwytyjąca odpowiedzi przychodzące z warstwy logiki biznesowej i w razie otrzymania sygnału o braku autoryzacji przekierowująca użytkownika na stronę logowania

funkcjonalność była potrzebna zarówno do usuwania użytkowników, jak i zadań, wyekstrahowano ją do osobnego, konfigurowalnego komponentu.

Tabela 6.15. Pakiet klas shared w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
ModalInformationService	Serwis	Klasa odpowiedzialna za wymianę informacji pomiędzy formularzami a komponentami, które je zawierają
ProposedImagesComponent	Komponent	Klasa wyświetlająca listę proponowanych zdjęć. Klasa znajduje się w pakiecie proposed-images
DeleteModalComponent	Komponent	Klasa wyświetlająca formularz odpowiedzialny za usuwanie obiektu o danym identyfikatorze. Klasa znajduje się w pakiecie delete-modal

Pakiet header (tabela 6.16) zawiera zaledwie jedną klasę, HeaderComponent, odpowiedzialną za wyświetlanie paska menu znajdującego się nad każdym ekranem zalogowanego użytkownika.

Tabela 6.16. Pakiet klas header w warstwie prezentacji

Nazwa klasy	Rodzaj	Zadanie
HeaderComponent	Komponent	Klasa odpowiedzialna za wyświetlanie paska menu znajdującego się nad każdym ekranem zalogowanego użytkownika

6.2. Implementacja

W tym rozdziale omówiono sposób realizacji założeń architektury na konkretnych przypadkach klas. Pokazano, w jaki sposób żądanie jest przetwarzane wewnątrz systemu. Przedstawiono w algorytm związany ze zmianą cen zadań. Wyjaśniono zasadę działania planistów oraz zabezpieczeń. Omówiono schemat relacyjny bazy danych. Zaprezentowano zawarte w CommOn formularze, czyli wejście i wyjście aplikacji. Zaprezentowano najważniejsze bądź najciekawsze elementy związane z zaimplementowanymi klasami.

6.2.1. Przetwarzanie żądań użytkownika

Omówienie działania całej aplikacji wewnątrz niniejszej pracy jest niemożliwe ze względu na jej ograniczony rozmiar. Aby jednak przekazać ogólny sposób interakcji między warstwami, zasady działania komponentów czy logikę idącą za generowanymi transakcjami baz danych, postanowiono zaprezentować przykład przetwarzania konkretnego żądania – stworzenia nowego zadania. Wszystko zaczyna się od wypełnienia przez użytkownika formularza związanego ze stworzeniem nowego zadania. Więcej na temat formularzy można znaleźć w rozdziale 6.2.3. Komponent odpowiedzialny za przyjmowanie danych od użytkownika – EditTaskModalComponent – tworzy nowy obiekt typu Task. Następnie posiadanej instancji serwisu TaskService zleca wykonanie funkcji addTask. Na listingu 1 pokazano jej implementację. Metoda ta wysyła żądanie HTTP (linia 8) pod adres URL systemu zakończony przyrostkiem /api/addTask, po czym subskrybuje to żądanie (linia 2), aby dostać oczekiwana odpowiedź. Dane wewnętrz ciala żądania są przesyłane w formacie JSON. Dane wysyłane z warstwy logiki biznesowej do warstwy prezentacji podobnie. W momencie otrzymania odpowiedzi na żądanie metoda addTask wysyła żądanie mające na celu odświeżyć posiadane informacje na temat zadań, w tym i tego nowo dodanego (linia 3).

Listing 1. Metoda wysyłająca nowe żądanie stworzenia zadania w TaskService

```
1 addTask(task: Task) {
2     this.addTaskCall(task).subscribe((_: Task) => {
3         this.fetchTasks()
4     });
5 }
```

```
6
7 addTaskCall(task: Task) {
8     return this.http.post<Task>('http://localhost:4200/api/addTask',
9         task, {withCredentials: true});
10}
```

Aby zaprezentować obsługę żądania pochodzącego z warstwy prezentacji, pokazano diagram klas (rysunek 6.3) związanych z zadaniami w warstwie logiki biznesowej. Nie zawarto na nim obiektów modelu ani klas, z których korzysta TaskPriceUpdateService, aby nie zaciemniać obrazu. Na diagramie znajdują się klasy (i interfejsy): TaskController, TaskService, TaskRepository, UndoneTasksAfterPeriodScheduler, TaskPriceUpdaterService oraz Task. Można zauważyc, że TaskRepository jest interfejsem, który nie posiada implementacji widocznej na diagramie. Dzieje się tak, ponieważ Spring [3] generuje implementację tej klasy.

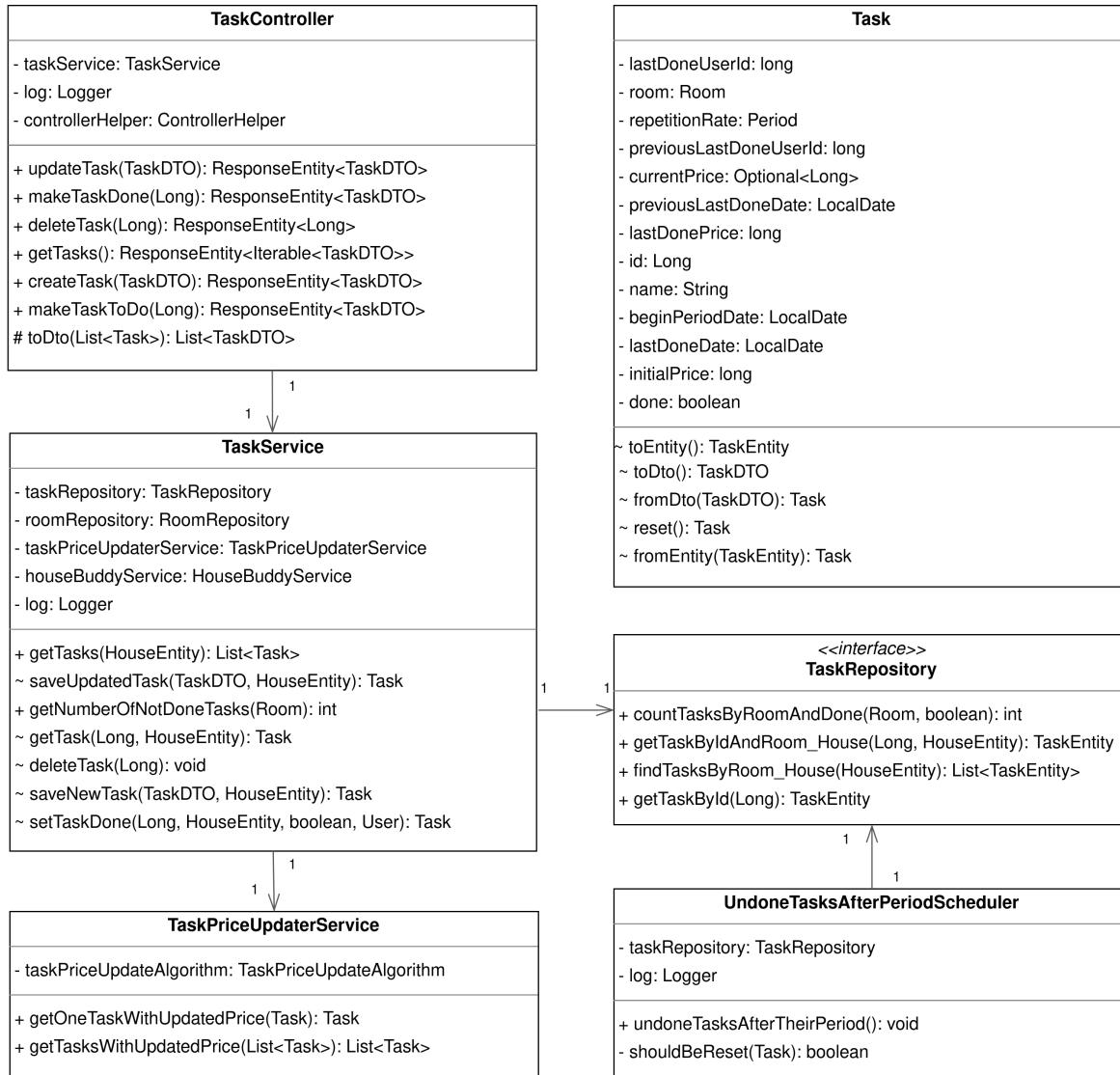
Żądaniem, które przychodzi do TaskController, jest stworzenie nowego zadania. Punktem wejścia żądania jest metoda createTask (widoczna na diagramie 6.3), która zostanie wywołana po wejściu użytkownika pod adres URL, na który zostało wysłane żądanie HTTP z warstwy prezentacji. Listing 2 ukazuje metodę createTask odpowiedzialną za jego obsługę. Adnotacja @PostMapping [81] (linia 1) służy do oznaczenia rodzaju żądania HTTP (np. POST lub GET) oraz wskazania ścieżki, która będzie oznaczała wywołanie tej metody. Adnotacja @RequestBody [81] służy do zdefiniowania oczekiwanej ciało żądania. Wewnątrz ciała metody wywoływany jest TaskService, który posiada metodę służącą do stworzenia nowego zadania (linia 4). Wywołana metoda saveNewTask zwraca obiekt domenowy Task, który należy przekształcić w DTO za pomocą metody toDto (linia 6).

Listing 2. Metoda tworząca nowe zadanie w TaskController

```
1 @PostMapping(path = "/addTask")
2     public ResponseEntity<TaskDTO> createTask(
3         @RequestBody TaskDTO taskDTO) {
4         Task task = taskService.saveNewTask(taskDTO,
5             controllerHelper.getMyHouse());
6         return new ResponseEntity<>(task.toDto(), HttpStatus.OK);
7     }
```

Metodą wołaną przez TaskController jest saveNewTask (widoczna na diagramie 6.3) znajdująca się w klasie TaskService. Jej implementacja znajduje się w listingu 3. Dostaje ona argument typu TaskDTO, który zawiera informacje na temat nowego zadania oraz HouseEntity, reprezentujący dom obecnie zalogowanego użytkownika. W pierwszej kolejności, za pomocą funkcji getRoomByIdAndHouse, pobierane są z bazy dane na

6. Projekt i implementacja



Rysunek 6.3. Diagram wybranych klas z pakietu tasks

temat pokoju, w którym dane zadanie ma zostać osadzone (linia 3). Teoretycznie dałoby się odnaleźć pokój po samym id, jednak włączenie w zapytanie domu powoduje, że użytkownik, nawet jeśli spreparuje jakąś złośliwą wiadomość i spróbuje zmodyfikować dane nieswojego domu, nie będzie miał możliwości modyfikować obiektów, do których nie powinien mieć dostępu. Jeśli wybrany pokój nie istnieje (linia 5) w tym domu, zadanie nie zostanie stworzone a metoda wyrzuci wyjątek IllegalArgumentException (linia 6). Jeśli wszystko przebiega pomyślnie, to tworzony jest nowy obiekt domenowy reprezentujący zadanie (linia 9) oraz ustawiany jest jego pokój (linia 10). Ostatecznie obiekt jest zapisywany w bazie danych (linia 12) przez TaskRepository (zarówno metoda save, jak i repozytorium są widoczne na diagramie 6.3) po wcześniejszej konwersji na typ TaskEntity. Serwis aktualizujący cenę zadania nie jest wołany, ponieważ tworzone jest nowe zadanie – jego cena jest taka sama, jak w momencie inicjalizacji.

Listing 3. Metoda tworząca nowe zadanie w TaskService

```

1   Task saveNewTask(TaskDTO taskDTO, HouseEntity myHouse) {
2       log.debug("Got taskDTO: {}", taskDTO);
3       Room room = roomRepository.getRoomByIdAndHouse(
4           taskDTO.getRoom().getId(), myHouse);
5       if (room == null) {
6           throw new IllegalArgumentException("Room with id: " +
7               taskDTO.getRoom().getId() + " does not exist");
8       }
9       Task task = Task.fromDto(taskDTO);
10      task.setRoom(room);
11      log.debug("Converted to task: {}", task);
12      return Task.fromEntity(taskRepository.save(task.toEntity()));
13  }

```

Podczas innych operacji wysyłających informacje o zadaniach używany jest TaskPriceUpdateService (widoczny na diagramie 6.3). Jego celem jest obliczenie obecnej ceny zadania. Używa on interfejsu TaskPriceUpdateAlgorithm, który definiuje, jak ma wyglądać algorytm zmiany ceny zadania. Jeśli programista ma ochotę stworzyć nową implementację, powinien stworzyć nową klasę dziedziczącą po TaskPriceUpdateAlgorithm. Wtedy podczas tworzenia serwisu TaskPriceUpdaterService można użyć nowo napisanej klasy, co w konsekwencji spowoduje, że nowy algorytm będzie używany w całej aplikacji. Interfejs TaskPriceUpdateAlgorithm narzuca zdefiniowanie tylko jednej metody: getNewPrice. Służy ona do wyliczenia ceny pojedynczego zadania. Obecnie używana implementacja to ExponentialTaskPriceUpdateAlgorithm. Sposób wyliczania nowej ceny zadania zgodnie z tym algorytmem wyraża się wzorem (1). Autorka wybrała taką funkcję ze względu na to, że w każdym kolejnym okresie niewykonania zadania jego cena rośnie o połowę względem ceny z początku okresu.

$$X(t) = \begin{cases} x_0 & \text{dla } t \leq T \\ x_0 \cdot \frac{3}{2}^{(\frac{t}{T}-1)} & \text{dla } t > T \end{cases} \quad (1)$$

gdzie:

x_0 : Cena początkowa zadania

t : Czas, który upłynął od ostatniego wykonania zadania

T : Okres, w jakim należy wykonać zadanie

Inną istotną z punktu widzenia zrozumienia działania aplikacji klasą jest

6. Projekt i implementacja

UndoneTasksAfterPeriodScheduler, widoczna na diagramie 6.3. Służy ona do zapewnienia funkcjonalności koniecznej do spełnienia wymagania WF 1.7, czyli automatycznej zmiany statusu zadań na *do wykonania* po upłynięciu okresu, w jakim mają zostać wykonane od czasu, w jakim ostatnio je wykonywano. Implementacja metody odpowiedzialnej za to zadanie znajduje się na listingu 4. Annotacja zapewniająca wykonywanie metody o określonym czasie nazywa się @Scheduled i jest zapewniana przez Spring [3] (linia 1). W tym przypadku zdefiniowano, że metoda będzie automatycznie uruchamiana codziennie o godzinie 23.50. W metodzie najpierw pobrano wszystkie zadania znajdujące się wewnętrz bazy danych (linia 3), następnie za pomocą strumienia, odfiltrowywane (linia 7) są tylko te zadania, które są zrobione, a data ich ostatniego wykonania jest odległa przynajmniej o okres od daty dzisiejszej. Tym zadaniem zajmuje się funkcja shouldBeReset (linia 16). W dalszej kolejności wyselekcjonowane zadania są resetowane (linia 9) oraz zapisywane z powrotem do bazy (linia 13).

Listing 4. Metoda zmieniająca statusy zadań w klasie UndoneTasksAfterPeriodScheduler

```
1  @Scheduled(cron = "0 50 23 * * *")
2  public void undoneTasksAfterTheirPeriod() {
3      Iterable<TaskEntity> allTasks = taskRepository.findAll();
4      List<TaskEntity> collect = StreamSupport
5          .stream(allTasks.spliterator(), false)
6          .map(Task::fromEntity)
7          .filter(this::shouldBeReset)
8          .logForScheduler())
9          .map(Task::reset)
10         .map(Task::toEntity)
11         .collect(Collectors.toList());
12     log.info("reset {} tasks", collect.size());
13     taskRepository.saveAll(collect);
14 }
15
16 private boolean shouldBeReset(Task task) {
17     return task.isDone() && task.getLastDoneDate().isBefore(
18         LocalDate.now().minus(
19             task.getRepetitionRate().minusDays(1)));
20     // minus 1 because also equal to beginPeriodDate is counted
21 }
```

Kolejną ciekawą klasą w warstwie logiki biznesowej jest JoinCodeGenerator. Implementacja znajduje się na listingu 5. Jest ona używana przez HouseService do generowania kodów dołączenia do domu. Jak widać, posiada ona metodę

generateNewJoinCode (linia 10), która generuje liczbę w zakresie 1000-9999. Zmienna random (linia 4), wstrzykiwana przez Springa, znajduje się w klasie BeanInjectionConfig zdefiniowanej przez autorkę. Aby ją utworzyć, użyto należącej do języka Java klasy SecureRandom, która jest specjalną klasą podającą kryptograficznie mocną liczbę [80]. Zanim nowo wygenerowany kod domu zostanie faktycznie użyty, HouseSevice sprawdza, czy taki kod nie jest już obecny. Jeśli jest, losowany jest nowy.

Listing 5. Klasa JoinCodeGenerator

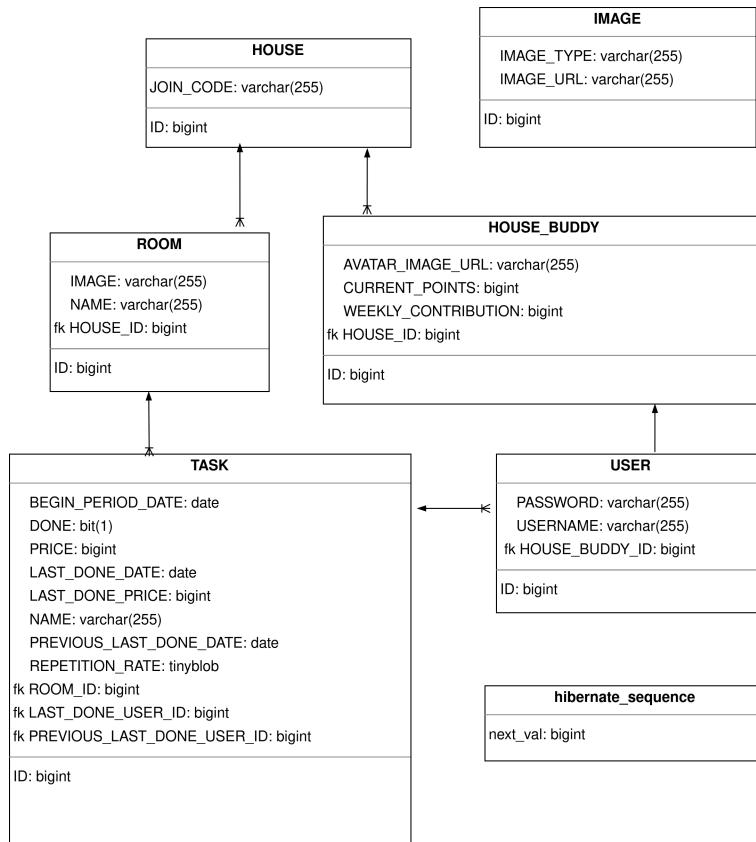
```
1  @Service
2  class JoinCodeGenerator {
3
4      private final Random random;
5
6      public JoinCodeGenerator(Random random) {
7          this.random = random;
8      }
9
10     String generateNewJoinCode() {
11         int generatedCode = this.random.nextInt(9000) + 1000;
12         //range 1000–9999
13         return Integer.toString(generatedCode);
14     }
15 }
```

W aplikacji CommOn występują różne mechanizmy zapewniania bezpieczeństwa przechowywanych danych użytkowników. Podstawowe mechanizmy zabezpieczania warstwy logiki biznesowej są zapewnione przez bibliotekę Spring Security [58]. Podstawową ochroną jest konieczność autoryzacji użytkownika przed obsługiwaniem jego żądań. W tym celu musi się zalogować przy użyciu strony udostępnianej przez warstwę prezentacji. Po stronie warstwy logiki biznesowej wystawione są punkty komunikacji, które nie wymagają wcześniejszego logowania, np. adres służący do logowania czy rejestracji. Po pomyślnym logowaniu użytkownik otrzymuje z powrotem token sesji, który następnie jest używany przy wykonywaniu żądań zamiast przesyłania za każdym razem hasła oraz nazwy użytkownika. Kolejnym zabezpieczeniem jest stosowanie kontekstowości podczas obsługi zapytań. Za pomocą klasy ControllerHelper pozyskiwany jest zalogowany użytkownik, który wysłał obsługiwane żądanie. Dzięki zadawaniu zapytań do bazy danych zawsze z użyciem kontekstu domu obecnego użytkownika nie jest on w stanie uzyskać informacji, do których nie ma praw (np. danych z innego domu), nawet jeśli spreparowałby żądanie ze złośliwym zamiarem.

6.2.2. Model relacyjny bazy danych

W celu przechowywania danych użytkownika skorzystano z bazy danych MySQL [5], opisanej w rozdziale 5.4. Komunikacja ze strony warstwy logiki biznesowej przebiega z użyciem biblioteki Hibernate [28] oraz Springa [3]. Interfejs CrudRepository, zapewniany przez Springa, na podstawie nazwy użytej funkcji generuje odpowiednie zapytanie SQL. Model relacyjny bazy danych zaprezentowano na rysunku 6.4. Jest on oparty na modelu ER zamieszczonym w rozdziale 6.1. Pola na dole każdej tabeli oznaczają klucz główny, są one typu bigint.

Tabela TASK, oprócz klucza głównego, posiada jeszcze trzy klucze obce: LAST_DONE_USER_ID, PREVIOUS_LAST_DONE_USER_ID oraz ROOM_ID. Tabela ROOM posiada klucz obcy HOUSE_ID, co wynika z jej relacji jeden do wielu z domem. Tabela HOUSE może mieć wiele pokoi oraz wielu użytkowników związanych z logiką biznesową. IMAGE nie posiada relacji. Tabele USER i HOUSE_BUDDY posiadają relację jeden do jednego, co wynika z decyzji projektowej, jaką było podzielenie użytkownika na dwie encje. Sekwencja hibernate_sequence jest używana do generowania identyfikatorów.



Rysunek 6.4. Diagram relacyjny bazy danych

Aby przetłumaczyć model obiektowy, używany w Javie, na model relacyjny, używany w bazie danych, użyto narzędzia Hibernate [28], o czym wspomniano w rozdziale 5.3.1.

W dalszej części tego rozdziału przedstawiono przykładową realizację klasy reprezentującej encję. Pozostałe klasy były tworzone analogicznie.

Kod przykładowej encji znajduje się na listingu 6. Adnotacja @Entity (linia 1) służy do oznaczenia obiektu, który może być zapisany w bazie danych. Pochodzi ona ze Springa [3]. Adnotacja @Data służy do generowania potrzebnych metod, takich jak metody dostępu do atrybutów (*ang. getter*), metody ustawiające atrybuty, które nie są stałe (*ang. final*), konstruktory wymaganych atrybutów oraz metody porównawcze (metodę hashCode oraz equals). Adnotacja ta, znajdująca się w linii 2, jest zapewniana przez Lomboka [29]. Adnotacja @Table (linia 3), zapewniana przez Springa, służy do zdefiniowania nazwy tabeli w bazie danych. Wewnątrz klasy HouseEntity znajdują się 4 atrybuty: identyfikator, kod dołączenia do domu, zbiór pokoi oraz zbiór użytkowników. Podczas zapisu do bazy danych identyfikatorem obiektu zostaje atrybut oznaczony adnotacją @Id (linia 6). W aplikacji CommOn przyjęto strategię generowania id podczas pierwszego zapisu do bazy. Świadczy o tym adnotacja @GeneratedValue (linia 7). Ostatnią adnotacją obecną przy pierwszym atrybutie jest @Column (linia 8), służąca do nadania kolumnom w bazie danych odpowiedniej nazwy. Wszystkie trzy adnotacje są dostarczane przez Springa. Adnotacja @JsonIgnore (linia 14) jest używana podczas tworzenia pliku JSON na podstawie danej klasy. Aby utworzyć bazodanową relację w Springu, należy zdefiniować ją w obu klasach powiązanych związkiem. Z tego względu, gdy obiekt jest konwertowany do pliku JSON, następuje nieskończona rekurencja. Aby temu zapobiec, stosowana jest adnotacja @JsonIgnore która zapobiega dodaniu danego pola do pliku. Ostatnią interesującą adnotacją jest @OneToMany (linia 15), która służy do zdefiniowania relacji bazodanowej jeden do wielu. Jak widać na listingu 6, dom może posiadać wiele pokoi oraz wielu użytkowników logiki biznesowej (typu HouseBuddy).

Listing 6. Klasa przykładowej encji - HouseEntity

```

1  @Entity
2  @Data
3  @Table(name = "HOUSE")
4  public class HouseEntity {
5
6      @Id
7      @GeneratedValue(strategy = GenerationType.AUTO)
8      @Column(name = "ID")
9      private Long id;
10
11     @Column(name = "JOIN_CODE")
12     private String joinCode;
13

```

```
14     @JsonIgnore
15     @OneToMany(mappedBy = "house", fetch = FetchType.EAGER,
16     cascade = CascadeType.MERGE)
17     private Set<Room> rooms = new HashSet<>();
18
19     @JsonIgnore
20     @OneToMany(mappedBy = "house", fetch = FetchType.EAGER,
21     cascade = CascadeType.MERGE)
22     private Set<HouseBuddy> houseBuddies = new HashSet<>();
23 }
```

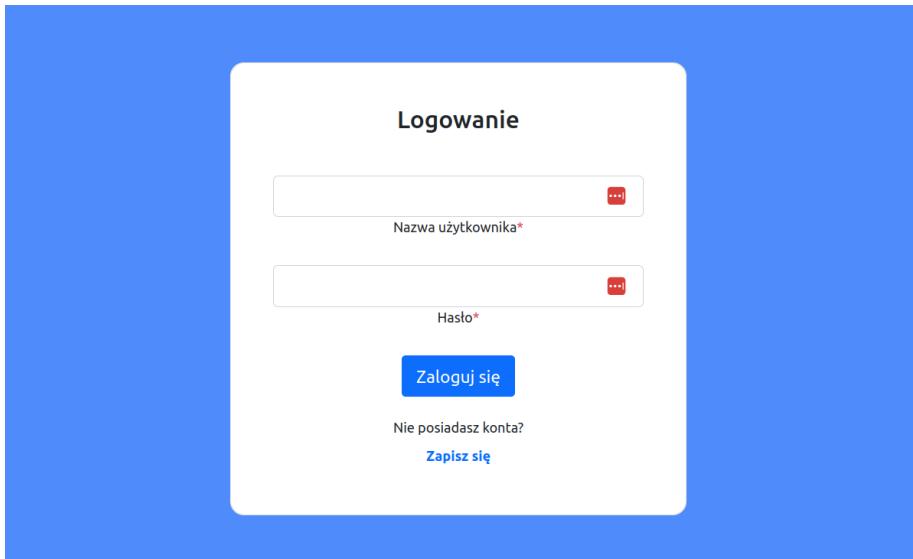
6.2.3. Wejście i wyjście

W kontekście aplikacji internetowych wejściem można nazwać akcje użytkownika, które wpływają na zmianę obiektów wewnątrz aplikacji. W przypadku wyjścia mamy do czynienia z wyświetleniem stosownej reakcji. CommOn używa wielu dostępnych sposobów zebrania informacji od użytkownika.

Przykładowym źródłem danych wejściowych jest przycisk zmieniający stan zadania na wykonane. Reakcją jest wizualne przeniesienie zadania do listy wykonanych, wyszarzenie go oraz zmodyfikowanie jego wartości, takich jak obecna cena, użytkownik, który je ostatnio wykonywał czy data wykonania. Pozostałe zmiany systemu w oparciu o przyciski są analogiczne, dlatego zdecydowano się ich tutaj nie przytaczać.

Inną formą interakcji z aplikacją są formularze. Pozwalają one użytkownikowi na wpisywanie różnych wartości, jednak podlegają też walidacji, tak aby wprowadzane dane miały sens. W aplikacji CommOn znajduje się 7 takich formularzy: logowanie, rejestracja, stworzenie lub zmiana zadania, zmiana liczby wymaganych punktów dla użytkownika na tydzień, zmiana hasła, stworzenie lub zmiana pokoju, zmiana zdjęcia profilowego. Aby zaprezentować dokładniej możliwości komunikowania się z CommOn, w dalszej części tego rozdziału omówiono wygląd oraz działanie tych formularzy.

Logowanie to pierwsza czynność, jaką użytkownik wykonuje. Na rysunku 6.5 pokazany jest formularz, który należy wypełnić w celu zalogowania. Wartości, jakie można w nim wpisać, są dowolne, ponieważ jeśli nie są poprawne, warstwa logiki biznesowej zwróci odpowiedź 401 Unauthorized, a na stronie wyświetli się komunikat „Logowanie nie powiodło się. Sprawdź nazwę użytkownika i hasło”. Jest to jedyna strona, która otrzyma odpowiedź 401 z serwera, ponieważ reszta jest zabezpieczona za pomocą klasy AuthInterceptorService. Znajduje się ona w pakiecie authentication (tabela 6.14) i jest odpowiedzialna za przechwytywanie informacji wracających z warstwy logiki biznesowej. Jej celem jest wykrycie sytuacji, w której niezalogowany użytkownik spróbuje



Rysunek 6.5. Formularz strony logowania

dostać się na zabezpieczoną podstronę aplikacji. W takim wypadku wykrywa odpowiedź serwera 401 Unauthorized i reaguje przekierowaniem na stronę logowania.

Kolejnym formularzem obecnym w CommOn jest formularz rejestracji. Posiada on trzy obowiązkowe pola (oznaczone symbolem „*”) oraz jedno pole opcjonalne. *Nazwa użytkownika* jest wymagana, co oznacza, że nie może być pusta. Podobnie sprawia się z polem *Hasło*. Ostatnią wymaganą informacją jest powtórzenie hasła, które ma obowiązek być identyczne z pierwotnie wpisany hasłem. Jeśli użytkownik chce dołączyć do istniejącego domu, powinien wypełnić nieobowiązkowe pole *Kod domu* za pomocą kodu podanego mu przez innego domownika. Jeśli to pole pozostanie puste, zostanie wygenerowany nowy dom, w którym zostanie umieszczony nowy użytkownik. Na rysunku 6.6 zaprezentowano formularz rejestracyjny z wszystkimi możliwymi błędami. Jak widać, wyświetlają się stosowne komunikaty w jednolitym stylu oraz przycisk zapisujący jest wyszarzony.

Inny formularz, zwany dalej formularzem zadania, służy do stworzenia lub zmiany zadania. Wszystkie jego pola muszą zostać wypełnione, co tak, jak w przypadku poprzedniego formularza, jest oznaczone za pomocą gwiazdek. Oprócz tego *Cena początkowa* oraz *Okres powtarzania zadania* mają nałożone dodatkowe ograniczenie, jakim jest bycie liczbą całkowitą większą od zera. Nazwa pokoju może być wybrana spośród istniejących w danym domu pokoi. Wszystkie komunikaty o błędach zostały pokazane na rysunku 6.7.

Następny formularz dotyczy zmiany liczby wymaganych punktów dla użytkownika na tydzień. Nie zostaną zaakceptowane liczby niecałkowite ani mniejsze od zera. W wypadku, w którym użytkownik wpisze coś niebędącego liczbą albo nie wpisze nic,

The screenshot shows a registration form titled "Zapisz się". It includes three input fields with validation messages: "Nazwa użytkownika*" (Username*) with "Nazwa użytkownika nie może być pusta" (The username cannot be empty), "Hasło*" (Password*) with "Hasło nie może być puste" (The password cannot be empty), and "Powtórz hasło*" (Repeat Password*) with "Powtórzone hasło musi być takie samo jak hasło!" (The repeated password must be the same as the password!). There is also a checked checkbox "Mam już dom, nie chcę generować nowego kodu" (I already have a home, I don't want to generate a new code) and a note "Skoro masz już dom wprowadź proszę jego kod abym mógł dodać Cię do niego :)" (Since you already have a home, please enter its code so I can add you to it :)). A "Kod domu" (Home code) input field is present, and a blue "Zapisz się" (Sign up) button at the bottom.

Rysunek 6.6. Formularz strony rejestracji ze wszystkimi możliwymi błędami

formularz zareaguje nakazem wpisania liczby. Wymagania oraz wyświetlane komunikaty są dokładnie takie same jak w przypadku formularza zadań.

Zmianą hasła użytkownika zajmuje się formularz *zmiana hasła*. Wymaga on, aby użytkownik podał następujące informacje: obecne hasło, nowe hasło oraz aby powtórzył nowe hasło. Każdy z pól jest obowiązkowe, jeśli nie zostanie wypełnione, przycisk wysyłający formularz nie będzie aktywny. Powtórzone hasło dodatkowo musi być takie samo jak nowe hasło.

Kolejny formularz służy do stworzenia lub zmiany nazwy pokoju i jest dalej nazywany formularzem pokoju. Jest on zaprezentowany na rysunku 6.8. Jak widać, istnieje obowiązek wpisania nazwy oraz możliwość zamieszczenia adresu URL zdjęcia. Istnieje również możliwość wyboru ze zdjęć wyselekcjonowanych przez autorkę. Za pomocą kliknięcia wybranego obrazka jego adres jest wklejany w pole *Ścieżka do zdjęcia*. Jeśli użytkownik spróbuje wpisać niepoprawny URL, formularz na to nie pozwoli i zaproponuje wybór z gotowych zdjęć lub pozostawienie pustego pola. Jeśli użytkownik

Rysunek 6.7. Formularz zadania ze wszystkimi możliwymi błędami

wybierze drugą opcję, w ramach zdjęcia zostanie umieszczony domyślny URL przedstawiający prosty symbol domu.

Aby zmienić zdjęcie profilowe, użytkownik używa formularza *zmiana zdjęcia profilowego*. Zawiera on tylko jedną informację do uzupełnienia – adres URL nowego zdjęcia. Użytkownik może samodzielnie znaleźć taki adres albo użyć jednego z zaproponowanych przez autorkę. Pod spodem wyświetli się lista proponowanych obrazów. Tak jak w przypadku formularza pokoju, tu również użytkownik może kliknąć wybrany obrazek. Jak w poprzednim formularzu, tak i w tym występuje walidacja URL.

Zasada działania wszystkich wyżej wymienionych formularzy jest niemal identyczna, dlatego w dalszej części omówiono jeden z nich zawierający większość funkcjonalności pozostałych. W listingu 7 znajduje się fragment kodu pliku .ts z klasy EditTaskModalComponent odpowiedzialnej za walidację formularzy. W listingu 8 znajduje się odpowiadający fragment kodu z pliku .html.

Podczas obsługi wejścia ze strony użytkownika aplikacja używa obiektów o nazwie FormGroup, zapewnianych przez Angulara. W przykładzie widać metodę inicjalizującą formularz domyślnymi wartościami. W linii 8 na listingu 7 tworzony jest nowy obiekt typu

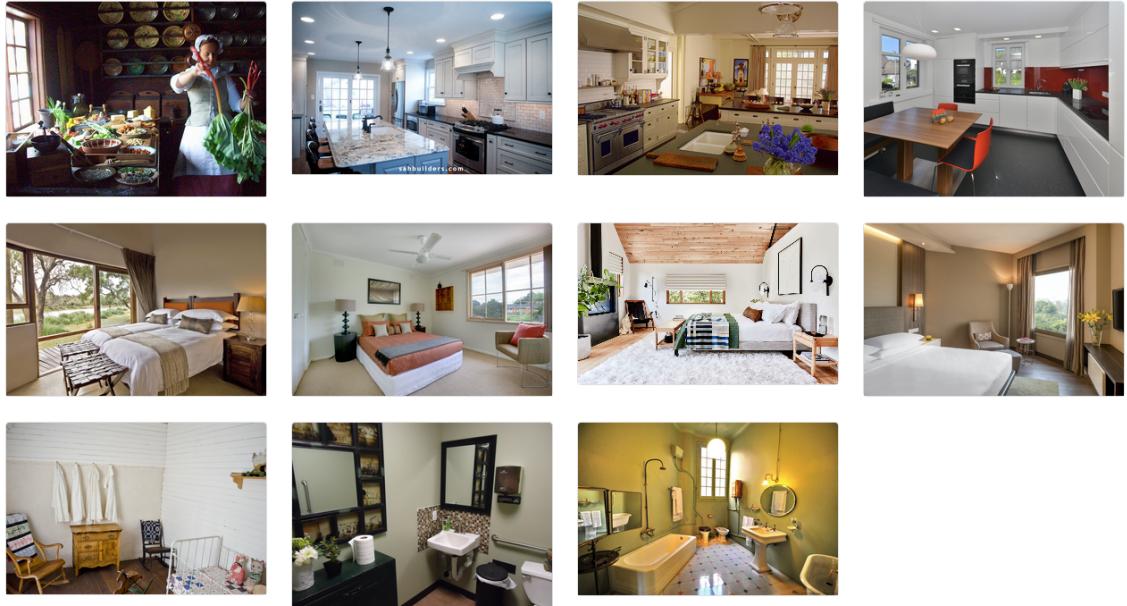
6. Projekt i implementacja

Stwórz nowy pokój

Nazwa Pokoju*
Kuchnia

Ścieżka do zdjęcia
https://upload.wikimedia.org/wikipedia/commons/b/b8/L_K%C3%BCche_2015.jpg

Jeśli nie masz pomysłu na zdjęcie może chcesz wybrać jedno z zaproponowanych przez nas:



Anuluj **Stwórz pokój**

Rysunek 6.8. Formularz pokoju

FormGroup, którego składowymi są obiekty FormControl. Jak widać w linii 10, oprócz domyślnej wartości można też przekazać listę validatorów. Obiekt o nazwie initialPrice posiada trzy validators: Validators.required, Validators.min(1) oraz Validators.pattern. Pierwszy z nich powoduje, że pole wejściowe nie może być puste. Drugi powoduje, że minimalną wartością musi być jeden. Ostatni stosuje napisane przez autorkę wyrażenie regularne zapewniające, że pole wejściowe będzie liczbą całkowitą.

Listing 7. Plik .ts klasy EditTaskModalComponent

```
1 ...
2 private initFormEmpty() {
3     let taskName = '';
4     let taskInitialPrice: number = 10;
5     let taskRoomName: string = '';
6     let repetitionRateInDays: number = 7;
7 }
```

```

8   this.taskForm = new FormGroup({
9     'name': new FormControl(taskName, Validators.required),
10    'initialPrice': new FormControl(taskInitialPrice,
11      [Validators.required, Validators.min(1),
12       Validators.pattern(/^\d+$/)]),
13    'roomName': new FormControl(taskRoomName,
14      Validators.required),
15    'repetitionRateInDays': new FormControl(repetitionRateInDays,
16      [Validators.required, Validators.min(1),
17       Validators.pattern(/^\d+$/)])
18  });
19 }
20 }

```

Na listingu 8 zademonstrowano użycie właściwości zapewnianych przez FormGroup i inne klasy angularowe podczas sprawdzania, czy i jakie pojawiają się błędy. Linia 1 oznacza miejsce, w które użytkownik będzie wpisywał dane. Walidacja znajduje się poniżej. W linii 6 sprawdzana jest ogólna poprawność pola. Linia 7 upewnia się, że pole zostało już kliknięte, aby nie wyświetlać komunikatów o błędach, zanim użytkownik zdąży kliknąć pole. Linia 14 jest aktywowana, gdy pole nie spełnia wymagania zdefiniowanego wspomnianym wyżej wyrażeniem regularnym. Sprawdzenie dodatniości liczby jest wykonywane w linii 20.

Listing 8. Plik .html klasy EditTaskModalComponent

```

1 <input
2   type="number"
3   id="initialPrice"
4   formControlName="initialPrice"
5   class="form-control">
6 <div *ngIf="!taskForm?.get('initialPrice')?.valid &&
7   taskForm?.get('initialPrice')?.touched &&
8   taskForm?.get('initialPrice')?.errors['required']"
9   class="alert alert-danger small">
10  To pole nie może być puste i musi być liczbą!
11 </div>
12 <div *ngIf="!taskForm?.get('initialPrice')?.valid &&
13   taskForm?.get('initialPrice')?.touched &&
14   taskForm?.get('initialPrice')?.errors['pattern']"
15   class="alert alert-danger small">

```

6. Projekt i implementacja

```
16      Wpisana wartość musi być liczbą całowitą!
17  </div>
18 <div *ngIf="!taskForm?.get('initialPrice')?.valid &&
19      taskForm?.get('initialPrice')?.touched &&
20      taskForm?.get('initialPrice')?.value < 1"
21      class="alert alert-danger small">
22      Cena musi być większa od 0!
23  </div>
24 }
```

7. Testy

Testowanie aplikacji to jedno z najważniejszych zadań wykonywanych podczas rozwoju nowego oprogramowania. Pozwala ono zwiększyć prawdopodobieństwo poprawności działania aplikacji oraz zapewnia solidny fundament do wprowadzania nowych funkcjonalności. W aplikacji CommOn zostały przeprowadzone zarówno testy automatyczne (głównie warstwa logiki biznesowej, ale również warstwa prezentacji), jak i manualne (wszystkie trzy warstwy). Przeprowadzano testy jednostkowe (głównie automatyczne), integracyjne (manualnie) oraz systemowe (manualnie). Szczegóły procesu testowania aplikacji przedstawiono w niniejszym rozdziale.

7.1. Testy automatyczne

Testy automatyczne są wyjątkowo szybkie i powtarzalne. Narzucają również zaplanowanie odpowiedniej architektury, ponieważ aby przetestować tylko wybrane fragmenty, kod musi być odpowiednio podzielony. Pozwalają na stworzenie powtarzalnych scenariuszy przy użyciu atrap (*ang. mock*). Przez ograniczenie czynnika ludzkiego podczas ich wykonywania, maleje ryzyko błędu.

7.1.1. Warstwa logiki biznesowej

Testy przeprowadzane na warstwie logiki biznesowej były zautomatyzowane w największym stopniu. Jest to częściowo zasługa TDD (*ang. test-driven development*) [45]. Jest to metodologia rozwoju oprogramowania kładąca nacisk na pisanie testów przed stworzeniem kodu produkcyjnego odpowiedzialnego za przejście testu [84].

Listing 9. Klasa abstrakcyjna ControllerTest

```
1 @ExtendWith(SpringExtension.class)
2 public abstract class ControllerTest {
3
4     @MockBean
5     protected TaskRepository taskRepository;
6
7     @MockBean
8     @Qualifier("userService")
9     protected UserDetailsService userDetailsService;
10
11    ...
12
13    @Autowired
14    private MockMvc mockMvc;
```

```
15  
16     protected MockMvc getMockMvc() {  
17         return mockMvc;  
18     }  
19 }
```

Podczas tworzenia aplikacji CommOn inspirowano się tą metodologią – najpierw pisano nieprzechodzący test, a potem go naprawiano. Oczywiście idealne zastosowanie TDD w praktyce zajmuje wyjątkowo dużo czasu, dlatego zdecydowano się raczej na wprowadzenie wersji implementującej dobrze funkcjonalność od razu zamiast na implementowanie zawsze minimalnej ilości kodu. Autorka przekonała się jednak, że ten sposób programowania jest bardzo użyteczny, ponieważ wymusza zastosowanie odpowiedniego podziału architektonicznego klas, aby dało się testować za pomocą atrap. Do definiowania oraz uruchamiania testów użyto biblioteki JUnit [31], natomiast do definiowania atrap skorzystano z biblioteki Mockito [64]. Biblioteki te opisano dokładniej w rozdziale poświęconym użytym technologiom 5.3.2.

Testy warstwy logiki biznesowej zostały podzielone w taki sposób, aby można było testować każdą funkcję oddziennie. Aby to osiągnąć, użyto biblioteki Mockito [64], która umożliwia imitowanie poszczególnych komponentów programu i przewidywalne zwracanie oczekiwanych wartości. Podstawowymi klasami, jakie testowano, są klasy kontrolerów oraz serwisów. Funkcjonalność repozytoriów, czyli obsługa zapytań do bazy danych, została zapewniona przez interfejs CrudRepository [81] znajdujący się w Springu [3]. Ze względu na to, że testowanie repozytoriów byłoby sprawdzaniem działania biblioteki, nie zdecydowano się na tworzenie testów automatycznych tego fragmentu oprogramowania. Szczegóły dotyczące architektury aplikacji i podziału klas na serwisy, kontrolery oraz repozytoria zostały omówione w rozdziale 6.1.

Ze względu na duże podobieństwo oraz pokrywające się fragmenty kodu testów kontrolerów wyekstrahowano nadziedziczoną klasę abstrakcyjną ControllerTest (pokazaną na listingu 9), po której dziedziczą wszystkie pozostałe klasy testujące kontrolery. Zdefiniowano wewnętrznie niej pola typów powtarzających się we wszystkich klasach testujących kontrolery, takich jak repozytoria i serwisy odpowiedzialne za obsługę zadań, domów, pokoi itp. Linia 1 zawiera adnotację @ExtendWith(SpringExtension.class) [85], służy ona do stworzenia kontekstu springowego podczas przeprowadzania testów. Jest on potrzebny podczas testowania kontrolerów, ale nie jest używany przy testowaniu innych klas. Adnotacja @MockBean [81] służy do stworzenia atrap zależności wstrzykiwanych do kontrolerów, które są tworzone w kontekście springowym (linia 4). Adnotacja Qualifier [81] służy do oznaczenia Springowi, jakiej klasy ma użyć w postaci wstrzykiwanej zależności. W tym wypadku autorka zaznacza, że w roli UserServiceDetail ma zostać użyty serwis UserService który został opisany w rozdziale 6.1. Klasa UserService,

zdefiniowana przez autorkę, dziedziczy po klasie UserDetailService zdefiniowanej przez Spring. Z tego względu potrzebna jest adnotacja, ponieważ w kontekście Springa automatycznie definiowana jest klasa UserDetailService, więc jeśli Spring [3] wykrywa, że istnieje więcej niż jedna klasa, która może wykonywać obowiązki UserDetailService, to słusznie komunikuje błąd, ponieważ nie wiadomo, której z nich należy użyć. Z tego względu trzeba wprost zaznaczyć, że należy użyć klasy UserService. Adnotacja @Qualifier znajduje się w linii 8. Ostatnią interesującą adnotacją wewnątrz klasy abstrakcyjnej ControllerTest jest @Autowired [81]. Służy ona do oznaczenia, którą zależność ma wstrzyknąć Spring [3].

Listing 10. Przykładowa klasa testująca kontroler - TaskControllerTest

```

1  @WebMvcTest(TaskController.class)
2  @AutoConfigureMockMvc(addFilters = false)
3  public class TaskControllerTest extends ControllerTest {
4
5      private static final String TASK_NAME = "name";
6      private static final int INITIAL_PRICE = 10;
7      // ... kolejne pola
8      private final HouseEntity house = new HouseEntity();
9      private final Room room = new Room(1L, ROOM_NAME,
10         ROOM_IMAGE_URL, house);
11
12     @Test
13     @SneakyThrows
14     public void shouldGetTaskById() {
15         // given
16         Task task = new Task(TASK_ID, TASK_NAME, INITIAL_PRICE,
17             NOT_DONE, room);
18         when(taskService.getTask(TASK_ID, controllerHelper
19             .getMyHouse())).thenReturn(task);
20
21         // then
22         getMocMvc().perform(get("/api/task?id=" + TASK_ID))
23             .andExpect(status().isOk())
24             .andExpect(content().json(asJsonString(task)))
25             .andDo(print());
26     }
27 }
```

7. Testy

W listingu 10 przedstawiono typowy test kontrolera. Testy te są oznaczone adnotacją @WebMvcTest [81]. Służy ona do automatycznej konfiguracji Spring Security [58] oraz MockMvc [63]. Aby móc testować kontrolery bez konieczności zalogowania użytkownika, czy innych obecnych w aplikacji dla bezpieczeństwa, ale nie potrzebnych w testach elementów, użyto adnotacji @AutoConfigureMockMvc(addFilters = false) [81]. Flaga addFilters ustawiona na fałsz ma za zadanie pozbyć się ograniczeń związanych z zabezpieczeniem kontaktu z aplikacją. Klasy testujące kontroler dziedziczą po opisanej powyżej klasie abstrakcyjnej ControllerTest. Adnotacja @Test [85] jest wykorzystywana przez JUnit [31] do oznaczenia metody jako testu. Adnotacja @SneakyThrows [86], zapewniana przez Lomboka [29], służy do obsługi wyjątków zadeklarowanych przez wszystkie funkcje znajdujące się wewnętrz oznaczonej metody. Zastosowano ją ze względu na walory estetyczne, jakie wprowadza ona do testów.

W tym teście najpierw definiowane jest zadanie, które powinien zwrócić kontroler (linia 16 na listingu 10). Następnie za pomocą funkcji when() definiowane jest zachowanie metody serwisu, który jest tylko atrapą skonstruowaną przez Mockito [64] (linia 18). Ostatecznie za pomocą klasy MockMvc [63] wykonywane jest zapytanie pod wskazany adres URL (linia 22). Sprawdzany jest status oraz ciało odpowiedzi (linie 23, 24).

Kolejnym rodzajem testowanych klas są klasy serwisów. Nie są one do siebie tak podobne i dlatego nie dziedziczą po wydzielonej klasie abstrakcyjnej. Na listingu 11 przedstawiono przykładowy test serwisu. Adnotacja @MockitoSettings [80] ma za zadanie skonfigurować Mockito [64] (linia 1). Pola wartości stałych używanych w wielu testach znajdują się pomiędzy 3 a 5 linią. Atrybuty, których testy używają, znajdują się pomiędzy 6 a 10 linią. Zmienna systemUnderTest wskazuje na serwis, który jest testowany w danej klasie (linia 11). Jest to konwencja, którą przyjęto, aby programiści czytającemu te testy w przyszłości łatwiej się było zorientować. Adnotacja @BeforeEach [85] oznacza funkcję, która ma zostać wywołana przed każdym testem. W wypadku klasy TaskServiceTest za każdym razem tworzymy testowany serwis z uwagi na to, że stworzenie zainicjowanie go wewnętrz klasy powoduje błąd – imitowane klasy są pobierane jako null.

Listing 11. Przykładowa klasa testująca serwis - TaskServiceTest

```
1  @MockitoSettings
2  class TaskServiceTest {
3      public static final String TASK_NAME = "TaskName";
4      ...
5      public static final long ROOM_ID = 1L;
6      @Mock
7      TaskRepository taskRepository;
8      ...
9      @Mock
```

```
10     RoomRepository roomRepository;
11     private TaskService systemUnderTest;
12
13     @BeforeEach
14     void setUp() {
15         systemUnderTest = new TaskService(taskRepository,
16             roomRepository, taskPriceUpdaterService);
17     }
18
19     @Test
20     void shouldSaveNewTask() {
21         // given
22         TaskDTO taskDTO = new TaskDTO(TASK_NAME, INITIAL_PRICE,
23             NOT_DONE, ROOM_ID);
24         when(taskRepository.save(any()))
25             .thenAnswer(returnsFirstArg());
26         when(roomRepository.getRoomByIdAndHouse(ROOM_ID, house))
27             .thenReturn(room);
28
29         // when
30         Task actual = systemUnderTest.saveNewTask(taskDTO, house);
31
32         // then
33         assertThat(actual)
34             .extracting(
35                 Task::isDone,
36                 Task::getInitialPrice,
37                 Task::getName,
38                 Task::getRoom)
39             .contains(
40                 taskDTO.isDone(),
41                 taskDTO.getPrice(),
42                 taskDTO.getName(),
43                 room);
44
45         verify(roomRepository, times(1))
46             .getRoomByIdAndHouse(ROOM_ID, house);
47         verify(taskRepository, times(1)).save(any());
```

Podczas testowania zastosowano również konwencję given-when-then (GWT) [87]. Test dzieli się na trzy fragmenty [87]. Pierwszy z nich, *given*, odpowiedzialny jest za zdefiniowanie stanu aplikacji przed rozpoczęciem wykonania interesującego nas w teście scenariusza. W wypadku testu TaskService tworzona jest instancja TaskDTO oraz dwie atrapy definiujące zachowanie komponentów wewnętrz testu. Kolejny etap to *when* zawierający kod, który ma zostać przetestowany. W tym wypadku jest to uruchomienie metody saveNewTask na testowanym komponencie. Ostatni, *then*, opisuje to, czego należy się spodziewać po otrzymanych wynikach. W rozpatrywanym przypadku potwierdzana identyczność otrzymanego obiektu z obiektem oczekiwany. Sprawdzane jest, czy funkcje getRoomByIdAndHouse i save zostały wywołane dokładnie raz i z podanymi argumentami.

Pozostałe testy kontrolerów oraz serwisów zostały zrealizowane w sposób analogiczny do przykładów przedstawionych powyżej, dlatego nie będą osobno omówione. Przeprowadzono testy każdej z metod znajdujących się w kontrolerach i serwisach. Poza tym sprawdzono poprawność generowania kodu dołączenia do domu oraz algorytm zmiany ceny zadania. W warstwie logiki biznesowej istnieje łącznie 78 testów, a pokrycie kodu wynosi 87,4%. Średni czas wykonania testów automatycznych warstwy logiki biznesowej wynosi około 4s podczas uruchomienia na maszynie autorki.

Ciekawym testem znajdującym się w warstwie logiki biznesowej jest parametryzowany test algorytmu zmieniającego ceny zadań – ExponentialTaskPriceUpdateAlgorithmTest, pokazany na listingu 12. Metoda shouldCorrectlyCalculateExponentialIncrement oznaczona adnotacją @ParameterizedTest [31] to test parametryzowany (linia 10). Oznacza to, że przyjmuje ona strumień (*ang. stream*) argumentów, na których wykonuje test. Aby otrzymać strumień wejściowy, musi być oznaczona za pomocą adnotacji @MethodSource [85] która w swoim argumencie podaje źródło danych, w tym przypadku jest to metoda provideStringsForIsBlank znajdująca się w linii 31.

Zwraca ona strumień argumentów zawierających po kolej: cenę początkową, ostatnią datę wykonania zadania, okres, w jakim należy wykonywać dane zadanie, obecny czas (jest on konfigurowalny ze względu na użycie atrapy zegara) oraz oczekiwana cenę (linia 32). W sekcji *given* testu jest ustawiana atrapa zegara (linia 16) oraz tworzona jest instancja testowanego algorytmu (linia 20). W sekcji *when* na podstawie podanych argumentów wyliczana jest prawdziwa (linia 24) cena, natomiast w ostatniej sekcji sprawdzone jest, że cena prawdziwa oraz oczekiwana są takie same (linia 28).

Listing 12. Klasa testująca ExponentialTaskPriceUpdateAlgorithmTest

```
1 @MockitoSettings
```

```
2 class ExponentialTaskPriceUpdateAlgorithmTest {
3     public static final LocalDate DATE = LocalDate.of(2022, 1, 1);
4     public static final Period DAY_PERIOD = Period.ofDays(1);
5     private static final Period WEEK_PERIOD = Period.ofWeeks(1);
6
7     @Mock
8     private Clock clock;
9
10    @ParameterizedTest
11    @MethodSource("provideStringsForIsBlank")
12    void shouldCorrectlyCalculateExponentialIncrement(int price,
13        LocalDate lastDoneDate, Period period, LocalDate dateNow,
14        int expectedPrice) {
15        //given
16        when(clock.instant()).thenReturn(dateNow.atStartOfDay()
17            .toInstant(ZoneOffset.UTC));
18        when(clock.getZone()).thenReturn(ZoneOffset.UTC);
19
20        ExponentialTaskPriceUpdateAlgorithm exponentialAlgorithm =
21        new ExponentialTaskPriceUpdateAlgorithm(clock);
22
23        //when
24        long actualPrice = exponentialAlgorithm
25            .getNewPrice(price, lastDoneDate, period);
26
27        //then
28        assertThat(actualPrice).isEqualTo(expectedPrice);
29    }
30
31    private static Stream<Arguments> provideStringsForIsBlank() {
32        return Stream.of(
33            Arguments.of(10, DATE, WEEK_PERIOD, DATE
34                .plusDays(0), 10),
35
36            Arguments.of(10, DATE, WEEK_PERIOD, DATE
37                .plusDays(WEEK_PERIOD.getDays() / 2), 10),
38
39            Arguments.of(10, DATE, DAY_PERIOD, DATE
```

```
40          .plusDays(DAY_PERIOD.getDays()) , 10) ,
41
42          Arguments.of(10, DATE, DAY_PERIOD, DATE
43          .plusDays(DAY_PERIOD.getDays() * 2) , 15) ,
44
45          // 22.5 but cast to int
46          Arguments.of(10, DATE, DAY_PERIOD, DATE
47          .plusDays(DAY_PERIOD.getDays() * 3) , 22) ,
48
49          Arguments.of(10, DATE, WEEK_PERIOD, DATE
50          .plusDays(10) , 11) , // 11.89 but cast to int
51
52          Arguments.of(8, DATE, DAY_PERIOD, DATE
53          .plusDays(DAY_PERIOD.getDays() * 2) , 12)
54
55      );
56  }
57 }
```

7.1.2. Warstwa prezentacji

Testy warstwy prezentacji posiadają zautomatyzowane testy logiki znajdującej się w serwisach. Najważniejszym zadaniem serwisów jest wysyłanie żądań HTTP do warstwy logiki biznesowej, dlatego na tym skupiały się przypadki testowe. Do definiowania testów oraz atrap użyto biblioteki Jasmine [26], natomiast w celu ich uruchomienia w terminalu skorzystano z narzędzia Karma [25]. Narzędzia te opisano dokładniej w rozdziale poświęconym użytym technologiom 5.2.2. Przykładowy test serwisu pokazano na listingu 13. Na początku testu definiowane są zmienne, które będą używane w kolejnych testach. Funkcja beforeEach, zapewniana przez Jasmine [26], służy zdefiniowaniu obiektów potrzebnych przed każdym testem oraz konfigurowaniu modułu testującego (linia 34). TestBed [88] to specjalna klasa służąca do konfigurowania i inicjalizacji środowiska testowego zapewniana przez Angulara [4]. Za pomocą metody configureTestingModule definiowane są serwisy potrzebne do przetestowania TaskService, czyli sam testowany serwis oraz wszystkie zależności potrzebne do jego stworzenia. W liniach 38 oraz 40 tworzone są jednak atrapy, które zostaną użyte do stworzenia testowanego obiektu. Takie rozwiązanie, zapewniane przez Jasmine [26], pozwoli manipulować zachowaniem wstrzykiwanych zależności, a w konsekwencji zapewni przewidywalność testów. Ostatnie dwie linie funkcji beforeEach zapewniają wstrzyknięcie gotowych serwisów do zmiennych używanych później w testach. Funkcja it, znajdująca się w linii 49, definiuje

rozpoczęcie testu. W pierwszej części wywołania znajduje się opis, który zostanie wyświetlony w ramach stosu wywołania, jeśli test się nie powiedzie. Drugim argumentem jest już funkcja anonimowa definiująca ciało testu. Jak widać, funkcja anonimowa jest wywoływana z parametrem done. Jest to wbudowany mechanizm Jasmine [26], który umożliwia testowanie asynchronicznych funkcji, takich jak między innymi zapytania do warstwy logiki biznesowej. Parametr done służy przekazaniu informacji, że nastąpiło wykonanie ciała funkcji asynchronicznej, dlatego właśnie na jego końcu wywoływana jest funkcja done(), a w przypadku niepowodzenia done.fail. W linii 62 znajduje się sprawdzenie, czy dokładnie raz zostało wywołane zapytanie pod odpowiedni adres i z odpowiednimi danymi.

Listing 13. Klasa testująca TaskService

```

1
2 describe('CustomersService', () => {
3   let service: TaskService;
4   let httpSpy: Spy<HttpClient>;
5
6   let house = new House(1, '1234');
7   let room = new Room(1, 'Room 1', 'image_url', house);
8   let room2 = new Room(2, 'Room 2', 'image_url_2', house);
9
10  let expectedTask: Task = new TaskBuilder()
11    .setId(1)
12    .setName('Task 1')
13    .setDone(false)
14    .setInitialPrice(10)
15    .setCurrentPrice(10)
16    .setRoom(room)
17    .setLastDoneDate(new Date())
18    .setRepetitionRateInDays(7)
19    .build();
20
21  let task2: Task = new TaskBuilder()
22    .setId(2)
23    .setName('Task 2')
24    .setDone(false)
25    .setInitialPrice(20)
26    .setCurrentPrice(20)
27    .setRoom(room2)
28    .setLastDoneDate(new Date())

```

7. Testy

```
28     .setRepetitionRateInDays(1)
29     .build();
30
31 const expectedTasks: Task[] =
32   [expectedTask, task2];
33
34 beforeEach((() => {
35   TestBed.configureTestingModule({
36     providers: [
37       TaskService,
38       { provide: HouseService,
39         useValue: createSpyFromClass(HouseService) },
40       { provide: HttpClient,
41         useValue: createSpyFromClass(HttpClient) }
42     ]
43   });
44
45   service = TestBed.inject(TaskService);
46   httpSpy = TestBed.inject<any>(HttpClient);
47 });
48
49 it('should return an expected list of customers', (done: DoneFn) => {
50   httpSpy.get.and.nextWith(expectedTasks);
51
52   service.fetchTasksCall().subscribe(
53     actualTasks => {
54       expect(actualTasks).toHaveLength(expectedTasks.length);
55       expect(actualTasks).toEqual(expectedTasks);
56       done();
57     },
58     done.fail
59   );
60
61
62   expect(httpSpy.get).toHaveBeenCalledWith('http://localhost:4200/api/tasks', Object(
63     withCredentials: true )));
```

7.2. Testy manualne

Manualne testy były przeprowadzane na każdym poziomie aplikacji, jednak w najmniejszym stopniu w warstwie logiki biznesowej (ze względu na duże pokrycie testami automatycznymi). Warstwa bazy danych była testowana manualnie podczas pisania kodu produkcyjnego. Sprawdzano, czy dodawane encje są poprawne. Korzystano z dwóch metod takiego sprawdzenia. Pierwszym sposobem było pisanie zapytań do bazy i sprawdzanie, czy zwracane wartości spełniają oczekiwania. Drugim było wizualne sprawdzenie wewnątrz zintegrowanego środowiska programistycznego (*ang. Integrated Development Environment, IDE*). Użyto środowiska IntelliJ [89], które ma możliwość wizualnego wyświetlania całej bazy danych oraz wartości w poszczególnych jej tabelach.

Przeprowadzono również testy manualne warstwy prezentacji. Nie tworzone testów sprawdzających poprawność wyświetlania komponentów, z tego względu zdecydowano się na ręczne testy zarówno podczas tworzenia kodu, jak i po jego zakończeniu. Przykładowo sprawdzano poprawność operacji zaliczenia i odznaczenia zadania. Napisano testy automatyczne sprawdzające logikę biznesową, jednak aby się upewnić o poprawności wykonywania operacji, sprawdzono przebieg całości ręcznie. Wpierw umieszczone wiadomości logujące obiekty wysyłane z warstwy prezentacji. Następnie, za pomocą debugera (*ang. debugger*), sprawdzano, czy informacje są poprawnie przetwarzane w warstwie logiki biznesowej. W momencie zapisania do bazy danych zmodyfikowanych obiektów sprawdzano, za pomocą specjalnego narzędzia w IDE IntelliJ [89], czy obiekty w bazie danych spełniają założenia. Cały proces powtarzano zarówno dla operacji wykonania zadania, jak i odznaczenia wykonania zadania.

Aby zrealizować wymaganie niefunkcjonalne WNF 6, zdecydowano się na uruchomienie aplikacji w przeglądarkach Google Chrome [21], Mozilla Firefox [22] oraz Opera [23]. W każdej z nich CommOn działa bez zarzutu. Przeprowadzono również testy sprawdzające poprawność żądania na poziomie każdej warstwy. Za pomocą debugera oraz narzędzia podglądu bazy danych zapewnianego przez IntelliJ [89] monitorowano przebiegi poszczególnych żądań. Za pomocą stopera sprawdzano również czas odpowiedzi aplikacji. Mieści się on w zakresie wymagania WNF 3.

8. Przykład użycia

W niniejszym rozdziale opisano przykładowe użycie aplikacji. Skupiono się na aspekcie wizualnym i opisie interfejsu użytkownika. Scenariusz opisanego przykładu składa się z następujących kroków:

1. Uruchomienie aplikacji
2. Stworzenie pierwszego konta i domu
3. Stworzenie drugiego konta należącego do tego samego domu
4. Edycja zdjęcia profilowego
5. Stworzenie kilku pokoi
6. Stworzenie kilku zadań
7. Próba usunięcia zadania
8. Edycja zadania
9. Zaliczenie zadania
10. Odznaczenie zaliczenia zadania
11. Sprawdzenie wyników
12. Upływ czasu – dynamiczna zmiana punktów za zadania
13. Odejmowanie punktów użytkownikom
14. Usunięcie konta

Aby uruchomić aplikację, należy pobrać kod źródłowy z repozytoriów warstwy prezentacji (https://github.com/HasuNoHana/common_frontend) oraz warstwy logiki biznesowej (<https://github.com/HasuNoHana/common-backend>). Pościągnięciu, przy pomocy Gita [75], kodu obu warstw należy się zapoznać z instrukcjami obecnymi w plikach README. Wymagane jest zainstalowanie wymienionych w nich narzędzi, są to:

1. Ubuntu [52]
2. Docker [36]
3. Docker Compose [76]
4. Maven [30]
5. Node Package Manager (npm) [90]
6. Node.js [91]
7. Angular [4]

Wymagane wersje są podane w plikach README. Po upewnieniu się, że posiadane wersje są odpowiednie, należy uruchomić skrypty startujące znajdujące się w repozytoriach (warstwa danych – start_db.sh z repozytorium common-backend, warstwa logiki biznesowej – start.sh, warstwa prezentacji – start.sh). Zapewnią one zainstalowanie odpowiednich pakietów, ściągnięcie oraz konfigurację bazy danych i na

koniec uruchomienie aplikacji. Po poprawnym uruchomieniu aplikacji należy wejść pod adres URL <http://localhost:4200/>, który przekieruje użytkownika na stronę logowania.

The screenshot shows a registration form titled "Zapisz się". It consists of three input fields: "Nazwa użytkownika*" (Username*), "Hasło*" (Password*), and "Powtórz hasło*" (Repeat password*). Each field has a red "..." icon at its right end. Below the fields is a checkbox labeled "□ Mam już dom, nie chcę generować nowego kodu" (I already have a home, I don't want to generate a new code). At the bottom is a blue "Zapisz się" (Register) button.

Rysunek 8.1. Strona rejestracji

Jeśli użytkownik spróbuje wpisać ręcznie adres innej, zabezpieczonej podstrony, zostanie on przekierowany z powrotem na ekran logowania. Mechanizm przekierowania nieco dokładniej opisano w rozdziale 6.2.3. Aby móc stworzyć nowe konto, należy kliknąć „Zapisz się”. Użytkownik zostanie wtedy przeniesiony do strony rejestracji (rysunek 8.1).

Użytkownik wpisuje nazwę „Ala” oraz hasło „maKota”, jednak podczas powtarzania się pomylił. W związku z tym formularz nie jest poprawny i przy próbie zapisania wyświetli się komunikat o błędzie pokazany na rysunku 8.2. Użytkownik, przeczytawszy komunikat, rozumie gdzie jest błąd i poprawia powtórzone hasło. Po poprawieniu przycisk zapisania staje się możliwy do wcisnięcia. Użytkownik rejestruje się i zostaje przeniesiony z powrotem na stronę logowania. Po poprawnym wpisaniu nazwy użytkownika oraz hasła następuje przekierowanie do pustej listy zadań nowo powstałego domu. Jak widać na obrazku 8.3, wyświetlany jest na niej przycisk służący do tworzenia nowego zadania oraz dwie puste listy zawierające zadania do wykonania, oraz już wykonane zadania. Listy są zbudowane w identyczny sposób.

Testowanie dalszych funkcji systemu wymaga stworzenia drugiego konta użytkownika należącego do tego samego domu. W tym celu należy sprawdzić kod dostępu obecnego domu. Można to zrobić poprzez kliknięcie rozsuwanego przycisku „Więcej” znajdujący

8. Przykład użycia

The screenshot shows a registration form titled "Zapisz się". It includes fields for "Nazwa użytkownika*" (username) containing "Ala", "Hasło*" (password) containing "*****", and "Powtóż hasło*" (repeat password) containing "****". A red error message box at the bottom states: "Powtórzone hasło musi być takie samo jak hasło!". Below the form is a checkbox labeled "Mam już dom, nie chcę generować nowego kodu". At the bottom is a blue "Zapisz się" button.

Rysunek 8.2. Strona rejestracji z błędem niepoprawnego hasła

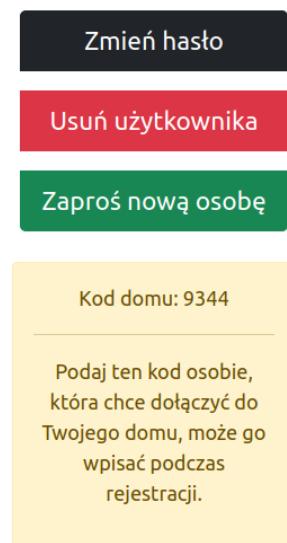
się w prawym górnym rogu ekranu i wybranie opcji „Profil”. Wewnątrz zakładki profil znajduje się przycisk służący do zapraszania nowych osób do domu. Użytkownik Ala kliką go, po czym przekazuje kod domu innemu użytkownikowi pragnącemu dołączyć do tego samego domu. Na rysunku 8.4 widać komunikat wyświetlający kod domu – 9344.

W dalszej kolejności następuje stworzenie nowego użytkownika, który zamiast tworzyć nowy dom, pragnie się przyłączyć do już istniejącego. Po wylogowaniu Ali (przycisk wylogowania znajduje się pod opcją „Profil” na rysunku 8.3) tworzony jest nowy użytkownik o nazwie „Jacek”, haśle „lubiAle” oraz kodzie domu „9344”. Wypełniony formularz pokazany jest na rysunku 8.5. Po zalogowaniu Jacek może sprawdzić, że faktycznie znajduje się w tym samym domu, co Ala, poprzez wejście w zakładkę „Wyniki” i obejrzenie listy mieszkańców. Na rysunku 8.6 zaprezentowano nazwy osób znajdujących się w tym samym domu, co obecnie zalogowany użytkownik. Zgodnie z oczekiwaniami, lista zawiera dwoje użytkowników: Alę i Jacka.

W celu zmiany zdjęcia profilowego użytkownik przemieszcza się do swojego profilu (tak jak na rysunku 8.3), po czym kliką w przycisk „Edytuj” znajdujący się bezpośrednio pod jego domyślnym zdjęciem profilowym. Aplikacja ukazuje formularz (zaprezentowany

The screenshot shows the CommOn application interface. At the top, there is a navigation bar with links: 'CommOn', 'Zadania', 'Pokoje', and 'Wyniki'. On the right side of the top bar are 'Więcej' (More) with a crown icon, 'Profil' (Profile), and 'Wyloguj' (Logout). Below the navigation bar is a green button labeled 'Stwórz nowe zadanie' (Create new task). The main area displays a section titled 'Zadania do wykonania' (Tasks to perform) with a note: 'Zaznacz zadanie, aby oznaczyć je jako wykonane' (Check the task to mark it as completed). Below this is a table header with columns: 'Nazwa' (Name), 'Cena początkowa' (Initial price), 'Obecna cena' (Current price), 'Pokój' (Room), 'Ostatnio wykonane' (Last performed), and 'Wykonane przez' (Performed by). The next section is titled 'Wykonane zadania' (Completed tasks) with a note: 'Odnacz wykonane przez siebie zadanie, jeśli się pomyliłeś' (Check the task you performed yourself if you made a mistake). It also has a similar table header.

Rysunek 8.3. Pusta lista zadań i zakładka „Profil”



Rysunek 8.4. Komunikat wyświetlający kod domu

na rysunku 8.7), w którym użytkownik może wpisać własny adres URL zdjęcia lub wybrać jeden z wyselekcjonowanych przez autorkę. Użytkownik wybiera zdjęcie kotwicy i zapisuje zmiany.

Po udanej zmianie zdjęcia użytkownik kliką zakładkę „Pokoje” znajdującą się w menu głównym, po czym tworzy nowe pomieszczenie. Formularz nowego pomieszczenia pokazany jest na rysunku 6.8. Użytkownik wpisał nazwę „Kuchnia” oraz wybrał jedno z proponowanych zdjęć. Jeśli użytkownik wpisałby niepoprawny składniowo adres URL, przycisk do zapisywania formularza pozostałby nieaktywny. Jest natomiast możliwość

8. Przykład użycia

The screenshot shows a registration form titled "Zapisz się". It includes fields for "Nazwa użytkownika*" (Jacek), "Hasło*" (*****), "Powtóż hasło*", a checked checkbox for "Mam już dom, nie chcę generować nowego kodu", a note about adding an existing home's code, and a "Kod domu" field containing "9344". A blue "Zapisz się" button is at the bottom.

Rysunek 8.5. Formularz rejestracji użytkownika z istniejącym domem

Mieszkaniec

Ala

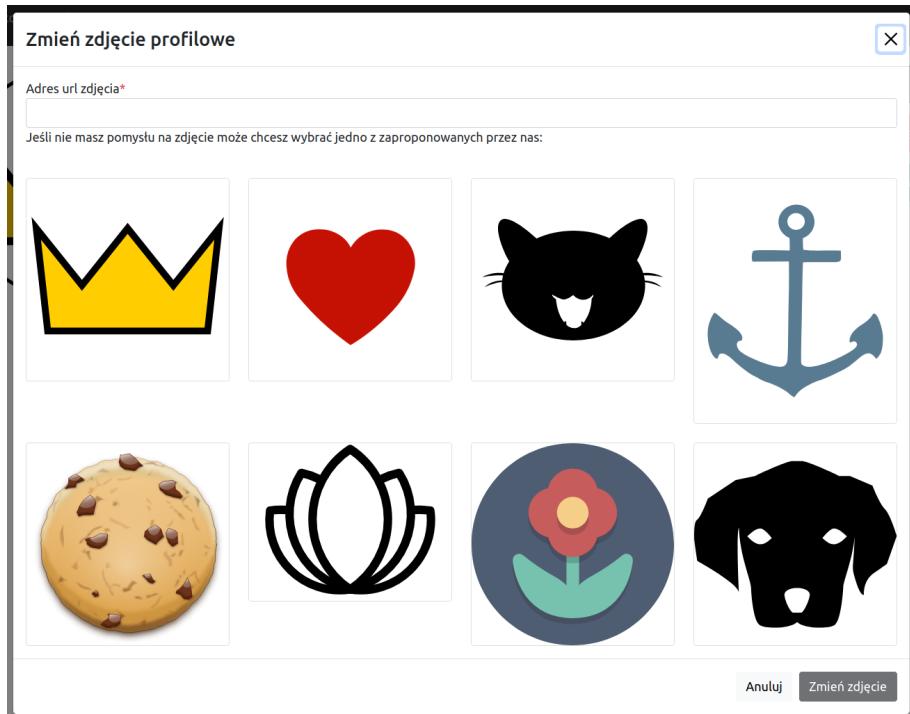
Jacek

Rysunek 8.6. Lista użytkowników wewnętrz obecnego domu

niewpisywania adresu. W tym przypadku zostanie ustawione domyślne zdjęcie. W podobny sposób użytkownik stworzył jeszcze dwa pomieszczenia – łazienkę i sypialnię.

W tym momencie lista pokoi w domu wygląda tak, jak na rysunku 8.8. Jak widać, zakładka „Pokoje” pokazuje listę pomieszczeń obecnie znajdujących się w domu oraz przycisk służący do tworzenia nowych pomieszczeń. Każdy pokój posiada zdjęcie oraz nazwę. Pod spodem znajduje się zdanie, które jest dynamicznie zmieniane w zależności od liczby zadań do wykonania w danym pokoju. Obecnie żaden pokój nie posiada zadań, jednak w kolejnym kroku się to zmieni. Każdy pokój posiada też zestaw przycisków służących do wyświetlania szczegółów pokoju, edycji oraz usuwania.

W następnej kolejności użytkownik wraca do listy zadań, aby stworzyć kilka z nich. Po kliknięciu przycisku odpowiedzialnego za dodanie nowego obowiązku ukazuje się



Rysunek 8.7. Formularz zmiany zdjęcia profilowego

Rysunek 8.8. Lista pokoi

formularz. Podczas wypełniania go użytkownik przez przypadek postawił minus przed ceną. Wymagane jest, aby nazwa zadania nie była pusta, zarówno cena początkowa, jak i okres były liczbami naturalnymi większymi od 0 oraz aby nazwa pokoju nie była pusta. Ta ostatnia jest możliwa do wybrania z listy pomieszczeń, które obecnie istnieją w domu. Z powyższych względów dane zawierające ujemną wartość ceny nie są poprawne, przez

8. Przykład użycia

co zostanie wyświetlony komunikat informujący o tym fakcie użytkownika oraz przycisk zapisujący pozostałe nieaktywne. Komunikat jest widoczny na rysunku 8.9. Użytkownik zidentyfikował problem, po czym uzupełnił formularz poprawnymi danymi widocznymi na rysunku 8.10.

The screenshot shows a modal dialog titled "Zmień istniejące zadanie". It contains fields for "Nazwa*" (Name) with the value "Zmyć naczynia", "Cena początkowa*" (Initial price) with the value "-10", and "Okres powtarzania zadania w dniach*" (Repeating period in days) with the value "7". Below the price field, a red error message box displays the text "Cena musi być większa od 0!". The "Pokój*" (Room) field has the value "Kuchnia". At the bottom are two buttons: "Anuluj" (Cancel) and a dark grey "zmień zadanie" (Change task) button.

Rysunek 8.9. Komunikat o błędzie w przypadku ujemnej ceny zadania

This screenshot shows the same modal dialog as in Figure 8.9, but with corrected data. The "Cena początkowa*" field now contains the value "10", which is no longer highlighted in red. The rest of the fields and layout are identical to the previous screenshot.

Rysunek 8.10. Poprawnie uzupełniony formularz zadania

W następnej kolejności użytkownik stworzył jeszcze kilka zadań: mycie naczyń, gotowanie obiadu, mycie lodówki, ścielenie łóżka. Wszystkie zadania oprócz ścielenia łóżka miały ustalony okres wykonania na tydzień. Jest to istotne w kontekście upływu czasu i zmiany cen zadania. Efekt działań użytkownika pokazano na rysunku 8.11. W każdym wierszu można znaleźć istotne informacje o konkretnym zadaniu, takie jak:

nazwa zadania, cena początkowa, cena obecna (obliczana zgodnie ze wzorem przedstawionym w rozdziale 6.2), pokój, w którym znajduje się to zadanie, data ostatniego wykonania oraz osoba, która wykonała to zadanie po raz ostatni. Po lewej stronie wiersza znajduje się przycisk służący do oznaczania zadania jako wykonane lub odwrotnie. Po prawej stronie wiersza znajduje się rozsuwane menu (przycisk „...”) zawierające możliwość edycji bądź usunięcia zadania, widoczne na rysunku 8.11.

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input type="checkbox"/> Zmyć naczynia	20	20	Kuchnia	2023-01-24	...
<input type="checkbox"/> Ugotować obiad	40	40	Kuchnia	2023-01-24	...
<input type="checkbox"/> Wymyć lodówkę	30	30	Kuchnia	2023-01-24	...
<input type="checkbox"/> Pościelić łóżko	10	10	Sypialnia	2023-01-24	...

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
Wykonane zadania Odznacz wykonane przez siebie zadanie, jeśli się pomyliłeś					Edytuj Usun

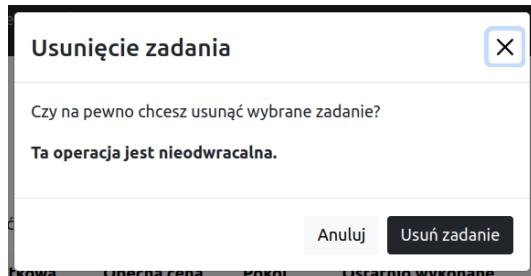
Rysunek 8.11. Lista zadań po uzupełnieniu przez użytkownika

Ze względu na to, że zadania zostały dopiero co stworzone, nie zawierają informacji o tym, kto je ostatnio wykonywał. Jego obecna cena jest równa cenie początkowej, a data ostatniego wykonania wskazuje dzień utworzenia zadania.

Jeśli użytkownik postanowi usunąć zadanie, pojawi się komunikat ostrzegający go, że ta operacja jest nieodwracalna (rysunek 8.12). Dopiero w przypadku, w którym użytkownik potwierdzi, zadanie zostanie usunięte. Jeśli użytkownik zamknie okno, wybierze opcję anulowania lub kliknie gdziekolwiek poza obręb wyświetlanego komunikatu, zadanie nie zostanie usunięte. Użytkownik postanowił zmienić cenę istniejącego zadania o nazwie „Zmyć naczynia” z 10 punktów na 20. W tej sytuacji wyświetlany jest ten sam formularz, co podczas tworzenia zadania, jedyną różnicą jest to, że pola od razu są wypełnione danymi zmienianego zadania (rysunek 8.10). Walidacja zmiany zadania jest taka sama jak tworzenia nowego zadania, dlatego nie można wprowadzić niepoprawnych danych.

W kolejnym kroku użytkownik zaznacza część zadań jako wykonane. Na rysunku 8.13 można obejrzeć konsekwencje jego działań. Jak widać, poprawnie zmodyfikowane zadanie „Zmyć naczynia” znajduje się obecnie w liście zadań zrobionych. Pojawiło się

8. Przykład użycia



Rysunek 8.12. Komunikat widoczny przy próbie usunięcia zadania

również imię użytkownika, który je wykonał. Podobnie sprawa ma się w kontekście zadania „Wymyć lodówkę”.

A screenshot of the application interface showing the "Zadania do wykonania" (Tasks to do) and "Wykonane zadania" (Completed tasks) sections.

Zadania do wykonania
Zaznacz zadanie, aby oznaczyć je jako wykonane

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input type="checkbox"/> Ugotować obiad	40	40	Kuchnia	2023-01-24	...
<input type="checkbox"/> Pościelić łóżko	10	10	Sypialnia	2023-01-24	...

Wykonane zadania
Oznacz wykonane przez siebie zadanie, jeśli się pomyliłeś

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input checked="" type="checkbox"/> Zmyć naczynia	20	20	Kuchnia	2023-01-24	Jacek
<input checked="" type="checkbox"/> Wymyć lodówkę	30	30	Kuchnia	2023-01-24	Jacek

Rysunek 8.13. Lista zadań po edycji oraz wykonaniu części z nich

Kolejną czynnością wykonaną przez użytkownika było odznaczenie wykonania zadania „Zmyć naczynia”. Na rysunku 8.14 można zobaczyć, jak aplikacja zareagowała. Jak widać, zadanie „Zmyć naczynia” powróciło do zadań do zrobienia, a oznaczenie użytkownika, który ostatnio je wykonał, znikło.

Po wykonaniu dwóch zadań użytkownik kliką zakładkę „Wyniki”, aby sprawdzić, ile punktów uzyskał. Rysunek 8.15 ukazuje tę sytuację. Zakładka udostępnia informacje o punktach zdobytych zarówno przez zalogowanego użytkownika, jak i przez pozostałych mieszkańców domu. Znajduje się tu również zdjęcie profilowe, wiadomość powitalna, liczba zadań wykonanych w danym tygodniu przez użytkownika oraz pasek postępu

8. Przykład użycia

CommOn Zadania Pokoje Wyniki Więcej ▾

[Stwórz nowe zadanie](#)

Zadania do wykonania
Zaznacz zadanie, aby oznaczyć je jako wykonane

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input type="checkbox"/> Zmyć naczynia	20	20	Kuchnia	2023-01-24	...
<input type="checkbox"/> Ugotować obiad	40	40	Kuchnia	2023-01-24	...
<input type="checkbox"/> Pościelić łóżko	10	10	Sypialnia	2023-01-24	...

Wykonane zadania
Oznacz wykonane przez siebie zadanie, jeśli się pomyliłeś

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input checked="" type="checkbox"/> Wymyć lodówkę	30	30	Kuchnia	2023-01-24	Jacek

Rysunek 8.14. Lista zadań po odznaczeniu zadania

obrazujący, jak dużo jeszcze brakuje do spełnienia celu, jakim jest 100 punktów w danym tygodniu. Wartość liczby punktów wymaganych do zdobycia w ciągu tygodnia może być ustalona niezależnie dla każdego użytkownika. Na rysunku 8.15 widać, że użytkownik Jacek ma zdjęcie profilowe przedstawiające kotwicę i wykonał jedno zadanie warte łącznie 30 punktów. Uzyskał już 30% punktów potrzebnych w tym tygodniu. Użytkownik Ala nie zyskał jeszcze żadnych punktów, co jest widoczne na liście mieszkańców na rysunku 8.15.

CommOn Zadania Pokoje Wyniki Więcej ▾



Cześć Jacek!
Zadania wykonane w tym tygodniu: 1
Liczba punktów: 30/100

30%

Punkty zebrane przez mieszkańców

Mieszkaniec	Punkty zebrane	Punkty wymagane
Ala	0	100
Jacek	30	100

Rysunek 8.15. Zakładka „Wyniki”

Po upłynięciu trzech dni Jacek ponownie loguje się do aplikacji. Lista zadań znajduje

8. Przykład użycia

się na rysunku 8.16. Jak widać zmieniła się tylko wartość zadania „Pościelić łóżko” ze względu na to, że okres jego wykonywania wynosi jeden dzień, a pozostałych zadań 7 dni. Po trzech dniach cena zadania powinna zostać dwukrotnie pomnożona przez 1,5. zgodnie ze wzorem przedstawionym w rozdziale 6.2, co powinno dać w wyniku wartość 22,5. Ze względu na to, że liczba punktów powinna być całkowita, za każdym razem wynik jest zaokrąglany w dół. Jak widać na rysunku 8.16, cena zadania wynosi 22 punkty, czyli zgodnie z oczekiwaniami.

The screenshot shows the CommOn application interface. At the top, there is a navigation bar with links: ComOn, Zadania, Pokoje, Wyniki, and a 'Więcej' button with a dropdown arrow. Below the navigation bar, there is a green button labeled 'Stwórz nowe zadanie'. The main content area is divided into two sections: 'Zadania do wykonania' and 'Wykonane zadania'.
Zadania do wykonania:
A heading 'Zadania do wykonania' followed by a sub-instruction 'Zaznacz zadanie, aby oznaczyć je jako wykonane'. A table lists three tasks:

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input type="checkbox"/> Zmyć naczynia	20	20	Kuchnia	2023-01-24	...
<input type="checkbox"/> Ugotować obiad	40	40	Kuchnia	2023-01-24	...
<input type="checkbox"/> Pościelić łóżko	10	22	Sypialnia	2023-01-24	...

Wykonane zadania:
A heading 'Wykonane zadania' followed by a sub-instruction 'Oznacz wykonane przez siebie zadanie, jeśli się pomyliłeś'. A table lists one task:

Nazwa	Cena początkowa	Obecna cena	Pokój	Ostatnio wykonane	Wykonane przez
<input checked="" type="checkbox"/> Wymyć lodówkę	30	30	Kuchnia	2023-01-24	...

Rysunek 8.16. Lista zadań po upływie trzech dni

Po upływie tygodnia użytkownikom są odejmowane punkty. Zakładając, że żaden z użytkowników nie wykonał więcej zadań oraz że Ala zmieniła swoje punkty wymagane z 80 na 100, po tygodniu zawartość zakładki „Wyniki” wygląda jak na rysunku 8.17. Jak widać, każdemu z użytkowników zostały odjęte jego punkty wymagane. Ala posiada -80 punktów, a Jacek -70. W momencie, w którym użytkownik posiada ujemną liczbę punktów, pasek postępu staje się czerwony i pokazuje, jak wielu punktów brakuje do wyrównania.

Jacek postanowił usunąć swoje konto, przechodzi więc do swojego profilu (rysunek 8.18), po czym kliką przycisk „Usuń użytkownika”. Wyświetlone zostaje okno podobne tego na rysunku 8.12 z tą różnicą, że pytanie o potwierdzenie dotyczy użytkownika, a nie zadania. Jacek potwierdza chęć usunięcia użytkownika. Aplikacja usuwa użytkownika, po czym następuje przekierowanie na stronę logowania.



Punkty zebrane przez mieszkańców

Mieszkaniec	Punkty zebrane	Punkty wymagane	...
Ala	-80	80	...
Jacek	-70	100	...

Rysunek 8.17. Zakładka „Wyniki” po upływie tygodnia



Rysunek 8.18. Zakładka „Profil”

9. Podsumowanie

Nierówny podział pracy w gospodarstwach domowych, zachęcenie dzieci do nauki wykonywania obowiązków oraz wyrównanie liczby zadań realizowanych przez wszystkich członków rodzin z dorosłymi dziećmi to istotne kwestie związane ze sprawiedliwym podziałem obowiązków w domu. Duża część z tych problemów była doświadczeniem samej autorki, dlatego doskonale rozumie ona ich powagę. W celu rozwiązania nawiązanych problemów w niniejszej pracy stworzono aplikację internetową CommOn pomagającą w zarządzaniu obowiązkami domowymi.

W rozdziale 2 dokładniej omówiono problemy, jakie aplikacja stara się rozwiązać, przedstawiono również podstawy psychologiczne, na których potem była ona budowana. W ramach przeglądu dziedziny (rozdział 3) przeanalizowano rynek, aby upewnić się, że tworzenie nowej aplikacji ma sens. Z przeprowadzonego przeglądu wynika, że na rynku jest niska, którą CommOn z powodzeniem mogłoby wypełnić. Aby aplikacja sprostała postawionym oczekiwaniom, w rozdziale 4 zdefiniowano zarówno wymagania funkcjonalne, jak i niefunkcjonalne oraz zaprezentowano diagramy przypadków użycia. Technologie użyte podczas pisania aplikacji, których wybór można prześledzić w rozdziale 5, są sprawdzone przez wielu programistów. Są to między innymi TypeScript [2] i Angular [4], użyte do napisania warstwy prezentacji, Java [1] i Spring [3], zastosowane podczas rozwoju warstwy logiki biznesowej oraz baza danych MySQL [5] w roli warstwy danych. W rozdziale 6, poświęconym projektowi i implementacji, znaleźć można szczegółowe informacje zarówno o ogólnej architekturze aplikacji, podjętych decyzjach czy zastosowanych wzorcach projektowych, jak również szczegółowy opis działania obsługiwanych zadań i implementacji konkretnych klas wraz z przykładami. Informacje na temat testów przeprowadzanych na aplikacji w celu upewnienia się, że działa poprawnie, są opisane w rozdziale 7. Dzięki nim stwierdzono, że CommOn spełnia wszystkie postawione jej wymagania. Na koniec, w rozdziale 8, omówiono przykład użycia wraz z zaprezentowaniem graficznego interfejsu użytkownika.

Aplikacja CommOn posiada wiele zalet jak, chociażby możliwy dostęp przez komputer, pozwalający na uczestnictwo całej rodziny, także dzieci i osób starszych. Intuicyjny i estetyczny interfejs sprawia, że użytkownik chce w aplikacji spędzać czas i nie zostaje zniechęcony przez toporność systemu obsługi. Dynamiczna zmiana cen zadań wprowadza zasady wolnego rynku, które działają na korzyść zarówno mieszkańców – którzy wykonują zadania, jakie naprawdę lubią – jak i domu – zadania niewykonywane przez dłuższy okres mają wyższy priorytet. Wyświetlanie wyników dodatkowo motywuje użytkowników do dalszej pracy i w klarowny sposób pokazuje, ile jeszcze pracy zostało do zrobienia. Pomimo że aplikacja CommOn spełnia postawione przed nią wymagania, posiada również wady. Można by poprawić mechanizmy bezpieczeństwa, takie jak np.

wprowadzenie komunikacji za pomocą protokołu HTTPS, który obecnie nie jest wykorzystywany z powodu braku odpowiedniego certyfikatu. Kolejnym minusem jest brak automatycznych testów integracyjnych, które pozwoliłyby na szybszy i mniej podatny na błędy rozwój aplikacji.

W ramach analizowania perspektyw na przyszłość zidentyfikowano obszary, w których można by rozwinąć CommOn. Przede wszystkim, aby stworzyć rozwiązanie kompleksowe, następnym krokiem mogłoby być stworzenie mobilnego odpowiednika aplikacji CommOn, aby mogła ona rywalizować ze wszystkimi aplikacjami poświęconymi tematyce zarządzania obowiązkami na rynku. Bardzo interesujące mogłoby się okazać wprowadzenie, innych niż wykładowiczy, algorytmów zmiany ceny zadania, zbadanie ich wpływu na zaangażowanie użytkownika oraz pozwolenie na wybór algorytmu w każdym domu. Oczywiście można również dodać nowe funkcjonalności, takie jak np. podpowiadanie zadań, które użytkownik może chcieć stworzyć. Aby zwiększyć poziom zaangażowania użytkowników, można by wprowadzić mechanizm grywalizacji związany z ciągłym postępuem, np. zdobywanie poziomów.

Bibliografia

- [1] Oracle, *Java*, Dostęp zdalny (22.07.2022): <https://www.java.com/en/>, 2022.
- [2] Microsoft, *TypeScript*, Dostęp zdalny (19.07.2022): <https://www.typescriptlang.org/>, 2022.
- [3] VMware, *Spring*, Dostęp zdalny (22.11.2022): <https://spring.io/>, 2022.
- [4] Google, *Angular*, Dostęp zdalny (21.07.2022): <https://angular.io/>, 2022.
- [5] Oracle, *MySQL*, Dostęp zdalny (19.07.2022): <https://dev.mysql.com/>, 2022.
- [6] Vaillant George E., McArthur Charles C., Bock Arlie, *Grant Study of Adult Development, 1938-2000*, Dostęp zdalny (29.11.2022): <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/48WRX9>, 2022.
- [7] Anna Marianowska, "Uwarunkowania odraczania dorosłości w okresie transformacji ustrojowej w Polsce", w *Przestrzenie i miejsca edukacji dorosłych w Polsce*, Wydawnictwo FRSE, 2018, s. 94–115.
- [8] Oxford University Dictionary, *Gamification*, Dostęp zdalny (29.11.2022): <https://www.oxfordlearnersdictionaries.com/definition/english/gamification?q=gamification>, 2022.
- [9] Google, *How Google Search Works?*, Dostęp zdalny (10.01.2023): <https://www.google.com/search/howsearchworks/>, 2023.
- [10] ——, *Android Apps for Google Play*, Dostęp zdalny (10.01.2023): <https://play.google.com/store/games>, 2023.
- [11] LoopLoop Apps, *Tody*, Dostęp zdalny (1.02.2022): <https://play.google.com/store/apps/details?id=com.looploop.tody&hl=pl&gl=US>, 2022.
- [12] Woohoo Software, *Clean My House*, Dostęp zdalny (1.02.2022): <https://play.google.com/store/apps/details?id=com.woohoosoftware.cleanmyhouse&hl=pl&gl=US>, 2022.
- [13] M.M Productions, *Cleaning Checklist*, Dostęp zdalny (1.02.2022): <https://play.google.com/store/apps/details?id=com.mmproductions.cleaningchecklist&hl=pl&gl=US>, 2022.
- [14] 23apps.com, *Chores App*, Dostęp zdalny (1.02.2022): <https://play.google.com/store/apps/details?id=com.apps23.household&hl=pl&gl=US>, 2022.
- [15] AppMates UG, *flatify*, Dostęp zdalny (1.02.2022): <https://play.google.com/store/apps/details?id=flatify.com.flatify&hl=pl&gl=US>, 2022.
- [16] OurHome, *OurHome*, Dostęp zdalny (1.02.2022): <https://play.google.com/store/apps/details?id=com.getfairshare.ourhome&hl=pl&gl=US>, 2022.
- [17] OurHome, Dostęp zdalny (9.01.2023): <https://app.ourhomeapp.com/#tasks>, 2023.
- [18] Lindsey Parker, *Chore Apps: The best 5 house chore apps of 2021 reviewed*, Dostęp zdalny (10.01.2023): <https://thetoday.app/blog/chore-apps-the-best-5-house-chore-apps-of-2021-reviewed/>, 2023.

9. Bibliografia

- [19] LoopLoop Aps, *Tody*, Dostęp zdalny (8.01.2023): <https://choreapp.wingboat.com/>, 2023.
- [20] James Clear, *Atomic Habits*. Wielka Brytania: ydawnictwo Penguin Random House, 2018.
- [21] Google, *Google Chrome*, Dostęp zdalny (23.01.2023): <https://www.google.com/chrome/>, 2023.
- [22] Mozilla Corporation, *Mozilla Firefox*, Dostęp zdalny (23.01.2023): <https://www.mozilla.org/en-US/firefox/new/>, 2023.
- [23] Opera Norway, *Opera*, Dostęp zdalny (29.01.2023): <https://www.opera.com/pl>, 2023.
- [24] Bootstrap team, *Bootstrap*, Dostęp zdalny (22.07.2022): <https://getbootstrap.com/>, 2022.
- [25] Friedel Ziegelmayer and other contributors, *Karma*, Dostęp zdalny (01.08.2022): <https://karma-runner.github.io/latest/index.html>, 2022.
- [26] Pivotal Labs, *Jasmine*, Dostęp zdalny (01.08.2022): <https://jasmine.github.io/>, 2022.
- [27] VMware, *Spring Boot*, Dostęp zdalny (22.11.2022): <https://spring.io/projects/spring-boot>, 2022.
- [28] Red Hat, *Hibernate*, Dostęp zdalny (22.07.2022): <https://hibernate.org/>, 2022.
- [29] The Project Lombok Authors, *Project Lombok*, Dostęp zdalny (25.07.2022): <https://projectlombok.org/>, 2022.
- [30] The Apache Software Foundation, *Maven*, Dostęp zdalny (25.07.2022): <https://maven.apache.org/>, 2022.
- [31] Kent Beck, Erich Gamma, David Saff and Kris Vasudevan, *JUnit*, Dostęp zdalny (27.07.2022): <https://junit.org/junit5/>, 2022.
- [32] Marc R. Hoffmann, Evgeny Mandrikov and Mirko Friedenhagen, *Jacoco*, Dostęp zdalny (25.07.2022): <https://www.jacoco.org/jacoco/trunk/doc/>, 2022.
- [33] GitHub, *GitHub*, Dostęp zdalny (21.07.2022): <https://github.com/>, 2021.
- [34] —, *GitHub Actions*, Dostęp zdalny (05.08.2022): <https://github.com/features/actions>, 2022.
- [35] Sonar Source, *Sonar Cloud*, Dostęp zdalny (22.08.2022): <https://sonarcloud.io/>, 2022.
- [36] Docker, *Docker*, Dostęp zdalny (02.08.2022): <https://www.docker.com/>, 2022.
- [37] Netscape, *Javascript Standard*, Dostęp zdalny (19.07.2022): <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>, 2022.
- [38] Nikita Kozlov, *Static vs Dynamic typing*, Dostęp zdalny (21.07.2022): <https://dev.to/kozlovzxc/js-interview-in-2-minutes-static-vs-dynamic-typing-2d5k>, 2021.
- [39] Facebook, *React*, Dostęp zdalny (21.07.2022): <https://reactjs.org/>, 2022.
- [40] Evan You, *Vue*, Dostęp zdalny (21.07.2022): <https://vuejs.org/>, 2022.

- [41] Shaumik Daityari, *Angular vs React vs Vue*, Dostęp zdalny (21.07.2022): <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/#license>, 2022.
- [42] Hiren Dhaduk, *Best Frontend Frameworks for Web Development in 2022*, Dostęp zdalny (21.07.2022): <https://www.simform.com/blog/best-frontend-frameworks/>, 2021.
- [43] Google, *Angular Material*, Dostęp zdalny (21.07.2022): <https://material.angular.io/>, 2022.
- [44] SmartBear Software, *Behaviour-driven Development*, Dostęp zdalny (28.11.2022): <https://cucumber.io/docs/bdd/>, 2019.
- [45] BrowserStack, *Test-driven Development*, Dostęp zdalny (28.11.2022): <https://www.browserstack.com/guide/what-is-test-driven-development>, 2022.
- [46] Asim Hussain, *Jasmine & Karma*, Dostęp zdalny (01.08.2022): <https://codecraft.tv/courses/angular/unit-testing/jasmine-and-karma/>, 2022.
- [47] Frank Moraes, *HTML*, Dostęp zdalny (28.11.2022): <https://html.com/>, 2022.
- [48] W3Schools, *CSS*, Dostęp zdalny (28.11.2022): <https://www.w3schools.com/css/>, 2022.
- [49] Sean Grogg, *Why do we need a back-end in web development? Can't the front-end directly send requests to the database?*, Dostęp zdalny (22.07.2022): <https://www.quora.com/Why-do-we-need-a-back-end-in-web-development-Cant-the-front-end-directly-send-requests-to-the-database>, 2022.
- [50] Python Software Foundation, *Python*, Dostęp zdalny (22.07.2022): <https://www.python.org/>, 2022.
- [51] Microsoft, *C*, Dostęp zdalny (22.07.2022): <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>, 2022.
- [52] Canonical, *Ubuntu*, Dostęp zdalny (22.07.2022): <https://ubuntu.com/>, 2022.
- [53] Microsoft, *Windows*, Dostęp zdalny (22.07.2022): <https://www.microsoft.com/pl-pl/>, 2022.
- [54] Baeldung, *Why choose Spring as your Java Framework*, Dostęp zdalny (25.08.2022): <https://www.baeldung.com/spring-why-to-choose>, 2022.
- [55] Vaskaran Sarcar, *Java Design Patterns, A Hands-On Experience with Real-World Examples*. New York, 233 Spring Street: Springer Science+Business Media, 2018.
- [56] JRebel, *Top 4 Java Web Frameworks Revealed*, Dostęp zdalny (22.07.2022): <https://www.jrebel.com/blog/java-web-framework-usage-stats>, 2015.
- [57] Oracle, *JDBC*, Dostęp zdalny (28.11.2022): <https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>, 2022.
- [58] VMware, *Spring Security*, Dostęp zdalny (03.12.2022): <https://docs.spring.io/spring-security/reference/index.html>, 2022.
- [59] Gradle, *Gradle*, Dostęp zdalny (25.07.2022): <https://gradle.org/>, 2022.
- [60] JRebel, *Ant vs Maven vs Gradle: Java Build Tools Comparison*, Dostęp zdalny (25.07.2022): <https://www.jrebel.com/blog/java-build-tools-comparison>, 2021.

9. Bibliografia

- [61] The Apache Software Foundation, *Maven - Build Life Cycle*, Dostęp zdalny (25.07.2022):
https://www.tutorialspoint.com/maven/maven_build_life_cycle.htm, 2022.
- [62] TutorialsPoint, *Hibernate - ORM Overview*, Dostęp zdalny (25.07.2022): https://www.tutorialspoint.com/hibernate/orm_overview.htm, 2022.
- [63] VMware, *MockMvc*, Dostęp zdalny (20.12.2022): <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/web/servlet/MockMvc.html>, 2022.
- [64] Szczepan Faber and friends, *Mockito*, Dostęp zdalny (25.11.2022): <https://site.mockito.org/>, 2022.
- [65] Oracle, *Getting Started with MySQL*, Dostęp zdalny (19.07.2022): <https://dev.mysql.com/doc/mysql-getting-started/en/>, 2022.
- [66] The PostgreSQL Global Development Group, *PostgreSQL*, Dostęp zdalny (20.07.2022): <https://www.postgresql.org/>, 2022.
- [67] SQLite Consortium, *SQLite*, Dostęp zdalny (20.07.2022): <https://www.sqlite.org/index.html>, 2022.
- [68] O'Reilly, *How MySQL Compares to Other Databases*, Dostęp zdalny (20.07.2022):
<https://www.oreilly.com/library/view/mysql-reference-manual/0596002653/ch01s09.html>, 2022.
- [69] Ormuco, *Most Popular Databases in 2020*, Dostęp zdalny (20.07.2022): <https://ormuco.com/blog/most-popular-databases>, 2022.
- [70] Mike Wolfe, *MySQL vs. SQLite*, Dostęp zdalny (20.07.2022):
<https://towardsdatascience.com/mysql-vs-sqlite-ba40997d88c5>, 2021.
- [71] Read Hat, *What is a CI/CD pipeline?*, Dostęp zdalny (05.08.2022): <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>, 2022.
- [72] GitHub, *GitHub Container Registry*, Dostęp zdalny (22.08.2022): <https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry>, 2022.
- [73] Sonar Source, *Sonar Source*, Dostęp zdalny (22.08.2022):
<https://www.sonarsource.com/>, 2022.
- [74] —, *Sonar Lint*, Dostęp zdalny (22.08.2022): <https://www.sonarlint.org/>, 2022.
- [75] Junio Hamano i inni, *Git*, Dostęp zdalny (02.08.2022): <https://git-scm.com/>, 2022.
- [76] Docker, *Docker Compose*, Dostęp zdalny (02.08.2022): <https://docs.docker.com/compose/>, 2022.
- [77] —, *Docker Hub Container Image Library*, Dostęp zdalny (22.08.2022): <https://www.docker.com/products/docker-hub/>, 2022.
- [78] Robert Cecil Martin, *Clean Code*. Pearson Education, 2008.
- [79] Laurentiu Spilca, *Spring Security In Action*. Manning Publications, 2020.

- [80] Oracle, *Java Docs*, Dostęp zdalny (23.01.2023):
<https://docs.oracle.com/en/java/>, 2023.
- [81] VMware, *Spring Docs*, Dostęp zdalny (23.01.2023):
<https://docs.spring.io/spring-boot/docs/current/api/index.html>, 2023.
- [82] Inji Wijegunaratne and George Fernandez, *Distributed Applications Engineering*. Springer, 1998.
- [83] Amiya Ranjan Rout, *Difference Between @Component, @Repository, @Service, and @Controller Annotations in Spring*, Dostęp zdalny (24.01.2023):
<https://www.geeksforgeeks.org/difference-between-component-repository-service-and-controller-annotations-in-spring/>, 2023.
- [84] Michael Herman, *What is Test-Driven Development?*, Dostęp zdalny (06.01.2023):
<https://testdriven.io/test-driven-development/>, 2023.
- [85] JUnit, *Junit Docs*, Dostęp zdalny (23.01.2023):
<https://junit.org/junit5/docs/current/api/>, 2023.
- [86] The Project Lombok Authors, *Lombok Docs*, Dostęp zdalny (23.01.2023):
<https://projectlombok.org/features/>, 2023.
- [87] Daniel Terhorst-North i Chris Matts, *Given-When-Then*, Dostęp zdalny (07.01.2023):
<https://martinfowler.com/bliki/GivenWhenThen.html#footnote-ivan>, 2023.
- [88] Angular, *Angular Docs*, Dostęp zdalny (23.01.2023):
<https://angular.io/docs>, 2023.
- [89] Jet Brains, *IntelliJ*, Dostęp zdalny (06.01.2023):
<https://www.jetbrains.com/idea/>, 2023.
- [90] npm, Inc., *npm*, Dostęp zdalny (28.01.2023):
<https://www.npmjs.com/>, 2023.
- [91] Node.js Collaborators, *Node.js*, Dostęp zdalny (28.01.2023):
<https://nodejs.org/en/>, 2023.

Wykaz symboli i skrótów

MVC – (ang. *Model-View-Controller*)

IDE – (ang. *Integrated Development Environment*)

BDD – (ang. *behavior - driven development*)

TDD – (ang. *Test-driven development*)

GWT – (ang. *Given-When-Then*)

DTO – (ang. *Data Transfer Object*)

HTTP – (ang. *Hypertext Transfer Protocol*)

SQL – (ang. *Structured Query Language*)

REST – (ang. *Representational state transfer*)

HTML – (ang. *HyperText Markup Language*)

CSS – (ang. *Cascading Style Sheets*)

JSON – (ang. *JavaScript Object Notation*)

URL – (ang. *Uniform Resource Locator*)

Spis rysunków

4.1 Przypadki użycia związane z kontem użytkownika	21
4.2 Przypadki użycia związane z zadaniami	21
4.3 Przypadki użycia związane z pokojami	22
4.4 Przypadki użycia związane z punktami	23
4.5 Przypadki użycia związane z czasem	24
6.1 Realizacja architektury trójwarstwowej w CommOn	34
6.2 Diagram ER bazy danych	35
6.3 Diagram wybranych klas z pakietu tasks	48
6.4 Diagram relacyjny bazy danych	52
6.5 Formularz strony logowania	55
6.6 Formularz strony rejestracji ze wszystkimi możliwymi błędami	56
6.7 Formularz zadania ze wszystkimi możliwymi błędami	57
6.8 Formularz pokoju	58
8.1 Strona rejestracji	73
8.2 Strona rejestracji z błędem niepoprawnego hasła	74
8.3 Pusta lista zadań i zakładka „Profil”	75
8.4 Komunikat wyświetlający kod domu	75
8.5 Formularz rejestracji użytkownika z istniejącym domem	76
8.6 Lista użytkowników wewnętrz obecnego domu	76
8.7 Formularz zmiany zdjęcia profilowego	77
8.8 Lista pokoi	77
8.9 Komunikat o błędzie w przypadku ujemnej ceny zadania	78
8.10 Poprawnie uzupełniony formularz zadania	78
8.11 Lista zadań po uzupełnieniu przez użytkownika	79
8.12 Komunikat widoczny przy próbie usunięcia zadania	80
8.13 Lista zadań po edycji oraz wykonaniu części z nich	80
8.14 Lista zadań po odznaczeniu zadania	81
8.15 Zakładka „Wyniki”	81
8.16 Lista zadań po upływie trzech dni	82
8.17 Zakładka „Wyniki” po upływie tygodnia	83
8.18 Zakładka „Profil”	83

Spis tabel

3.1 Definicje pojęć stosowanych w aplikacji	14
3.2 Porównanie funkcji oferowanych przez konkurencyjne aplikacje	18
4.1 Wymagania funkcjonalne	19
4.2 Wymagania niefunkcjonalne	24
6.1 Pakiet klas tasks w warstwie logiki biznesowej	36
6.2 Pakiet klas updateAlgorithms w warstwie logiki biznesowej	37
6.3 Pakiet klas rooms w warstwie logiki biznesowej	37
6.4 Pakiet klas users w warstwie logiki biznesowej	38
6.5 Pakiet klas house_buddy w warstwie logiki biznesowej	38
6.6 Pakiet klas autorization w warstwie logiki biznesowej	39
6.7 Pakiet klas images w warstwie logiki biznesowej	39
6.8 Pakiet klas houses w warstwie logiki biznesowej	40
6.9 Pakiet klas security w warstwie logiki biznesowej	40
6.10 Pakiet klas tasks w warstwie prezentacji	42
6.11 Pakiet klas rooms w warstwie prezentacji	42
6.12 Pakiet klas profile w warstwie prezentacji	43
6.13 Pakiet klas houses w warstwie prezentacji	44
6.14 Pakiet klas authentication w warstwie prezentacji	45
6.15 Pakiet klas shared w warstwie prezentacji	45
6.16 Pakiet klas header w warstwie prezentacji	46