

WYŻSZA SZKOŁA TECHNOLOGII
INFORMATYCZNYCH W
KATOWICACH

WYDZIAŁ INFORMATYKI
KIERUNEK: INFORMATYKA

NOWAK MARCIN
NR ALBUMU 08255
STUDIA NIESTACJONARNE

**Projekt i implementacja
aplikacji wspomagającej
zarządzanie budżetem
domowym**

PRZEDMIOT: PROJEKT SYSTEMU INFORMATYCZNEGO
POD KIERUNKIEM
MGR. JACEK ŻYWICZOK
W ROKU AKADEMICKIM 2022/23

Katowice 2022



Spis treści

1	<i>Wstęp</i>	4
2	<i>Charakterystyka i analiza problemu</i>	6
3	<i>Analiza istniejących rozwiązań</i>	11
3.1	<i>Budżet papierowy lub arkusz kalkulacyjny</i>	12
3.2	<i>Systemy bankowe - Przegląd wydatków w Santander</i>	15
3.3	<i>Aplikacje dedykowane - Intuit Mint</i>	17
4	<i>Koncepcja własnego rozwiązania</i>	21
4.1	<i>Koncepcja rozwiązania użytkowego</i>	21
4.2	<i>Koncepcja rozwiązania technologicznego</i>	22
5	<i>Projekt ogólny</i>	25
5.1	<i>Specyfikacja wymagań funkcjonalnych i nie-funkcjonalnych</i>	25
5.2	<i>Architektura systemu</i>	27
5.3	<i>Metody i narzędzia realizacji</i>	27
5.4	<i>Koncepcja przechowywania danych</i>	29
5.5	<i>Projekt interfejsu użytkownika</i>	31
6	<i>Dokumentacja techniczna</i>	36
6.1	<i>Schemat rzeczywistej struktury systemu</i>	36
6.2	<i>Wybrane fragmenty kodu aplikacji</i>	41
6.3	<i>Wybrane fragmenty kodu bazy danych</i>	48
7	<i>Testy i weryfikacja systemu</i>	53
7.1	<i>Najciekawsze wykryte błędy</i>	55
8	<i>Przykładowy scenariusz wykorzystania systemu</i>	60
9	<i>Zakończenie</i>	61

Rozdział 1

Wstęp

Finanse są dziedziną nauki ekonomicznej która zajmuje się rozporządzaniem pieniędzmi [6]. Nauka ta w podobnym zakresie a różnej skali dotyczy państw, dużych przedsiębiorstw, małych działalności gospodarczych jak i zwykłych obywateli - w efekcie jest to dziedzina o stosunkowo prostych podstawach jednak niesamowicie skomplikowana w każdym aspekcie w którym można ją zagłębić. Wiedza z tego zakresu staje się szczególnie przydatna w momencie dynamicznych zmian sytuacji ekonomicznej, wtedy nierazko decyduje ona o jakości oraz stanie życia poszczególnych osób fizycznych, rentowności przedsiębiorstw czy stabilności państw [7]. W przypadku państw i firm przeważnie do zarządzania budżetem oddelegowane są dedykowane całe zespoły lub dedykowani eksperci z tej dziedziny. Jednak osoby zarządzające budżetem domowym najczęściej dysponują wyłącznie nabitym doświadczeniem i na ogół stosują podejście intuicyjne, rzadko jeśli wogóle wspomagając się jakimkolwiek narzędziami które ułatwiałyby to zadanie. Część z nich może poszukiwać pożytecznych treści o tematyce finansowej w Internecie, jednakże rozpoczynając zaznajamianie się z tematyką mogą mieć spore trudność ich przystępnością oraz wyłuskaniem źródeł dobrej jakości informacji w natłoku materiałów błędnych, słabych merytorycznie, nieaktualnych czy też nastawionych na marketing ponad poprawność.

Celem pracy jest zaprojektowanie i realizacja modułu analitycznego aplikacji która ułatwi jej użytkownikom zarządzanie budżetem domowym poprzez dostarczenie narzędzia do analizy wpływów i wydatków, wizualizacji trendów oraz automatycznie kategoryzujące wprowadzone dane. W zamierzeniu aby ułatwić obsługę wymagać będzie minimalnej wiedzy i konfiguracji ze strony użytkownika, dostarczając mu jednocześnie możliwie najlepsze narzędzia. Będzie to aplikacja przeglądarkowa napisana w języku Python [49][5], wykorzystująca frameworki: Bootstrap [61], Flask [63], WTForms [65], jinja2 [66], oraz bibliotekę chart.js [67].

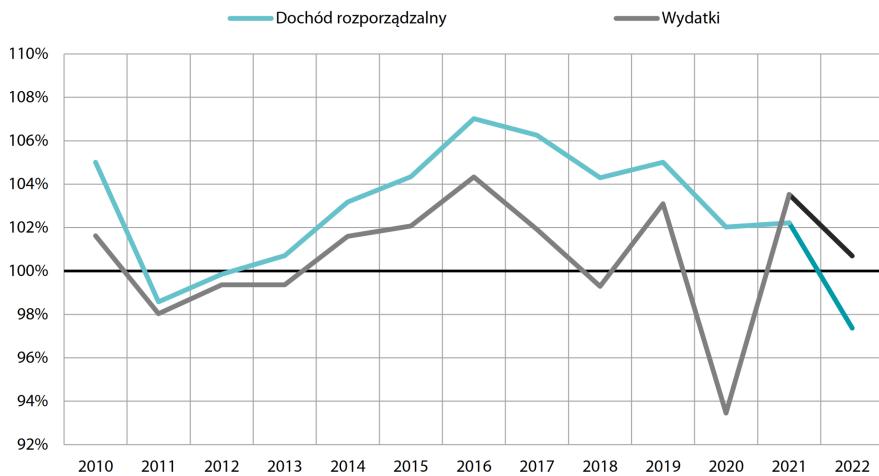
Rozdział drugi zawiera rozwinięcie charakterystyki i motywacji problemu krótko zaznaczonej we wstępie pracy. Rozdział trzeci to analiza istniejących rozwiązań. Rozdział czwarty opisuje koncepcję własnego rozwiązania. Rozdział piąty to ogólny zarys projektu. W rozdziale szóstym znajduje się dokumentacja techniczna. Rozdział siódmy to opis testów i weryfikacji systemu. Rozdział ósmy opisuje przykładowy scenariusz wykorzystania systemu. Ostatecznie rozdział dziewiąty jest zakończeniem pracy.

Rozdział 2

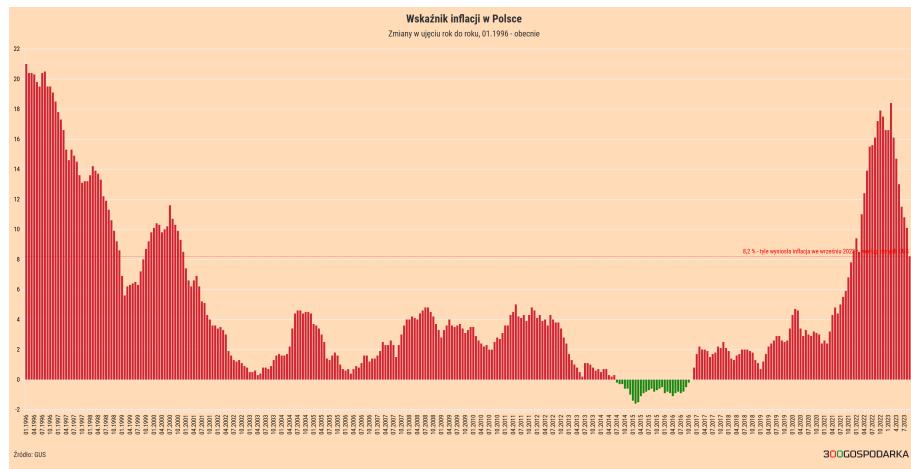
Charakterystyka i analiza problemu

Na dzień dzisiejszy wiedza z zakresu finansów oferowana w ramach systemu edukacji publicznej jest znikoma [9]. Sytuacja ta utrzymuje się od dawna, dlatego spora część obywateli Polski słabo orientuje się w kwestii finansów osobistych, ekonomii i przedsiębiorczości. Istnieje wiele aktywnie działających programów które wprowadzają uczestników w świat finansów poprzez przedstawienie podstawowych zagadnień z dziedziny ekonomii i podstaw inwestowania [9]. Część działań ma na celu zbudować w uczestnikach świadomość ogólnej sytuacji ekonomicznej jednakże jak wynika z badań Banku Pekao [10], większość rodziców stwierdza że nie posiada wystarczającej wiedzy o finansach żeby przekazać ją dzieciom, co pozwala wysnuć wniosek iż sami zarządzają finansami rodzinnymi korzystając raczej z intuicji i własnego doświadczenia aniżeli solidnych podstawa teoretycznych. Tego rodzaju podejście na wyczucie, działa przez większość czasu, wydaje się że nie ma większego wpływu na życie gdy sytuacja ekonomiczna jest spokojna - zmiana podejścia pozwala wtedy co prawda więcej zaoszczędzić, jednak w zasadzie jest to opcjonalne. Kiedy jednak na rynku czy to lokalnym czy globalnym sytuacja staje się bardziej trudniejsza, co może powodować nagły wzrost inflacji przeważnie koszta życia rosną niewspółmiernie do zarobków [8][12], a tym samym dopięcie finansów osobistych i domowych tak, by bilans wyszedł dodatni wymaga więcej uwagi i wiedzy. W ostatnich latach (2019-2023) miało miejsce kilka zdarzeń które dotknęły światową gospodarkę. W momencie pisania tej pracy takimi wydarzeniami są pandemia Covid 19, działania zbrojne na terenie Ukrainy oraz wojna na bliskim wschodzie. Zdarzenia te jak wynika z badań Krajowego Rejestru Długów [11] u prawie połowy polaków wywołała poczucie zagrożenia biedą, podczas gdy jedynie 22% twierdzi że jest spokojna o swoją sytuację finansową. Raport Warsaw Enterprise Institute [13] wykazuje natomiast spadek realnych płac (uwzględniających zarobki oraz wydatki) średnio o 2%, a w niektórych grupach możliwe

5-11% w latach 2020-2022 co wskazuje na wysokie zapotrzebowanie na narzędzia i edukację w zakresie budżetowania.



Rysunek 2.1: Główny Urząd Statystyczny, Dynamika realnych dochodów i wydatków na 1 osobę w gospodarstwach domowych w latach 2010–2022 [8]



Rysunek 2.2: 300gospodarka.pl, Inflacja w Polsce w latach 1996-2023 dane GUS

Planowanie domowego budżetu jest podstawowym narzędziem które pomaga utrzymać wydatki w ryzach. Składa się z dwóch etapów - po pierwsze metody lub narzędzia które ułatwiają zarządzanie budżetem domowym oraz znajomości podstaw zarządzania finansami. Dlatego dla osób rozpoczynających budżetowanie ważne jest przedstawienie w możliwie prostej, zwięzłej i przystępnej formie już gotowych opracowanych rozwiązań które można zastosować aby świadomie

zarządzać sytuacją finansową własnego domostwa. W najprostszym wariantie na budżet [14][17][15][18][16][3] składają się: wpływy czyli dochody ze wszystkich źródeł, zobowiązania czyli płatności stałe jak rachunki czy raty kredytów oraz wydatki które są zróżnicowane. Skrupulatne zbieranie danych z pewnego okresu pozwala określić ogólną sytuację, a w miarę wydłużania zakresu czasu dostępnych danych i zwiększania ich precyzja wyłaniają się trendy co umożliwia prognozowanie przyszłej sytuacji. Istnieje wiele różnych podejść do tworzenia budżetu - od najbardziej ogólnych które skupiają się wyłącznie na określeniu bilansu wydatków oraz wpływów, po najbardziej szczegółowe analizy wydatków na poszczególne kategorie czy nawet produkty. Każde z podejść ma swoje dobre strony, i w gruncie rzeczy wybór odpowiedniego podejścia jest wyłącznie kwestią preferencji.

OPŁATY		ART. SPOŻ.		CHEMIA		ZDROWIE		TRANSPORT		ODPOCZYNEK		INNE (rozślecko)		
ZA CO	ILE	ZA CO	ILE	ZA CO	ILE	ZA CO	ILE	ZA CO	ILE	ZA CO	ILE	ZA CO	ILE	
KREDYT	750 zł	LIDL	30	PROSZEK	50	OKULISTA	120	PALIWO	385	KINO	60	UBRANIE	30	
PRAD	120	BIEDRONKA	50	TABL. DOŁŻN. MIESZ.	40	APTEKA	30	BILETY AUTOBUS	190	PUB	55	ŚPIERKI	20	
WODA	60	MIESNY	20	DOMESTOS	30	APTEKA	50			HOBBY	40	BAGIANKA	30	
GAZ	60	MIESNY	10									PREZENT	80	
INTERNET	40	TESCO	30	PASTA	10	APTEKA	68							
TELEFON S	35	LIDL	70	SŁOŃCZENI	20									
TELEFON T	40	LIDL	145	KREM DO RĘK	5									
TV	45	MIESNY	20											
CZYNISZ	550	REAL	110											
ŁÓDZBOK	300	CUKIERNA	47											
		CUKIERNA	13											
		WĘZŁOWINA	15											
		ZABKA	110											
		FRESH	92											
		RYNEK	110											
		BIERONKA	85											
		BIEDRA	100											
		REAL	188											
SUMA	2000 zł		1420 zł		160 zł		268 zł		575 zł		155 zł		160 zł	4738 zł

Rysunek 2.3: oszczędzaniepieniedzyblog.pl, Najprostszy budżet - wydatki

WYNAGRODZENIE ZA PRACĘ - MĄŻ	2500
WYNAGRODZENIE ZA PRACĘ - ŻONA	2000
PRACA DODATKOWA	50
PIENIĄDZEM ODKRZECIÓW	200
INNE	10
SUMA	4760 zł

WYNAGRODZENIE ZA PRACĘ - MĄŻ	2520
WYNAGRODZENIE ZA PRACĘ - ŻONA	2000
PRACA DODATKOWA	
PIENIĄDZEM ODKRZECIÓW	100
INNE	
SUMA	4620

WYNAGRODZENIE ZA PRACĘ - MĄŻ	2500
WYNAGRODZENIE ZA PRACĘ - ŻONA	2300
PRACA DODATKOWA	10
PIENIĄDZEM ODKRZECIÓW	
INNE	
SUMA	4810 zł

$$4760 + 4620 + 4810 = 14190 / 3 \approx 4730 \text{ zł}$$

Rysunek 2.4: oszczedzaniepieniedzyblog.pl, Najprostszy budżet - przychody

Poza ukazaniem ogólnego obrazu sytuacji finansowej w budżecie uwzględnić można cele jak spłata zadłużenia, czy planowany znaczny wydatek, oraz limity pomagające ograniczyć wydatki i cele przychodów zwiększące ilość dostępnych środków finansowych. W literaturze przedmiotowej opisano także wiele przydatnych podejść oraz zasad jak wstępny podział wydatków na podstawie priorytetów [1][3] czy uwzględnienie oszczędzania w formie podejścia najpierw zapłać sobie [1][2] które można zastosować jako strategie zarządzania finansami domowymi aby usprawnić budżet lub osiągnąć zamierzony cel. Jednak aby zastosować daną strategię trzeba ją najpierw znać, a jak wynika z informacji opisanych wcześniej poziom wiedzy z zakresu finansów w Polsce oceniany jest jako słaby, dlatego narzędzia do zarządzania finansami powinny, w najprostszej formie udostępniać takie informacje w łatwo przystępnej formie, lub przy bardziej zaawansowanym podejściu posiadać wbudowane mechanizmy które pozwoliliłyby użytkownikowi wybrać i zastosować strategię bez potrzeby jej dogłębnej znajomości.

Z uwagi na tematykę zwyczajowo są to dane bardzo wrażliwe, zatem wymagają odpowiednich zabezpieczeń. Idealną opcją dla potencjalnych użytkowników byłoby gdyby jako jedyni mieli dostęp do prywatnych danych, oraz mogli sami precyzyjnie decydować komu je udostępniają. Przy znaczającej statystycznie liczbie użytkowników dane zebrane w aplikacji po odpowiedniej pełnej nieodwracalnej anonimizacji i uśrednieniu mogą posłużyć do modelowania wydatków obywateli danych regionów, sektorów, segmentów gospodarki lub nawet ogólnie całego państwa, co z kolei można wykorzystać zwrotnie w samej aplikacji aby porównać model wydatków użytkownika do adekwatnej średniej i zwrócić uwagę na obszary w których pozytyw-

nie od niej odstaje działając jako pozytywne wzmocnienie dobrego nawyku [19]. Tego typu modelowanie jest już przeprowadzane przez Główny Urząd Statystyczny, którego raporty publikowane są co roku, jest to więc potencjalne źródło najbardziej precyzyjnych danych, które mogłyby zostać wykorzystane do porównań.

Rozdział 3

Analiza istniejących rozwiązań

W tym rozdziale przedstawiona zostanie analiza wybranych, obecnie dostępnych rozwiązań rozważanego w pracy problemu, jej celem jest określenie wady i zalety tych rozwiązań. Jako że budżetowanie jest problemem tak starym jak sam wynalazek pieniądza, historycznie powstało wiele różnych rozwiązań których celem jest je ułatwić.

Podstawową i najprostszą formą budżetu jest zapis na papierze czy chociażby w formie księgi zawierającej przychody i wydatki [14]. Sposób ten zostanie przeanalizowany ponieważ do niedawna była to główna metoda prowadzenia budżetu i mimo postępu cyfryzacji i informatyzacji nadal jest szeroko stosowany. Po części zastąpiły go rozwiązania komputerowe w formie różnych aplikacji które wymagają mniejszych lub większych nakładów pracy od użytkownika. Okazjonalnie tego typu zestawienia prowadzone są dziś także w arkuszach kalkulacyjnych.

Pierwszą w pełni cyfrową opcją są same witryny kont bankowych [16] na których klient często może kategoryzować poszczególne transakcje i wyświetlać podsumowania oraz określić zakładany budżet. Rozwiązania te są dostępne dla każdego klienta danego banku dla tego warto się im przyjrzeć, jako przykład posłuży portal banku Santander - centrum24.pl [21].

Na kolejną kategorię rozwiązań składają się aplikacje dedykowane do zarządzania budżetem [17]. Systemy tego typu po wprowadzeniu danych udostępniają użytkownikowi cały wachlarz dodatkowych specjalistycznych opcji i narzędzi. Omówione zostaną dwa przykłady tego rodzaju aplikacji - Intuit mint [22] oraz Goodbudget [25]. Na rynku dostępnych jest wiele więcej rozwiązań przez co użytkownik ma dowolny wybór, jednak świadomy wybór odpowiedniej opcji wymaga od użytkownika dokładnego przeglądu i porównania kilku aplikacji.

3.1 Budżet papierowy lub arkusz kalkulacyjny

Grupa ta obejmuje wiele różnorodnych narzędzi, nierzadko darmowych, lub takich, które użytkownik posiada do innych celów. Przykładowe opcje obejmują proste rozpiski i podsumowania na kartkach, arkusze kalkulacyjne jak Microsoft Excel, Google Sheets, LibreOffice Calc - przykłady na rysunkach 3.1 oraz 3.2. Największą zaletą tych rozwiązań jest prostota dzięki której z tego typu rozwiązania jest w stanie skorzystać w zasadzie każdy jednakże odpowiedzialność za manualne utrzymanie i dbanie o jakość czy spójność danych spoczywa wyłącznie na użytkowniku który jest jednocześnie autorem budżetu. Jedynie w podejściach cyfrowych okazjonalnie znaleźć można dodatki służące do automatyzacji części funkcji, niemniej jednak użytkownicy posiadający odpowiednią wiedzę i umiejętności mogą przygotować tego typu mechanizmy osobiste - przykładowo w arkuszach kalkulacyjnych jako formuły czy nawet skrypty (np. VBA w Excell) jeżeli narzędzie ma taką funkcję. Udostępniają także każde możliwe podejście znane użytkownikowi oraz całkowitą wolność wyboru chociażby kategoryzacji. Z uwagi na niski próg wejścia w sieci dostępnych jest wiele poradników oraz szablonów, choć paradoksalnie jednocześnie jest to wada ponieważ początkującemu użytkownikowi trudno się odnaleźć w sporej ilości prezentowanych opcji. Zależnie od wykorzystanej technologii (uwzględniając także budżet papierowy) mogą być dostępne zarówno lokalnie jak i w przeglądarce.

Na uwagę zasługuje również fakt iż podejście takie niejako mimo chodem uczy użytkownika podstaw finansów i zmusza do refleksji nad swoją sytuacją, co może zaowocować wypracowaniem własnych spersonalizowanych systemów dostosowanych pod swoje potrzeby i dopasowanych do preferowanych metod pracy lepiej niż pozostałe dostępne rozwiązania.

ROK 2020	ŁĄCZNIE W CAŁYM ROKU: ŚREDNIO MIESIĘCZNE	100%	15%	36%
		6 900 zł 575 zł	21 400 zł 1 783 zł	28 300 zł 2 358 zł
Miesiąc	Wydatki nieregularne	Potrzeba	Zachcianka	Suma
1	Wyjazd rodzinny na narty	- zł	6 000 zł	6 000,00 zł
2	Dzieci - wyjazd na obóz zimowy	- zł	1 200 zł	1 200,00 zł
2	Oplata za II semestr angielskiego dla dzieci	600 zł	- zł	600,00 zł
2	Urodziny - moje	200 zł	300 zł	500,00 zł
3	Składka roczna - ubezpieczenie mieszkania	200 zł	- zł	200,00 zł
3	Użytkowanie wieczyste- mieszkanie	410 zł	- zł	410,00 zł
3	Użytkowanie wieczyste- garaż	80 zł	- zł	80,00 zł
3	Podatek od nieruchomości - mieszkanie	110 zł	- zł	110,00 zł
3	Rośliny na taras - dosadzanie	- zł	200 zł	200,00 zł
4	Dodatkowe wydatki na Święta Wielkanocne	300 zł	200 zł	500,00 zł
4	Przegląd roczny samochodu do dowodu	100 zł	- zł	100,00 zł
4	Rocznny serwis samochodu (olej, filtry, itp.)	700 zł	- zł	700,00 zł
4	Wymiana opon na letnie	80 zł	- zł	80,00 zł
4	Polisa OC/AC - samochód	400 zł	1 000 zł	1 400,00 zł
4	Podatek dochodowy - dopłata	- zł	- zł	0,00 zł
4	Urodziny - dziecko 1	200 zł	300 zł	500,00 zł
5	Weekend majowy - wyjazd	- zł	800 zł	800,00 zł
5	Komuna dziecka	1 200 zł	1 000 zł	2 200,00 zł
6	Dzieci - wyjazd na letnią kolonię	- zł	2 400 zł	2 400,00 zł
6	Urodziny - żona	200 zł	300 zł	500,00 zł
7	Wyjazd rodzinny na wakacje	- zł	3 200 zł	3 200,00 zł
8	Dodatkowe przybory szkolne dla dzieci (wyprawka)	600 zł	600 zł	1 200,00 zł
9	Składki w szkole	240 zł	200 zł	440,00 zł
9	Urodziny - dziecko 2	200 zł	300 zł	500,00 zł
10	Wymiana opon na zimowe	80 zł	- zł	80,00 zł
11	Wyjazd do rodziny na "Wszystkich Świętych"	- zł	300 zł	300,00 zł
11	Zakup prezentów na Boże Narodzenie	300 zł	800 zł	1 100,00 zł
12	Wyjazd do rodziny na Święta Bożego Narodzenia	- zł	1 000 zł	1 000,00 zł
12	Dodatkowe wydatki Świąteczne	500 zł	300 zł	800,00 zł
12	Sylwester w domu lub Bal Sylwestrowy	200 zł	1 000 zł	1 200,00 zł

Rysunek 3.1: <https://marciniwuc.com/budzet-domowy-05-wydatki-nieregularne/>
Przykładowy budżet w aplikacji Excel

Co ciekawe, na tym podejściu zbudowana jest także aplikacja Tiller [31], służy jako interfejs który pozwala użytkownikowi na migrację danych bankowych do prywatnego arkusza kalkulacyjnego w Excel lub Google Sheets, w którym tworzy predefiniowany szablon z wizualizacjami i podsumowaniami.

Budżet domowy

2013.09

wpisz aktualny miesiąc

Zarobki w ubiegłym miesiącu:

6700

kwota na rekę (netto)

Lista wydatków stałych:

Lista wydatków jednorazowych (tylko w tym miesiącu):

PODRECZNIKI SZKOLNE	320
PRZYKROTY SZKOLNE	70
PLECAK	45
PRZEGŁAD ANTA	900
	1335
Saldo finalne:	1115

Jeśli saldo jest dodatnie, to brawo! Tyle pieniędzy możesz zaoszczędzić >> Strona 3.

Jeśli saldo jest ujemne, to oznacza, że o taką kwotę musisz uszczuplić swoje oszczędności (Fundusze nieregularnych wydatków) lub ograniczyć koszty.

Rysunek 3.2: <https://jakoszczedzaczepieniadze.pl/prosty-budzet-domowy> Przykładowy budżet na papierze

3.2 Systemy bankowe - Przegląd wydatków w Santander

Wraz z rozwojem technologii i cyfryzacji posiadanie konta bankowego stało się praktyczne wymogiem. Na rynku istnieje szeroki ich wybór, dlatego jak każda nowoczesna firma, także i banki starają się wyjść naprzeciw oczekiwaniom klienta i zachęcić go dodatkowymi funkcjami. W efekcie, choć jest to jedynie opcja dodatkowa, część z nich wdrożyła u siebie usługę pomagającą użytkownikowi w analizie finansów i planowaniu budżetu.

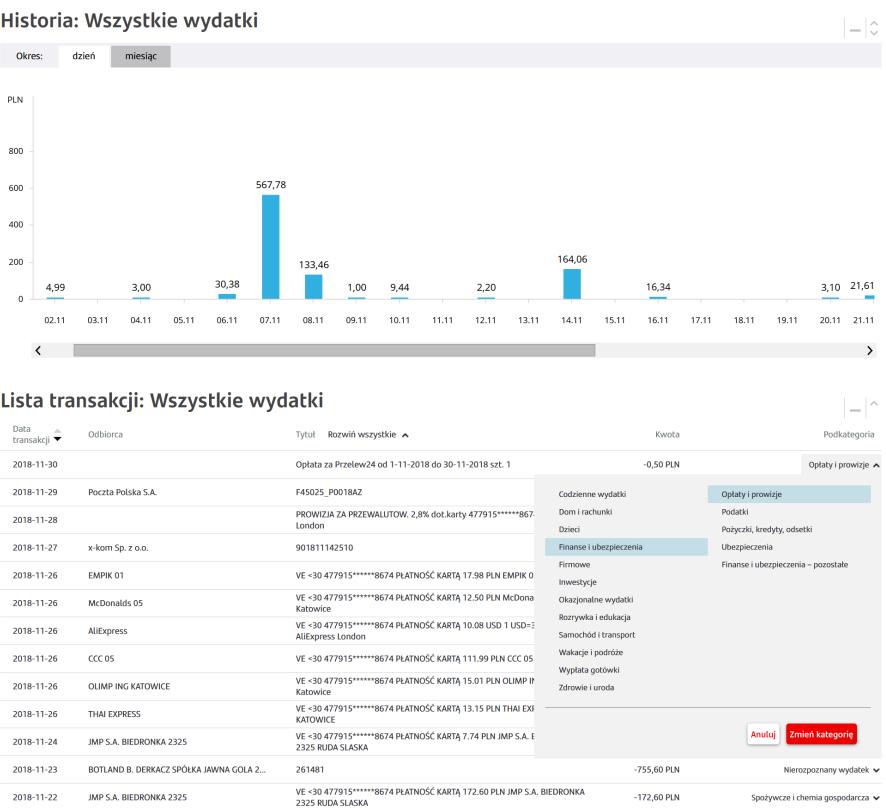
The screenshot shows the Santander 'Historia' (History) section of the centrum24.pl website. At the top, there's a navigation bar with links like 'Finanse', 'Twoje sprawy', 'Przelewy', 'Historia', 'Kantor Santander', 'Fundusze inwestycyjne', 'Uslugi partnerów', 'Rachunek maklerski', and 'Oferta'. Below the navigation is a search bar and filter options for date range ('Od' to 'do'), transaction type ('Wydatki i wpływy'), and status ('Wszystkie'). The main area displays a table of transactions with columns for date, recipient/source, title, amount, and actions. The table lists various transactions such as payments to AliExpress, McDonald's, and Poczta Polska, along with their amounts and descriptions. At the bottom of the table, there are download buttons for PDF, CSV, or printing.

Rysunek 3.3: Centrum24.pl, Santander historia transakcji

Jako przykład posłuży system Przegląd wydatków na portalu centrum24.pl banku Santander, zaprezentowany na rysunku 3.6. Usługa stara się na bieżąco przypisywać transakcje dokonane na koncie klienta do dość ogólnych kategorii, jednocześnie użytkownik może je dowolnie zmienić. Kategorie mają dwa poziomy szczegółowości, dzięki czemu są grupowane jednak w pewnym stopniu pozwalają ukazać szczegółowo. Jak widać na rysunku 3.5 użytkownikowi prezentowany jest wykres słupkowy pokazujący ilość wydanych pieniędzy na poszczególne kategorie lub grupy kategorii, oraz wykres pokazujący sumę wydatków w danym okresie i wszystkie transakcje które wchodzą w ich skład.

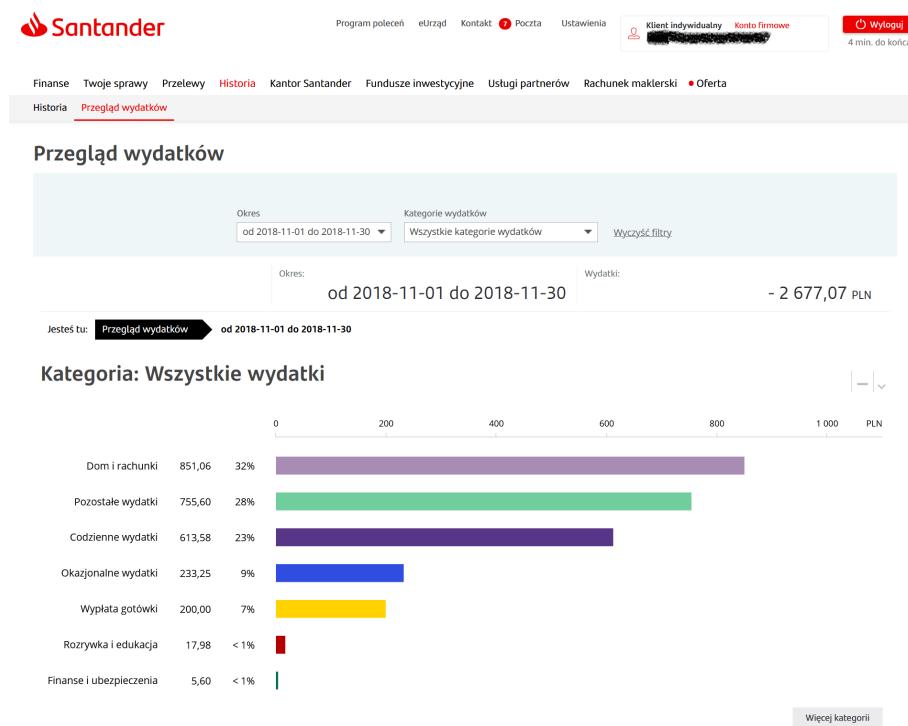


Rysunek 3.4: Centrum24.pl, Santander przegląd kategorii wydatków



Rysunek 3.5: Centrum24.pl, Santander kategoryzacja wydatków

System ma także swoje ograniczenia. Niestety prezentuje wyłącznie wydatki, tym samym jedyne miejsce w którym użytkownik może podejrzeć swoje przychody jest historia konta, jednak jak widać na rysunku 3.3 jest to jedynie suma w wybranym okresie, ewentualnie zestawienie transakcji wpływów pozostawiając analizę w gestii użytkownika.



Rysunek 3.6: Centrum24.pl, Santander przegląd wydatków

3.3 Aplikacje dedykowane - Intuit Mint

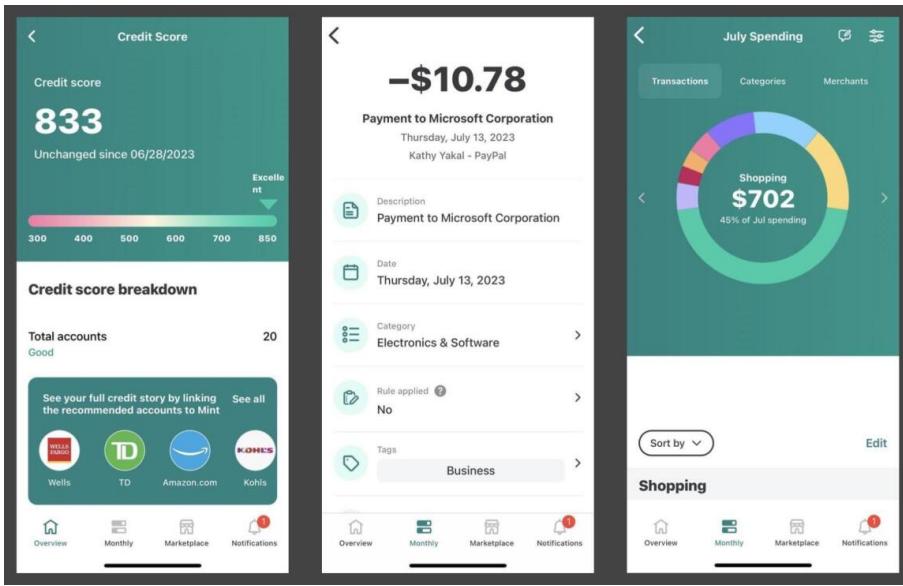
W trakcie pisania pracy dostawca aplikacji zdecydował się wycofać aplikacje z początkiem roku 2024 i zachęcić użytkowników do migracji na swoją platformę Credit Karma [24], która pozbawiona jest funkcji budżetowania. Mimo to przykład pozostaje aktualny ponieważ aplikacja jest uznawana przez wielu użytkowników i recenzentów za jedną z najlepszych w kategorii finansów, dlatego warto zwrócić uwagę na jej zalety i wady, zwłaszcza że jej wycofanie tworzy na rynku pewną niszę.

Aplikacja Intuit Mint [22][23] dostępna jest w wersji darmowej z reklamami lub płatnej, na systemy mobilne Android i iOS oraz

przeglądarki. Po uruchomieniu Mint prezentuje użytkownikowi ekran z podsumowaniem finansów w obecnym miesiącu w postaci zakładek które grupują informacje z kilku kategorii, m.in.: Wartość netto, Wydatki, Inwestycje. Aby zasilić dane użytkownik musi dać aplikacji dostęp do swoich kont bankowych i inwestycyjnych, co wiele recenzentów uznał za niesamowicie wygodne jednak jednocześnie przez to nie jest to aplikacja dla ludzi dbających o prywatność - jest to także główny powód dlaczego przegląd tej aplikacji oparty jest na recenzjach i poradnikach opublikowanych w internecie. W zakładce Transakcje zgromadzone są także wszystkie płatności zarejestrowane na udostępnionych aplikacji kontach, z opinii długotrwałych użytkowników wynika że jakość automatycznej kategoryzacji jest słaba [26] [27][28] [29][30]. Użytkownik może je oznaczać etykietami (tag), dodawać do nich notatki czy wykluczać z zestawień.

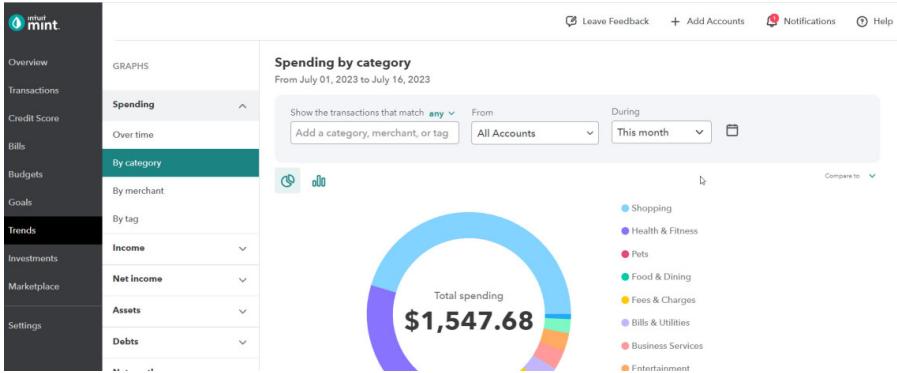
W zakładce Miesiąc aplikacja generuje podsumowanie przychodów i wydatków, pozwala utworzyć tygodniowe cele wydatków oraz budżet (a nawet kilka jednocześnie) przy użyciu prostego kreatora.

Kreator budżetu wydaje się bardzo prosty w obsłudze, samodzielnie szacuje przychód na podstawie dostępnych danych który następnie użytkownik zatwierdza lub nadpisuje dowolną wartością, w drugim kroku użytkownik definiuje samodzielnie listę kategorii oraz podkategorii wydatków co niestety wymaga od wcześniejszego przygotowania. Użytkownik może dodać cel finansowy podając nazwę, wartość, oraz datę kiedy powinien być spełniony, następnie wybiera konto bankowe które chce z nim powiązać - jego spełnienie śledzi porównując ilość środków na koncie z wyznaczonym celem. Dodatkową opcją są przypomnienia o opłacie nadchodzących rachunków w postaci powiadomień push, wiadomości e-mail lub wydarzeń w kalendarzu.



Rysunek 3.7: Mint w wersji mobilnej

Wersja w przeglądarce pozwala dodać nieruchomości i pojazdy zaliczane do całkowitej wartości netto, porady poprawy zdolności kredytowej i wartość inwestycji na dodanych rachunkach inwestycyjnych. Zakładka Trends zawiera kilka predefiniowanych wykresów wizualizujących dane, zawierających dość ogólne dane.



Rysunek 3.8: Mint w wersji na przeglądarki

Mint integruje także dodatkowe usługi jak wyliczanie zdolności kredytowej przez usługę firmy TransUnion, czy negocjację umów przez usługę firmy BillShark. Aplikacja sama w sobie nie zawiera zintegrowanego samouczka bądź instrukcji, informacje wymagane zdaniem twórców skrótnie prezentowane są kontekstowo w miejscu w którym są wymagane.

Podsumowując aplikacja Mint jest prostota w obsłudze, wymaga minimum dodatkowej uwagi od użytkownika, jest bardzo przejrzysta, estetyczna i nowoczesna. Jednak wstępna konfiguracja wymaga od użytkownika nadania jej dostępu do kont które zawierają wiele wrażliwych danych co jest problemem z uwagi na prywatność, ponadto użytkownik musi samodzielnie z góry określić wydatki na poszczególne kategorie zanim aplikacja udostępni mu wartościowe informacje o budżecie.

Rozdział 4

Koncepcja własnego rozwiązania

Rozdział ten opisuje koncepcję rozwiązania problemów opisanych we wstępie na podstawie wniosków wyciągniętych z analizy istniejących rozwiązań tak, by proponowane rozwiązanie posiadało możliwie jak najwięcej zalet, a jednocześnie nie posiadało wad a także uzupełniało braki w obecnie dostępnych na aplikacjach.

4.1 Koncepcja rozwiązania użytkowego

Projekt aplikacji zakłada realizację aplikacji internetowej która umożliwia interakcję z wieloma użytkownikami jednocześnie. Aplikacja przechowywać będzie dane finansowe, które są danymi wrażliwymi więc wymagają szczególnej ochrony, dlatego dostęp do nich zostanie ograniczony do minimum - będzie wymagany w krótkich okresach komunikacji aplikacji z bazą podczas zapisu danych z pamięci podręcznej aplikacji oraz pobierania danych do wyświetlenia użytkownikowi. Aplikacja udostępniać będzie użytkownikom interfejs do wprowadzania, edycji i usuwania danych finansowych jak przychody, rachunki (płatności stałe), wydatki oraz kategorie po których będą grupowane - typy produktów i produkty. Użytkownikom udostępnione zostaną predefiniowane raporty złożone z wizualizacji, statystyk i danych analitycznych oraz podane w przystępny sposób informacje o podstawach zarządzania finansami i budżetem.

Istnieje też pewna liczba mile widzianych funkcji, które z uwagi na poziom złożoności systemu i stosunkowo krótki czas implementacji są obecnie poza zakresem projektu, jednak mogą w przyszłości zostać zrealizowane. Są to między innymi: modyfikacja interfejsu tak, by użytkownik mógł ocenić użyteczność funkcji oraz panel zgłoszeń propozycji i problemów, co pozwoli ukierunkować rozwój aplikacji w stronę najbardziej przydatnych i potrzebnych w danym momencie rozwiązań. W dalszym etapie rozwoju aplikacji można także wdrożyć

moduł predykcji przyszłych wydatków w oparciu o dane historyczne. Kolejnym obszarem z potencjałem rozwoju jest wzbogacenie wachlarza metod wprowadzania danych - aby ułatwić użytkowanie aplikacji można utworzyć moduł importu danych z plików w popularnym standardowym formacie jak CSV [39], a w dalszej perspektywie nawet funkcje ekstrakcji danych z obrazów co umożliwi wprowadzanie danych bezpośrednio z faktur, zdjęć, rachunków, paragonów, pasków wynagrodzenia. Możliwość określenia własnych progów wydatków które będą uwzględniane na wizualizacjach. Można także wdrożyć automatyczną kategoryzację wydatków na proponowane w literaturze przedmiotowej zbiory wydatków niezbędnych, potrzebnych i zachcianek [3]. Ostatnią przewidzianą na ten moment funkcją dodatkową jest możliwość definiowania celów finansowych które wpłyną na prezentację danych w raportach aby użytkownik mógł śledzić ich postępy i realizację.

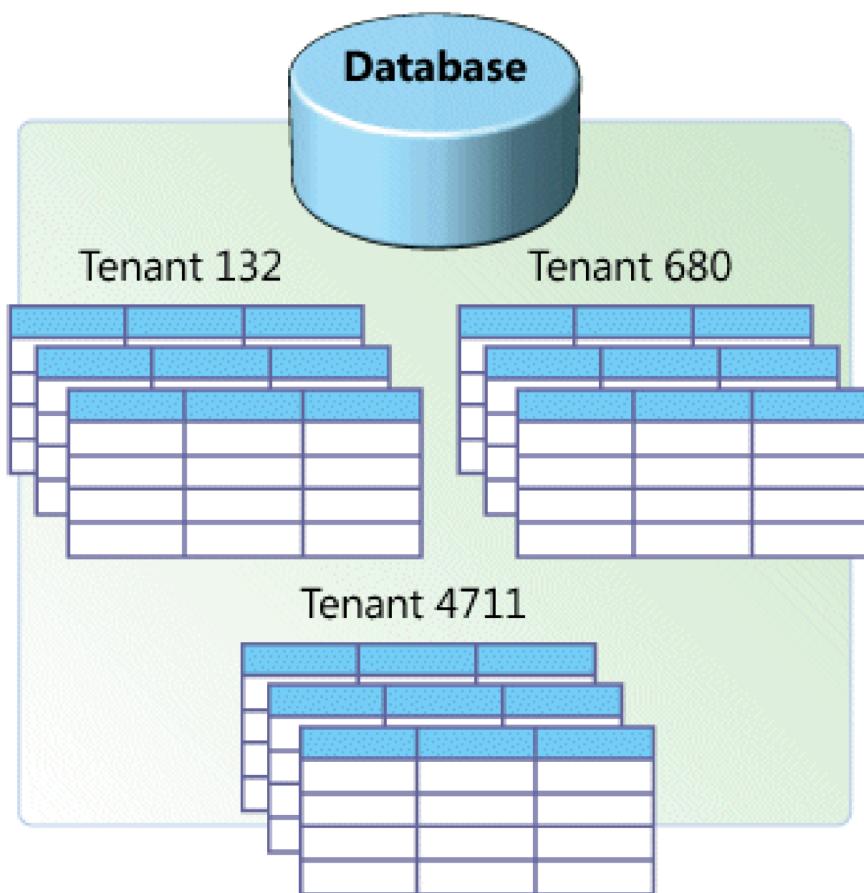
W fazie rozwoju aplikacji na potrzeby pracy inżynierskiej projekt będzie udostępniony publicznie na licencji open source, dlatego wstępny projekt interfejsu będzie w języku angielskim aby poszerzyć grono potencjalnych użytkowników, poprawić czytelność projektu i ułatwić współpracę podczas rozwijania kodu w dalszych etapach. Nie jest to natomiast docelowa jedyna wersja językowa - implementację wyboru wersji językowej, tłumaczenie interfejsu na kilka popularnych języków (manualnie lub maszynowo) pozostawiono jako funkcję dodatkową, opcjonalną.

4.2 *Koncepcja rozwiązania technologicznego*

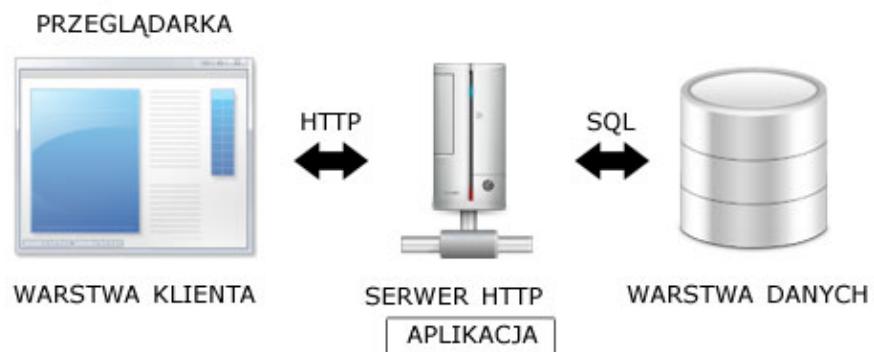
Proponowane rozwiązanie powinno przyjąć formę trójwarstwowej aplikacji przeglądarkowej jak zaprezentowano na rysunku 4.2. Ponieważ z aplikacji ma korzystać wielu użytkowników jednocześnie rodzi to wymóg wielodostępności o podejściu architektury multitenant [20], która pozwala korzystać wielu użytkownikom z tej samej bazy aplikacji o różnym stopniu prywatności danych zależnym od zastosowanego poziomu separacji - osobne bazy, wspólna baza i osobne schematy, wspólny schemat podział na poziomie rekordu. Z uwagi na prywatność danych każdy z użytkowników docelowo będzie korzystał z własnego schematu w bazie danych aplikacji przechowywanej na serwerze co przedstawia rysunek 4.1, jest to jednak rozwiązanie które trudno wdrożyć, dlatego w fazie projektowej, która jest przedmiotem tej pracy, zastosowano uproszczenie w postaci pojedynczego domyślnego schematu danych dla każdego użytkownika (słowniem: wszyscy użytkownicy mają dostęp do tych samych danych). Jest to jednak

pierwsza poważna zmiana, której wymaga system po udostępnieniu wszystkich funkcji które określono jako krytyczne dla pełnego działania minimalnej wersji aplikacji.

Etap projektowania bazy danych wymagać będzie sporych nakładów pracy dlatego aby go uprościć, początkowo zastosowana zostanie technologia SQLite [43]. Jest to jednak technologia maksymalnie uproszczona, nie ma w niej funkcji tworzenia osobnych schematów ani zarządzania dostępem do danych, dlatego docelowo struktura bazy zostanie przeniesiona do PostgreSQL [46]. Dopiero po migracji do nowej technologii możliwe będzie zastosowanie podejścia z osobnymi schematami jednak będzie to zmiana stosunkowo pracochłonna.



Rysunek 4.1: Microsoft, Architektura współdzielona baza, rozdzielne schematy



Rysunek 4.2: <http://framework.gigr.pl/> Architektura Trójwarstwowa

Rozdział 5

Projekt ogólny

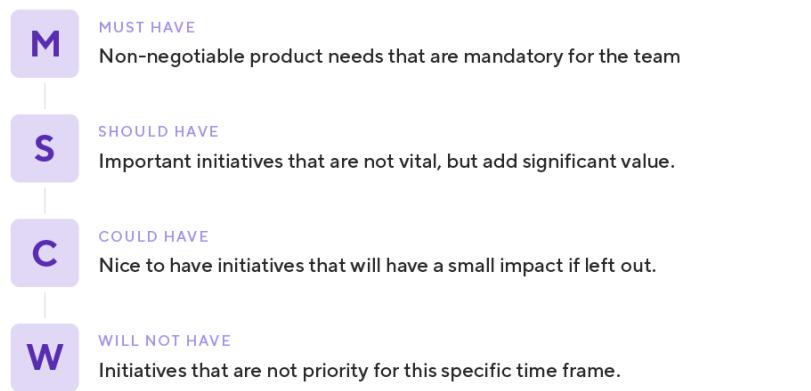
Rozdział ten opisuje ogólną koncepcję organizacji systemu, plan jego architektury, przechowywanie danych, ogólny plan interfejsu użytkownika oraz metody i narzędzia realizacji.

5.1 Specyfikacja wymagań funkcjonalnych i niefunkcjonalnych

Zestawienie funkcji które powinien spełniać program, wzbogacone o informacje które z nich zostały spełnione. Podzielone na listy wymagań niefunkcjonalnych i funkcjonalnych. Nagłówki z powodu objętości zostały skrócone, legenda:

PRIOR - Priorytet w kategorii MoSCoW [32], rysunek 5.1

IMPLEMENTED - Oznaczenie czy funkcję wdrożono (Implemented)



Rysunek 5.1: Pirorytetyzacja MoSCoW

Tabela 5.1: Wymagania niefunkcjonalne

Funkcja	PRIORITET	IMPLEMENTACJA	Opis
Prosty interfejs	M	TAK	Prosty, minimalistyczny interfejs
Plik konfiguracji	M	TAK	Konfiguracja ustawień użytkownika
Poradniki	M	TAK	Poradniki zarządzania finansami
Rejestr zdarzeń	S	NIE	Logi z działania aplikacji
Informacja zwrotna	S	NIE	Kontrolki informacji zwrotnej i opinii użytkownika
Lokalizacje	C	NIE	Wersje językowe interfejsu do wyboru
Personalizacja interfejsu	W	NIE	Personalizacja interfejsu użytkownika np. kolory-styka, układ

Tabela 5.2: Wymagania funkcjonalne

Funkcja	PRIORITET	IMPLEMENTACJA	Opis
Aplikacja WEB	M	TAK	Dostęp do aplikacji z poziomu przeglądarki
Wiele użytkowników	M	TAK	Wsparcie dla wielu użytkowników jednocześnie
Dodawanie danych	M	TAK	Dodawanie danych
Podsumowanie wydatków	M	TAK	Okresowe podsumowanie wydatków
Podsumowanie przychodów	M	TAK	Okresowe podsumowanie przychodów
Bilans okresowy	M	TAK	Okresowy bilans zysków i strat
Definiowanie produktów	M	TAK	Definiowanie produktów
Definiowanie przychodów	M	TAK	Definiowanie przychodów
Definiowanie typów produktów	M	TAK	Definiowanie typów produktów
Weryfikacja danych	M	TAK	Potwierdzenie jakości danych
Panel konfiguracyjny	S	TAK	Osobny panel konfiguracyjny
Import danych	S	NIE	Import danych w standardowym formacie np. CSV
Konteneryzacja	S	TAK	Konteneryzacja aplikacji w środowisku Docker
Statystyki typów	C	TAK	Statystyki wydatków na dany typ produktu
Statystyki produktów	C	TAK	Statystyki wydatków na dany produkt
Zaawansowane wprowadzanie danych	C	NIE	Zaawansowane metody wprowadzania danych np.: zdjęcia, skany
Definiowanie typów przychodów	C	NIE	Definiowanie typów przychodów
Eksport danych	C	NIE	Eksport danych do standardowego formatu
Własne progi	C	NIE	Progi wydatków określone przez użytkownika w ustawieniach
Cele oszczędnościowe	C	NIE	Cele oszczędnościowe określone przez użytkownika w ustawieniach
Porady	C	TAK	Porady na podstawie danych wprowadzonych do aplikacji pomagające użytkownikom w poprawie usprawnień budżetu
Predykcja	W	NIE	Predykcja trendów na podstawie wprowadzonych danych
Autokategoryzacja	W	NIE	Automatyczna kategoryzacja wydatków na niezbędne, potrzebne i zachcianki

5.2 Architektura systemu

Przedmiotem projektu będzie system klasy internetowej - monolityczna aplikacja internetowa dostępna w przeglądarce o architekturze trójwarstwowej [4], którą można będzie umieścić na dedykowanym serwerze lub jeśli dodatkowy cel konteneryzacji zostanie zrealizowany - na dowolnej maszynie na której udostępniony będzie kod źródłowy oraz zainstalowany Docker. Aplikacja podzielona będzie na interfejs użytkownika (frontend) odpowiadający za interakcję z użytkownikiem i walidację danych, kod na serwerze (backend) zawierający logikę działania aplikacji i komunikację z warstwą dostępu do danych którą będzie baza danych aplikacji i użytkowników. Warstwy aplikacji będą komunikować się między sobą zgodnie z obowiązującym standardem - frontend będzie mieć dostęp wyłącznie do backendu, z kolei backend poza odpowiadaniem na żądania frontendu komunikować się będzie z warstwą przechowywania danych którą stanowi baza. Wszystkie komponenty aplikacji docelowo działać będą na pojedynczej maszynie, jednak nic nie stoi na przeszkodzie aby w przyszłości jeżeli zajdzie taka potrzeba wydzielić dla każdego z nich dedykowane maszyny, natomiast konteneryzacja aplikacji umożliwia zmianę modelu z monolitycznego na mikroserwisy i skalowanie horyzontalne.

Aby uprościć logikę aplikacji i poprawić jej wydajność ciężar przetwarzania danych zostanie przerzucony na warstwę bazy danych. Zadanie to przejmą widoki napisane w języku SQL [45] które odpytane przez aplikacje efektywnie przetwarzają dane w locie.

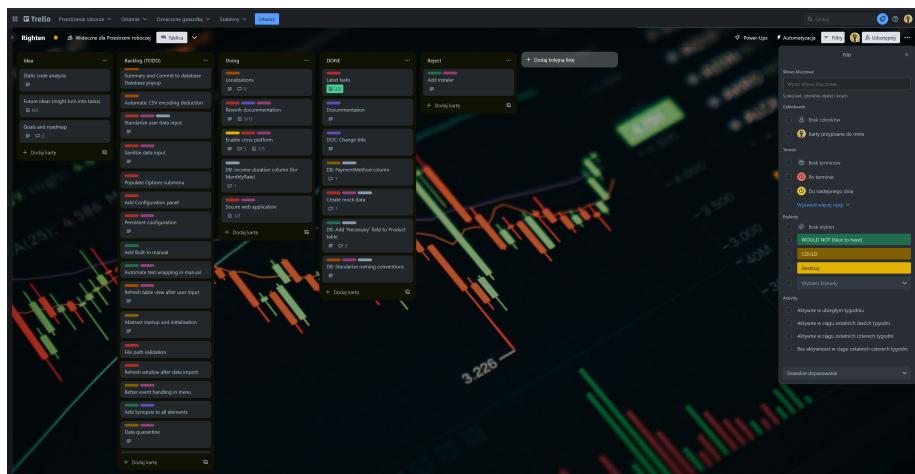
5.3 Metody i narzędzia realizacji

Plan projektu zakłada podejście w metodyce LEAN Software Developpment [36] która umożliwia przyrostowe doskonalenie projektu [37], z wykorzystaniem metody Kanban [35] za pomocą usługi Trello [50] widocznej na rysunku 5.2 do rejestracji, planowania realizacji i śledzenia zadań. Zdaniem autora pracy podejście takie umożliwia szybki rozwój ponieważ w miarę rozwoju projektu możliwości aplikacji poszerzają się dzięki małym, ciągłym udoskonaleniom już obecnych funkcji oraz stopniowym dodawaniu nowych.

O kolejności realizacji prac zadecyduje priorytet określony w klasyfikacji MoSCoW [32] poglądowo przedstawiony na rysunku 5.1, natomiast wymóg i czas rozpoczęcia prac wyznaczy podejście oparte o matrycę Eisenhowera [33]. Dzięki temu wszystkie funkcje które pozwolą dostarczyć Minimalny Wystarczający Produkt (MVP,

Minimal Viable Product) [34] będą na bieżąco widoczne jako zadania najpilniejsze a zatem prace mające na celu ich wdrożenie realizowane niezwłocznie.

Zgodnie z dobrymi praktykami inżynierii informatycznej kod rozwijanej aplikacji będzie przechowywany w systemie kontroli źródła git [55], i opublikowany na portalu GitHub [56] w repozytorium Righten [59] - z poprzednimi wersjami projektu można się zapoznać w repozytorium DatabaseShenanigans [57] a jego dokumentacją w Budgeter [58]. W projektowanym rozwiążaniu preferowane będą technologie i rozwiązania darmowe oraz open source.



Rysunek 5.2: Tablica zadań projektu w usłudze Trello

Kod aplikacji napisany zostanie w język Python [49], głównie przez wzgląd na walory edukacyjne i prostej, ekspresywnej składni co zwiększy czytelność kodu i zmniejszy poziom złożoności aplikacji.

W warstwie interfejsu użytkownika (frontend) większość wykorzystywanych narzędzi jest ze sobąści połączona ponieważ wybór jednej technologii wpływa na kolejne, które z nią współpracują. W zamierzeniu poszczególne strony generowane będą przez silnik szablonów jinja2 [66] który pozwala definiować część elementów strony dynamicznie, wybrano go z uwagi ponieważ jego składnia podobna jest do języka Python. Framework Bootstrap [61] umożliwia wzbogacenie interfejsu o funkcje napisane w JavaScript [60] który posłuży także do wypełniania szablonów stron danymi, a dzięki temu możliwe będzie także generowanie wizualizacji z pomocą biblioteki chart.js [67]. Do obsługi formularzy wprowadzania danych posłuży framework WTForms [65].

Ponieważ z projekt zakłada realizację w miarę prostej aplikacji, aby nie serwer sieciowy (backend) obsługiwać będzie framework

Flask [63] wraz z dodatkowymi wtyczkami do obsługi poszczególnych funkcji (m.in.: Flask-Login [64] do zarządzania dostępem do stron w ramach sesji użytkowników) oraz określonymi w trakcie pisania aplikacji wymaganymi bibliotekami. Jako alternatywę rozpatrywano framework Django [62] który udostępnia wiele dodatkowych mechanizmów zwiększających wydajność i bezpieczeństwo kosztem bardziej złożonej konfiguracji. Po rozważeniu opcji stwierdzono jednak że w chwili obecnej jest to projekt na małą skalę w którym liczy się przejrzystość kodu za czym wydają się przemawiać zalety prostszego z rozwiązań, pozostawiając jednocześnie Django jako ewentualność na przyszłość jeżeli projektowana aplikacja zostanie wykorzystana na większą skalę.

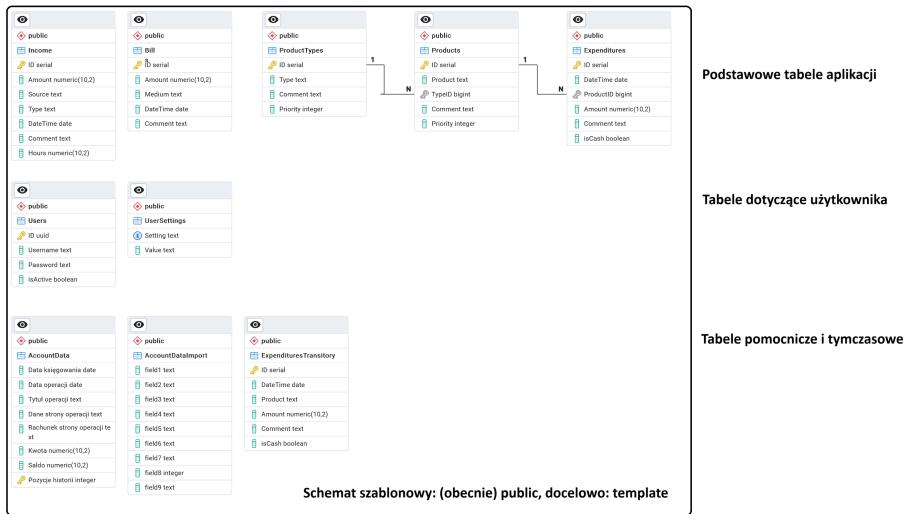
Dla uproszczenia etapu projektowania wstępnie w warstwa bazy danych będzie projektowana na lokalnej instancji SQLite3 [43], w dalszej części projektu zostanie zmigrowana do docelowej technologii jaką jest PostgreSQL [46]. Rozwiążanie takie przyjęto ponieważ instancja PostgreSQL do działania wymaga serwera oraz osobnej aplikacji pgAdmin [47] do zarządzania nią co zwiększa nakłady pracy wymagane do wdrożenia funkcji podstawowych aplikacji.

Domyślnie rozwiązanie będzie działało natywnie na serwerze lub maszynie wirtualnej, jednak opcjonalnie w ramach dalszego rozwoju przewidziano aby środowisko aplikacji powoływać dynamicznie z wykorzystaniem platformy uruchomieniowej Docker [48].

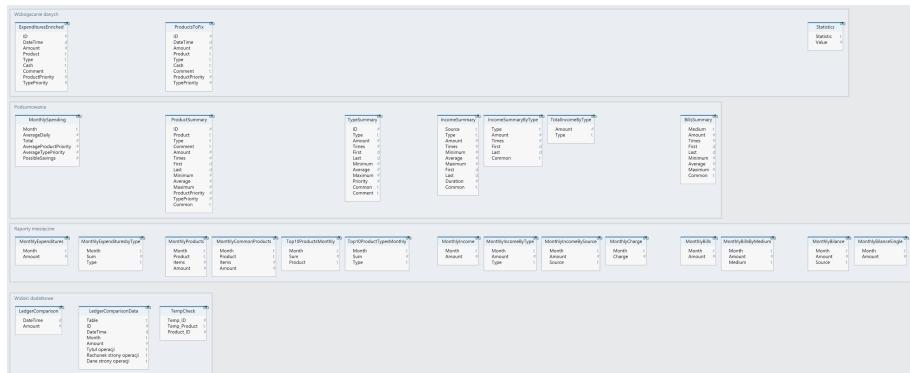
Do tworzenia dokumentacji wykorzystany zostanie pakiet narzędzi open source, między innymi będą to: StarUML [51] do tworzenia diagramów, pgAdmin [47] do tworzenia diagramów na podstawie encji bazy danych. Dokumentacja zostanie spisana w języku LaTex [52] wraz z gamą oficjalnych rozszerzeń dostępnych w sieci - zarówno kod jak i dokumentacja spisana w środowisku VSCode [54].

5.4 *Koncepcja przechowywania danych*

Podstawowy model schematu bazy danych przedstawia rysunek 5.3, który prezentuje Diagram Związków Encji (ERD, Entity-Relationship Diagram) tabel w szablonowym schemacie bazy danych aplikacji. Widoki prezentuje rysunek 5.4, przetwarzają one dane które prezentowane są użytkownikowi w aplikacji. Z racji na przyjęty model przyrostowy aplikacji liczba i logika widoków może ulec zmianie w toku prac nad projektem.



Rysunek 5.3: Diagram Związków Encji (ERD, Entity-Relationship Diagram)



Rysunek 5.4: Widoki w bazie danych aplikacji

Podstawowe tabele aplikacji: Tabela Income to zbiór przychodów, natomiast tabela Bills to zbiór okresowych wydatków stałych. Tabela ProductTypes zawiera dane o typach produktów, Tabela Products zawiera dane o Produktach które powiązane są z typami. Tabela Expenditures przechowuje zbiór wydatków okazjonalnych na określone produkty.

Tabele dotyczące użytkownika: W tabeli Users przechowywane są dane o użytkownikach aplikacji jak UUID, nazwa, hash hasła i ustawienia administracyjne, ich identyfikatorem jest UUID. Tabela UserSettings zawiera ustawienia prywatne użytkownika - jej struktura jest rozwijana wraz z rozwojem funkcji projektu.

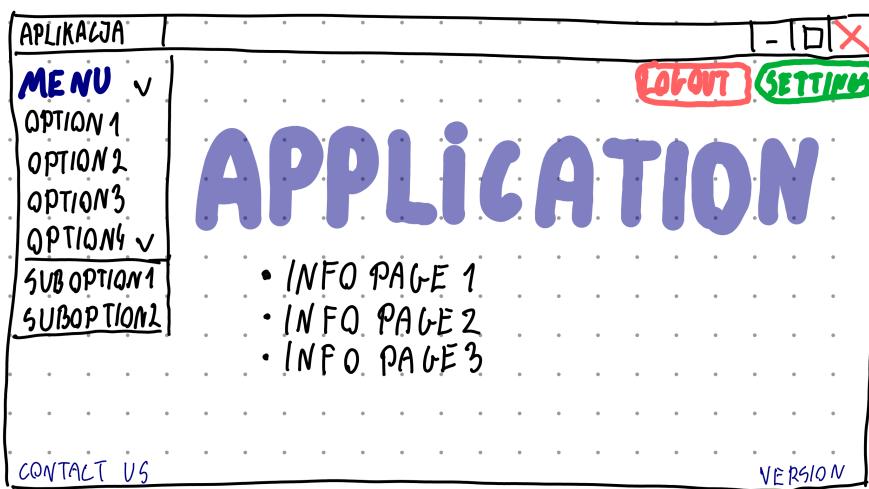
Tabele pomocnicze i tymczasowe: są to tabele robocze do zarządzania danymi w trakcie rozwoju, ich ilość, nazwy, struktura

oraz przeznaczenie mogą się zmieniać. Kiedy projektowana funkcja zostanie ukończona tabela wchodzi do użytku w ramach którejś z wcześniej opisanych kategorii.

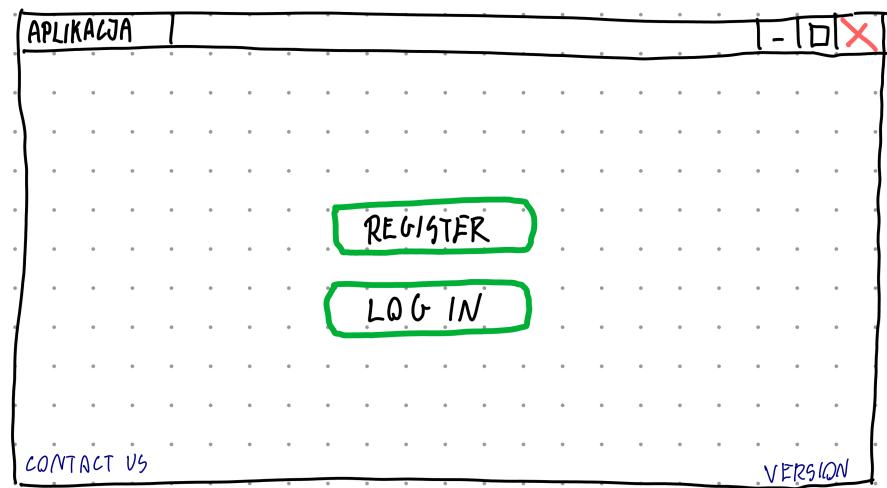
5.5 Projekt interfejsu użytkownika

Interfejs użytkownika będzie prosty, minimalistyczny aby ułatwić użytkownikowi poruszanie się po aplikacji i zmniejszyć obciążenie poznawcze. Dzięki temu opcje powinny być łatwo dostępne, a ryzyko że użytkownik się pogubi lub przeoczy opcję której poszukuje minimalne.

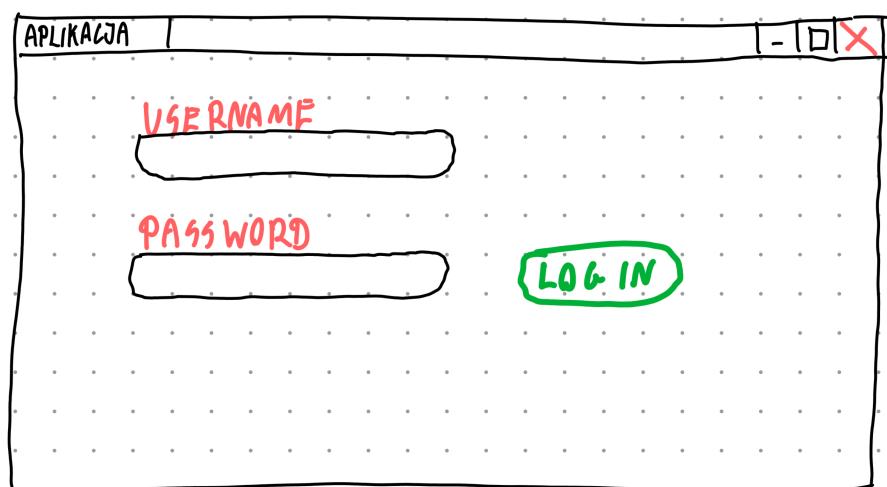
Pierwszym ekranem który napotka użytkownik jest tak zwany splash page którego projekt prezentuje rysunek 5.5, po wejściu w opcję login użytkownik trafi na ekran startowy który prezentuje rysunek 5.6. Rysunek 5.7 prezentuje ekran logowania. Zalogowany użytkownik będzie mógł zmienić swoje ustawienia na ekranie ustawień zobrazowanym przez rysunek 5.8 - funkcja ta jest funkcją dodatkową, zostanie zrealizowana jeśli wszystkie funkcje spełniające podstawowe założenia i wymagane do prawidłowego działania aplikacji zostaną wdrożone. Dodatkowo w aplikacji powstaną ekranu poradnika na wzór prezentowanego na rysunku 5.9 z podstawowymi informacjami o zarządzaniu finansami - we wstępnej fazie aby pokazać potencjał rozwiązania ekran będzie tylko jeden, pozostałe pozostaną puste lub zostaną wypełnione domyślnym tekstem, który w miarę rozwoju aplikacji zastąpią docelowe informacje ze sprawdzonych źródeł o dobrej reputacji.



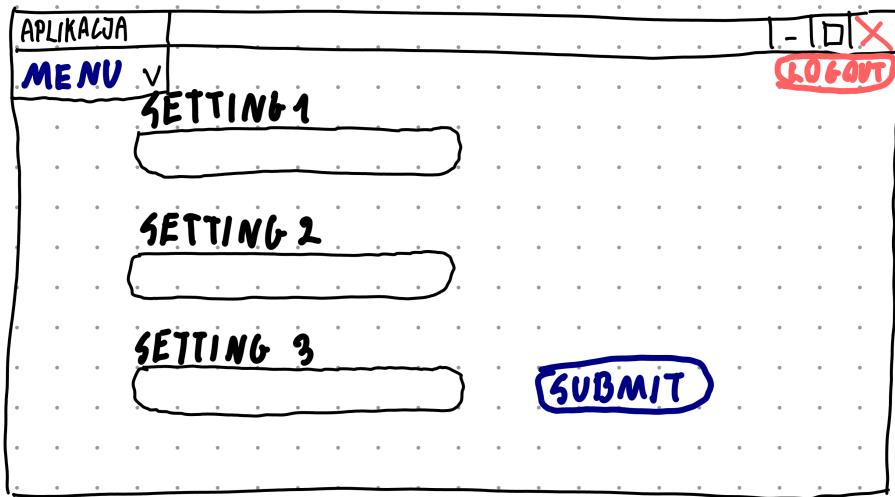
Rysunek 5.5: Projekt ekranu początkowego (splashscreen)



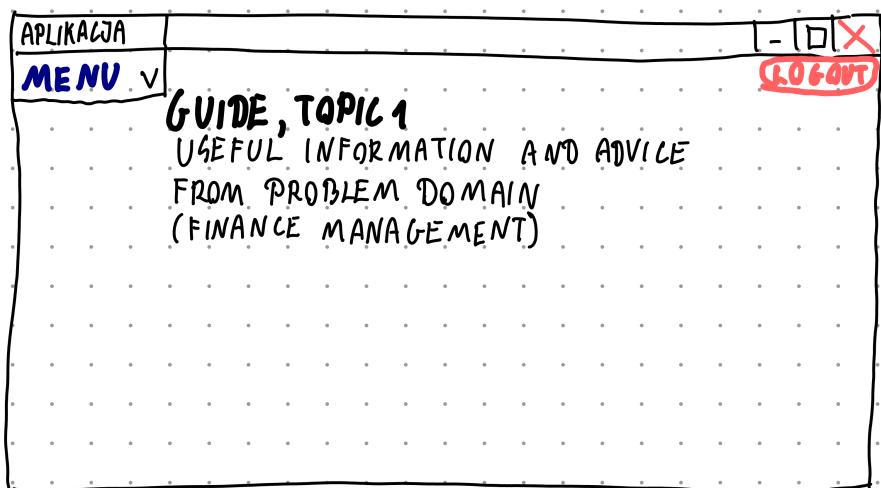
Rysunek 5.6: Projekt ekranu pierwszej interakcji



Rysunek 5.7: Projekt ekranu logowania



Rysunek 5.8: Projekt ekranu dodawania danych



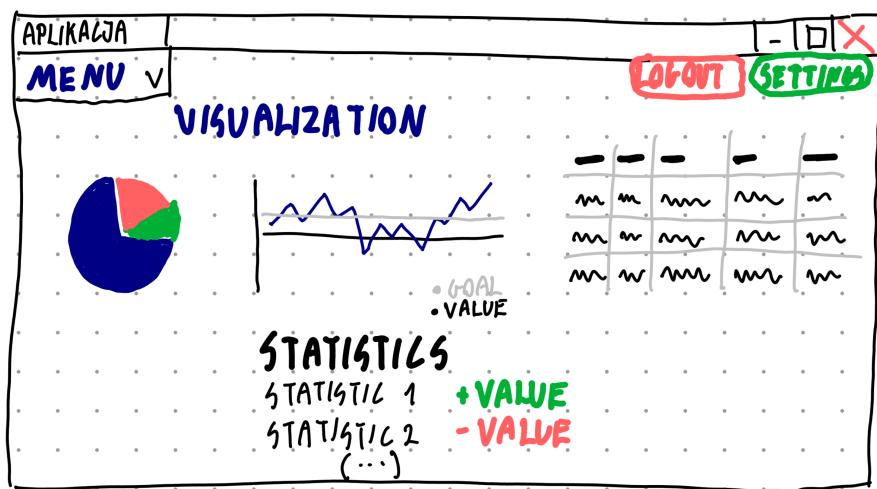
Rysunek 5.9: Projekt ekranu poradnika

Ekran zarządzania danymi prezentuje rysunek 5.10, jest to ekran który pozwala edytować dane dodane przez ekran dodawania danych przedstawiony na rysunku 5.11. W obu ekranach dane wprowadzane do aplikacji będą walidowane po stronie klienta przez logikę zaszytą w formularzach do wprowadzania danych dostarczoną przez wybrane rozwiązanie.

Rysunek 5.10: Projekt ekranu zarządzania danymi

Rysunek 5.11: Projekt ekranu dodawania danych

Na pozostałe ekranы aplikacji składać się będą różnego typu wizualizacje, poglądowo prezentowane przez projekt na rysunku 5.12. Liczba i złożoność wizualizacji zależeć będzie od nakładów pracy wymaganych do ich wdrożenia. Na wstępie użytkownikom udostępnione zostaną wyłącznie podstawowe wizualizacje, natomiast bardziej złożone i nowe uzupełniane będą z czasem w trakcie rozwoju aplikacji, w oparciu o uwagi użytkowników. Takie podejście zagwarantuje że aplikacja będzie maksymalnie przydatna do celów do których została stworzona - pomocy użytkownikom przy zarządzaniu budżetem.



Rysunek 5.12: Projekt ekranu dodawania danych

Rozdział 6

Dokumentacja techniczna

Rozdział ten zawiera dokumentację techniczną projektu. Dokumentacja jest opatrzona stosownymi diagramami, opisuje strukturę projektu wraz z ogólnym opisem przepływu wykonania kodu oraz logiki aplikacji. Znajdują się tu także opisy wybranych fragmentów kodu jak klasy, metody i funkcje oraz algorytmy.

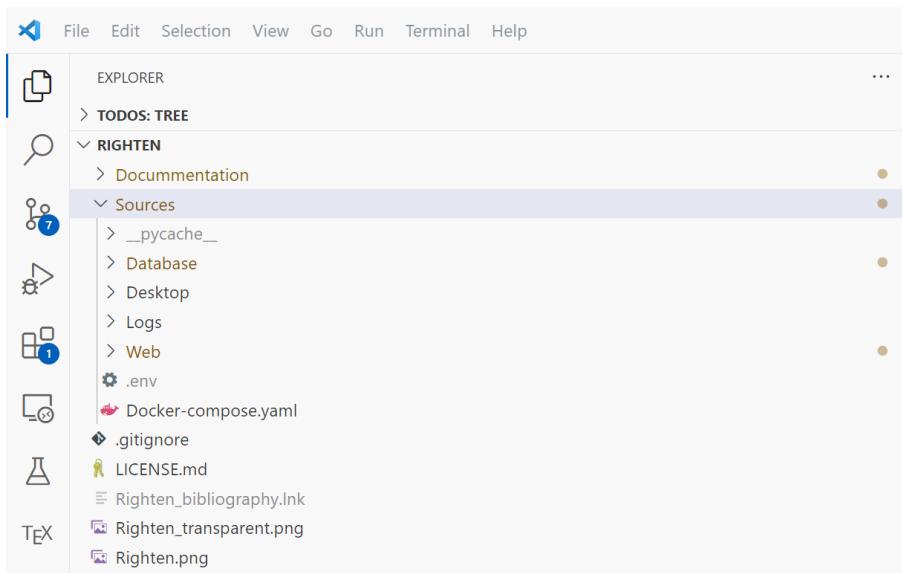
6.1 *Schemat rzeczywistej struktury systemu*

Struktura projektu w repozytorium Righten [59] prezentuje rysunek 6.1. Zgodnie z obowiązującymi dobrymi praktykami dla poprawy czytelności projekt podzielono na katalogi zależenie od ich funkcji. W głównym katalogu znajdują się katalogi Documentation który zawiera niniejszą dokumentację, Sources zawierający kod źródłowy aplikacji oraz plik licencji LICENSE.md który określa że obecnie oprogramowanie udostępniane jest na licencji GNU GENERAL PUBLIC LICENSE Version 2, June 1991 (może to jednak ulec zmianie w przyszłości).

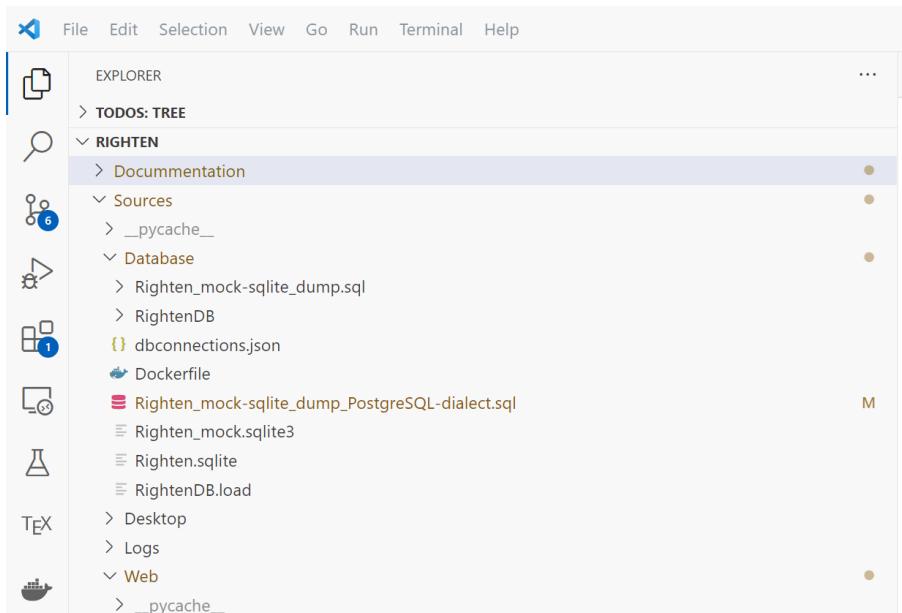
Zawartość katalogu Sources to katalog Desktop z poprzednią wersją projektu w formie aplikacji Budgeter na komputery osobiste - z jej kodem zapoznać się można w repozytorium DatabaseShenanigans [57] - nie jest to część aplikacji a osobny projekt, dlatego zostanie pominięta w dalszej części dokumentacji. Katalog Database zawiera wszystkie pliki bazy danych projektu, Logs przeznaczono na logi aplikacyjne do wykorzystania pod przyszłą funkcję rejestrowania zdarzeń, Web zawiera kod przeglądarkowej wersji aplikacji która jest głównym celem niniejszego projektu. Plik Docker-Compose.yaml zostanie omówiony później.

W skład zawartości katalogu Database przedstawionego na rysunku 6.2 wchodzą wstępna struktura bazy Righten.sqlite, jej wersję Righten_mock.sqlite3 wypełnioną danymi na potrzeby testów i rozwoju aplikacji, plik Righten_mock-sqlite-dump_PostgreSQL-dialect.sql

wykorzystany jako skrypt inicjalizacyjny bazy PostgreSQL zawartej w katalogu RightenDB, którym zmigrowano strukturę między technologiami oraz plik Dockerfile który zostanie omówiony w dalszej części dokumentacji.

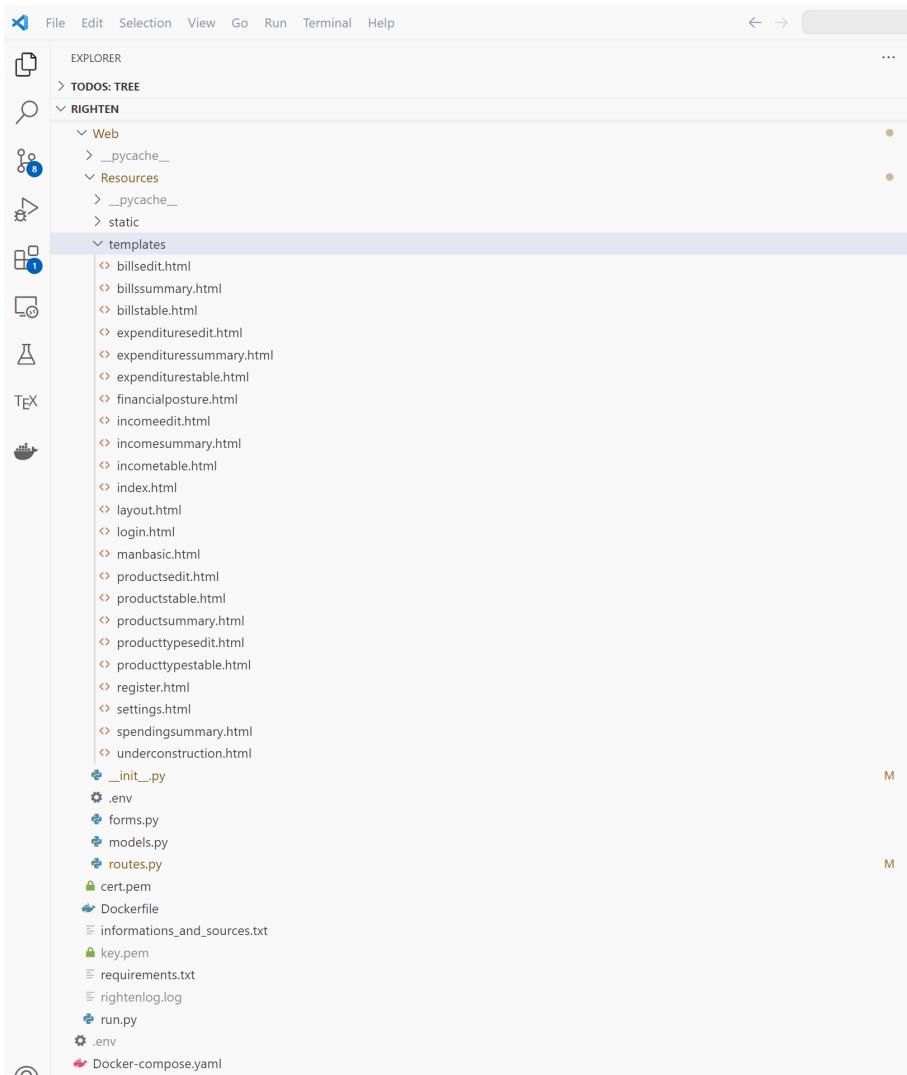


Rysunek 6.1: Struktura katalogu projektu



Rysunek 6.2: Struktura katalogu projektu - katalog Database

Katalog Web przedstawiony na rysunku 6.3 zawiera pełny kod aplikacji przeglądarkowej. Komunikacja zabezpieczona jest wygenerowanym na potrzeby projektu certyfikatem OpenSSL - plik cert.pem zawiera klucz publiczny (certyfikat), natomiast key.pem klucz prywatny którym aplikacja szyfruje ruch, dzięki czemu aplikacja do komunikacji wykorzystuje protokół HTTPS. Plik run.py służy do uruchamiania aplikacji, a wszystkie jej zasoby ładowane są z katalogu Resources. Pliki Dockerfile oraz requirements.txt omówione zostaną w dalszej części dokumentacji.



Rysunek 6.3: Struktura katalogu projektu - katalog Web

Katalog Resources także widoczny na rysunku 6.3 zawiera pliki ważne dla działania aplikacji, przedstawione w kolejności wywoływanego. Plik .env zawiera zmienne środowiskowe aplikacji, `__init__.py` to tak zwana fabryka aplikacji - inicjalizuje wymagane zmienne oraz konfigurację aplikacji, na końcu wczytując plik routes.py który definiuje ścieżki dostępne w przeglądarce i całą logikę która kryje się pod nimi. Ścieżki aplikacji dla użytkownika rozumiane jako strony w przeglądarce wykorzystują szablony zdefiniowane w katalogu templates, ponadto mogą wykorzystywać modele i formy. Modele danych definiuje plik models.py w którym na podstawie obiektów w bazie danych stworzono klasy które używane są w aplikacji zarówno do odpytywania bazy jak i przechowania w niej informacji. Plik forms.py natomiast przechowuje definicje i logikę form do wprowadzania danych wyświetlanych użytkownikowi na poszczególnych stronach. W katalogu static znajdują się statyczne elementy stron - obecnie jest to jedynie ikona aplikacji w formie pliku favicon.ico.

Napotkane po drodze pliki Docker-compose.yaml którego zawartość zawiera listing 6.1 oraz Dockerfile służą do powoływania środowiska aplikacji w formie kontenerów na platformie uruchomieniowej Docker. Infrastruktura aplikacji składa się z trzech komponentów powoływanych i konfigurowanych w pliku Docker-compose.yaml. Pierwszym komponentem jest baza danych PostgreSQL której konfigurację opisuje plik Sources\Database\Dockerfile, drugim jest system zarządzania bazą danych pgAdmin opisany w samym Docker-compose.yaml, ostatnim i najważniejszym sama aplikacja zdefiniowana w pliku Sources\Web\Dockerfile. Ponieważ katalog w którym kontener bazy przechowuje bazę danych oraz kontener aplikacji przechowuje logi mapowane są w ich definicjach do ścieżek na maszynie gospodarza dane te są przechowywane na stałe i nie są gubione podczas usunięcia, awarii kontenerów. Gotowe środowisko aplikacji uruchomione w Docker przedstawia rysunek 6.4.

Podczas uruchomienia kontenera aplikacji na systemie instalowane są wszystkie wymagane paczki bibliotek systemowych, wymienione są w pliku Sources \Web\requiremets.txt. Są to paczki Flask, flask-login, flask_bcrypt, flask_sqlalchemy, Flask-WTF, python-dotenv, cryptography, pyopenssl oraz psycopg2_binary.

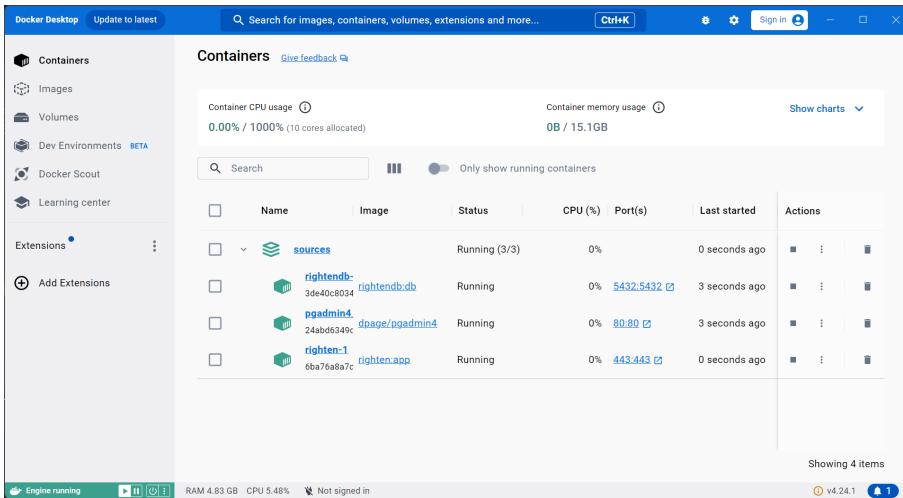
```

01 | #https://github.com/felipewom/docker-compose -
02 |   postgres
03 | services:
04 |   righten:
05 |     image:          'righten:app'
06 |     build:         ./Web/
07 |     restart:       always
08 |     command:      sh -c "sleep 10s ; python3
09 |       ./run.py" #Waiter script
10 |     ports:
11 |       - 443:443
12 |     stop_signal:    SIGINT #flask requirement
13 |     environment:
14 |       FLASK_SERVER_PORT: '443'
15 |     volumes:
16 |       - ./Logs:/logs
17 |     depends_on:
18 |       - rightendb
19 |
20 |   rightendb:
21 |     image:          'rightendb:db'
22 |     build:         ./Database/
23 |     restart:       always
24 |     ports:
25 |       - 5432:5432
26 |     environment:
27 |       POSTGRES_USER:   postgres
28 |       POSTGRES_PASSWORD: postgres
29 |       POSTGRES_DB:     RightenDB #DB created at
30 |       initialization
31 |       #Share catalog with container to persist data
32 |     volumes:
33 |       - ./Database/RightenDB/:/var/lib/postgresql/
34 |         data/
35 |         #Initialize data at first startup
36 |         - ./Database/Righten_mock-
37 |           sqlite_dump_PostgreSQL-dialect.sql:/docker-
38 |             entrypoint-initdb.d/init.sql
39 |
40 |   pgadmin:
41 |     image:          dpage/pgadmin4
42 |     container_name: pgadmin4_container
43 |     restart:       always
44 |     environment:
45 |       #Cardinal sin #1: Thou shall not divulge
46 |       your password. Done for sake of clarity
47 |       PGADMIN_DEFAULT_EMAIL: admin@righten.com
48 |       PGADMIN_DEFAULT_PASSWORD: testpass
49 |     ports:
50 |       - "80:80"
51 |     #Persistent server configuration
52 |     volumes:
53 |       - ./Database/dbconnections.json:/pgadmin4/
54 |         servers.json

```

Listing 6.1: Definicja infrastruktury aplikacji w Docker-compose.yaml

Definicja infrastruktury aplikacji w Docker-compose.yaml na podstawie której powoływane jest całe środowisko aplikacji.



Rysunek 6.4: infrastruktura aplikacji Righten uruchomiona w Docker

6.2 Wybrane fragmenty kodu aplikacji

W sekcji tej opisano po jednym przykładzie każdej z głównych funkcji programu.

Podczas projektowania aplikacji wielowarstwowych odwiecznym problemem jest utrzymanie spójnego modelu danych pomiędzy warstwami. Wraz z rozwojem projektu i pojawianiem się nowych potrzeb format oraz zakres danych wymaganych w interfejsie użytkownika mogą się zmieniać, jednocześnie niezależnie zmianom podlegają także struktury w bazie danych. W efekcie zmiany wprowadzane z czasem w modelu danych w obu warstwach wymagają coraz większych nakładów pracy po stronie logiki serwera aplikacji aby usługa działała poprawnie. W projekcie Righten zastosowano mechanizm refleksji [41] zaprezentowany na listingu 6.2 który ma za zadanie utrzymać spójność modelu danych między warstwami.

```

01 |     with app.app_context():
02 |         Base = automap_base()
03 |         Base.prepare(autoload_with=db.engine, reflect=True)
04 |         #Tables
05 |         Income=Base.classes.Income
06 |         Bills=Base.classes.Bills
07 |         ProductTypes=Base.classes.ProductTypes
08 |         Products=Base.classes.Products
09 |         Expenditures=Base.classes.Expenditures
10 |         #(...)
11 |
12 |         #Views
13 |         ExpendituresEnriched=Table("ExpendituresEnriched", db.
14 |             metadata, autoload_with=db.engine)
15 |         MonthlyBilance=Table("MonthlyBilance", db.metadata,
16 |             autoload_with=db.engine)
17 |         MonthlyBills=Table("MonthlyBills", db.metadata,
18 |             autoload_with=db.engine)
19 |         #(...)
20 |         #Map used to generalize functions - to refer to table by
21 |         #objectname alone
22 |         tables={
23 |             "Income": Income,
24 |             "Bills" : Bills,
25 |             "ProductTypes": ProductTypes,
26 |             "Products" : Products,
27 |             "Expenditures" : Expenditures,
28 |             #(...)
29 |             "ExpendituresEnriched" : ExpendituresEnriched,
30 |             "MonthlyBilance" : MonthlyBilance,
31 |             "MonthlyBills" : MonthlyBills
32 |             #(...)
33 |         }

```

Listing 6.2: Refleksja obiektów bazy

Refleksja obiektów bazy na klasy do użytku w warstwie aplikacji. Na uwagę zasługuje fakt iż mechanizm wczytywania tabel oraz widoków jest inny. Ponadto zmapowane obiekty klas przechowuje tabela pomocnicza tables używana w funkcjach których działanie jest niezależne od struktury wewnętrznej tabeli.

Formularze do wprowadzania danych przez użytkownika zdefiniowane w pliku forms.py wykorzystują framework WTForms. Ponieważ część danych w tabelach podstawowych się pokrywa, aby uprościć kod stworzono klasę bazową CommonForm która definiuje wszystkie pola wspólne i skorzystano z dziedziczenia aby w klasach pochodnych móc definiować wyłącznie pozostałe pola. Przykład tego mechanizmu prezentuje listing 6.3.

```

01 |     #Base class with common elements
02 |     class CommonForm(FlaskForm):
03 |         datetime = StringField("DateTime",
04 |                               validators=[DataRequired(),
05 |                                           Regexp("(:(?19|20)\\\
06 |                                         d\\d)-(0?[1-9]|1[012])-([12][0-9]|3[01]|0?[1-9])",
07 |                                           message="Date
08 |                                         must be in format: YYYY-MM-DD")
09 |                               ],
10 |                               default=date.today().isoformat()
11 |                               )
12 |
13 |         amount = DecimalField("Amount", validators=[
14 |             DataRequired()])
15 |         comment = StringField("Comment", validators[])
16 |         submit = SubmitField("SubmitField")
17 |
18 |     class IncomeInputForm(CommonForm):
19 |         with app.app_context():
20 |             types=[]
21 |             sources=[]
22 |             for type in db.session.query(Income.Type).distinct():
23 |                 types.append(type[0])
24 |                 for source in db.session.query(Income.Source).
25 |                     distinct():
26 |                         sources.append(source[0])
27 |
28 |             type = SelectField("Type", validators=[DataRequired()],
29 |                               choices=types,
30 |                               )
31 |             #NICE-TO-HAVE: display only sources viable for
32 |             #specified type
33 |             source = SelectField("Source", validators=[DataRequired(
34 |                               ),
35 |                               choices=sources
36 |                               ])

```

Listing 6.3: Formularze do wprowadzania danych przez użytkownika

Klasa bazowa CommonForm która zawiera pola wspólne dla wszystkich tabel oraz przykład klasy pochodnej IncomeInputForm która definiuje pola szczegółowe dla danej tabeli. Każde z pól określa klasę w której zdefiniowano jego nazwę, validatory oraz dane źródłowe do uzupełnienia list rozwijanych jeśli są wymagane. Ponieważ komunikacja w aplikacji jest zabezpieczona zapytania do bazy musi wykonać sama aplikacja co wymaga instrukcji `with app.app_context()`.

Strony zdefiniowane w pliku routes.py, wśród nich na wyróżnienie zasługują dwa główne typy: strony do zarządzania danymi widoczne na listingu 6.5 przez które użytkownik wprowadza, modyfikuje i usuwa swoje dane w aplikacji oraz strony z podsumowaniami jak zaprezentowana na listingu 6.4 które prezentują użytkownikowi dane w formie różnorodnych wizualizacji.

Znaczniki czasu w aplikacji przechowywane są w formacie zgodnym ze standardem ISO 8601 [38] ponieważ jednolity format ułatwia obsługę i przetwarzanie danych.

Dane między warstwami backend i frontend muszą być przesyłane w standardowym formacie aby zapewnić współpracę z dostępnymi na rynku bibliotekami i uprościć implementacje funkcji które będą je obsługiwać. Na potrzeby aplikacji wybrano format JSON [40].

```
01 |     @app.route("/incomesummary")
02 |     @flask_login.login_required
03 |     def incomesummary():
04 |         IncomeSummarydata=db.session.query(TotalIncomeByType).
05 |             all()
06 |         incometypesummary=[]
07 |         incometypes=[]
08 |         Summary = db.session.query(IncomeSummary).all()
09 |         for sum, type in IncomeSummarydata:
10 |             incometypesummary.append(sum)
11 |             incometypes.append(type)
12 |
13 |         IncomeOverTime=db.session.query(MonthlyIncome).all()
14 |         monthlyincomedata=[]
15 |         for Month, Amount in IncomeOverTime:
16 |             monthlyincomedata.append({"x": Month, "y": Amount})
17 |
18 |         IncomeTypesByTime=db.session.query(MonthlyIncomeByType).
19 |             all()
20 |         IncomeTypesByTimeDataset=createchartdataset(
21 |             IncomeTypesByTime)
22 |
23 |         return render_template("incomesummary.html",
24 |                               title="Income",
25 |                               IncomeSummarydata=json.dumps(
26 |                                   incometypesummary, cls=DecimalEncoder),
27 |                               IncomeSummarylabels=json.dumps(
28 |                                   incometypes, cls=DecimalEncoder),
29 |                               MonthlyIncome=json.dumps(
30 |                                   monthlyincomedata, cls=DecimalEncoder),
31 |                               IncomeTypesByTimeDataset=json.
32 |                               dumps(IncomeTypesByTimeDataset, cls=DecimalEncoder),
33 |                               Summary=Summary
34 | )
```

Listing 6.4: Strony podsumowań na przykładzie /incomesummary

Kazda strona prezentacji danych ma odmienny układ projektowany z myślą o funkcji jaką będzie pełnić dla użytkownika tak, aby przedstawić dane w sposób prosty, jasny i nie zaciemniać obrazu nadmiarem informacji. Każda z nich podobnie do przykładowej pobiera dane z bazy, przetwarza je do formatu kolekcji prezentowanych na grafach (osie x i y), lub w formie zestawów etykiet i wartości do prezentacji w formie tabel lub wykresów kołowych. Następnie warstwa serwera przetwarza dane do formatu json [40] którym wysyłane są do szablonów i prezentowane użytkownikowi przez frontend.

```

01 | #Basic data display
02 | -----
03 | @app.route("/income", methods=["GET", "POST"])
04 | @flask_login.login_required
05 | def income():
06 |     form = IncomeInputForm()
07 |     entries = db.session.query(Income).order_by(Income.
08 |         DateTime.desc(), Income.ID.desc()).all()
09 |     if form.validate_on_submit():
10 |         entry = Income(DateTime=date.fromisoformat(form.
11 |             datetime.data),
12 |                         Amount=form.amount.data,
13 |                         Type=form.type.data,
14 |                         Source=form.source.data,
15 |                         Comment=form.comment.data)
16 |         addtodb(entry)
17 |         return redirect(redirect_url(url_for("income", next
18 |             ="income")))
19 |     return render_template("incometable.html", title="Income",
20 |             entries=entries, form=form)

```

Listing 6.5: Strony zarządzania danymi na przykładzie /income

Aplikacja udostępnia użytkownikowi wprowadzanie danych przez dedykowany formularz (przykład opisano w listingu 6.3), oraz wyświetla dane już wprowadzone przez użytkownika wraz z opcjami usunięcia i edycji każdego z rekordów. Aplikacja sprawdza poprawność danych wprowadzonych przez użytkownika i wysyła je do bazy.

Przykładowy szablon strony wymaga szerszego opisu ponieważ zależnie od jego przeznaczenia, składać się może z kilku różnych sekcji. Punktem wyjścia jest szablon layout.html, który decyduje o ogólnym wyglądzie aplikacji - to w nim osadzone są elementy głównego paska nawigacyjnego menu i najważniejsze funkcje które powinny być dostępne wszędzie. Szablon ten rozszerza szablony poszczególnych stron. Większość stron podobnie jak na przykładowym listingu 6.6 w pewien sposób wyświetla dane przekazane tekstowo w formacie JSON [40]. Następnie w bloku skryptowym powstają z nich obiekty klas z biblioteki Chart.js [67] co prezentuje listing 6.7 które wyświetlane są w ramach wizualizacji zdefiniowanych w szablonie jak na przykładzie listingu 6.8. Na części stron występują także formularze do wprowadzania danych 6.9 oraz zestawienia danych podstawowych użytkownika przechowywane w bazie przedstawione na listingu 6.10 dzięki którym użytkownik może wyświetlić, edytować oraz usunąć wcześniej wprowadzone dane. Ponieważ każda tabela podstawowa w bazie danych aplikacji ma inny zestaw pól, powstały oddzielne strony edycji danych dla każdej z tabel, natomiast dla usunięcia utworzono jedną wspólną funkcję generyczną dla wszystkich tabel.

```

01 |   {%
02 |     block javascript %}
03 |     <script>
04 |       let IncomeSummarydata = JSON.parse({{
05 |         IncomeSummarydata | toJSON }})
06 |         let IncomeSummarylabels = JSON.parse({{
07 |           IncomeSummarylabels | toJSON }})
08 |             let MonthlyIncome = JSON.parse({{ MonthlyIncome | toJSON }})
09 |               let IncomeTypesByTimeDataset = JSON.parse({{
10 |                 IncomeTypesByTimeDataset | toJSON }})
11 |                   (...)
```

Listing 6.6: Przekazywanie danych do szablonu stron

Początek bloku skryptowego JavaScript [60] strony, frontend odbiera dane wysłane z backendu i przetwarza (parsuje) dane z tekstu do obiektu w formacie JSON [40].

```

01 |   let IncomeOverTime = new Chart(document.getElementById("IncomeOverTime"), {
02 |     type: "line",
03 |     data: {
04 |       datasets: [{
05 |         label: "Income",
06 |         data: MonthlyIncome,
07 |       }]
08 |     }
09 |   })
10 |
11 |   let IncomeTypesOverTime = new Chart(document.getElementById("IncomeTypesOverTime"), {
12 |     type: "line",
13 |     data: {
14 |       datasets: IncomeTypesByTimeDataset
15 |     }
16 |   })
```

Listing 6.7: Tworzenie obiektów wizualizacji w szablonach stron

Tworzenie obiektów klas chart.js.

```

01 |   <div class="col-xs-10, col-sm-6">
02 |     <div class="card card-style mb-2">
03 |       <div class="card-body">
04 |         <div class="chart-container" style="position: relative;">
05 |           <canvas id="IncomeOverTime"></canvas>
06 |         </div>
07 |       </div>
08 |     </div>
09 |   </div>
```

Listing 6.8: Panale wizualizacji w szablonach stron

Komponenty które służą do prezentacji przygotowanych wizualizacji użytkownikowi.

```

01 |     <div class="col-sm-10 ml-auto mr-auto">
02 |         <form action="" method="POST">
03 |             {{ form.csrf_token() }}
04 |             <fieldset class="form-group">
05 |                 <legend class="mb-4">Add Income</legend>
06 |                 <div class="row form-group">
07 |                     <div class="col form-group">
08 |                         {{ form.datetime.label(class="form-control-
label") }}
09 |                         {% if form.datetime.errors %}
10 |                             {{ form.datetime(class="form-control form-
control-sm is-invalid") }}
11 |                             {% for error in form.datetime.errors %}
12 |                                 <div class="invalid-feedback">
13 |                                     <span>{{error}}</span>
14 |                                 </div>
15 |                             {% endfor %}
16 |                             {% else %}
17 |                             {{ form.datetime(class="form-control form-
control-sm") }}
18 |                             {% endif %}
19 |                         </div>
20 |                         <!--(...)-->
21 |
22 |                         <div class="col form-group">
23 |                             {{ form.source.label(class = "form-control-
label") }}
24 |                             {{ form.source(class = "form-control form-
control-sm") }}
25 |                         </div>
26 |
27 |                         <div class="col form-group">
28 |                             {{ form.comment.label(class = "form-control-
label") }}
29 |                             {{ form.comment(class = "form-control form-
control-sm") }}
30 |                         </div>
31 |
32 |                         <div class="col">
33 |                             {{ form.submit(class="btn btn-success",
value="Add record")}}
34 |                         </div>
35 |                     </div>
36 |                 </fieldset>
37 |             </form>
38 |         </div>
39 |         <br></br>

```

Listing 6.9: Formularze wprowadzania w szablonach stron

Formularze wprowadzania danych występują na stronach zarządzania danymi. Każde pole formularza ma odpowiednią kontrolkę z validacją danych i obsługę błędów oraz przycisk do zatwierdzenia formularza który wysyła wprowadzone przez użytkownika dane do aplikacji. Komunikację zabezpiecza token CSRF (Cross-Site Request Forgery) [42], dzięki czemu aplikacja nie przyjmuje żądań HTTP wysłanych z zewnątrz aplikacji.

```

01 |     <div class="col-sm-10 ml-auto mr-auto">
02 |         <H2>Incomes</H2>
03 |         <table class="table" >
04 |             <thead class="thead-dark" >
05 |                 <tr>
06 |                     <th scope="col">ID</th>
07 |                     <th scope="col">DateTime</th>
08 |                     <th scope="col">Amount</th>
09 |                     <th scope="col">Type</th>
10 |                     <th scope="col">Source</th>
11 |                     <th scope="col">Comment</th>
12 |                     <th scope="col">Edit</th>
13 |                     <th scope="col">Delete</th>
14 |                 </tr>
15 |             </thead>
16 |             <tbody>
17 |                 {% for entry in entries %}
18 |                     <tr>
19 |                         <td>{{entry.ID}}</td>
20 |                         <td>{{entry.DateTime.strftime("%Y-%m-%d
")}}</td>
21 |                         <td>{{entry.Amount}}</td>
22 |                         <td>{{entry.Type}}</td>
23 |                         <td>{{entry.Source}}</td>
24 |                         <td>{{entry.Comment}}</td>
25 |                         <td><a href="{{ url_for('incomeedit',
26 |                             table='Income', entry_id = entry.ID )}" class="btn btn-
outline-warning btn-sm" id="EditButton">Edit</a></td>
26 |                         <td><a href="{{ url_for('delete',
27 |                             table='Income', entry_id = entry.ID )}" class="btn btn-
outline-danger btn-sm" id="DeleteButton">Delete</a></td
28 |                         >
29 |                     </tr>
30 |                 {% endfor %}
31 |             </tbody>
32 |         </table>
33 |     </div>

```

Listing 6.10: Zestawienia danych w szablonach stron

Aplikacja na podstawie dostarczonego w postaci JSON obierktu z rekordami w locie tworzy tabelę z wszystkimi danymi a także linkami do edycji oraz usunięcia dla każdego z rekordów.

6.3 Wybrane fragmenty kodu bazy danych

Baza danych aplikacji stworzona została na podstawie wymagań, następnie była stopniowo rozszerzana, modyfikowana i usprawniana w trakcie rozwoju projektu - zarówno kodu aplikacji, kolejnych iteracyjnych usprawnień wprowadzanych do samych założeń funkcji oraz w odpowiedzi na wykryte błędy i nowe wymogi danych powstałe w trakcie implementacji. W toku prac tabele podstawowe rozszerzono o dodatkowe pola jak isCash w tabeli Expenditures której defini-

cję prezentuje listing 6.12 (przed zmianami 6.11), oraz większość widoków.

```
01 | CREATE TABLE [Expenditures] (
02 |     [ID]           INTEGER,
03 |     [DateTime]    DATETIME,
04 |     [ProductID]   INTEGER UNSIGNED,
05 |     [Amount]       DOUBLE,
06 |     [Comment]      TEXT DEFAULT NULL,
07 |     PRIMARY KEY([ID] AUTOINCREMENT),
08 |     FOREIGN KEY([ProductID]) REFERENCES [Products]([ID])
09 | );
```

Listing 6.11: Tabela Expenditures - wersja wstępna

Tabela Expenditures zawiera wydatki wprowadzone przez użytkownika. Listing przedstawia pierwszą wstępnią wersję tabeli w technologii SQLite3 [43].

```
01 | CREATE TABLE IF NOT EXISTS public."Expenditures"
02 | (
03 |     "ID" integer NOT NULL DEFAULT nextval('
04 |         Expenditures_ID_seq'::regclass),
05 |     "DateTime" date,
06 |     "ProductID" bigint,
07 |     "Amount" numeric(10,2),
08 |     "Comment" text COLLATE pg_catalog."default",
09 |     "isCash" boolean DEFAULT false,
10 |     CONSTRAINT "Expenditures_pkey" PRIMARY KEY ("ID"),
11 |     CONSTRAINT "Expenditures_ProductID_fkey" FOREIGN KEY ("
12 |         ProductID")
13 |         REFERENCES public."Products" ("ID") MATCH SIMPLE
14 |         ON UPDATE NO ACTION
15 |         ON DELETE NO ACTION
16 | )
17 |
18 | TABLESPACE pg_default;
19 | ALTER TABLE IF EXISTS public."Expenditures"
20 |     OWNER to postgres;
```

Listing 6.12: Tabela Expenditures - wersja wstępna

Tabela Expenditures zawiera wydatki wprowadzone przez użytkownika. Listing przedstawia finalną wersję tabeli w technologii PostgreSQL [46].

Zgodnie z założeniami projektu widoki przejęły ciężar przetwarzania danych. W trakcie rozwoju projektu ulegały szeregowi zmian. Uzupełniano je, czego przykładem może być pola Cash, ProductPriority i TypePriority w widoku ExpendituresEnriched wymienionym na listingu 6.13 - we wstępnej wersji projektu pola te nie występowały ponieważ funkcje które je wykorzystują powstały później etapie. Powstały także zupełnie nowe widoki jak ProductSummary widoczny na listingu 6.14 który oblicza przydatne dane statystyczne

prezentowane później użytkownikowi zamiast surowych danych z tabeli podstawowej o produktach. Wstępnie zaprojektowane skomplikowane widoki podzielono także na mniejsze aby uprościć zapytania jeśli logika przetwarzania kilku widoków zawierała części wspólne - między innymi MonthlyCommonProducts który podzielono na podstawowy MonthlyProducts oraz MonthlyCommonProducts widoczne na listingu 6.15 którego logika stała się dzięki temu o wiele krótsza a więc i łatwiejsza w utrzymaniu.

```
01 | CREATE VIEW "ExpendituresEnriched" AS
02 | SELECT
03 |     "Expenditures"."ID"          AS "ID",
04 |     "Expenditures"."DateTime"    AS "DateTime",
05 |     "Expenditures"."Amount"      AS "Amount",
06 |     "Products"."Product"        AS "Product",
07 |     "ProductTypes"."Type"       AS "Type",
08 |     (CASE WHEN
09 |         ("Expenditures"."isCash"=FALSE)
10 |         THEN 'NO' ELSE 'YES' END) AS "Cash",
11 |     "Expenditures"."Comment"    AS "Comment",
12 |     "Products"."Priority"       AS "ProductPriority", --Nowe
13 |     "ProductTypes"."Priority"   AS "TypePriority"      --Nowe
14 |     pole
15 |     FROM "Expenditures"
16 |     LEFT JOIN "Products"
17 |     ON "Expenditures"."ProductID"="Products"."ID"
18 |     LEFT JOIN "ProductTypes"
19 |     ON "Products"."TypeID"="ProductTypes"."ID"
20 |     ORDER BY "DateTime";
```

Listing 6.13: Widok rozwinięty w toku projektu - ExpendituresEnriched

Widok ExpendituresEnriched prezentuje dane o wydatkach użytkownika w czytelny sposób, wzbogacone o dodatkowe pola które nie występują w tabeli podstawowej.

```

01 | CREATE VIEW "ProductSummary" AS
02 | SELECT
03 |     "Products"."ID"                                AS "ID"
04 |     , "Products"."Product"                         AS "Product"
05 |     , "ProductTypes"."Type"                       AS "Type"
06 |     , "Products"."Comment"                        AS "Comment"
07 |     , COALESCE("Summary"."Amount", 0)             AS "Amount"
08 |     , COALESCE("Summary"."BoughtTimes", 0)        AS "Times"
09 |     , "Summary"."FirstBought"                    AS "First"
10 |     , "Summary"."LastBought"                     AS "Last"
11 |     , "Summary"."Minimum"                        AS "Minimum"
12 |     , "Summary"."Average"                        AS "Average"
13 |     , "Summary"."Maximum"                        AS "Maximum"
14 |     , "Products"."Priority"                     AS "ProductPriority"
15 |     , "ProductTypes"."Priority"                 AS "TypePriority"
16 |     , COALESCE("Summary"."Common", 'Absent')    AS "Common"
17 | FROM "Products"
18 | LEFT JOIN (SELECT
19 |     *
20 |     ,(CASE WHEN ("BoughtTimes">>(
21 |         SELECT
22 |             AVG("BoughtTimes") AS "Average"
23 |             FROM (
24 |                 SELECT
25 |                     "Product"
26 |                     , Round(SUM("Amount"), 2) AS "Amount"
27 |                     , COUNT(TO_CHAR("DateTime", 'YYYY-MM-DD')) AS "BoughtTimes"
28 |                     , MAX(TO_CHAR("DateTime", 'YYYY-MM-DD')) AS "LastBought"
29 |                     , MIN(TO_CHAR("DateTime", 'YYYY-MM-DD')) AS "FirstBought"
30 |                 FROM "ExpendituresEnriched"
31 |                 GROUP BY "Product"
32 |                 ORDER BY "BoughtTimes" DESC)))
33 |             THEN 'Common' ELSE 'Uncommon' end) AS "Common"
34 |             FROM (
35 |                 SELECT
36 |                     "Product"
37 |                     , Round(SUM("Amount"), 2)                               AS "Amount"
38 |                     , COUNT(TO_CHAR("DateTime", 'YYYY-MM-DD')) AS "BoughtTimes"
39 |                     , MAX("DateTime")                                     AS "LastBought"
40 |                     , MIN("DateTime")                                    AS "FirstBought"
41 |                     , ROUND(AVG("Amount"), 2)                            AS "Average"
42 |                     , MIN("Amount")                                     AS "Minimum"
43 |                     , MAX("Amount")                                     AS "Maximum"
44 |                 FROM "ExpendituresEnriched"
45 |                 GROUP BY "Product"
46 |                 ORDER BY "BoughtTimes" DESC))
47 |             AS "Summary"
48 |             ON "Products"."Product"="Summary"."Product"
49 |             LEFT JOIN "ProductTypes" ON
50 |                 "Products"."TypeID"="ProductTypes"."ID";

```

Listing 6.14: Widok nowy - ProductSummary

Widok ProductSummary zawiera podsumowanie o zakupionych przez użytkownika produktach w formie danych statystycznych.

```

01 | CREATE VIEW "MonthlyProducts" AS
02 | SELECT *
03 | FROM (
04 |     SELECT TO_CHAR("DateTime", 'YYYY-MM') AS "Month",
05 |             "Product",
06 |             COUNT("Product") AS "Items",
07 |             ROUND(SUM("Amount"), 2) AS "Amount"
08 |     FROM "ExpendituresEnriched"
09 |     GROUP BY "Product", "Month"
10 |     ORDER BY "Month" DESC, "Amount" DESC
11 | );
12 |
13 | CREATE VIEW "MonthlyCommonProducts" AS
14 | SELECT *
15 | FROM "MonthlyProducts"
16 | WHERE "Items">>(
17 |     SELECT AVG("Items")
18 |     FROM "MonthlyProducts");

```

Listing 6.15: Widok podzielony - MonthlyCommonProducts

Początkowo był to jeden widok jednak aby wyliczyć średnią ilość produktów zapytanie wymagało powielenia logiki.

Ponadto część przetwarzania z samej aplikacji stopniowo przeniesiona zostaje do widoku Statistics, którego wynik prezentowany jest użytkownikowi w jednym z paneli. Widok ten zawiera w sobie niezwiązane ze sobą zapytania które dostarczają jednak przydatnych z punktu widzenia użytkownika informacji - wyniki z przykładowych danych testowej wersji aplikacji obrazuje rysunek 6.5.

The screenshot shows the pgAdmin interface. In the Object Explorer, under the 'public' schema, there is a 'Statistics' node which is expanded, showing various statistics like 'FIRE savings requirement', 'Average bills year to date', etc. A query window is open with the following SQL:

```

1 SELECT * FROM public."Statistics"
2

```

The results pane displays the following data:

Statistic text	Value
FIRE savings requirement	1686129.00
Average bills year to date	2440.28
Average income year to date	10453.46
Net Worth	227754.89
Average spending year to date	5620.43
Average expenditures year to date	3180.15

Rysunek 6.5: infrastruktura aplikacji Righten uruchomiona w Docker

Rozdział 7

Testy i weryfikacja systemu

Niniejszy rozdział opisuje proces testowania oraz weryfikacji systemu. Aby zapewnić możliwie szybką implementację zaprojektowanych funkcji zastosowano najprostszą metodę testów którą są testy manualne. Testowaniu podlegały zarówno pojedyncze funkcje, moduły jak i aplikacja jako całość, zależnie od etapu rozwoju aplikacji oraz typu implementowanej zmiany. Przykłady wykrytych błędów zaprezentowano w kolejnej sekcji tego rozdziału.

Po migracji aplikacji na środowisko uruchomieniowe Docker oraz zmianie technologii bazy danych z SQLite na PostgreSQL testowanie a zarazem i rozwój aplikacji stały się bardziej skomplikowane. Ponieważ kod uruchamiany na platformie Docker jest niejako gotowym środowiskiem wdrożonej aplikacji, standardowe funkcje środowiska programistycznego jak możliwość zatrzymania wykonywanego kodu czy inspekcji wartości zmiennych są niedostępne. Z tego powodu w konfiguracji zaszyto przełącznik w formie zmiennej debug_version, cały kod z nim związany rozsiany jest po kilku plikach - ich wycinki prezentuje listing 7.1.

```

01 |     #Zawartosc .env
02 |     FLASK_SQLALCHEMY_DATABASE_URI=postgresql+psycopg2://
03 |             postgres:postgres@rightendb:5432/RightenDB
04 |
05 |     #zawartosc __init__.py
06 |     import os
07 |     from dotenv import load_dotenv
08 |     basepath = os.path.abspath(os.path.dirname(__file__))
09 |     load_dotenv(os.path.join(basepath, '.env'))
10 |
11 |     app=Flask(__name__)
12 |     app.config.from_prefixed_env() #Reads FLASK_* from .env and
13 |             .flaskenv
14 |     version="debug_local"      #Decyduje o technologiach
15 |
16 |     #Nadpisanie polaczenia do bazy danych aplikacji w trakcie
17 |             rozwoju
18 |     if version=="debug_local":
19 |         app.config['SQLALCHEMY_DATABASE_URI']='sqlite:///E:\\\\
20 |             Projects\\\\Git\\\\Righten\\\\Sources\\\\Database\\\\Righten_mock
21 |                 .sqlite3' #Local SQLite3
22 |
23 |     db=SQLAlchemy(app)

```

Listing 7.1: Fragmenty kodu prezentujące mechanizm debug_version

Mechanizm ten decyduje o wartości zmiennych środowiskowych wskazujących ważne pliki wymagane do uruchomienia aplikacji. W efekcie zmieniając wartość w jednym miejscu można przełączyć aplikację w tryb testowania lub wdrożenia wykorzystujące rozbieżne technologie.

Ponieważ ciężar przetwarzania danych aplikacji położono na warstwie bazy danych, to logika zawarta zapytaniach z których złożone są obiekty bazy danych jak widoki podlegała najbardziej rygorystycznym testom. Po utworzeniu zapytania do bazy danych aplikacji wprowadzano zestaw danych testowych, dla których wyniki przetwarzania przygotowano ręcznie oraz z wykorzystaniem narzędzi z pakietu Google Docs [53]. Wyniki uzyskane ręcznie porównywano następnie z danymi wynikowymi przygotowanej funkcji, co pozwalało na bieżąco eliminować błędy w najbardziej istotnej logice aplikacji. Na etapie testów jako bazę danych aplikacji wykorzystano manualnie utworzoną instancję SQLite3 [43], oraz narzędzie DB Browser for SQLite [44] do interakcji z bazą danych. Baza ta na dalszym etapie projektu zmigrowana została do technologii PostgreSQL [46] i narzędzia pgAdmin [47]. Finalnie bazę danych wypełniono danymi testowymi które służą jako przykład informacji wprowadzonych przez użytkownika z pomocą specjalnie napisanego w tym celu skryptu. Skrypt ten powielał wybraną losowo część rekordów zmieniając wskazane dane w określonym zakresie - przykładowo cena*liczbaLosowaWZakresie(0.8,1.2).

W warstwie backend przeprowadzano głównie manualne testy akceptacyjne dla każdej implementowanej funkcji aby określić czy działa zgodnie z założeniami wstępnymi projektu. Test każdej wdrożonej funkcji wymagał uruchomienia aplikacji i potwierdzenia jej prawidłowego działania. Część funkcji wymagała zmiany w interfejsie, w takich przypadkach test sprowadzał się do wykonania określonej akcji i potwierdzenia że jej efekt jest zgodny z założonym. Weryfikacja zależała od wdrażanej funkcji, kilka z wybranych przykładów opisano poniżej.

Sprawdzenie czy dane wprowadzone przez interfejs trafiły do bazy aplikacji.

Po dodaniu panelu logowania weryfikacja obejmowała próbę wejścia do panelu wizualizacji bez logowania, próby wprowadzenia błędnych poświadczeń (tak nazwy użytkownika jak i hasła) i weryfikację czy zalogowany użytkownik może prawidłowo poruszać się po aplikacji. Test po dodaniu zabezpieczenia sesji użytkownika opisano w kolejnej sekcji jako ciekawy przypadek.

Warstwa interfejsu użytkownika podlegała głównie manualnych testach integracyjnych, testach nawigacji między elementami aplikacji, wizualnej inspekcji dodanych elementów oraz interakcji poszczególnych kontrolek w przeglądarce. Wykryte błędy sprowadzały się głównie do sytuacji w których ułożenie elementów odbiega od założeń projektowych, wybrane elementy interfejsu były nieczytelne lub interakcja z nimi nie działała prawidłowo. Testy obejmowały także określenie jak bardzo czytelny, intuicyjny i prosty w obsłudze jest interfejs aplikacji, jednakże ponieważ jest to kwestia subiektywna, zarówno oceny jak i zmian dokonywano arbitralnie. Wśród wykrytych błędów najczęstszym były błędy formatowania tekstu, czy elementy które wstępnie nie mieściły się w wyznaczonym obszarze. Ich naprawa natomiast sprowadzała się do zmiany odpowiedzialnego atrybutu w szablonie HTML i ponownej weryfikacji.

7.1 *Najciekawsze wykryte błędy*

W miarę rozwoju projektu podczas implementacji poszczególnych funkcji wykryto i wyeliminowano wiele błędów, niektóre jednak wyróżniają się od reszy na kilka ciekawych sposobów a zatem zasługują na szczególną.

Pierwszą kategorią tego typu błędów wykrytych w aplikacji były błędy logiki nie widoczne na pierwszy rzut oka. Za przykład posłuży obliczanie miesięcznej średniej dziennych wydatków. Ponieważ pytanie budowane było etapami w których stopniowo rozbudowywano je

o dodatkowe informacje, jego logika była dość zawiła. Na kolejnych fazach projektu podczas usprawnień warstwy bazy danych zapytanie zostało zoptymalizowane - części wspólne przeniesiono do osobnego podzapytania zbierającego dane dzienne, z którego następnie obliczano dane miesięczne. Efektem prac był kod widoczny na listingu 7.2, który mimo iż na pierwszy rzut oka wygląda prawidłowo oblicza średnie miesięczne a średnie ze średnich dziennych. Błąd logiki wykryto podczas ręcznej weryfikacji danych testowych, a logika po naprawie błędu (i zmianie dialekta SQL) widoczna jest na listingu 7.3.

```
01 |   SELECT
02 |     strftime('%Y-%m',[Day])                      AS [Month]
03 |     ,ROUND(AVG([DailyAmount]))                   AS [AverageDaily]
04 |     ,ROUND(SUM([DailyAmount]))                  AS [Total]
05 |     ,ROUND(AVG([DailyCashPercentage]))        AS [CashPercentage]
06 |     ,ROUND(AVG([AverageDailyProductPriority]))
07 |       AS [AverageProductPriority]
08 |     ,ROUND(AVG([AverageDailyTypePriority]))
09 |       AS [AverageTypePriority]
10 |     ,ROUND(SUM([PossibleSavings]),2)           AS [PossibleSavings]
11 |   FROM (
12 |     SELECT
13 |       strftime('%Y-%m-%d',[DateTime])          AS [Day]
14 |       ,SUM([Amount])                          AS [DailyAmount]
15 |       ,SUM([Amount]*[ProductPriority])/SUM([Amount])
16 |         AS [AverageDailyProductPriority]
17 |       ,SUM([Amount]*[TypePriority])/SUM([Amount])
18 |         AS [AverageDailyTypePriority]
19 |       ,SUM((CASE WHEN ([ProductPriority]<(
20 |         SELECT [Value]
21 |         FROM [UserSettings]
22 |         WHERE [Setting] = "ProductPriorityTarget"))
23 |           THEN [Amount] ELSE 0 END))    AS [PossibleSavings]
24 |       ,AVG((CASE WHEN [Cash] = 'YES'
25 |           THEN 100 ELSE 0 END))      AS [DailyCashPercentage]
26 |     FROM [ExpendituresEnriched]
27 |     WHERE [Amount]>0
28 |     GROUP BY [DateTime]
29 |   )
30 |   GROUP BY [Month]
31 |   ORDER BY [Month] ASC;
```

Listing 7.2: Widok MonthlySpending - średnia średnich dziennych (dialekt SQLite)

Zapytanie SQL w dialekcie SQLite sprzed migracji bazy na technologię PostgreSQL. Przykładowo: jeśli w miesiącu zakupiono 50 produktów, jednego dnia 2 za gotówkę (DailyCashPercentage 100%), w drugim dniu 48 kartą (DailyCashPercentage 0%), widok prezentował informację że 50% produktów miesięcznie zakupiono za gotówkę ($0+100/2=50$), podczas gdy faktycznie wartość ta wynosiła 4% ($2/50$).

```

01 | --Create intermediary table
02 | WITH "Daily" AS (
03 |   --Average daily data by month
04 |   SELECT
05 |     TO_CHAR("DateTime", 'YYYY-MM') AS "Month"
06 |     ,ROUND(AVG("DailyAmount"),2) AS "AverageDaily"
07 |   FROM (
08 |     --Get daily amounts
09 |     SELECT
10 |       "DateTime",
11 |       SUM("Amount") AS "DailyAmount"
12 |     FROM "ExpendituresEnriched"
13 |     GROUP BY "DateTime"
14 |   )
15 |   GROUP BY "Month"
16 | )
17 |
18 | --Necessary for Alias
19 | SELECT
20 |   "Monthly".*,
21 |   "Daily"."AverageDaily"
22 | FROM (
23 |   SELECT
24 |     TO_CHAR("DateTime", 'YYYY-MM') AS "Month"
25 |     ,ROUND(SUM("Amount"),2) AS "Total"
26 |     ,ROUND(AVG(CASE WHEN "Cash"='YES'
27 |                   THEN 100 ELSE 0 END)) AS "CashPercentage"
28 |     ,ROUND(AVG("ProductPriority"))
29 |       AS "AverageProductPriority"
30 |     ,ROUND(AVG("TypePriority"))
31 |       AS "AverageTypePriority"
32 |     ,ROUND(SUM(CASE WHEN ("ProductPriority"><CAST((
33 |                   SELECT "Value"
34 |                   FROM "UserSettings"
35 |                   WHERE "Setting"='ProductPriorityTarget')
36 |                   AS NUMERIC))
37 |                   THEN "Amount" ELSE 0 END),2)
38 |       AS "PossibleSavings"
39 |   FROM "ExpendituresEnriched"
40 |   GROUP BY "Month"
41 |   ORDER BY "Month" ASC
42 | ) AS "Monthly"
43 | LEFT JOIN "Daily" ON "Monthly"."Month"="Daily"."Month";

```

Listing 7.3: Widok MonthlySpending - poprawiona średnia średnich dziennych (dialekt PostgreSQL)

Zapytanie SQL w dialekcie PostgreSQL po migracji bazy. Zapytanie oblicza osobno średnie miesięczne oraz dodaje średnie dzienne.

Z ciekawszych problemów które wykryto w warstwie backend na uwagę zasługuje problem z przechowywaniem haseł użytkowników. W trakcie implementacji panelu logowania użytkownika we wstępnej wersji dane wprowadzone przez użytkownika były szyfrowane funkcją bcrypt, a następnie porównywano skrót (hash) hasła wprowadzonego z przechowywanym w bazie danych. Mimo wpisywania prawidłowego hasła uwierzytelnianie za każdym razem kończyło się niepowodze-

niem, ponieważ funkcja bcrypt automatycznie generuje także sól - generowany pseudolosowo fragment danych dodawany do hasła przed zaszyfrowaniem aby zwiększyć entropię (losowość danych) - więc w efekcie porównywane hasła zawsze były różne. Po wgłębianiu się w dokumentację biblioteki zmieniono kod aplikacji tak, by wykorzystać funkcję bcrypt.check_password_hash która porównuje zapisany skrót hasła z danymi w formie jawnej. Poprawiony fragment kodu prezentuje listing 7.4.

```
01 |     @app.route('/login', methods=['GET', 'POST'])
02 |     def login():
03 |         form=LoginForm()
04 |         if request.method == "POST" and form.validate_on_submit():
05 |             username=form.username.data
06 |             usr=db.session.query(Users).filter_by(username=username).first()
07 |             #if user does not exist password is not checked
08 |             if usr and bcrypt.check_password_hash(usr.Password,
09 |                 form.password.data.encode('utf-8')):
10 |                 user = User()
11 |                 user.id = username
12 |                 flask_login.login_user(user)
13 |                 flash("User logged in", "success")
14 |                 return redirect(redirect_url(url_for("index",
15 |                     next="index")))
16 |             flash("Failed to log in", "danger")
17 |             return redirect(redirect_url(url_for("login", next=
"index")))
18 |     return render_template("login.html", title="Login",
form=form)
```

Listing 7.4: Panel logowania - weryfikacja zgodności hasła

Funkcja ta wykorzystuje sól zapisaną na początku docelowego skrótu, dodając go do testowanego hasła, hashując je i porównując ciągi - w wyniku zwraca jedynie informację czy hash jest prawidłowy.

W warstwie prezentacji danych natomiast ciekawy problem wywiązał się podczas poprawy zabezpieczeń w aplikacji. Po dodaniu tokenu CSRF (Cross-site Request Forgery) aplikacja straciła dostęp do bazy danych. Ponieważ dane wymagane do połączenia aplikacji z bazą danych oraz instancja silnika który utrzymuje połączenie i wykonuje zapytania jest częścią konfiguracji wstępnej aplikacji, zabezpieczenie sesji w ramach aplikacji z wykorzystaniem klucza szyfrowania spowodowało blokowanie wykonywanych poza sesją prób połączenia z bazą funkcji która mapowała model bazy danych na obiekty wykorzystywane w aplikacji działającej w osobnym module models.py. Rozwiążanie problemu wymagało importu we wskazanym module obiektu app reprezentującego aplikację w pamięci i dodania polecenia

with app.context które wykonuje polecenia zawarte w bloku kodu w ramach sesji aplikacji. Problematyyczny fragment modułu wraz z opisanymi zmianami zaprezentowano na listingu 7.5.

```
01 |     from Resources import db, app
02 |     #(...)
03 |     with app.app_context():
04 |         Base = automap_base()
05 |         Base.prepare(autoload_with=db.engine, reflect=True)
06 |         #Tables
07 |         Income=Base.classes.Income
08 |         Bills=Base.classes.Bills
09 |         ProductTypes=Base.classes.ProductTypes
10 |         #(...)
```

Listing 7.5: Fragment modułu models.py - kouminkacja z bazą z wykorzystaniem kontekstu aplikacji

Wprowadzenie zabezpieczenia CSRF wymagało zmiany sposobu mapowania obiektów bazy danych na obiekty aplikacji.

Rozdział 8

Przykładowy scenariusz wykorzystania systemu

Rozdział 9

Zakończenie

Celem pracy było zaprojektowanie i realizacja modułu analitycznego aplikacji ułatwiającej jej użytkownikom zarządzanie budżetem domowym dostarczając narzędzia do analizy wpływów i wydatków, wizualizacji trendów oraz automatycznie kategoryzującej wprowadzone dane.

Bibliografia

- [1] George S. Clason, (2021) Najbogatszy człowiek w Babilonie, ISBN: 978-83-67060-04-2
- [2] Marcin Iwuć, (2020) Finansowa Forteca, ISBN: 978-83-958468-0-9
- [3] Krzysztof Piotr Łabenda, (2011) Budżet domowy pod kontrolą. Jak rozsądnie wydawać, oszczędzać i inwestować pieniądze, ISBN: 978-83-246-3626-6
- [4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, (2010) Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku
- [5] Al Sweigart, Automate the Boring Stuff with Python
<https://automatetheboringstuff.com/#toc>
- [6] Wikipedia, Nauki Ekonomiczne
https://pl.wikipedia.org/wiki/Nauki_ekonomiczne
- [7] Wikipedia, Economic collapse
https://en.wikipedia.org/wiki/Economic_collapse
- [8] Główny Urząd Statystyczny, Budżety gospodarstw domowych w 2022 roku
<https://stat.gov.pl/obszary-tematyczne/warunki-zycia/dochody-wydatki-i-warunki-zycia-ludnosci/budzety-gospodarstw-domowych-w-2022-roku,9,21.html>
- [9] Łukasz Grygiel, Jak wygląda edukacja finansowa dzieci i młodzieży w Polsce?
<https://web.archive.org/web/20230529115945/https://lukaszgrygiel.com/edukacja-finansowa-dzieci-i-mlodziezy/>
- [10] Bank Pekao, Raport Banku Pekao: „Dziecięcy świat finansów - jak rynek finansowy odpowiada na potrzeby

najmłodszych klientów". <https://www.pekao.com.pl/o-banku/aktualnosci/d4e423aa-0ba4-4bde-8a0a-7ff3a17a9793/raport-banku-pekaod-zieciecy-swiat-finansow-jak-rynek-finansowy-odpowiada-na-potrzeby-najmłodszych-klientow.html>

- [11] Krajowy Rejestr Długów, Portfel statystycznego Polaka w pandemii <https://krd.pl/centrum-prasowe/raporty/2022/portfel-statystycznego-polaka-w-pandemii>
- [12] Warsaw Enterprise Institute, [RAPORT] Żegnajcie niskie ceny? Koszty życia i poziom cen w Polsce na tle krajów UE w latach 2010–2022 <https://wei.org.pl/2022/aktualnosci/wiktorwojciechowski/raport-zegnajcie-niskie-ceny-koszty-zycia-i-poziom-cen-w-polsce-na-tle-krajow-ue-w-latach-2010-2022/>
- [13] Warsaw Enterprise Institute, [RAPORT] Jak inflacja zubaża Polaków?
<https://wei.org.pl/2023/publikacje/raporty/mateusz-benedyk/raport-jak-inflacja-zubaza-polakow/>
- [14] Opcje24, Budżetowanie <https://www.opcje24h.pl/budżetowanie-przewodnik-planowanie-budzetu/>
- [15] /marciniwuc.com, Budżet domowy krok po kroku <https://marciniwuc.com/budzet-domowy-krok-po-kroku/>
- [16] ING Bank Śląski, Jak zapanować nad budżetem domowym?
<https://spolecznosc.ing.pl/-/Blog/Jak-zapanowa%C4%87-nad-bud%C5%BCetem-domowym/ba-p/3968>
- [17] The Balance, Understanding Budgeting & Personal Finance
<https://www.thebalancemoney.com/personal-finance-budget-4802696>
- [18] www.mint.intuit.com, Budgeting 101
<https://mint.intuit.com/blog/category/budgeting/>
- [19] Simply Scholar, Ltd., Positive Reinforcement: What Is It and How Does It Work? <https://www.simplypsychology.org/positive-reinforcement.html>
- [20] Frederick Chong, Gianpaolo Carraro, and Roger Wolter, Microsoft Corporation, Multi-Tenant Data Architecture
<https://ramblingsofraju.com/wp-content/uploads/2016/08/Multi-Tenant-Data-Architecture.pdf>

-
- [21] Santander, Centrum24.pl <https://www.centrum24.pl/>
 - [22] Inuit inc., Mint <https://mint.intuit.com/>
 - [23] Ryan McGregor, Mint Budgeting App: How to Setup and Use a Budget (BEST WAY)
https://www.youtube.com/watch?v=rQ_5v3BUBqQ
 - [24] theverge.com, Mint is shutting down, and it's pushing users toward Credit Karma
<https://mint.intuit.com/blog/mint-app-news/intuit-credit-karma-welcomes-minters/>
 - [25] Dayspring Partners, Goodbudget
<https://goodbudget.com/>
 - [26] <https://www.cnbc.com/select/best-free-budgeting-tools/>
<https://www.cnbc.com/select/best-free-budgeting-tools>
 - [27] <https://www.forbes.com/advisor/banking/best-budgeting-apps/> <https://www.forbes.com/advisor/banking/best-budgeting-apps/>
 - [28] <https://www.tomsguide.com/best-picks/best-budgeting-apps> <https://www.tomsguide.com/best-picks/best-budgeting-apps>
 - [29] <https://www.investopedia.com/best-budgeting-apps-5085405> <https://www.investopedia.com/best-budgeting-apps-5085405>
 - [30] <https://www.nerdwallet.com/article/finance/best-budget-apps> <https://www.nerdwallet.com/article/finance/best-budget-apps>
 - [31] Tiller Inc., Tiller <https://www.tillerhq.com/>
 - [32] Product Plan, MOSCOW Prioritization <https://www.productplan.com/glossary/moscow-prioritization/>
 - [33] Praca.pl, Matryca Eisenhowera - czym jest, zasada, priorytyzacja zadań https://www.praca.pl/poradniki/rynek-pracy/matryca-eisenhowera-czym-jest,zasa-da,priorytyzacja-zadan_pr-2012.html
 - [34] Wikipedia, Minimal Viable Product https://en.wikipedia.org/wiki/Minimum_viable_product

-
- [35] Lean Action Plan, Kanban – układ nerwowy sterowania produkcją w koncepcji Lean Manufacturing
<https://leanactionplan.pl/kanban/>
 - [36] Wikipedia, Lean software development https://pl.wikipedia.org/wiki/Lean_software_development
 - [37] Wikipedia, Model Przyrostowy
https://pl.wikipedia.org/wiki/Model_przyrostowy
 - [38] NASA.gov, A summary of the international standard date and time notation
<https://fits.gsfc.nasa.gov/iso-time.html>
 - [39] Y. Shafranovich, SolidMatrix Technologies, Inc., Common Format and MIME Type for Comma-Separated Values (CSV) Files <https://www.rfc-editor.org/rfc/rfc4180>
 - [40] json.org, Introducing JSON
<https://www.json.org/json-en.html>
 - [41] Uniwersytet w Białymostku Wydział Informatyki Justyna Korycińska, Programowanie refleksyjne <http://ii.uwb.edu.pl/rudnicki/wp-content/uploads/2016/02/P25.pdf>
 - [42] OWASP foundation, Cross-Site Request Forgery Prevention Cheat Sheet https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
 - [43] sqlite.org, SQLite <https://www.sqlite.org/index.html>
 - [44] sqlitebrowser.org, DB Browser for SQLite
<https://sqlitebrowser.org>
 - [45] wikipedia.org, SQL - Structured Query Language
<https://en.wikipedia.org/wiki/SQL>
 - [46] postgresql.org, PostgreSQL
<https://www.postgresql.org/>
 - [47] pgadmin.org, pgAdmin <https://www.pgadmin.org/>
 - [48] docker.com, Docker <https://www.docker.com/>
 - [49] python.org, Python <https://www.python.org/>
 - [50] Atlassian, Trello.com <https://trello.com/>
 - [51] MKLabs Co.,Ltd, StarUML <https://staruml.io/>

-
- [52] The LaTeX Project <https://www.latex-project.org/>
 - [53] Google Docs <https://docs.google.com/>
 - [54] Microsoft, Visual Studio Code
<https://code.visualstudio.com/>
 - [55] git-scm.com, git <https://git-scm.com/>
 - [56] <https://github.com> <https://github.com>
 - [57] [github.com MarcinNowak94, DatabaseShenanigans](https://github.com/MarcinNowak94/DatabaseShenanigans) <https://github.com/MarcinNowak94/DatabaseShenanigans>
 - [58] [github.com MarcinNowak94, budgeter](https://github.com/MarcinNowak94/budgeter)
<https://github.com/MarcinNowak94/budgeter>
 - [59] [github.com MarcinNowak94, Righten](https://github.com/MarcinNowak94/Righten)
<https://github.com/MarcinNowak94/Righten>
 - [60] Mozilla, JavaScript <https://developer.mozilla.org/en-US/docs/Web/javascript>
 - [61] getbootstrap.com, Bootstrap <https://getbootstrap.com/>
 - [62] Django Software Foundation, Django
<https://www.djangoproject.com/>
 - [63] flask.palletsprojects.com, Flask
<https://flask.palletsprojects.com/en/3.0.x/>
 - [64] <https://pypi.org/project/Flask-Login/>, Flask-Login
<https://pypi.org/project/Flask-Login/>
 - [65] wtforms.readthedocs.io, WTForms
<https://wtforms.readthedocs.io/en/3.1.x/>
 - [66] jinja.palletsprojects.com, Jinja
<https://jinja.palletsprojects.com/en/3.1.x/>
 - [67] www.chartjs.org, Chart.js <https://www.chartjs.org/>

Spis rysunków

2.1	Główny Urząd Statystyczny, Dynamika realnych dochodów i wydatków na 1 osobę w gospodarstwach domowych w latach 2010–2022 [8]	7
2.2	300gospodarka.pl, Inflacja w Polsce w latach 1996–2023 dane GUS	7
2.3	oszczedzaniepieniedzyblog.pl, Najprostszy budżet – wydatki	8
2.4	oszczedzaniepieniedzyblog.pl, Najprostszy budżet – przychody	9
3.1	https://marciniwuc.com/budzet-domowy-05-wydatkinieregularne/ Przykładowy budżet w aplikacji Excel	13
3.2	https://jakoszczedzaczepieniadze.pl/prosty-budzet-domowy Przykładowy budżet na papierze	14
3.3	Centrum24.pl, Santander historia transakcji	15
3.4	Centrum24.pl, Santander przegląd kategorii wydatków	16
3.5	Centrum24.pl, Santander kategoryzacja wydatków	16
3.6	Centrum24.pl, Santander przegląd wydatków	17
3.7	Mint w wersji mobilnej	19
3.8	Mint w wersji na przeglądarki	19
4.1	Microsoft, Architektura współdzielona baza, rozdzielne schematy	23
4.2	http://framework.gigr.pl/ Architektura Trójwarstwowa	24
5.1	Pirorytetyzacja MoSCoW	25
5.2	Tablica zadań projektu w usłudze Trello	28
5.3	Diagram Związków Encji (ERD, Entity-Relationship Diagram)	30
5.4	Widoki w bazie danych aplikacji	30
5.5	Projekt ekranu początkowego (splashscreen)	31
5.6	Projekt ekranu pierwszej interakcji	32
5.7	Projekt ekranu logowania	32

5.8	Projekt ekranu dodawania danych	33
5.9	Projekt ekranu poradnika	33
5.10	Projekt ekranu zarządzania danymi	34
5.11	Projekt ekranu dodawania danych	34
5.12	Projekt ekranu dodawania danych	35
6.1	Struktura katalogu projektu	37
6.2	Struktura katalogu projektu - katalog Database . . .	37
6.3	Struktura katalogu projektu - katalog Web	38
6.4	infrastruktura aplikacji Righten uruchomiona w Docker	41
6.5	infrastruktura aplikacji Righten uruchomiona w Docker	52

Spis tabel

5.1 Wymagania niefunkcjonalne	26
5.2 Wymagania funkcjonalne	26

Listings

6.1	Definicja infrastruktury aplikacji w Docker-compose.yaml	40
6.2	Refleksja obiektów bazy	42
6.3	Formularze do wprowadzania danych przez użytkownika	43
6.4	Strony podsumowań na przykładzie /incomesummary	44
6.5	Strony zarządzania danymi na przykładzie /income	45
6.6	Przekazywanie danych do szablonu stron	46
6.7	Tworzenie obiektów wizualizacji w szablonach stron	46
6.8	Panale wizualizacji w szablonach stron	46
6.9	Formularze wprowadzania w szablonach stron	47
6.10	Zestawienia danych w szablonach stron	48
6.11	Tabela Expenditures - wersja wstępna	49
6.12	Tabela Expenditures - wersja wstępna	49
6.13	Widok rozwinięty w toku projektu - ExpendituresEnriched	50
6.14	Widok nowy - ProductSummary	51
6.15	Widok podzielony - MonthlyCommonProducts	52
7.1	Fragmenty kodu prezentujące mechanizm debug_version	54
7.2	Widok MonthlySpending - średnia średnich dziennych (dialekt SQLite)	56
7.3	Widok MonthlySpending - poprawiona średnia średnich dziennych (dialekt PostgreSQL)	57
7.4	Panel logowania - weryfikacja zgodności hasła	58
7.5	Fragment modułu models.py - kouminkacja z bazą z wykorzystaniem kontekstu aplikacji	59