



POLITECHNIKA KRAKOWSKA im. T. Kościuszki
Wydział Inżynierii Elektrycznej i Komputerowej



Kierunek studiów: Informatyka w inżynierii komputerowej

STUDIA STACJONARNE

INŻYNIERIA PROGRAMOWANIA

PROJEKT

Bartosz Szarłowicz
Maciej Walczak
Antoni Wałach

„TWORZENIE OPROGRAMOWANIA DLA FIRMY KURIERSKIEJ”

Prowadzący projekt:
Prof. Dr hab. Inż. Sergii Telenyk

Kraków, 2023

Spis treści

Wstęp	4
Cel projektu	4
Opis problemu	4
Założenia projektu	5
Zakres produktu	7
Wymagania	7
Wymagania funkcjonalne	7
Wymagania нефункционалне	8
Cele biznesowe	9
Wpływ oprogramowania na biznes	11
Analiza ograniczeń	13
Analiza ryzyka	15
Plan biznesowy	18
Analiza wzorców architektonicznych i wybór optymalnego wzorca	21
Główne funkcje	21
Architektura mikroservisów	22
Architektura monolityczna	22
Architektura MVC (Model-View-Controller)	23
Architektura warstwowa	24
Wybór optymalnego wzorca	24
Powody wybrania architektury mikroservisów	25
Rodzaje infrastruktur projektu	26
Modelowanie powiązań struktury i architektury projektu	27
Wzorzec Komponentowo-Łącznikowy	27
Wzorzec Alokacyjny	27
Wzorzec Modułowy	28
Technologie	28
Interfejs użytkownika - UI	29
Narzędzia programistyczne	29
Słownik	30
Architektura systemu	35
Diagram klas	37
Interfejs admina	37
Interfejs klient	39

Klasa kurier	41
Procesy biznesowe	43
Rejestracja użytkownika	43
Kod źródłowy - rejestracja użytkownika	44
Interfejs użytkownika (UI) - rejestracja użytkownika	44
Logowanie użytkownika	45
Kod źródłowy - logowanie użytkownika	45
Interfejs użytkownika (UI) - logowanie użytkownika	46
Wysyłanie paczki	47
Kod źródłowy - wysyłanie paczki	47
Interfejs użytkownika (UI) - wysyłanie paczki	48
Aktualizacja statusu przesyłki	49
Zmiana danych użytkownika	50
Wypłata pieniędzy	52
Wpłata pieniędzy na konto	54
Wydruk ZPL	56
Przyjmowanie zamówień	58
Historia i ocena zamówień	60
Baza danych	63
Schemat bazy danych	64
Wyniki fazy testowania	65
Rozwiązanie napotkanych problemów	66
Proces wdrożenia projektu	68
Perspektywy rozwoju projektu	70
Dokumentacja użytkowa	72
Widok klienta	72
Widok kuriera	77
Widok administratora	80

Wstęp

Dokument jest dokumentacją projektu przygotowanego na przedmiot Inżynieria Programowania. Projekt realizuje zespół: Bartosz Szarłowicz, Antoni Wałach, Maciej Walczak.

Cel projektu

Oprogramowanie ma na celu usprawnienie procesu wysyłania paczek poprzez prosty, intuicyjny interfejs. Zakres projektu obejmuje stworzenie kompleksowego systemu obsługi wysyłek, który umożliwi klientom szybkie wystawienie paczki do wysłania, a kurierom wygodną realizację przesyłek. Naszym priorytetem jest zapewnienie kompleksowego narzędzia do obsługi klientów oraz zapewnienie kurierom pracy w elastyczny sposób.

Opis problemu

Z perspektywy klienta:

Tradycyjne metody wysyłki paczek często sprawiają klientom i kurierom wiele trudności. Długie kolejki, skomplikowane procedury, daleka droga do punktu wysyłki - to tylko część problemów, z którymi się borykamy na rynku usług kurierskich. Istnieje potrzeba usprawnienia tego procesu, zapewniając klientom łatwość, szybkość i wygodę wysyłania paczki, jednocześnie umożliwiając kurierom wygodne realizowanie zleceń oraz elastyczność pracy. Ten problem zmotywował nas do stworzenia aplikacji, która stawia sobie za cel rozwiązanie tych wyzwań poprzez prosty i intuicyjny interfejs, dostarczając kompleksowy system obsługi wysyłek.

Z perspektywy firmy kurierskiej:

Firma kurierska obsługuje duże ilości przesyłek każdego dnia, co generuje liczne trasy dostaw i wymaga efektywnego zarządzania flotą pojazdów. Ręczne planowanie tras i koordynacja dostaw może być czasochłonne i podatne na błędy, co prowadzi do opóźnień, zwiększa koszty paliwa oraz obniża ogólną efektywność operacyjną firmy.

Aplikacja kurierska może wprowadzić automatyczne planowanie tras, wykorzystując zaawansowane algorytmy optymalizacyjne. Dzięki zbieraniu danych na temat lokalizacji paczek, pojazdów oraz informacji na temat ruchu drogowego, aplikacja może dynamicznie dostosowywać trasy dostaw w czasie rzeczywistym.

Założenia projektu

Skoncentrowanie na Użytkowniku: Projekt opiera się na filozofii "Użytkownik w centrum", gdzie każda decyzja projektowa i funkcjonalność aplikacji jest opracowana z myślą o zapewnieniu optymalnego doświadczenia dla klientów i kurierów. Prostota obsługi, szybkość działania oraz łatwość nawigacji są kluczowymi założeniami.

Elastyczność i Rozszerzalność: Aplikacja projektowana jest z myślą o łatwej rozszerzalności o nowe funkcjonalności w przyszłości. System ma być elastyczny, pozwalając na dodawanie nowych opcji i usprawnień zgodnie z ewoluującymi potrzebami klientów oraz dynamicznym rynkiem usług kurierskich.

Bezpieczeństwo Danych i Ustabilizowana Wydajność: Bezpieczeństwo danych osobowych klientów oraz stabilność działania aplikacji to priorytety. Projekt zakłada zastosowanie najnowszych standardów bezpieczeństwa oraz regularne testy wydajności, aby zapewnić płynne działanie platformy nawet przy wzroście liczby użytkowników.

Uniwersalność i Kompatybilność: Aplikacja ma działać na różnych urządzeniach i przeglądarkach, aby zapewnić szeroki dostęp do usług kurierskich. Kompatybilność na wielu platformach jest kluczowa, umożliwiając obsługę klientów o różnych preferencjach w korzystaniu z technologii.

Sprawne Zarządzanie Zleceniami i Klientami: Priorytetem jest stworzenie intuicyjnych paneli administracyjnych dla klientów i kurierów, zapewniających szybkie i skuteczne zarządzanie zleceniami oraz danymi użytkowników.

Koszty:

- **Rozwój aplikacji:** Szacowany koszt na rozwój aplikacji kurierskiej wynosi 120 000 zł, uwzględniając zespół programistów, testowanie, infrastrukturę IT itp.
- **Wsparcie i utrzymanie:** Roczny koszt utrzymania aplikacji oraz wsparcia technicznego na poziomie 180 000 zł.
- **Bezpieczeństwo danych:** Inwestycja w systemy bezpieczeństwa danych, szacowany koszt 50 000 zł.

Terminy:

- **Rozwój aplikacji:** Planowany czas rozwinięcia pierwszej wersji aplikacji to 6 miesięcy.
- **Etap testów i poprawek:** Po rozwinięciu aplikacji, przewiduje się 2 miesiące na testy, poprawki i optymalizacje.
- **Wdrożenie:** Planowany termin wdrożenia aplikacji na rynek to 8 miesięcy od rozpoczęcia projektu.

Wymagania do zespołu:

- **Programiści Full Stack:** trzy osób do pracy nad front-endem, back-endem i bazą danych.
- **Testerzy:** dwie osoby odpowiedzialnych za testy manualne i automatyzację testów.
- **Architekt systemowy:** dwie osoby do nadzorowania i projektowania całej infrastruktury.
- **Specjaliści ds. bezpieczeństwa:** zespół ekspertów do zapewnienia bezpieczeństwa danych.

Technologie:

- **Front-end:** Wykorzystanie React.js dla interfejsu użytkownika.
- **Back-end:** Użycie Node.js w zależności od preferencji i skali aplikacji.
- **Baza danych:** MySQL dla trwałego przechowywania danych.
- **Bezpieczeństwo:** Implementacja protokołów SSL/TLS, zabezpieczeń warstwy aplikacji, szyfrowania danych.

Zakres produktu

Zakresem produktu jest utworzenie kompleksowej aplikacji webowej, której towarzyszyć będzie specjalnie dostosowany system bazodanowy. Ten system będzie obsługiwany przez wielu użytkowników jednocześnie. Naszym celem jest zaprojektowanie interfejsu, który będzie nie tylko prosty i intuicyjny dla osób z doświadczeniem w obsłudze zaawansowanych technologii, ale również dla tych, którzy nie są biegli w korzystaniu z nowoczesnych narzędzi internetowych. Chcemy, aby użytkownicy o różnym stopniu zaawansowania technologicznego mieli łatwy dostęp do funkcji aplikacji, co pozwoli im wygodnie i efektywnie korzystać z wszystkich dostępnych usług.

Zakres produktu obejmuje stworzenie kompleksowej aplikacji typu Software as a Service (SaaS), zintegrowanej z dedykowanym systemem bazodanowym obsługiwany przez wielu użytkowników jednocześnie.

W naszym projekcie uzasadnione będzie wybranie modelu SaaS, ponieważ umożliwia on szybkie wdrożenie, elastyczne skalowanie i dostępność funkcji online, co przyczyni się do lepszej obsługi klienta oraz efektywności operacyjnej.

Wymagania

Wymagania funkcjonalne

1. Możliwość rejestracji konta klienta oraz kuriera.
2. Formularz umożliwiający klientom wprowadzenie danych paczki i adresu dostawy.
3. Panel administracyjny dla zarządzania paczkami i użytkownikami.
4. Interfejs dla kurierów umożliwiający akceptację zleceń.
5. Interfejs dla klientów umożliwiający wyświetlanie informacji o własnych przesyłkach.
6. Możliwość zarządzania kontami klienta oraz kuriera.
7. System ocen.
8. System sprawdza poprawność wprowadzonych danych.

Wymagania niefunkcjonalne

- 1. Interfejs użytkownika i doświadczenie użytkownika zapewniające prostotę i intuicyjność obsługi.**
 - Średnia liczba kliknięć potrzebnych do złożenia zamówienia - 8.
 - Średni czas rejestracji nowego użytkownika – 2 minuty.
- 2. Wysoka dostępność systemu i stabilność aplikacji.**
 - Czas działania systemu bez awarii - 99% dostępności rocznie.
 - Czas reakcji na awarie i naprawy - średnio 30 minut na rozwiązanie problemu krytycznego.
- 3. Szybkość działania platformy nawet przy dużym obciążeniu.**
 - Średni czas odpowiedzi serwera przy 1000 równoczesnych żądaniach - poniżej 500ms.
 - Wydajność dostawy danych przy dużym obciążeniu - 95% danych dostarczonych w czasie krótszym niż 2 sekundy.
- 4. Odporność na błędy, obsługa błędów krytycznych.**
 - Średni czas do zidentyfikowania krytycznego błędu - mniej niż 30 minut.
 - Procentowe ograniczenie dostępu do systemu w przypadku błędu krytycznego - mniej niż 1% użytkowników.
- 5. Optymalność, niskie obciążenie oraz zużycie pamięci sprzętu używającego aplikacji.**
 - Zużycie pamięci RAM w czasie normalnej pracy - średnio poniżej 100 MB.
 - Zużycie zasobów procesora w czasie obciążenia - poniżej 30% przy maksymalnym obciążeniu.
- 6. Dostępny oraz kompatybilny na różnych przeglądarkach.**
 - Procent użytkowników korzystających z aplikacji na różnych przeglądarkach - obsługa co najmniej 95% popularnych przeglądarek.
 - Czas rozwoju nowych funkcji dla każdej przeglądarki - maksymalnie dodatkowe 20% czasu.
- 7. Rozszerzalny, ma możliwość dodania nowych funkcjonalności w przyszłości.**
 - Średni czas wprowadzenia nowej funkcji od momentu zgłoszenia pomysłu - mniej niż 2 tygodnie.
 - Liczba nowych funkcji wprowadzonych rocznie - minimum 10 większych aktualizacji rocznie.

Cele biznesowe

- **Zwiększenie Wydajności Operacyjnej:** Celem jest stworzenie narzędzia, które usprawni procesy logistyczne związane z wysyłką paczek. Poprzez zoptymalizowanie i usprawnienie sposobu nadawania i dostarczania przesyłek, firma może skrócić czas realizacji zleceń, co prowadzi do zwiększenia wydajności.
- **Poprawa Obsługi Klienta:** Aplikacja ma na celu zapewnić klientom prosty i intuicyjny sposób wysyłania paczek oraz śledzenia ich statusu. Poprzez oferowanie łatwego dostępu do informacji i szybkich procesów, firma zamierza podnieść jakość obsługi klienta.
- **Zwiększenie Konkurencyjności:** Przywiązanie uwagi do potrzeb klientów poprzez szybkość, bezpieczeństwo i przejrzystość procesu wysyłki paczek pozwoli firmie wyróżnić się na tle konkurencji. Tworzenie innowacyjnych rozwiązań w obszarze usług kurierskich umożliwi zdobycie przewagi rynkowej.
- **Rozwój Relacji z Klientem:** Aplikacja umożliwi śledzenie historii wysyłek oraz gromadzenie opinii. To pozwoli na lepsze dostosowanie oferty do oczekiwań klientów oraz zapewnienie spersonalizowanego podejścia.
- **Elastyczność w Rozwoju Produktu:** Tworzenie aplikacji, która jest łatwa w rozbudowie i dostosowywaniu do ewoluujących potrzeb rynkowych. Firma zamierza reagować na zmieniające się trendy i zapotrzebowanie, dodając nowe funkcje i ulepszając istniejące, aby utrzymać aplikację na czołowej pozycji.
- **Zapewnienie Bezpieczeństwa Danych:** Istotnym celem jest zagwarantowanie bezpieczeństwa danych osobowych klientów i informacji dotyczących przesyłek. Dążenie do najwyższych standardów bezpieczeństwa danych stanie się kluczowym elementem budowania zaufania klientów do platformy.

Struktura Organizacyjna:

- **Zarządzanie centralne:** Kierownictwo odpowiedzialne za strategię, rozwój aplikacji i zarządzanie.
- **Działy funkcjonalne:** IT (rozwój aplikacji), obsługa klienta, rekrutacja kurierów.

Obowiązki Funkcjonalne:

- **Zarząd:** Określenie strategii rozwoju, zarządzanie zasobami.
- **Zespół IT:** Tworzenie, rozwój i utrzymanie aplikacji.
- **Obsługa klienta:** Wsparcie klientów, rozwiązywanie problemów z dostawami.
- **Rekrutacja:** Pozyskiwanie i szkolenie nowych kurierów.

Opis Usług:

- **Dostawa paczek:** Szybka i terminowa dostawa przesyłek.
- **Wsparcie klienta:** Pomoc i obsługa klienta 24/7.
- **Rekrutacja kurierów:** Werbowanie, szkolenie i zarządzanie kurierami.

Metody Pracy:

- **Agile Development:** Ciągłe dostosowywanie i rozwijanie aplikacji zgodnie z potrzebami rynku.
- **Kanban Workflow:** Monitoring zleceń i dostaw w czasie rzeczywistym.
- **Customer-Centric Approach:** Skupienie na doświadczeniu klienta i szybkiej reakcji na jego potrzeby.

Interesariusze:

- **Klienci:** Osoby wysyłające paczki oraz odbiorcy.
- **Kurierzy:** Osoby dostarczające przesyłki.
- **Właściciele/fundatorzy:** Zarząd firmy kurierskiej.
- **Partnerzy biznesowi:** Firmy, które współpracują z aplikacją kurierską.

Analiza Interakcji:

- **Zarządzanie klientami:** Obsługa, wsparcie i budowanie relacji.
- **Współpraca z kurierami:** Rekrutacja, szkolenia, ocena wydajności.
- **Koordinacja wewnętrzna:** Efektywna komunikacja między działami, ciągły przepływ informacji.

Cele Organizacji:

- **Doskonałe doświadczenie klienta:** Szybkość i niezawodność dostaw, profesjonalna obsługa klienta.
- **Optymalizacja logistyki:** Skuteczne zarządzanie trasami i magazynami, minimalizacja czasu dostaw.
- **Rozwój rynkowy:** Poszerzenie oferty usług, wejście na nowe rynki i rozwój partnerstw biznesowych.

Wpływ oprogramowania na biznes

Wprowadzenie odpowiednio zaprojektowanego oprogramowania dla firmy kurierskiej może przynieść szereg korzyści biznesowych, które przekładają się na efektywność operacyjną, zadowolenie klientów i wzrost zysków.

1. Optymalizacja kosztów operacyjnych

Uzasadnienie: Oprogramowanie do zarządzania trasami pozwala na zoptymalizowanie planu dostaw, co przekłada się na skrócenie tras i zmniejszenie kosztów paliwa.

Efekt biznesowy: Redukcja kosztów operacyjnych przyczynia się bezpośrednio do zwiększenia rentowności, co stanowi istotny element biznesowego sukcesu.

2. Zwiększona efektywność procesów:

Uzasadnienie: Automatyzacja procesów, takich jak przyjmowanie zamówień, alokacja tras, i monitorowanie dostaw, eliminuje zbędne ręczne czynności i redukuje ryzyko błędów.

Efekt biznesowy: Poprawa efektywności operacyjnej pozwala na obsłużenie większej liczby przesyłek przy mniejszych nakładach pracy, co z kolei prowadzi do oszczędności czasu i zasobów.

3. Zwiększenie satysfakcji klienta:

Uzasadnienie: Oprogramowanie umożliwia klientom śledzenie przesyłek w czasie rzeczywistym, zapewniając transparentność i pewność co do czasu dostawy.

Efekt biznesowy: Zadowoleni klienci są bardziej skłonni do powtórnych zakupów i polecania usług firmy kurierskiej innym, co przekłada się na zwiększenie przychodów.

4. Szybsza reakcja na zmiany i zapotrzebowanie rynkowe:

Uzasadnienie: Oprogramowanie dostarcza danych analitycznych na temat operacji kurierskich, co umożliwia szybkie dostosowywanie się do zmieniających się warunków rynkowych.

Efekt biznesowy: Możliwość elastycznego reagowania na nowe trendy rynkowe i potrzeby klientów pozwala utrzymać konkurencyjność i zdobyć przewagę nad innymi firmami kurierskimi.

5. Skalowalność i zwiększenie przepustowości:

Uzasadnienie: Oprogramowanie skalowalne umożliwia dostosowanie się do wzrostu liczby przesyłek i rozszerzenia zakresu usług.

Efekt biznesowy: Zwiększenie przepustowości przekłada się na zwiększone przychody, zwłaszcza w okresach zwiększonego popytu, takich jak okresy świąteczne czy kampanie promocyjne.

Analiza ograniczeń

- **Zasięg Geograficzny:** Istniejące ograniczenia dotyczące obszarów dostaw mogą skutkować niemożnością zapewnienia usług kurierskich we wszystkich lokalizacjach. Niektóre obszary mogą być poza zasięgiem lub mogą mieć ograniczoną dostępność kurierów, co prowadzi do nierówności w dostępie do usług dla klientów w niektórych regionach.
- **Ograniczenia Czasowe:** W godzinach szczytu lub w określonych dniach tygodnia, np. podczas świąt, ograniczenia czasowe mogą wpływać na szybkość i dostępność dostawy. To może stanowić problem dla klientów oczekujących na szybką realizację przesyłek.
- **Ograniczenia Dotyczące Produktów:** Nie wszystkie rodzaje przesyłek lub kategorii produktów mogą być obsługiwane przez aplikację. Istnieje możliwość, że niektóre paczki mogą nie być dopuszczalne lub niedostępne do wysyłki za pośrednictwem platformy kurierskiej.
- **Koszty:** Koszty dostawy mogą się różnić w zależności od odległości, czasu dostawy czy wartości paczki. Wysokie koszty mogą być decydującym czynnikiem dla klientów przy wyborze usług kurierskich.
- **Bezpieczeństwo Danych:** Ochrona danych osobowych klientów, takich jak adresy dostawy czy informacje płatnicze, stanowi istotne wyzwanie. Zagrożenia dla bezpieczeństwa danych mogą wpływać na zaufanie użytkowników do platformy kurierskiej.
- **Zależność od Dostawców Zewnętrznych:** Aplikacja może polegać na sieci kurierów zewnętrznych. Ograniczenia związane z dostępnością, jakością usług lub ich ilością mogą wpłynąć na sprawność realizacji zleceń oraz jakość obsługi.

Nazwa Ograniczenia	Dotkliwość Ograniczenia	Możliwe Rozwiązania Problemu
Zasięg Geograficzny	Wysoka	<ul style="list-style-type: none"> Rozbudowa sieci kurierów w trudno dostępnych obszarach. Wprowadzenie promocji dla klientów w regionach o niskiej dostępności kurierów. Umożliwienie klientom wyboru alternatywnych miejsc dostawy.
Ograniczenia Czasowe	Średnia	<ul style="list-style-type: none"> Dynamiczne dostosowanie cen dostawy w zależności od pory dnia. Zwiększenie floty kurierów w godzinach szczytu. Wprowadzenie specjalnych promocji lub rabatów w okresach o dużej ograniczonej dostępności.
Ograniczenia Dotyczące Produktów	Niska	<ul style="list-style-type: none"> Klarowna prezentacja listy obsługiwanych produktów przed składaniem zamówienia. Informowanie klientów o niedostępności niektórych produktów w danej lokalizacji. Wprowadzenie alternatywnych opcji dostawy dla niedozwolonych produktów.
Koszty	Wysoka	<ul style="list-style-type: none"> Wprowadzenie elastycznych taryf dostawy w zależności od odległości. Program lojalnościowy z rabatami dla stałych klientów. Okazjonalne promocje, zwłaszcza w okresach o mniejszym popycie na usługi kurierskie.
Bezpieczeństwo Danych	Wysoka	<ul style="list-style-type: none"> Wdrożenie zaawansowanych protokołów szyfrowania danych. Edukacja użytkowników w zakresie bezpiecznego korzystania z aplikacji. Regularne audyty bezpieczeństwa danych i szybka reakcja na wszelkie incydenty związane z naruszeniem prywatności.
Zależność od Dostawców Zewnętrznych	Wysoka	<ul style="list-style-type: none"> Współpraca z różnymi dostawcami, aby zminimalizować ryzyko związane z jednym dostawcą. Stała ocena i monitorowanie jakości usług dostawców zewnętrznych. Rozbudowa własnej floty kurierów w przypadku niedostępności zewnętrznych dostawców.

Analiza ryzyka

Aplikacja:

- Nieprawidłowe Użytkowanie: Użytkownicy mogą popełniać błędy przy wprowadzaniu danych lub korzystać z funkcji w sposób nieprzewidziany, co może skutkować błędnymi informacjami o przesyłkach lub problemami z realizacją zleceń.
- Brak Kompatybilności wstecznej: Aktualizacje aplikacji mogą generować niekompatybilność z wcześniejszymi wersjami, co spowoduje trudności dla użytkowników korzystających z starszych wersji.

Użytkownicy:

- Nieautoryzowany Dostęp: Istnieje ryzyko prób nieuprawnionego dostępu do systemu, co może prowadzić do zagrożenia poufności danych klientów i kurierów.
- Niewłaściwe Użytkowanie Systemu: Błędne korzystanie z aplikacji przez użytkowników może powodować problemy w obsłudze przesyłek lub generować nieprawidłowe informacje.

Projektowanie Systemu:

- Przekroczenie Budżetu: Koszty tworzenia aplikacji mogą przekroczyć założony budżet, co może skomplikować lub opóźnić dalszy rozwój projektu.
- Niedotrzymanie Terminu Utworzenia Aplikacji: Trudności techniczne lub złożoność projektu mogą spowodować opóźnienia w dostarczeniu finalnej wersji aplikacji.

Baza Danych:

- Zaburzenie Integralności Danych: Możliwe uszkodzenia lub utrata danych wpłyną negatywnie na poprawność obsługi paczek i danych klientów.
- Nieautoryzowany Dostęp do Bazy Danych: Ataki hakerskie lub próby nieuprawnionego dostępu mogą narazić prywatność i bezpieczeństwo danych użytkowników.

Serwer:

- Awaria Serwera: Problemy techniczne, awarie sprzętowe lub problemy z zasilaniem mogą spowodować niedostępność aplikacji, uniemożliwiając obsługę zleceń.
- Błędy Ludzkie: Ludzkie pomyłki w zarządzaniu danymi mogą prowadzić do uszkodzenia lub utraty danych, co wpłynie na spójność i dokładność informacji o przesyłkach.

Nazwa Ryzyka	Prawdopodobieństwo	Dotkliwość Ryzyka	Możliwe Rozwiązania Problemu
Nieprawidłowe Użytkowanie	Średnie	Średnia	<ul style="list-style-type: none"> Dostarczenie jasnych instrukcji obsługi dla użytkowników. Implementacja walidacji danych wejściowych w celu ograniczenia błędów. Regularne szkolenia dla użytkowników dotyczące poprawnego korzystania z aplikacji.
Brak Kompatybilności wstecznej	Niskie	Wysoka	<ul style="list-style-type: none"> Planowanie aktualizacji z uwzględnieniem kompatybilności wstecznej. Informowanie użytkowników o zmianach i udostępnianie instrukcji aktualizacji. Zapewnienie wsparcia technicznego dla użytkowników korzystających z starszych wersji aplikacji.
Nieautoryzowany Dostęp	Wysokie	Wysoka	<ul style="list-style-type: none"> Wdrożenie zaawansowanych środków bezpieczeństwa, takich jak dwuskładnikowa autentykacja. Monitoring aktywności w celu wczesnego wykrywania prób nieautoryzowanego dostępu. Regularne audyty bezpieczeństwa i aktualizacje zabezpieczeń.
Niewłaściwe Użytkowanie Systemu	Średnie	Średnia	<ul style="list-style-type: none"> Dostarczenie instrukcji obsługi i szkoleń dla użytkowników. Implementacja funkcji ostrzegających przed potencjalnie nieprawidłowym użyciem. System powiadomień o błędach, aby użytkownicy byli informowani o ewentualnych problemach.
Przekroczenie Budżetu	Średnie	Wysoka	<ul style="list-style-type: none"> Staranne planowanie i kontrola kosztów w trakcie całego procesu projektowania. Monitorowanie postępów budżetowych na bieżąco i dostosowywanie planów działania. Wprowadzenie rezerwy budżetowej na wypadek nieprzewidzianych wydatków.

Niedotrzymanie Terminu Utworzenia Aplikacji	Średnie	Wysoka	<ul style="list-style-type: none"> • Realistyczne oszacowanie czasu potrzebnego na każdy etap projektu. • Regularne oceny postępów i dostosowywanie planów w razie potrzeby. • Zespołowa współpraca i efektywne zarządzanie zasobami w celu uniknięcia opóźnień.
Zaburzenie Integralności Danych	Średnie	Wysoka	<ul style="list-style-type: none"> • Regularne tworzenie kopii zapasowych danych. • Implementacja mechanizmów sprawdzania integralności danych. • Szybka reakcja na wszelkie sytuacje, które mogą prowadzić do uszkodzenia lub utraty danych.
Nieautoryzowany Dostęp do Bazy Danych	Wysokie	Wysoka	<ul style="list-style-type: none"> • Silne zabezpieczenia bazy danych, w tym szyfrowanie danych. • Nadzór nad dostępem do bazy danych i identyfikacja podejrzanej aktywności. • Regularne aktualizacje zabezpieczeń i monitorowanie nowych zagrożeń.
Awaria Serwera	Średnie	Wysoka	<ul style="list-style-type: none"> • Redundancja serwerów i systemów zasilania. • Regularne testy wydajności i monitorowanie obciążenia serwerów. • Plan awaryjności z klarownymi procedurami przywracania usług w razie awarii.
Błędy Ludzkie	Niskie	Średnia	<ul style="list-style-type: none"> • Implementacja systemów zabezpieczających przed przypadkowymi usunięciami danych. • Szkolenia dla personelu w zakresie bezpiecznego zarządzania danymi. • Ustanowienie procedur weryfikacyjnych przed wprowadzeniem istotnych zmian w danych.

Plan biznesowy

1. Finanse:

- **System płatności:** Integracja różnych metod płatności (karty, portfele cyfrowe, przelewy), zapewniając bezpieczne i szybkie transakcje.
- **Rozliczenia dla kurierów:** Automatyczne generowanie rozliczeń dla kurierów na podstawie wykonanych dostaw.
- **Zarządzanie fakturami i opłatami:** Centralne zarządzanie fakturami, opłatami dostaw i prowizjami.
- **Raportowanie finansowe:** Generowanie raportów z transakcji, analiza przychodów i wydatków.
- **Śledzenie transakcji:** Zapewnienie możliwości śledzenia statusu płatności dla klientów i kurierów.
- **Zarządzanie budżetem:** Narzędzia umożliwiające kontrolę wydatków i przychodów.

2. Personel:

- **Rekrutacja:** Platforma rekrutacyjna dla kurierów z możliwością aplikowania online.
- **Szkolenia:** Dostęp do materiałów edukacyjnych dla nowych kurierów oraz aktualizacje szkoleń.
- **Ocena wydajności:** Narzędzia do oceny wydajności kurierów na podstawie czasu dostawy, opinii klientów itp.
- **Zarządzanie kontraktami i umowami:** Elektroniczne podpisywanie umów i zarządzanie dokumentami.

3. Reklama:

- **Marketing internetowy:** Wykorzystanie mediów społecznościowych, reklam Google, e-mail marketing.
- **Kampanie reklamowe:** Tworzenie kampanii reklamowych promujących korzyści z korzystania z usług kurierskich.
- **Zarządzanie wizerunkiem marki:** Monitorowanie opinii klientów i reagowanie na opinie online.
- **Programy lojalnościowe:** Tworzenie programów nagród dla stałych klientów i kurierów.

4. Księgowość:

- **Monitorowanie finansów:** Śledzenie przepływów finansowych w czasie rzeczywistym.
- **Rozliczenia z dostawcami:** Automatyczne rozliczanie płatności dla partnerów i dostawców.
- **Zarządzanie rachunkami:** Elektroniczne generowanie faktur i zarządzanie płatnościami.
- **Audyt finansowy:** Dostęp do danych księgowych dla audytorów zewnętrznych w celu weryfikacji.

5. Zarządzanie relacjami z klientami (CRM):

- **Baza danych klientów:** Przechowywanie danych klientów, historii zamówień i preferencji.
- **Obsługa zgłoszeń i reklamacji:** System obsługi klienta umożliwiający zgłaszanie problemów i ich rozwiązywanie.
- **Ocena satysfakcji:** Zbieranie opinii od klientów i reakcja na nie.
- **Personalizacja doświadczenia użytkownika:** Dostosowanie interfejsu aplikacji do preferencji klientów.

6. Zarządzanie dostawami:

- **Śledzenie paczek:** Zapewnienie możliwości śledzenia paczek w czasie rzeczywistym dla klientów i kurierów.
- **Optymalizacja tras:** Wybór najbardziej optymalnej trasy dostawy z uwzględnieniem ruchu ulicznego i czasu dostawy.

7. Planowanie:

- **Optymalizacja harmonogramów:** Planowanie dostaw zgodnie z popytem i dostępnością kurierów.
- **Prognozowanie popytu:** Wykorzystanie danych historycznych do przewidywania popytu w różnych obszarach.
- **Planowanie strategiczne:** Określenie długoterminowych celów rozwoju i strategii biznesowej.

8. Analiza statystyk i prognozy:

- **Monitorowanie danych dotyczących dostaw:** Analiza danych związanych z dostawami w celu optymalizacji procesu.
- **Analiza trendów rynkowych:** Śledzenie zmian i trendów w branży kurierskiej.
- **Raportowanie wydajności:** Generowanie raportów na podstawie analizy danych.

9. Opłaty:

- **System opłat:** Transparentne wyświetlanie opłat dla klientów i kurierów.
- **Kalkulacje kosztów dostawy:** Obliczanie kosztów dostawy w zależności od różnych czynników.
- **Zarządzanie stawkami:** Elastyczne zarządzanie stawkami opłat w zależności od lokalizacji, rodzaju paczki itp.

Analiza wzorców architektonicznych i wybór optymalnego wzorca

Główne funkcje

Elementy strukturalne	Główne funkcje
Aplikacja klienta	<ul style="list-style-type: none">- Możliwość złożenia zamówienia na dostawę w dowolnym miejscu i czasie.- Real-time monitorowanie statusu kuriera w trakcie dostawy.- Wybór różnych metod płatności, takich jak karta kredytowa, płatność online, gotówka przy odbiorze itp.- Przeglądanie historii zamówień, potwierdzeń dostaw i paragonów.
Aplikacja kuriera	<ul style="list-style-type: none">- Aplikacja umożliwiająca kurierom zarządzanie zamówieniami, trasą dostawy i komunikację z klientami.- Automatyczne powiadomienia o nowych zamówieniach i ich przyjęcie do realizacji.- Aktualizowanie statusu dostawy w czasie rzeczywistym, dostarczanie informacji klientowi o postępach.- Przeglądanie historii dostaw, ocen i recenzji od klientów w celu poprawy jakości usług.
Baza danych	<ul style="list-style-type: none">- Przechowuje informacje o zarejestrowanych użytkownikach, w tym dane personalne i preferencje.- Zawiera historię zamówień i oceny dla każdego użytkownika.- Przechowuje dane dotyczące kurierów, w tym ich dostępność, lokalizację, status i historię ocen.- Zapisuje historię wszystkich dostaw wraz z danymi dotyczącymi ocen i recenzji.

Architektura mikroserwisów

Architektura mikroserwisów to podejście projektowania oprogramowania, które zakłada rozbięcie systemu na mniejsze, niezależne i autonomiczne serwisy zwane mikroserwisami. Każdy z tych serwisów stanowi odrębny komponent, odpowiedzialny za konkretne zadanie lub funkcjonalność. Mikroserwisy komunikują się ze sobą poprzez interfejsy programistyczne (API), często wykorzystując protokoły komunikacyjne typu HTTP lub komunikację asynchroniczną.

Zalety:

- **Skalowalność:** Możliwość niezależnego skalowania poszczególnych mikroserwisów w zależności od obciążenia.
- **Łatwość w utrzymaniu:** Łatwiejsze zarządzanie i aktualizacja poszczególnych mikroserwisów.
- **Technologiczna heterogeniczność:** Możliwość użycia różnych technologii dla różnych mikroserwisów.

Wady:

- **Złożoność komunikacji:** Konieczność skomplikowanej komunikacji między mikroserwisami.
- **Zarządzanie transakcjami:** Trudności w zarządzaniu transakcjami między mikroserwisami.
- **Koszty infrastrukturalne:** Potrzeba rozbudowanej infrastruktury do monitorowania i zarządzania mikroserwisami.

Architektura monolityczna

Architektura monolityczna to tradycyjne podejście projektowania oprogramowania, w którym cała aplikacja jest rozwijana, wdrażana i utrzymywana jako jedna, zintegrowana jednostka. Wszystkie komponenty, zarówno warstwa prezentacji, jak i logika biznesowa oraz dostęp do danych, są zawarte w jednym obszarze.

Zalety:

- **Prostota wdrożenia:** Łatwość w wdrożeniu jednej jednostki.
- **Łatwość w debugowaniu:** Proste debugowanie, ponieważ cała aplikacja działa w jednym środowisku.
- **Zarządzanie stanem aplikacji:** Możliwość zarządzania stanem aplikacji w jednym miejscu.

Wady:

- **Skalowalność pionowa:** Trudność w pionowym skalowaniu do obsługi większego obciążenia.
- **Ograniczona elastyczność:** Mniejsza elastyczność w aktualizacjach i rozbudowie systemu.
- **Wysoki poziom skomplikowania:** W miarę wzrostu aplikacji, wzrasta złożoność.

Architektura MVC (Model-View-Controller)

Architektura MVC dzieli aplikację na trzy główne komponenty: Model (reprezentuje dane i logikę), View (odpowiada za prezentację danych) i Controller (przetwarza żądania użytkownika).

Zalety:

- **Separacja odpowiedzialności:** Klarowne oddzielenie warstwy prezentacji od logiki biznesowej.
- **Łatwość testowania:** Łatwość w testowaniu poszczególnych komponentów.
- **Współpraca zespołowa:** Ułatwia podział pracy między różnymi zespołami, skupionymi na różnych aspektach aplikacji.

Wady:

- **Złożoność w implementacji:** Wprowadza pewną złożoność w implementacji i zrozumieniu relacji między komponentami.
- **Zjawisko grubego kontrolera:** Może prowadzić do powstania "grubego kontrolera", co jest trudne do utrzymania.
- **Skomplikowane aplikacje:** Dla bardziej skomplikowanych aplikacji, może wymagać dodatkowych wzorców architektonicznych.

Architektura warstwowa

Architektura warstwowa, znana również jako architektura wielowarstwowa, to struktura projektowa, która dzieli system na logicznie uporządkowane warstwy. Każda warstwa ma określone zadania i odpowiedzialności, a komunikacja między nimi odbywa się zgodnie z ustalonymi regułami. To podejście pomaga w organizacji i zarządzaniu złożonymi systemami, a także ułatwia utrzymanie i rozwijanie aplikacji.

Zalety:

- **Separacja odpowiedzialności:** Klarowne oddzielenie zadań między różnymi warstwami.
- **Łatwość wzajemnego zastępowania:** Możliwość łatwego zastępowania lub aktualizowania poszczególnych warstw.
- **Łatwość wzajemnego zrozumienia:** Łatwość zrozumienia struktury aplikacji poprzez podział na jasno zdefiniowane warstwy.

Wady:

- **Złożoność komunikacji między warstwami:** Konieczność skomplikowanej komunikacji między różnymi warstwami.
- **Ryzyko zależności między warstwami:** Ryzyko tworzenia zbyt silnych zależności między warstwami.
- **Potencjalne obciążenie wydajnościowe:** Może wprowadzić dodatkowe obciążenie wydajnościowe z powodu komunikacji między warstwami.

Wybór optymalnego wzorca

W kontekście dynamicznej i rozwiniętej branży kurierskiej, gdzie różnorodność usług, dostępność kurierów i szybka reakcja na zmieniające się warunki są kluczowe, architektura mikroserwisów wydaje się być najbardziej odpowiednia. Pozwala ona na skalowalność, elastyczność i niezależność poszczególnych usług, co może być kluczowe w zaspokajaniu zróżnicowanych potrzeb klientów na różnych obszarach geograficznych. Jednakże, ważne jest, aby skonsultować tę decyzję zespołem projektowym i uwzględnić specyficzne szczegóły projektu oraz wymagania klienta.

Powody wybrania architektury mikroserwisów

Skalowalność:

Architektura mikroserwisów umożliwia elastyczne skalowanie poszczególnych usług w zależności od zmieniających się obciążeń. W przypadku branży kurierskiej, gdzie ilość zamówień może się znacznie zmieniać w różnych godzinach dnia czy sezonach, taka elastyczność jest kluczowa dla efektywnego zarządzania zasobami.

Różnorodność Usług:

Branża kurierska często oferuje różnorodne usługi, takie jak dostawy ekspresowe, standardowe czy usługi kurierskie dla firm. Architektura mikroserwisów pozwala na niezależne wdrożenie i zarządzanie poszczególnymi usługami, co umożliwia dostosowanie architektury do specyfiki każdej z oferowanych usług.

Niezależność Poszczególnych Usług:

Każdy mikroserwis stanowi odrębną jednostkę, co oznacza, że mogą być rozwijane, aktualizowane i wdrażane niezależnie od innych. To pozwala na szybkie wprowadzanie zmian, co jest kluczowe w środowisku, gdzie dostosowanie do nowych trendów rynkowych czy szybka reakcja na konkurencję są kluczowe.

Elastyczność Rozwoju:

Mikroserwisy pozwalają na elastyczny rozwój systemu, umożliwiając dostosowywanie się do ewoluujących potrzeb rynku kurierskiego. Nowe funkcje czy usługi mogą być wprowadzane bez konieczności przerwy w funkcjonowaniu pozostałych elementów systemu.

Niezależność Technologiczna:

Poszczególne mikroserwisy mogą być napisane w różnych językach programowania i korzystać z różnych technologii. To pozwala na wybór najlepszego narzędzia do każdego zadania, co może być istotne w zależności od specyfiki danej usługi kurierskiej.

Wyzwania i Konsultacje:

Przed podjęciem ostatecznej decyzji, zaleca się konsultacje z zespołem projektowym, aby ocenić specyficzne potrzeby branży kurierskiej i dostosować architekturę mikroserwisów do konkretnych wymagań projektu. Warto również uwzględnić wymagania klienta i zapewnić, że wybrany wzorzec architektoniczny będzie odpowiadał ich oczekiwaniom.

Rodzaje infrastruktury projektu

IaaS - model usług chmurowych, w którym dostawca chmury zapewnia fundamentalne zasoby infrastrukturalne, takie jak przestrzeń dyskowa, moc obliczeniowa, sieci, a użytkownik zarządza systemem operacyjnym, aplikacjami i danymi.

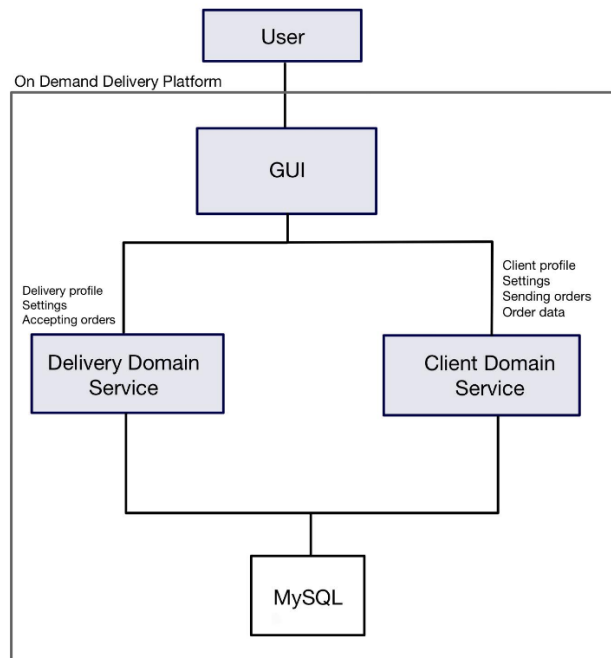
Własna Infrastruktura - Własna infrastruktura, znana również jako "on-premises" lub "on-prem", to model, w którym organizacja utrzymuje i zarządza własnym centrum danych oraz sprzętem informatycznym. W tym przypadku, wszystkie serwery, urządzenia sieciowe, magazyny danych i inne zasoby są fizycznie umieszczone w obiekcie firmy.

Hybrydowa Infrastruktura - Hybrydowa infrastruktura to połączenie własnej infrastruktury z rozwiązaniami chmurowymi, umożliwiające przenoszenie danych i aplikacji między lokalnym centrum danych a chmurą.

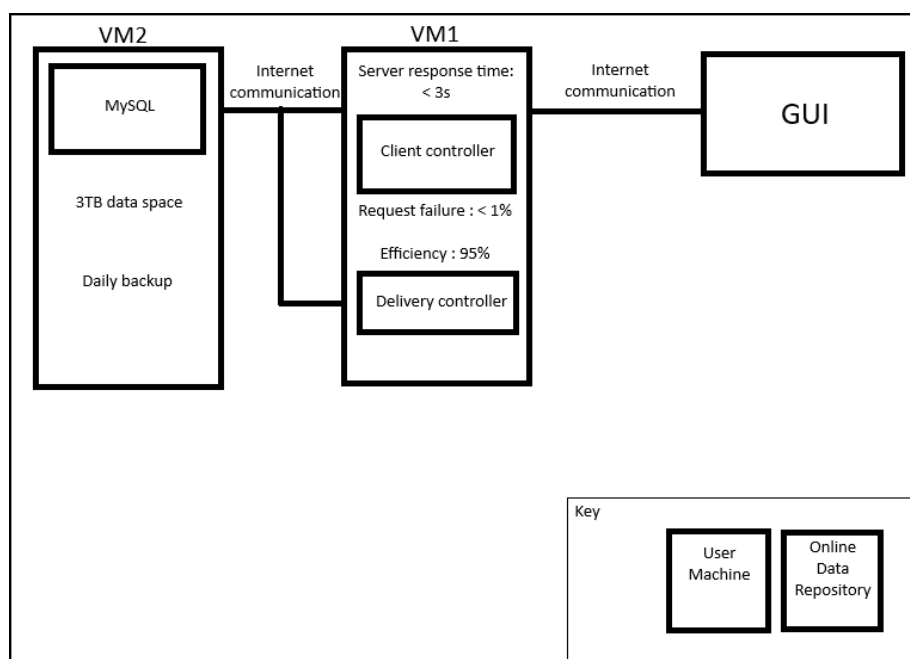
Dla naszego produktu, model Połączenia Własnej IT-Infrastruktury i IaaS (hybrydowa infrastruktura) może być najlepszym wyborem, umożliwiając elastyczność i kontrolę nad kluczowymi aspektami operacyjnymi, jednocześnie wykorzystując korzyści zasobów chmurowych, takie jak skalowalność i dostępność.

Modelowanie powiązań struktury i architektury projektu

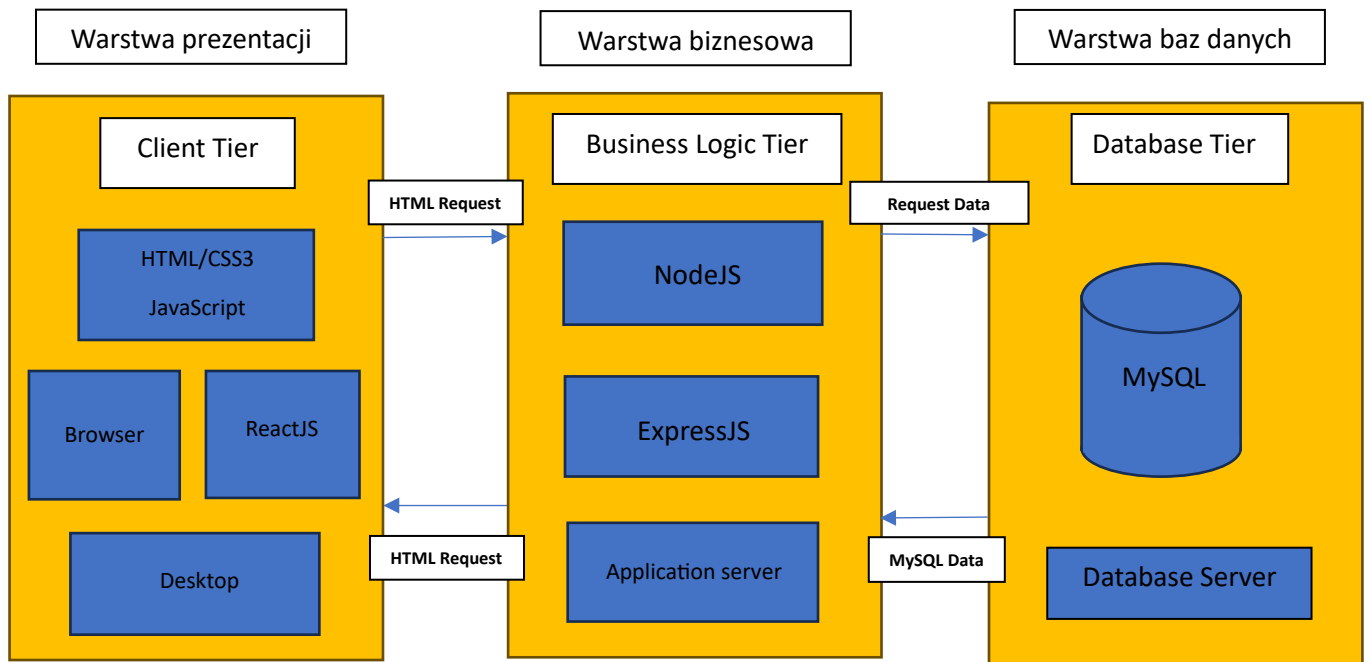
Wzorzec Komponentowo-Łącznikowy



Wzorzec Alokacyjny



Wzorzec Modułowy



Technologie

React	React jest biblioteką JavaScript, służy do budowy interfejsów użytkownika. React pozwala na efektywne tworzenie stron interaktywnych, dynamicznych stron internetowych.
Node.js	Node.js to środowisko wykonawcze JavaScript po stronie serwera. Umożliwia pisanie aplikacji serwerowych w języku JavaScript.
Express	Express to framework Node.js, jest często używany jako warstwa pośrednia (middleware) w aplikacjach webowych Node.js. Umożliwia definiowanie tras, obsługę żądań HTTP i wiele innych.
MySQL	MySQL to system zarządzania bazą danych relacyjnych. MySQL jest powszechnie stosowany do przechowywania danych w aplikacjach internetowych.

Interfejs użytkownika- UI

Interfejs użytkownika w produkcie został stworzony przy użyciu technologii React.js, przy wsparciu Bootstrapa i CSS3. React.js jest efektywną biblioteką JavaScript do budowy interfejsów użytkownika, zaprojektowaną z myślą o deklaratywnym podejściu do programowania komponentowego. Język JavaScript został wykorzystany do dynamicznego renderowania elementów interfejsu oraz obsługi interakcji użytkownika.

Bootstrap, jako framework front-endowy, umożliwia szybkie i responsywne projektowanie interfejsów, a także dostarcza gotowych komponentów, co przyspiesza proces tworzenia estetycznych i funkcjonalnych widoków aplikacji. Jego zastosowanie pozwala na osiągnięcie spójnego i profesjonalnego wyglądu interfejsu użytkownika.

Dodatkowo, użyto CSS3 do dostosowania stylów i nadania unikalnego charakteru interfejsowi. CSS3 oferuje zaawansowane funkcje, takie jak animacje, gradienty czy cieniowanie, co pozwala na tworzenie atrakcyjnych wizualnie elementów interfejsu.

Kombinacja tych technologii – React.js, JavaScript, Bootstrap oraz CSS3 – umożliwiła efektywne stworzenie interaktywnego, responsywnego i estetycznego interfejsu użytkownika dla aplikacji kurierskiej. Dzięki temu użytkownicy mogą korzystać z intuicyjnego i funkcjonalnego środowiska, co wpływa pozytywnie na doświadczenie obsługi aplikacji.

Narzędzia programistyczne

Visual Paradigm	Narzędzie do modelowania oprogramowania, umożliwiające tworzenie diagramów ERD.
Git	System kontroli wersji, pozwala śledzić zmiany w kodzie źródłowym i współpracować zespołowo.
Github	Platforma oparta na Git, zapewnia tworzenie repozytoriów, przegląd kodu i integrację zespołową.
DataGrip	Zarządzanie bazą danych, pisanie i testowanie zapytań SQL.
Visual Studio Code	Wieloplatformowy edytor kodu źródłowego, oferujący rozszerzenia dla wielu języków programowania.

Słownik

JavaScript to skryptowy język programowania wykorzystywany do tworzenia interaktywnych stron internetowych. Działa w przeglądarce, umożliwiając manipulację elementami strony oraz obsługę interakcji użytkownika. Jego dynamiczna natura pozwala na szybkie tworzenie funkcji bez konieczności kompilacji.

React to biblioteka JavaScript, która umożliwia tworzenie interfejsów użytkownika (UI) za pomocą komponentów. Dzięki jednokierunkowemu przepływowi danych i wydajnemu mechanizmowi Virtual DOM, React zapewnia reaktywność i efektywne renderowanie zmian, co pozwala tworzyć skalowalne i dynamiczne aplikacje internetowe.

JSX to rozszerzenie składni JavaScript, które umożliwia pisanie kodu HTML wewnątrz kodu JavaScript, co ułatwia tworzenie struktury interfejsu.

CSS (Cascading Style Sheets) to język używany do określania wyglądu i stylu elementów na stronach internetowych. Służy do definiowania m.in. kolorów, układu, rozmiarów oraz innych właściwości wizualnych elementów HTML. Jest kluczowym narzędziem w projektowaniu interfejsów użytkownika, umożliwiając kontrolę nad prezentacją treści na stronie internetowej.

Bootstrap to otwarta biblioteka front-endowa zawierająca zestaw narzędzi, stylów i komponentów zaprojektowanych w celu ułatwienia tworzenia responsywnych stron internetowych i aplikacji. Zawiera gotowe szablony, siatkę projektową, komponenty interfejsu użytkownika i wiele innych elementów, co pozwala programistom szybko tworzyć estetyczne i spójne witryny, dopasowane do różnych urządzeń i ekranów.

SQL (Structured Query Language) to język programowania używany do zarządzania danymi w relacyjnych bazach danych. Pozwala na tworzenie, modyfikowanie i zarządzanie bazami danych poprzez wykonywanie zapytań. SQL umożliwia manipulację danymi, takie jak dodawanie, usuwanie, aktualizacja rekordów oraz pobieranie danych z bazy poprzez zapytania SELECT, co czyni go kluczowym narzędziem dla administratorów baz danych i programistów.

Express jest minimalistycznym frameworkiem do tworzenia aplikacji internetowych w języku JavaScript działającym na platformie Node.js. Pozwala na szybkie tworzenie interfejsów API, zarządzanie trasami (routingiem) oraz obsługę żądań i odpowiedzi HTTP.

Pojęcia:

Backend to część aplikacji, która działa po stronie serwera i zajmuje się przetwarzaniem danych oraz logiką biznesową, nie będąc widoczną dla użytkowników. Odpowiada za obsługę żądań, komunikację z bazą danych i dostarczanie danych dla interfejsu użytkownika.

Frontend to część aplikacji, którą użytkownicy widzą i z której korzystają bezpośrednio. Odpowiada za interakcję z użytkownikiem, prezentację danych oraz wygląd strony internetowej lub aplikacji. Składa się z kodu HTML, CSS i JavaScriptu oraz używa różnych frameworków i bibliotek, aby stworzyć responsywny i interaktywny interfejs.

API, czyli Interfejs Programowania Aplikacji, to zestaw reguł, protokołów i narzędzi, które umożliwiają różnym aplikacjom komunikację między sobą. Stanowi mostek, który pozwala aplikacjom na wymianę danych i funkcjonalności.

Elementy API:

- **Endpointy:** Konkretnie punkty końcowe, do których aplikacje mogą wysyłać zapytania, np. /home lub /delivery.
- **Metody HTTP:** Określają, jakie operacje można wykonać na danym zasobie, np. GET do pobierania danych, POST do tworzenia, PUT do aktualizacji, DELETE do usuwania.

Zastosowania API:

- **Integracja usług:** Pozwala na integrację różnych usług, np. płatności, map, komunikatorów.
- **Tworzenie platform:** Umożliwia tworzenie platform, które udostępniają dane innym aplikacjom, np. platformy społecznościowe.

Autoryzacja to proces weryfikacji uprawnień użytkownika do dostępu do określonych zasobów w aplikacji. Polega na kontrolowaniu, czy dany użytkownik ma prawo do wykonania określonych operacji lub dostępu do konkretnych danych. Implementuje się ją na serwerze, często wykorzystując mechanizmy tokenów, sesji lub JWT, a także tworząc logikę biznesową, która decyduje o prawach dostępu dla poszczególnych użytkowników lub grup.

Uwierzytelnianie to proces potwierdzenia tożsamości użytkownika, czyli sprawdzenie, czy dana osoba jest tym, za kogo się podaje. Najczęściej realizowane jest poprzez proces logowania, gdzie użytkownik dostarcza swoje dane uwierzytelniające (login, hasło, token) w celu potwierdzenia tożsamości. Ten proces także ma miejsce po stronie serwera.

Zapytania do bazy danych to instrukcje wysyłane do bazy danych w celu pobrania, aktualizacji, usuwania lub dodawania danych. Służą do interakcji z bazą danych, umożliwiając manipulację danymi zapisanymi w bazie.

Typy zapytań:

- **Zapytania SELECT:** Służą do pobierania danych z bazy, np. **SELECT * FROM tabela** pobiera wszystkie rekordy z danej tabeli.
- **Zapytania INSERT:** Dodają nowe dane do bazy, np. **INSERT INTO tabela (kolumna1, kolumna2) VALUES (wartość1, wartość2).**
- **Zapytania UPDATE:** Aktualizują istniejące dane w bazie, np. **UPDATE tabela SET kolumna = wartość WHERE warunek.**
- **Zapytania DELETE:** Usuwają dane z bazy, np. **DELETE FROM tabela WHERE warunek.**

Routing w React polega na definiowaniu tras (routes) i ich komponentów, które renderują odpowiedni widok w zależności od adresu URL. Jest osiąganę za pomocą bibliotek takich jak react-router-dom, umożliwiając nawigację między różnymi częściami aplikacji poprzez dopasowanie ścieżki URL do odpowiedniego komponentu. Dzięki temu użytkownik może przemieszczać się po aplikacji, a komponenty są renderowane dynamicznie w zależności od adresu URL, co pozwala na budowanie bardziej interaktywnych interfejsów użytkownika.

Biblioteki takie jak axios w aplikacjach frontendowych, np. w React, są wykorzystywane do wysyłania żądań HTTP do serwera. Poprzez użycie metod takich jak **axios.get**, **axios.post** czy **axios.put**, możliwe jest wysłanie zapytań z różnymi metodami HTTP wraz z danymi, np. do pobrania, zapisania lub aktualizacji danych na serwerze. Po wysłaniu żądania, axios umożliwia obsługę odpowiedzi w postaci obiektu, co pozwala na dostęp i przetwarzanie danych otrzymanych z serwera, takich jak dane JSON lub tekst odpowiedzi HTTP.

useEffect to hook w React, który umożliwia wykonywanie efektów ubocznych w komponentach funkcyjnych. Jest używany do zarządzania efektami ubocznymi, takimi jak pobieranie danych z serwera, subskrypcje do zdarzeń, czy manipulacje DOM, uruchamianych w odpowiedzi na zmiany w komponencie lub podczas jego montowania, aktualizacji lub demontowania. Pozwala uniknąć pułapek związanych z cyklem życia komponentów klasowych, zapewniając kontrolę nad efektami ubocznymi w komponentach funkcyjnych w zależności od potrzeb i aktualizacji stanu.

Szyfrowanie, w kontekście funkcji **bcrypt.hash** używanej w kodzie, to proces zabezpieczania danych, takich jak hasła, poprzez przekształcenie ich w sposób nieodwracalny. Wykorzystując algorytm haszowania bcrypt, funkcja ta pobiera hasło w

postaci tekstu i za pomocą soli (salt) oraz algorytmu haszowania bcrypt, generuje zaszyfowaną wersję hasła. Proces ten jest nieodwracalny, co oznacza, że zaszyfowane hasło nie może być bezpośrednio odszyfrowane do postaci oryginalnego hasła, co wpływa na zwiększenie bezpieczeństwa przechowywanych danych.

Komponenty w React to niezależne, hermetyzowane bloki budujące interfejs użytkownika. Stanowią podstawową jednostkę struktury UI, umożliwiają wielokrotne wykorzystanie kodu oraz uporządkowane zarządzanie elementami interfejsu, co przyczynia się do modularności i łatwości w utrzymaniu aplikacji.

Layout to struktura i układ graficzny elementów na stronie internetowej lub w aplikacji. Określa, jak poszczególne elementy interfejsu użytkownika są rozmieszczone względem siebie i w jaki sposób są wyświetlane na ekranie. Elementy takie jak nagłówki, menu, treść czy stopka są ułożone w sposób zaplanowany, aby zapewnić czytelność, łatwość nawigacji i estetyczny wygląd strony.

Pojęcia domenowe:

ZPL (Zebra Programming Language) jest językiem programowania używanym do tworzenia etykiet, głównie w sektorze logistyki i magazynowania. Służy do definiowania wyglądu i zawartości etykiet. Pozwala na precyzyjne określenie układu graficznego, tekstu, kodów kreskowych i innych elementów, umożliwiając drukowanie etykiet z informacjami o produktach, kodami identyfikacyjnymi czy instrukcjami dotyczącymi przesyłek. Jest kluczowym narzędziem w branży logistycznej do oznaczania i identyfikacji towarów, ułatwiając identyfikację, śledzenie i zarządzanie przesyłkami.

Konwencje nazewnnicze:

W projekcie stosowane są zrozumiałe konwencje, gdzie elementy związane z administracją, klientem i obsługą dostaw mają spójne nazwy, np. **AdminOrders**, **ClientWallet**, **NavbarDelivery**. Zarówno funkcje, API, jak i komponenty na frontendzie i backendzie są nazwane zgodnie z ich funkcjonalnością, co znacząco ułatwia zrozumienie ich roli i zastosowania w projekcie. Takie podejście do nazewnictwa przyczynia się do przejrzystości kodu i ułatwia współpracę pomiędzy zespołem programistycznym.

Odwołania do literatury:

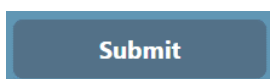
Oficjalna dokumentacja Bootstrapa - Podstawowe źródło zawierające szczegółowe informacje dotyczące komponentów, klas, poradników i przykładów użycia. [Oficjalna strona Bootstrapa](#) stanowi kluczowe źródło informacji.

Dokumentacja ZPL - Kluczowe źródło informacji, zawierające szczegółowe specyfikacje, instrukcje i przykłady związane z Zebra Programming Language. [Strona dokumentacji ZPL](#) zawiera podstawowe informacje na temat języka programowania ZPL.

Repozytoria – Zaimplementowanie wprowadzania numeru telefonu przy użyciu następującego [repozytorium](#).

Case studies:

- Przykładowe użycie biblioteki Bootstrap – przycisk. Urozmaicił on estetykę strony.



- Przykładowe użycie ZPL – umożliwił on generację etykiety która jest niezbędna w dziedzinie dostarczania przesyłek.



Architektura systemu

Świadczenia usług w modelu SaaS dla wielu firm kurierskich wymaga zastosowania jednej bazy danych, ale z wieloma schematami bazy danych.

Decyzja dotycząca Infrastruktury

Wybór infrastruktury hybrydowej umożliwia kombinację zarządzanej infrastruktury dostawcy SaaS i własnej infrastruktury. To pozwala na dostosowanie do specyficznych wymagań co do zasobów, skutecznej kontroli kosztów oraz większej elastyczności.

Izolacja Danych

Każda firma kurierska będzie miała swoje dane przechowywane w osobnym schemacie bazy danych. To pozwala na izolację danych między firmami, co jest istotne z punktu widzenia prywatności i bezpieczeństwa danych.

Łatwa Skalowalność

Dodanie nowej firmy kurierskiej może być prostą operacją dodania nowego schematu, a nie nowej bazy danych. To ułatwia zarządzanie rosnącą liczbą firm oraz umożliwia łatwe skalowanie systemu.

Współdzielenie Zasobów

Korzystanie z jednej bazy danych pozwala na efektywniejsze wykorzystanie zasobów serwera bazy danych. Skoro struktura danych między firmami jest podobna, można uniknąć redundancji i utrzymać jedną wspólną bazę.

Łatwiejsze Zarządzanie

Zarządzanie jedną bazą danych jest zazwyczaj łatwiejsze niż zarządzanie wieloma. To ułatwia utrzymanie, aktualizacje, tworzenie kopii zapasowych i inne operacje administracyjne.

Optymalizacja Zapytań

Jeśli istnieje potrzeba wspólnego wykonywania zapytań między firmami (na przykład analizy zbiorcze dla wszystkich firm), korzystanie z jednej bazy danych ułatwia takie operacje.

Interakcje Mikroserwisów

Mikroserwisy powinny komunikować się z bazą danych, uwzględniając odpowiedni schemat w zapytaniach. Kontenery mikroserwisów mogą być dynamicznie skalowane i zarządzane przez Kubernetes.

Bezpieczeństwo Danych

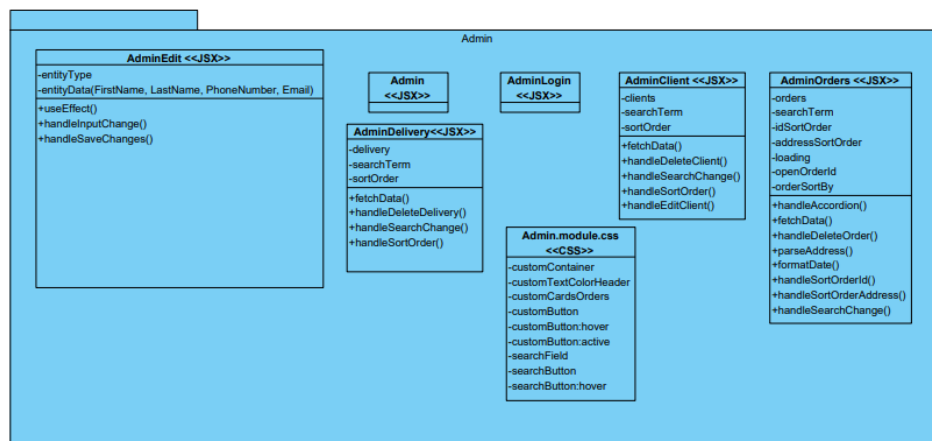
Wprowadzenie mechanizmów bezpieczeństwa, takich jak dostęp do schematu bazy danych tylko dla upoważnionych mikroserwisów, aby zabezpieczyć dane przed nieuprawnionym dostępem.

Takie podejście pozwoli na elastyczną obsługę wielu firm kurierskich, utrzymanie izolacji danych oraz efektywne zarządzanie infrastrukturą, co jest zgodne z zaleceniami dla modelu SaaS i infrastruktury hybrydowej.

Diagram klas

Interfejs admina

Diagram przedstawia strukturę klas aplikacji administracyjnej. Aplikacja składa się z następujących głównych klas:



Admin - klasa główna, która odpowiada za zarządzanie aplikacją.

AdminLogin - klasa odpowiedzialna za logowanie i wylogowywanie użytkowników.

AdminClient - klasa odpowiedzialna za zarządzanie klientami.

AdminOrders - klasa odpowiedzialna za zarządzanie zamówieniami.

AdminDelivery - klasa odpowiedzialna za zarządzanie dostawami.

Klasy te są podzielone na dwie grupy:

- Klasy interfejsu użytkownika (UI) - odpowiadają za wyświetlanie danych użytkownikowi.
- Klasy biznesowe - odpowiadają za obsługę logiki biznesowej aplikacji.

Klasy UI są reprezentowane przez następujące symbole:

- <<<JSX>>> - klasa React, która opisuje interfejs użytkownika.
- <<CSS>> - klasa CSS, która określa styl interfejsu użytkownika.

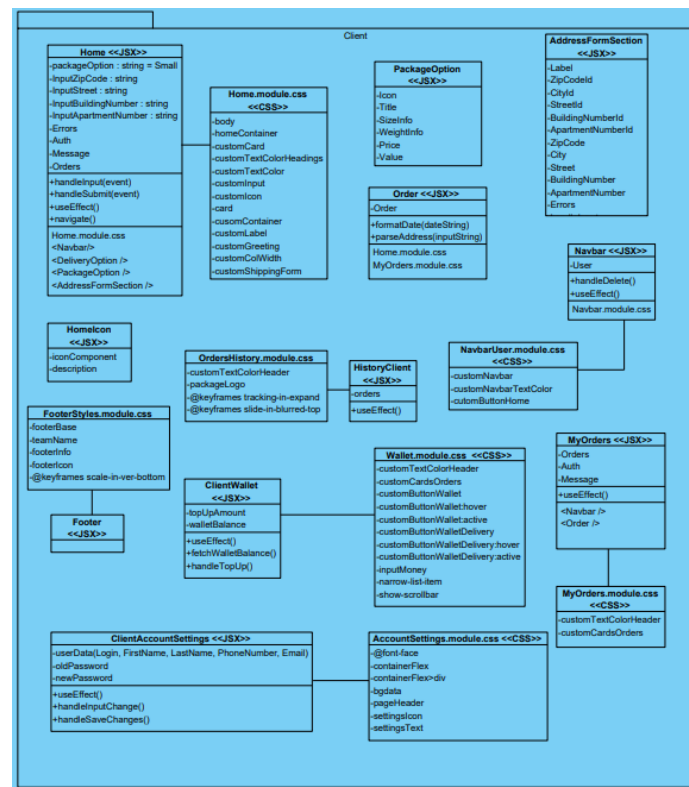
Klasy biznesowe są reprezentowane przez następujące symbole:

- -entity Type - określa typ danych, które przechowuje klasa.
- -entityData - określa dane przechowywane przez klasę.
- useEffect() - metoda React, która jest używana do odświeżania danych po zmianie stanu aplikacji.
- handleInputChange() - metoda, która jest używana do obsługi zmian danych wprowadzanych przez użytkownika.
- handleSaveChanges() - metoda, która jest używana do zapisywania zmian danych.
- fetchData() - metoda, która jest używana do pobierania danych z serwera.

Diagram przedstawia następujące zależności między klasami:

- Klasy UI są zależne od klas biznesowych. Oznacza to, że klasy UI korzystają z metod i danych klas biznesowych do wyświetlania danych użytkownikowi.
- Klasy biznesowe mogą być zależne od innych klas biznesowych. Oznacza to, że klasy biznesowe mogą korzystać z metod i danych innych klas biznesowych do obsługi logiki biznesowej aplikacji.
- Na przykład, klasa AdminClient jest zależna od klasy Admin. Oznacza to, że klasa AdminClient korzysta z metod i danych klasy Admin do wyświetlania danych użytkownikowi. Klasa AdminClient jest również zależna od klasy AdminData. Oznacza to, że klasa AdminClient korzysta z metod i danych klasy AdminData do obsługi logiki biznesowej aplikacji.

Interfejs klient



Home - komponent odpowiadający za wyświetlanie strony głównej aplikacji.

Navbar - komponent odpowiadający za wyświetlanie paska nawigacyjnego.

DeliveryOption - komponent odpowiadający za wyświetlanie listy opcji dostawy.

PackageOption - komponent odpowiadający za wyświetlanie listy opcji pakowania.

AddressFormSection - komponent odpowiadający za wyświetlanie formularza adresu.

Order - komponent odpowiadający za wyświetlanie zamówienia.

Komponenty te są podzielone na dwie grupy:

- Komponenty statyczne - odpowiadają za wyświetlanie statycznych danych, takich jak nawigacja czy lista opcji.
- Komponenty dynamiczne - odpowiadają za wyświetlanie dynamicznych danych, takich jak zamówienia.

Komponenty statyczne są reprezentowane przez następujące symbole:

- `<<JSX>>>` - klasa React, która opisuje komponent statyczny.
- `<<CSS>>` - klasa CSS, która określa styl komponentu statycznego.

Komponenty dynamiczne są reprezentowane przez następujące symbole:

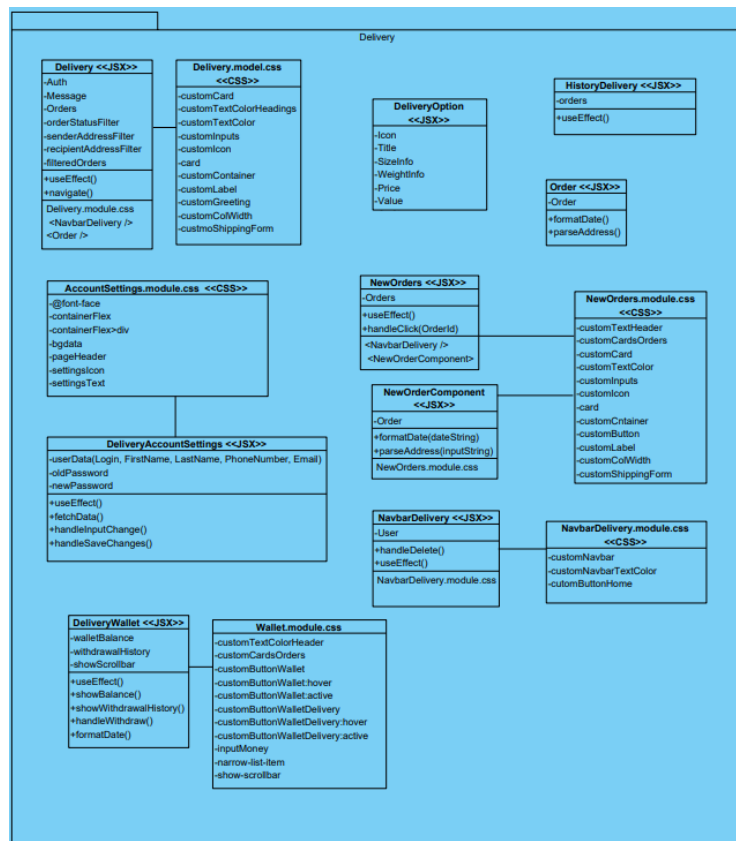
- `<Component />` - komponent React, który opisuje komponent dynamiczny.

Diagram przedstawia następujące zależności między komponentami:

- Komponenty statyczne mogą być zależne od innych komponentów statycznych. Oznacza to, że komponenty statyczne mogą korzystać z metod i danych innych komponentów statycznych do wyświetlania danych użytkownikowi.
- Komponenty dynamiczne mogą być zależne od komponentów statycznych. Oznacza to, że komponenty dynamiczne mogą korzystać z metod i danych komponentów statycznych do wyświetlania danych użytkownikowi.
- Komponenty dynamiczne mogą być zależne od siebie. Oznacza to, że komponenty dynamiczne mogą korzystać z metod i danych innych komponentów dynamicznych do wyświetlania danych użytkownikowi.
- Na przykład, komponent `Order` jest zależny od komponentu `AddressFormSection`. Oznacza to, że komponent `Order` korzysta z metod i danych komponentu `AddressFormSection` do wyświetlania danych użytkownika. Komponent `Order` jest również zależny od komponentu `PackageOption`. Oznacza to, że komponent `Order` korzysta z metod i danych komponentu `PackageOption` do wyświetlania danych o pakowaniu.

Klasa kurier

Diagram przedstawia strukturę aplikacji internetowej dla dostarczania paczek, która skupia się na obsłudze zamówień. Aplikacja składa się z następujących głównych komponentów



Delivery - komponent odpowiadający za wyświetlanie opcji dostawy.

Order - komponent odpowiadający za wyświetlanie zamówienia.

AccountSettings - komponent odpowiadający za wyświetlanie ustawień konta użytkownika.

NewOrders - komponent odpowiadający za wyświetlanie nowych zamówień.

DeliveryWallet - komponent odpowiadający za wyświetlanie portfela dostawczego.

Komponenty statyczne - odpowiadają za wyświetlanie statycznych danych, takich jak nawigacja czy lista opcji.

Komponenty dynamiczne - odpowiadają za wyświetlanie dynamicznych danych, takich jak zamówienia.

Komponenty statyczne są reprezentowane przez następujące symbole:

- `<<JSX>>` - klasa React, która opisuje komponent statyczny.
- `<<CSS>>` - klasa CSS, która określa styl komponentu statycznego.

Komponenty dynamiczne są reprezentowane przez następujące symbole:

- `<Component />` - komponent React, który opisuje komponent dynamiczny.

Procesy biznesowe

Rejestracja użytkownika

Proces rejestracji w aplikacji kurierskiej rozpoczyna się od strony startowej oraz wyrażenia chęci do rejestracji poprzez kliknięcie przycisku "Sing up" na stronie logowania w interfejsie graficznym (GUI). Pierwszym krokiem tego procesu jest wprowadzenie przez użytkownika danych osobowych, adresu e-mail, numeru telefonu, ustawienia hasła do nowego konta oraz wyboru pomiędzy kontem klienta lub kontem kurierskim.

Po wprowadzeniu danych, użytkownik zatwierdza swoje dane poprzez naciśnięcie odpowiedniego przycisku, "Sign up", i oczekuje na odpowiedź serwera dotyczącą powodzenia lub niepowodzenia procesu rejestracji.

Serwer, odbierając prośbę o utworzenie nowego konta użytkownika, sprawdza bazę danych, aby upewnić się, czy podany adres e-mail jest dostępny. Jeśli dane zostały poprawnie wprowadzone, serwer tworzy nowy wpis w bazie danych w tabeli "User". Wynik operacji jest przesyłany z serwera do aplikacji klienckiej.

Następnie na interfejsie graficznym wyświetla się informacja o sukcesie (lub błędzie) całego procesu rejestracji.

Kod źródłowy- rejestracja użytkownika

```
app.post("/signup", (req, res) => {
  bcrypt.hash(req.body.password.toString(), salt, (err, hash) => {
    if (err) return res.json({ Error: "Error for hashing password" });
    const values = [
      req.body.username,
      req.body.fName,
      req.body.lName,
      req.body.phoneNumber,
      req.body.email,
      hash,
    ];
  });
  if ("userType" in req.body) {
    if (req.body.userType.toString() === "user") {
      const qClient =
        "INSERT INTO client (`Login`,`FirstName`,`LastName`,`PhoneNumber`,`Email`,`Password`) VALUES (?)";
      db.query(qClient, [values], (err, data) => {
        if (err) {
          console.error(err);
          return res.status(500).json("Error");
        }
        return res.json(data);
      });
    } else if (req.body.userType.toString() === "delivery") {
      const qDelivery =
        "INSERT INTO delivery (`Login`,`FirstName`,`LastName`,`PhoneNumber`,`Email`,`Password`) VALUES (?)";
      db.query(qDelivery, [values], (err, data) => {
        if (err) {
          console.error(err);
          return res.json({ Error: "Inserting data Error in server" });
        }
        return res.json(data);
      });
    } else {
      return res.json("Invalid userType");
    }
  } else {
    console.log("Nie ma userType");
  }
});
});
```

Interfejs użytkownika (UI)- rejestracja użytkownika

The image shows a mobile application interface for user registration. The background is a solid blue color. In the center, there is a white card with a light beige border. The card has a title 'Sign Up' in bold black text. Below the title, there are six input fields, each with a label above it: 'Username', 'First Name', 'Last Name', 'Phone Number', 'Email', and 'Password'. Each input field has a placeholder text 'Enter [field name]'. Below the input fields, there are two radio buttons with labels 'Client' and 'Delivery'. At the bottom of the card, there is a green button with the text 'Sign up' and a white button with the text 'Login'.

Logowanie użytkownika

Proces logowania w aplikacji kurierskiej rozpoczyna się od strony startowej oraz wyrażenia chęci do zalogowania poprzez kliknięcie przycisku "Log in" na stronie logowania w interfejsie graficznym (GUI). Pierwszym krokiem tego procesu jest wprowadzenie przez użytkownika adresu email oraz hasła.

Po wprowadzeniu danych, użytkownik zatwierdza swoje dane poprzez naciśnięcie odpowiedniego przycisku, "Log in", i oczekuje na odpowiedź serwera dotyczącą poprawności wprowadzonych danych.

Serwer, odbierając prośbę o zalogowanie, sprawdza bazę danych, aby upewnić się, czy podany adres e-mail istnieje i jest zgodny z wprowadzonym przez użytkownika, tak samo dzieje się w przypadku hasła. Jeśli dane zostały poprawnie wprowadzone, użytkownik zostaje zalogowany i przekierowany do strony głównej.

Kod źródłowy- logowanie użytkownika

```
app.post("/login", (req, res) => {
  const { userType, email, password } = req.body;

  let query, table;

  if (userType === "user") {
    query = "SELECT * FROM client WHERE Email = ?";
    table = "client";
  } else if (userType === "delivery") {
    query = "SELECT * FROM delivery WHERE Email = ?";
    table = "delivery";
  } else {
    return res.json({ login: false, message: "Invalid user type" });
  }

  const values = [email];

  db.query(query, values, (err, data) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ Error: "Internal Server Error" });
    }

    if (data.length > 0) {
      bcrypt.compare(password.toString(), data[0].Password, (err, response) => {
        if (err) {
          console.error(err);
          return res.status(500).json({ Error: "Internal Server Error" });
        }

        if (response) {
          const { Login, ID } = data[0];
          const token = jwt.sign({ name: Login, id: ID }, "jwt-secret-key", {
            expiresIn: "1d",
          });

          res.cookie("token", token);

          req.session.username = Login;
          req.session.id = ID;

          return res.json({
            login: true,
            username: req.session.username,
            id: req.session.id,
            userType: table,
          });
        } else {
          return res.json({ login: false, message: "Password not matched" });
        }
      });
    } else {
      return res.json({ login: false, message: "Fail" });
    }
  });
});
```

Interfejs użytkownika (UI)- logowanie użytkownika

The image displays two side-by-side mockups of a user login interface, illustrating a design choice for background color. Both mockups feature a central 'Log In' form with a light beige background and rounded corners. The form includes a title 'Log In' and two tabs: 'Client' (active) and 'Delivery'. Below the tabs are two input fields: 'Email' with the placeholder text 'Enter Email' and 'Password' with the placeholder text 'Enter Password'. A dark green 'Log In' button is positioned below the input fields, and a 'Create Account' link is located at the bottom of the form. The left mockup has a solid blue background, while the right mockup has a solid orange background. Both mockups include a small back arrow icon in the top-left corner.

Wysyłanie paczki

Pierwszym krokiem tego procesu jest określenie punktu docelowego paczki - adres dostarczenia lub paczkomat.

Następnie użytkownik wybiera rozmiar paczki, mając do wyboru trzy opcje: mała, średnia lub duża. Po wybraniu rozmiaru, użytkownik wypełnia dwa formularze z danymi adresowymi odbiorcy oraz nadawcy.

Po wprowadzeniu danych, użytkownik zatwierdza szczegóły paczki poprzez naciśnięcie odpowiedniego przycisku, na przykład "Submit", i oczekuje na potwierdzenie serwera dotyczące przyjęcia danych paczki.

Serwer, odbierając prośbę o wysłanie paczki, sprawdza wprowadzone dane w bazie danych, upewniając się, że adres odbiorcy jest poprawny, a także analizuje wybrany rozmiar paczki. Jeśli dane zostały poprawnie wprowadzone, serwer rejestruje paczkę w systemie i przesyła użytkownikowi potwierdzenie wysłania paczki.

Użytkownik zostaje następnie poinformowany o sukcesie procesu wysyłki, a interfejs graficzny może wyświetlić dodatkowe informacje, takie jak numer przesyłki czy oczekiwany czas dostarczenia. Ten proces zapewnia efektywną i intuicyjną obsługę wysyłki paczek przez użytkowników aplikacji kurierskiej.

Kod źródłowy- wysyłanie paczki

```
app.post("/home", verifyUser, (req, res) => {
  const clientId = req.user.id;
  let date = new Date();
  console.log(clientId.toString());


  const values = [
    req.body.InputZipCode1.toString() +
    req.body.InputCity1.toString() +
    req.body.InputStreet1.toString() +
    req.body.InputBuildingNumber1.toString() +
    req.body.InputApartmentNumber1.toString(),
    req.body.InputZipCode2.toString() +
    req.body.InputCity2.toString() +
    req.body.InputStreet2.toString() +
    req.body.InputBuildingNumber2.toString() +
    req.body.InputApartmentNumber2.toString(),
    date.toISOString().slice(0, 19).replace("T", " "),
    clientId.toString(),
    req.body.packageOption.toString(),
  ];

  const q =
    "INSERT INTO `order` (`SenderAddress`, `RecipientAddress`, `Date`, `ClientID`, `OrderDetailsName`) VALUES (?)"; //change Date data type to DATETIME in DataBase

  console.log(req.body.packageOption.toString());

  db.query(q, [values], (err, data) => {
    if (err) {
      console.error(err);
      return res.status(500).json("Error");
    }
    return res.json(data);
  });
});
```


Interfejs użytkownika (UI)- wysyłanie paczki



My orders Chrupcio ▾


Search for orders...

Delivery destination



Address


The courier will deliver the parcel directly to the address



Shipping point

The courier will deliver the parcel at the shipping point

Pack size




Small

max. 10 x 40 x 65 cm

up to 10 kg

\$15




Medium

max. 20 x 40 x 65 cm

up to 20 kg

\$20



Large

max. 45 x 40 x 65 cm

up to 30 kg

\$25

Shipping details

Sender

Zip Code

38-400

City

Krosno

Street

Świerkowskie

Building

22

Apartment

Recipient

Zip Code

30-034

City

Kraków

Street

Lipirskięgię

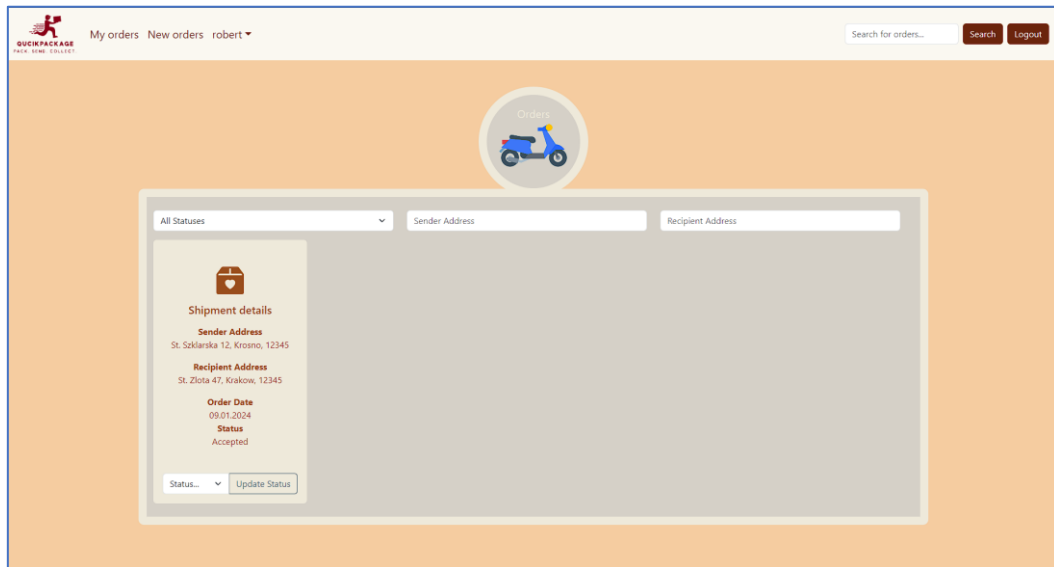
Building

7

Apartment

20

Aktualizacja statusu przesyłki



W powyższym kodzie, endpoint `app.put("/delivery")` obsługuje aktualizację statusu przesyłki. Po uwierzytelnieniu użytkownika, serwer przyjmuje identyfikator zamówienia (`orderId`) i nowy status (`orderstatusid`) z żądania PUT. Następnie wykonuje zapytanie SQL, aktualizując status przesyłki w bazie danych. Jeśli operacja powiedzie się, zwracane jest zapytanie HTTP z kodem 200 i komunikatem o sukcesie. Drugi endpoint `app.get("/delivery/orders")` służy do pobierania zamówień przypisanych do danego dostawcy, pomijając zamówienia ze statusem 5. Otrzymane dane są zwracane jako odpowiedź w formacie JSON. Oba endpointy wymagają uwierzytelnienia użytkownika.

```
app.get("/delivery/orders", verifyUser, (req, res) => {
  const deliveryId = req.user.ID;

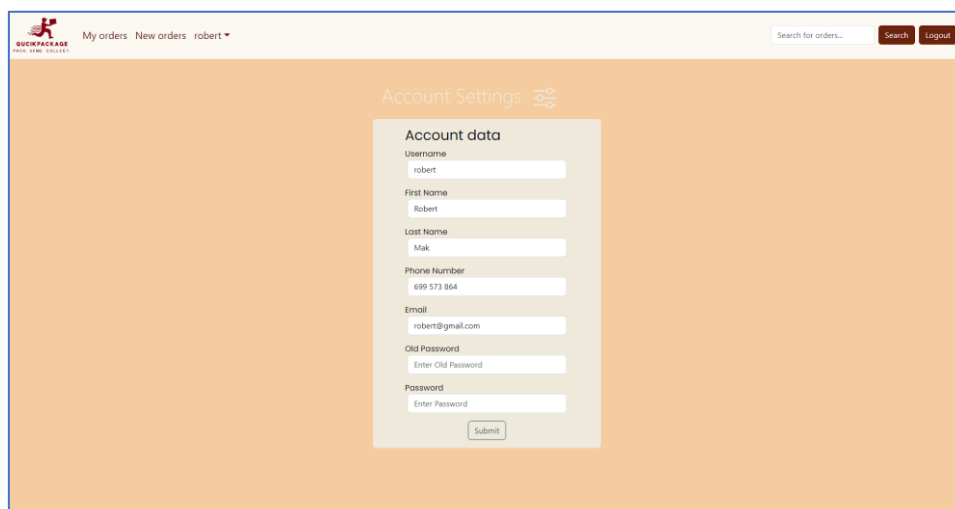
  const q =
    "SELECT `Order`.*, `OrderStatus`.`status` FROM `Order` LEFT JOIN `OrderStatus` ON `Order`.`OrderStatusID`=`OrderStatus`.`ID` WHERE `DeliveryID` = ? AND `Order`.`OrderStatusID` != 5";

  db.query(q, [deliveryId], (err, data) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ error: "Internal server error" });
    }
    return res.json(data);
  });
});
```

```
app.put("/delivery", verifyUser, async (req, res) => {
  const { orderId } = req.body;
  const { orderstatusid } = req.body;
  console.log(orderstatusid + " " + orderId);
  try {
    const updateStatusQuery =
      "UPDATE `Order` SET `OrderStatusID` = ? WHERE `ID` = ?";
    await db.promise().query(updateStatusQuery, [orderstatusid, orderId]);

    return res
      .status(200)
      .json({ message: "Order status updated successfully" });
  } catch (error) {
    console.error("Error updating order status:", error);
    return res.status(500).json({ error: "Internal server error" });
  }
});
```

Zmiana danych użytkownika



Kod Reacta AccountSettings obsługuje zmiany danych użytkownika poprzez interakcję użytkownika na formularzu. Po wprowadzeniu zmian, dane są przesyłane do serwera za pomocą żądań HTTP POST (do weryfikacji hasła) i PUT (do aktualizacji danych). Oto proces zmiany danych użytkownika:

Inicjalizacja Danych

Dane użytkownika, takie jak login, imię, nazwisko, numer telefonu, i email, są pobierane z serwera przy użyciu zapytania GET.

Wprowadzenie Zmian:

Użytkownik wprowadza zmiany w formularzu, w tym zmiany hasła, jeśli konieczne.

Weryfikacja Hasła

Jeśli użytkownik chce zmienić hasło, stary kod weryfikacyjny zostaje wysłany do serwera, który porównuje go z obecnym hasłem w bazie danych.

Aktualizacja Danych

Po weryfikacji hasła, nowe dane są przesyłane do serwera za pomocą zapytania PUT, a tam są aktualizowane w bazie danych.

```

app.post("/:end/verify-password", verifyUser, async (req, res) => {
  try {
    const userId = req.user.ID;
    const { OldPassword } = req.body;
    console.log(userId);

    const getUserQuery = "SELECT Password FROM Client WHERE ID = ?";
    const [userData] = await db.promise().query(getUserQuery, userId);
    console.log("Baza: " + userData[0].Password);
    console.log("OLD: " + OldPassword);

    const isOldPasswordCorrect = await bcrypt.compare(
      OldPassword,
      userData[0].Password
    );

    return res.status(200).json({ valid: isOldPasswordCorrect });
  } catch (error) {
    console.error("Error verifying old password:", error);
    return res.status(500).json({ error: "Internal server error" });
  }
});

```

```

app.put("/:end/update", verifyUser, async (req, res) => {
  const userId = req.user.ID;
  const { end } = req.params;
  const updatedEntityData = req.body;
  let entityType = "";

  if (end === "home") {
    entityType = "Client";
  } else {
    entityType = "Delivery";
  }
  console.log(updatedEntityData);
  try {
    // Check if the entity exists
    const checkEntityQuery = `SELECT * FROM ${entityType} WHERE ID = ?`;
    const [existingEntity] = await db.promise().query(checkEntityQuery, userId);

    if (!existingEntity || existingEntity.length === 0) {
      return res.status(404).json({ error: `${entityType} not found` });
    }

    // Update the entity data
    const updateEntityQuery = `UPDATE ${entityType} SET ? WHERE ID = ?`;
    let updateData = { ...updatedEntityData };

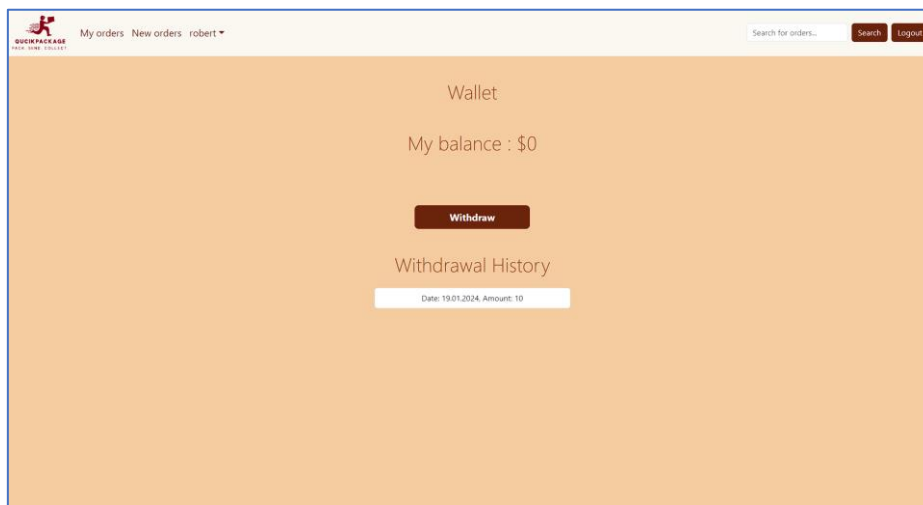
    // Hash the new password before updating, only if provided
    if (updateData.Password) {
      const saltRounds = 10;
      updateData.Password = await bcrypt.hash(updateData.Password, saltRounds);
    }

    await db.promise().query(updateEntityQuery, [updateData, userId]);

    return res
      .status(200)
      .json({ message: `${entityType} data updated successfully` });
  } catch (error) {
    console.error("Error updating ${entityType} data:", error);
    return res.status(500).json({ error: "Internal server error" });
  }
});

```

Wyplata pieniedzy



Kod Reacta obsługuje funkcjonalność portfela dostawcy, umożliwiając mu sprawdzenie salda, wypłatę środków oraz przeglądanie historii wypłat. Poniżej przedstawiam krótką analizę kluczowych elementów:

Pobieranie Salda

Endpoint `"/delivery/wallet"` pozwala na pobranie aktualnego salda portfela dostawcy z bazy danych.

Wyplata Środków

Funkcja `handleWithdraw` obsługuje proces wypłaty środków.

Po kliknięciu przycisku `"Withdraw"`, zostaje wykonane żądanie `POST` do endpointu `"/delivery/withdraw"`.

Jeśli operacja wypłaty zakończy się sukcesem, saldo portfela zostaje ustawione na zero, a historia wypłat jest aktualizowana.

Aktualizacja Salda po Dostarczeniu Paczki

Endpoint `"/delivery/updatewalletdelivery"` jest wywoływany przy aktualizacji statusu zamówienia.

Jeśli status zamówienia to `"5"` (dostarczone), pobierana jest nazwa paczki oraz jej cena.

Na podstawie ceny paczki, portfel dostawcy jest aktualizowany poprzez zapytanie `SQL`.

Aktualne saldo portfela i historia wypłat są wyświetlane w interfejsie.

```

app.post("/delivery/addWageHistory", verifyUser, (req, res) => {
  console.log("TEST")
  const deliveryId = req.user.ID;
  const { BankAccountNumber, AmountWage, PayDate } = req.body;

  const sql =
    "INSERT INTO WageHistory (BankAccountNumber, AmountWage, PayDate, DeliveryID) VALUES (?, ?, ?, ?)";
  const values = [BankAccountNumber, AmountWage, PayDate, deliveryId];

  db.query(sql, values, (err, result) => {
    if (err) {
      console.error("Błąd przy dodawaniu do WithdrawalHistory:", err);
      res
        .status(500)
        .json({
          success: false,
          message: "Błąd przy dodawaniu do WithdrawalHistory",
        });
    } else {
      res
        .status(200)
        .json({
          success: true,
          message: "Dodano do WithdrawalHistory",
          withdrawalID: result.insertId,
        });
    }
  });
});

```

```

app.put("/delivery/updatewalletdelivery", verifyUser, async (req, res) => {
  const deliveryID = req.user.ID;
  const { orderId } = req.body;
  const { orderstatusid } = req.body;

  if (orderstatusid == "5") {
    const packageNameQuery =
      "SELECT OrderDetailsName FROM `Order` WHERE ID = ?";

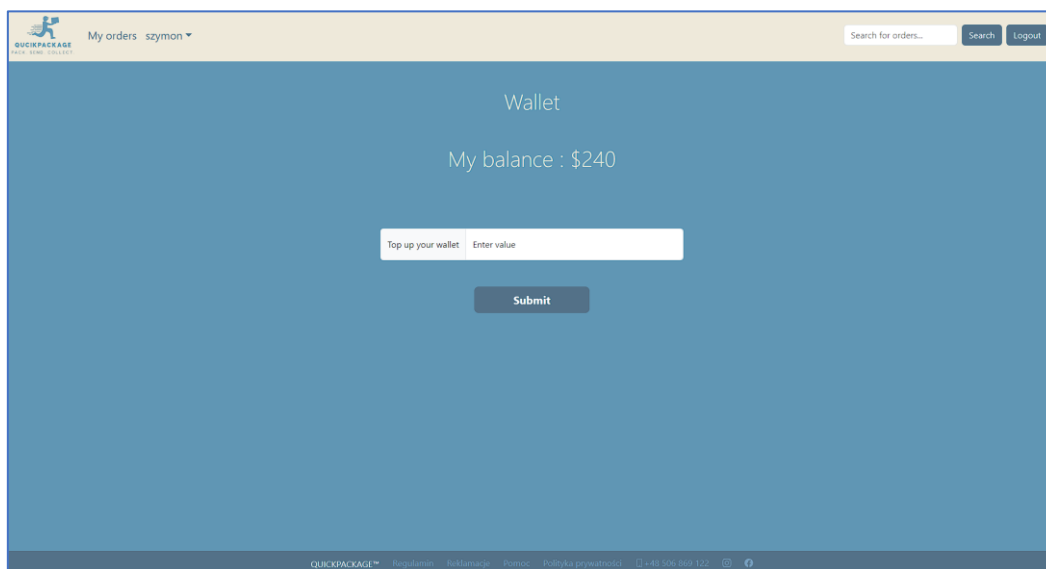
    // Pobieranie nazwy paczki na podstawie orderId
    db.query(packageNameQuery, [orderId], async (nameErr, nameResult) => {
      if (nameErr || nameResult.length === 0) {
        res.status(500).send("Error fetching package name");
      } else {
        const packageName = nameResult[0].OrderDetailsName;

        // Pobieranie ceny paczki na podstawie jej nazwy
        const packageCostQuery =
          "SELECT Price FROM OrderDetails WHERE Name = ?";
        db.query(packageCostQuery, [packageName], async (err, result) => {
          if (err || result.length === 0) {
            res.status(500).send("Error fetching package cost");
          } else {
            const packageCost = result[0].Price;
            const comissionPrice = packageCost * 0.5;

            // Kod aktualizujący portfel kuriera w bazie danych
            const updateWalletQuery =
              "UPDATE DeliveryWallet SET Balance = Balance + ? WHERE DeliveryID = ?";
            db.query(
              updateWalletQuery,
              [comissionPrice, deliveryID],
              (walletErr, walletResult) => {
                if (walletErr) {
                  res.status(500).send("Error updating courier's wallet");
                } else {
                  res.status(200).send("Courier's wallet updated successfully");
                }
              }
            );
          }
        });
      }
    });
  }
});

```

Wpłata pieniędzy na konto



Poniższy kod Reacta obsługuje funkcjonalność portfela klienta, umożliwiając mu sprawdzenie salda, doładowanie konta oraz przeglądanie historii transakcji. Poniżej przedstawiam krótką analizę kluczowych elementów:

Pobieranie Salda

Endpoint `"/home/walletBalance"` pozwala na pobranie aktualnego salda portfela klienta z bazy danych.

Doładowanie Konta

Funkcja `handleTopUp` obsługuje proces doładowania konta klienta.

Wprowadzona kwota doładowania jest sprawdzana, czy jest liczbą dodatnią. Następnie jest wysyłana na serwer za pomocą żądania POST do endpointu `"/home/topup"`.

Po wykonaniu udanego doładowania, strona jest przeładowywana za pomocą `window.location.reload()`.

Aktualizacja Salda po Doładowaniu

Endpoint `"/home/topup"` jest wywoływany po pomyślnym doładowaniu konta.

Saldo portfela klienta jest aktualizowane poprzez zapytanie SQL.

```

app.get("/home/walletBalance", verifyUser, (req, res) => {
  const clientId = req.user.ID;

  const selectQuery = "SELECT Balance FROM Wallet WHERE ClientID = ?";

  db.query(selectQuery, [clientId], (err, result) => {
    if (err) {
      console.error("Error fetching wallet balance:", err);
      return res.status(500).json({ error: "Internal server error" });
    }

    if (result.length === 0) {
      return res.status(404).json({ balance: 0 });
    }

    const walletBalance = result[0].Balance;
    return res.status(200).json({ balance: walletBalance });
  });
});

```

```

app.post("/home/topup", verifyUser, (req, res) => {
  const { amount } = req.body;
  const clientId = req.user.ID;
  //sprawdzenie czy amount jest liczbą dodatnia
  if (isNaN(amount) || amount <= 0) {
    return res.status(400).json({ error: "Invalid top-up amount" });
  }

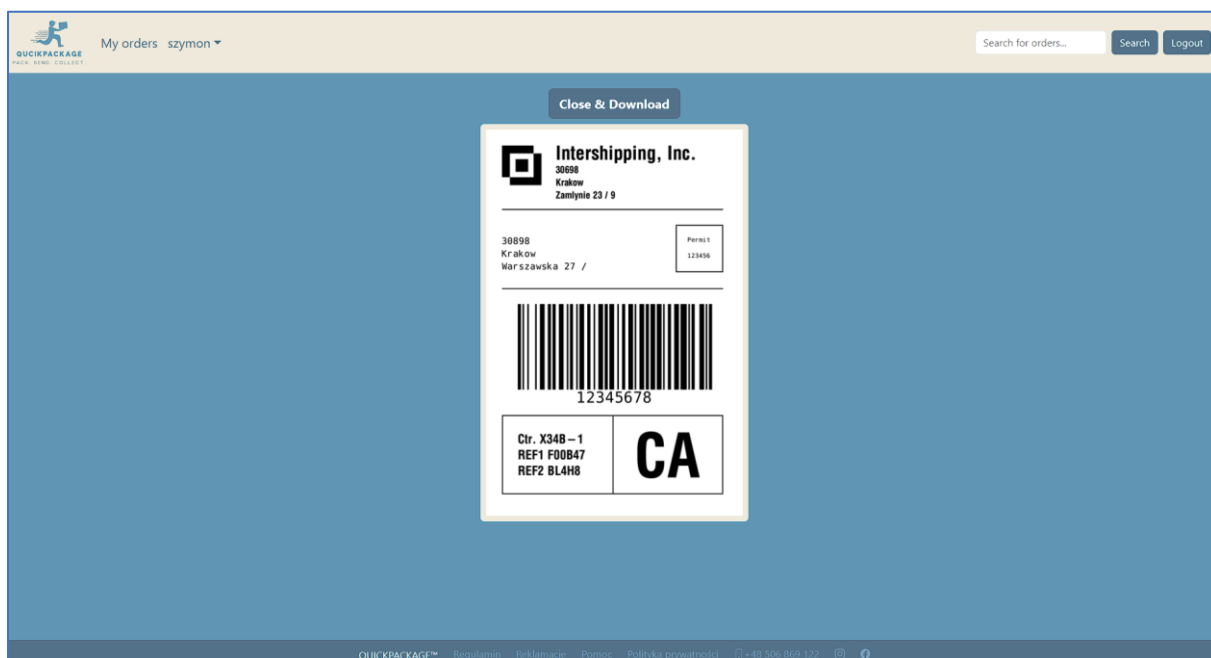
  const updateQuery =
    "UPDATE Wallet SET Balance = Balance + ? WHERE ClientID = ?";

  db.query(updateQuery, [amount, clientId], (err, data) => {
    if (err) {
      console.error("Error updating wallet balance:", err);
      return res.status(500).json({ error: "Internal server error" });
    }

    console.log("Wallet balance updated successfully");
    return res.json(data);
  });
});

```

Wydruk ZPL



Dla użytkownika generowanie ZPL odbywa się w trzech głównych krokach:

- Wybór Rozmiaru Paczki

Użytkownik ma możliwość wyboru rozmiaru paczki spośród trzech opcji: Small, Medium, i Large.

Z każdym rozmiarem paczki związana jest cena, która jest uwzględniana w saldzie portfela użytkownika.

- Wypełnienie Szczegółów Wysyłki

Użytkownik wprowadza szczegóły dotyczące adresu nadawcy i odbiorcy.

Warto zauważyć, że formularz zawiera walidację, więc użytkownik otrzymuje informacje zwrotne w przypadku błędów wprowadzonych danych.

- Wysłanie Przesyłki

Po poprawnym wypełnieniu formularza, użytkownik wysyła przesyłkę, a system generuje etykietę ZPL z danymi przesyłki.

Użytkownik może zobaczyć podgląd wygenerowanej etykiety na stronie.

Etykieta zawiera informacje o nadawcy, odbiorcy, kodzie kreskowym, a także pola na pozwolenie i inne dane.

Użytkownik ma możliwość pobrania wygenerowanej etykiety w formie pliku PNG.

Podsumowując, dla użytkownika proces generowania ZPL jest zintegrowany w interfejsie użytkownika aplikacji. Wybiera rozmiar paczki, wprowadza dane i odbiera gotową etykietę, która jest gotowa do druku i naklejenia na przesyłkę.

```
const handleGenerateZPL = async () => {
  try {
    const validationErrors = Validation(values);

    if (
      /validationErrors.InputZipCode1 &&
      /validationErrors.InputZipCode2 &&
      /validationErrors.InputCity1 &&
      /validationErrors.InputCity2 &&
      /validationErrors.InputStreet1 &&
      /validationErrors.InputStreet2 &&
      /validationErrors.InputBuildingNumber1 &&
      /validationErrors.InputBuildingNumber2
    ) {
```

```
    try {
      const generatedZPL = `^XA

^FX Top section with logo, name and address.
^CF0,60
^F050,50^GB100,100,100^FS
^F075,75^FR^GB100,100,100^FS
^F093,93^GB40,40,40^FS
^F0220,50^FDIntershipping, Inc.^FS
^CF0,30
^F0220,115^FD${convertToZPLString(values.InputZipCode1)}^FS
^F0220,155^FD${convertToZPLString(values.InputCity1)}^FS
^F0220,195^FD${convertToZPLString(
  values.InputStreet1
)}^FS
^F0220,195^FD${convertToZPLString(
  values.InputBuildingNumber1
)}^FS
^F050,250^GB700,3,3^FS

^FX Second section with recipient address and permit information.
^CFA,30

^F050,340^FD${convertToZPLString(values.InputZipCode2)}^FS
^F050,380^FD${convertToZPLString(values.InputCity2)}^FS
^F050,420^FD${convertToZPLString(
  values.InputStreet2
)}^FS
^F050,420^FD${convertToZPLString(
  values.InputBuildingNumber2
)}^FS
^F050,420^FD${convertToZPLString(values.InputApartmentNumber2)}^FS
^CFA,15
^F0600,300^GB150,150,3^FS
^F0638,340^FDPPermit^FS
^F0638,390^FD123456^FS
^F050,500^GB700,3,3^FS

^FX Third section with bar code.
^BYS,2,270
^F0100,550^BC^FD12345678^FS

^FX Fourth section (the two boxes on the bottom).
^F050,900^GB700,250,3^FS
^F0400,900^GB3,250,3^FS
^CF0,40
^F0100,960^FDctr. X348-1^FS
^F0100,1010^FDREF1 F00847^FS
^F0100,1060^FDREF2 BL4H8^FS
^CF0,190
^F0470,955^FDCA^FS

^XZ`;
```

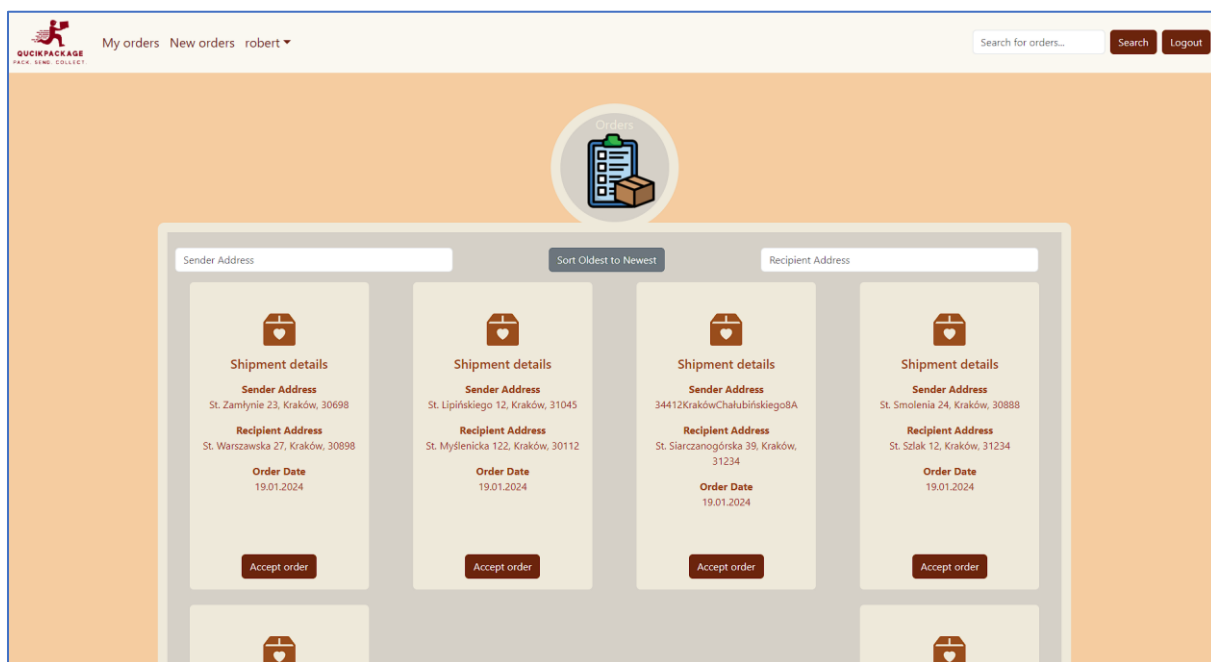
```
const response = await fetch(
  "http://api.labelary.com/v1/printers/8dpm/labels/4x6/0/",
  {
    method: "POST",
    headers: {
      Accept: "image/png",
      "Content-Type": "application/x-www-form-urlencoded",
    },
    body: `${generatedZPL}`,
  }
);

if (response.ok) {
  const imageUrl = URL.createObjectURL(await response.blob());
  setLabelImage(imageUrl);
  setZPLVisible(true);
} else {
  const errorMessage = await response.text();
  console.error("Error generating label:", errorMessage);
} catch (error) {
  console.error("Error generating label:", error);
}

} catch (err) {
  console.error("Error in handleGenerateZPL:", err);
}
};

const handleDownloadImage = () => {
  const link = document.createElement("a");
  link.href = labelImage;
  link.download = "label_image.png";
  link.click();
};
```

Przyjmowanie zamówień



Przyjmowanie Zamówień jako Kurier

Endpoint ten jest chroniony middlewarem `verifyUser`, co oznacza, że dostęp do niego mają jedynie uwierzytelnieni użytkownicy (kurierzy).

W celu uzyskania zamówień, kurier identyfikowany jest poprzez ID dostępu, a następnie wykonywane jest zapytanie SQL do bazy danych w celu pobrania wszystkich nieprzypisanych do innych kurierów zamówień.

Utworzony jest endpoint typu PUT `/delivery/neworders/:orderId`.

Kurier może przypisać sobie zamówienie, podając jego identyfikator w parametrze ścieżki (`:orderId`).

Przed aktualizacją zamówienia w bazie danych, sprawdzane jest, czy zamówienie istnieje i czy nie zostało wcześniej przypisane do innego kuriera. Jeśli tak, zwracany jest błąd 404.

Jeśli zamówienie spełnia warunki, to jest aktualizowane w bazie danych poprzez zapytanie SQL.

Status zamówienia ustawiany jest na "Accepted", a `DeliveryID` (ID kuriera) jest ustawiane na ID aktualnie zalogowanego kuriera.

W pliku klienta (frontend):

Strona Widoku Dostępnych Nowych Zamówień:

Kurier ma dostęp do strony NewOrders.

Na tej stronie wyświetlane są dostępne nowe zamówienia, z możliwością filtrowania według adresu nadawcy i odbiorcy.

Zamówienia są również sortowane według daty - od najnowszego do najstarszego lub od najstarszego do najnowszego, w zależności od aktualnie wybranej opcji sortowania.

Kurier ma możliwość kliknięcia na przycisk "Sort Oldest to Newest" / "Sort Newest to Oldest" w celu zmiany kierunku sortowania.

Interakcja z Zamówieniami:

Kurier ma możliwość przypisania sobie konkretnego zamówienia poprzez kliknięcie na odpowiednie zamówienie.

Po przypisaniu zamówienia, aktualizacja jest wysyłana na serwer, a widok zamówień jest odświeżany.

Zamówienia, które zostały przypisane do kuriera, są wyłączone z widoku dla innych kurierów.

To podejście umożliwia kurierowi przeglądanie i przyjmowanie dostępnych zamówień, co pozwala na efektywne zarządzanie przesyłkami i ich dostawą.

```
app.get("/delivery/neworders", verifyUser, (req, res) => {
  const deliveryId = req.user.ID;

  const q = "SELECT * FROM `Order` WHERE `DeliveryID` IS NULL";

  db.query(q, [deliveryId], (err, data) => {
    if (err) {
      console.error(err);
      return res.status(500).json("Error");
    }
    return res.json(data);
  });
});
```

```

app.put("/delivery/neworders/:orderId", verifyUser, async (req, res) => {
  const { orderId } = req.params;
  console.log("Received orderId:", orderId);
  const deliveryId = req.user.ID;
  console.log("DELIVERY", deliveryId);

  try {
    const orderQuery =
      "SELECT * FROM `Order` WHERE `ID` = ? AND `DeliveryID` IS NULL";
    const [order] = await db.promise().query(orderQuery, [orderId]);

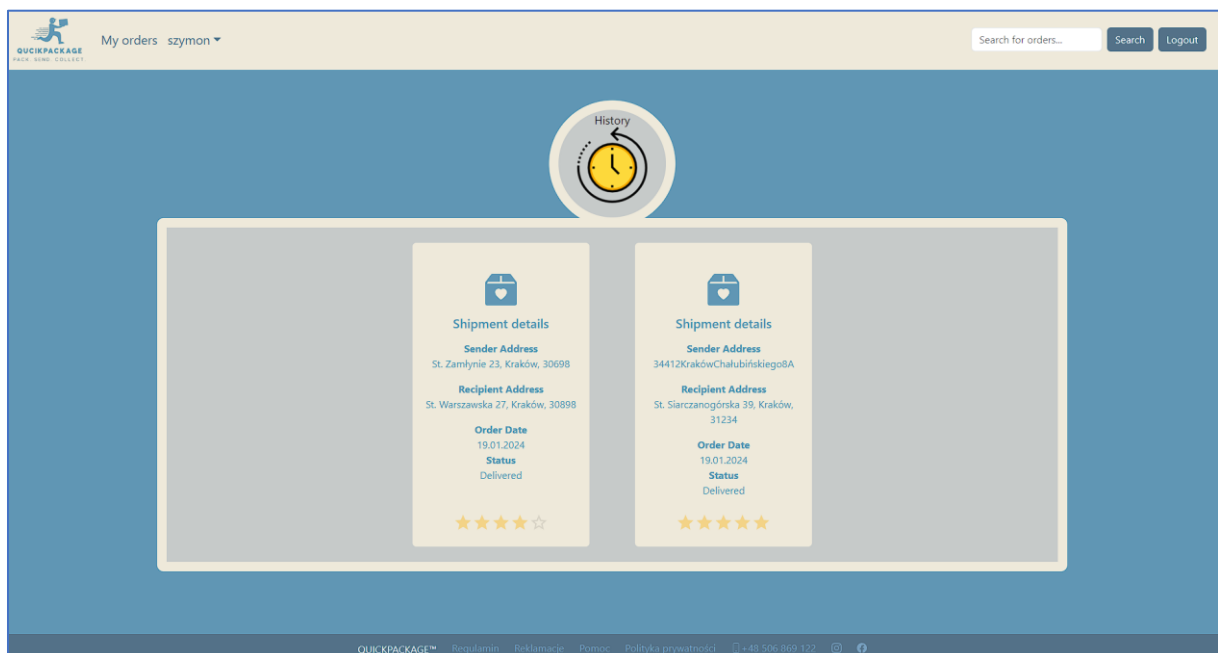
    if (!order || order.length === 0) {
      return res.status(404).json({
        error: "Order not found or already assigned to a delivery person",
      });
    }

    const updateQuery =
      "UPDATE `Order` SET `DeliveryID` = ?, `OrderStatusID` = 2 WHERE `ID` = ?";
    await db.promise().query(updateQuery, [deliveryId, orderId]);

    return res.status(200).json({ message: "Order updated successfully" });
  } catch (error) {
    console.error("Error updating order:", error);
    return res.status(500).json({ error: "Internal server error" });
  }
});

```

Historia i ocena zamówień



Komponent BasicRating obsługuje ocenianie zamówień poprzez interakcję klienta, zapisując oceny lokalnie i wysyłając aktualizacje do serwera.

Komponent Order umożliwia wyświetlanie szczegółów zamówienia, aktualizację statusu przez kuriera i wystawienie oceny przez klienta.

Strona HistoryClient prezentuje historię zamówień klienta, umożliwiając ocenę każdego zamówienia przy użyciu komponentu BasicRating.

Funkcje pomocnicze, takie jak formatowanie daty i adresu, zostały uwzględnione w komponencie Order.

Wszystkie interakcje z serwerem, takie jak aktualizacja statusu i oceny, są obsługiwane za pomocą odpowiednich endpointów na serwerze HTTP.

```
app.get("/home/history", verifyUser, (req, res) => {
  const clientId = req.user.ID;

  const q =
    "SELECT `Order`.*, OrderStatus.status FROM `Order` LEFT JOIN OrderStatus ON Order.OrderStatusID=OrderStatus.ID WHERE `ClientID` = ? AND Order.OrderStatusID = 5";

  db.query(q, [clientId], (err, data) => {
    if (err) {
      console.error(err);
      return res.status(500).json("Error");
    }
    return res.json(data);
  });
});
```

```
app.post("/home/rateOrder", verifyUser, async (req, res) => {
  const { orderId, orderRate } = req.body;

  try {
    const checkOrderQuery = "SELECT * FROM `Order` WHERE ID = ?";
    const [existingOrder] = await db
      .promise()
      .query(checkOrderQuery, [orderId]);

    if (!existingOrder || existingOrder.length === 0) {
      return res.status(404).json({ error: "Order not found" });
    }

    const addRatingQuery =
      "INSERT INTO Rate (OrderID, OrderRate) VALUES (?, ?)";
    await db.promise().query(addRatingQuery, [orderId, orderRate]);

    return res.status(200).json({ message: "Order rated successfully" });
  } catch (error) {
    console.error("Error rating order:", error);
    return res.status(500).json({ error: "Internal server error" });
  }
});
```

```

const BasicRating = ({ uniqueId }) => {
  const storedValue = localStorage.getItem('ratingValue_${uniqueId}');
  const storedIsDisabled = localStorage.getItem('isDisabled_${uniqueId}');

  const [value, setValue] = React.useState(
    storedValue ? JSON.parse(storedValue) : 0
  );
  const [isDisabled, setIsDisabled] = React.useState(
    storedIsDisabled ? JSON.parse(storedIsDisabled) : false
  );

  const handleRatingChange = (event, newValue) => {
    setValue(newValue);

    axios
      .post("http://localhost:8081/home/rateOrder", {
        orderId: uniqueId,
        orderRate: newValue,
      })
      .then((response) => {
        console.log("Order rated successfully:", response.data);
      })
      .catch((error) => {
        console.error("Error rating order", error);
      });
    handleToggleDisabled();
  };

  const handleToggleDisabled = () => {
    setIsDisabled(!isDisabled);
  };

  React.useEffect(() => {
    localStorage.setItem('ratingValue_${uniqueId}', JSON.stringify(value));
    localStorage.setItem('isDisabled_${uniqueId}', JSON.stringify(isDisabled));
  }, [value, isDisabled, uniqueId]);

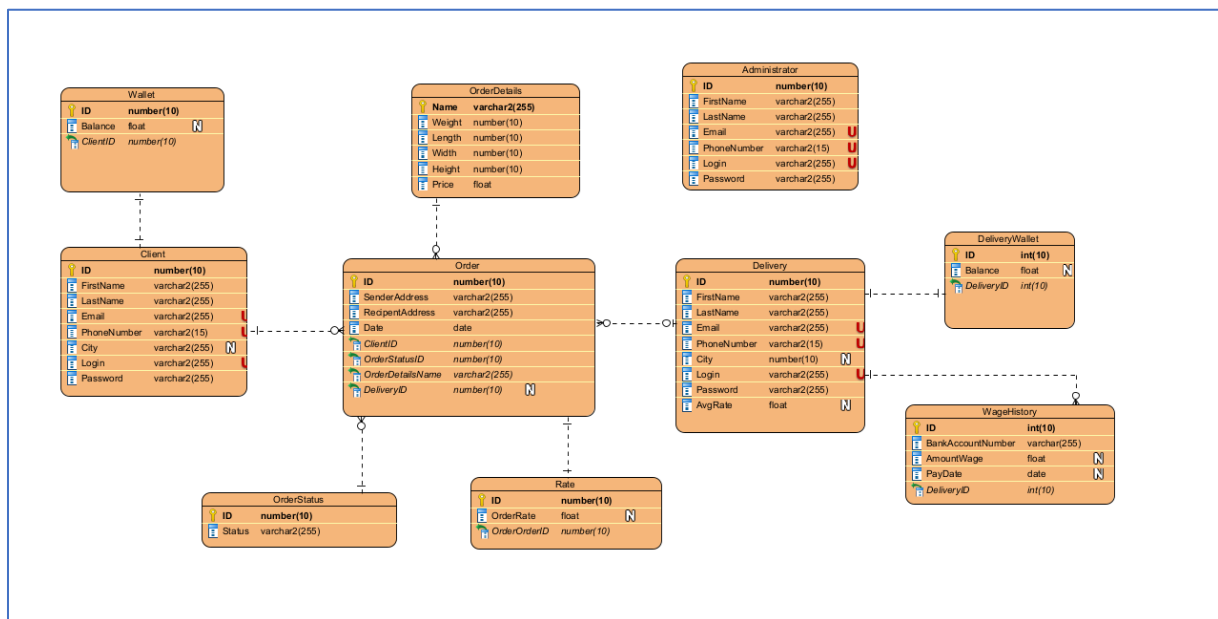
  return (
    <Box>
      <Box>
        <Text>
          "a > legend": { mt: 2 },
        </Text>
      </Box>
      <Rating>
        name={`simple-controlled_${uniqueId}`}
        value={value}
        onChange={handleRatingChange}
        size="large"
        disabled={isDisabled}
      </Rating>
    </Box>
  );
};

```

Baza danych

Client	zawiera informacje o klientach, takie jak imię, nazwisko, unikalny adres e-mail, numer telefonu, miasto zamieszkania, unikalna nazwa użytkownika i hasło. Każdy klient ma przypisane automatycznie generowane ID, które pełni rolę klucza głównego.
Order	reprezentuje zamówienia, zawierając informacje o adresie nadawcy i odbiorcy, dacie zamówienia, identyfikatorze klienta, statusie zamówienia, szczegółach zamówienia, oraz dostawcy. Każde zamówienie posiada unikalne ID jako klucz główny.
Delivery	przechowuje dane dostawców, takie jak imię, nazwisko, unikalny adres e-mail, numer telefonu, identyfikator miasta, unikalna nazwa użytkownika, hasło oraz średnia ocena dostawcy. Dostawcy identyfikowani są za pomocą automatycznie generowanego ID, które pełni funkcję klucza głównego.
Wallet	zawiera informacje o portfelach klientów, w tym unikalne ID, saldo oraz przypisany klient.
DeliveryWallet	przechowuje informacje o portfelach dostawców, takie jak unikalne ID, saldo oraz przypisany dostawca.
WageHistory	zawiera historię wypłat dla dostawców, obejmując numer konta bankowego, kwotę wypłaty, datę wypłaty oraz identyfikator dostawcy. Każdy wpis ma unikalne ID jako klucz główny.
OrderDetails	przechowuje szczegóły dotyczące zamówień, takie jak nazwa, waga, długość, szerokość, wysokość i cena. Klucz główny to nazwa.
Rate	przechowuje oceny zamówień, z każdą oceną identyfikowaną przez unikalne ID, wartość oceny oraz powiązaną z nią identyfikację zamówienia.
OrderStatus	przechowuje różne statusy zamówień, z każdym statusem identyfikowanym za pomocą automatycznie generowanego ID jako klucza głównego.
Administrator	zawiera informacje o administratorach systemu, takie jak imię, nazwisko, unikalny adres e-mail, numer telefonu, unikalna nazwa użytkownika i hasło. Każdy administrator ma przypisane automatycznie generowane ID, pełniące rolę klucza głównego.

Schemat bazy danych



Wyniki fazy testowania

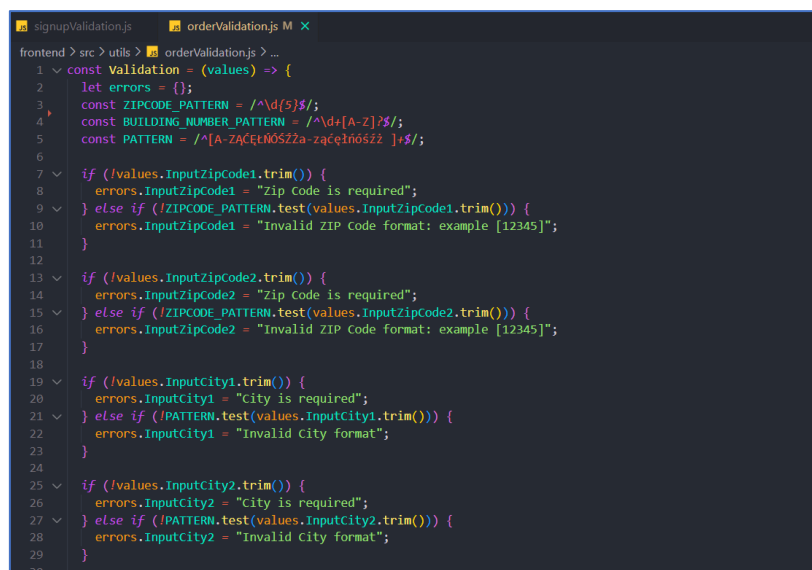
W trakcie uzupełniania formularza zauważono, że wprowadzone znaki polskie nie są poprawnie interpretowane jako ich odpowiedniki w postaci dużych liter. To utrudniało poprawne wypełnienie formularza, zwłaszcza w przypadku pól wymagających wprowadzenia dużych liter, takich jak imię czy nazwisko. Zaleca się poprawę tej funkcjonalności, aby umożliwić użytkownikom komfortowe i poprawne korzystanie z formularza bez konieczności unikania znaków polskich.

The screenshot displays a 'Shipping details' form with two main sections: 'Sender' and 'Recipient'. Each section contains input fields for 'Zip Code', 'City', 'Street', 'Building', and 'Apartment'. The 'City' field for the 'Sender' is currently filled with 'Łódź', which has triggered a red error message: 'City must start with an uppercase letter'. The 'Recipient' section shows the same fields filled with 'Łódź', 'Eqwea', and '22'. A 'Submit' button is located at the bottom center of the form. Above the form, there are three price tags: '\$15', '\$20', and '\$25'.

Shipping details					
Sender			Recipient		
Zip Code 12334	City Łódź		Zip Code 12334	City Łódź	
City must start with an uppercase letter			City must start with an uppercase letter		
Street Atetwtvet	Building 123	Apartment	Street Eqwea	Building 22	Apartment
<div>Submit</div>					

Rozwiązanie napotkanych problemów

Rozwiązaniem problemu było zmienienie stałej PATTERN w celu uwzględnienia poprawnego rozpoznawania dużej litery w imionach miast zawierających polskie znaki diakrytyczne.



```
1 const Validation = (values) => {
2   let errors = {};
3   const ZIPCODE_PATTERN = /^d{5}$/;
4   const BUILDING_NUMBER_PATTERN = /^d+[A-Z]?$/;
5   const PATTERN = /^[A-ZĄĆĘŁŃÓŚŻŁa-ąćęłńóśżł]+$/;
6
7   if (!values.InputZipCode1.trim()) {
8     errors.InputZipCode1 = "Zip Code is required";
9   } else if (!ZIPCODE_PATTERN.test(values.InputZipCode1.trim())) {
10    errors.InputZipCode1 = "Invalid ZIP Code format: example [12345]";
11  }
12
13  if (!values.InputZipCode2.trim()) {
14    errors.InputZipCode2 = "Zip Code is required";
15  } else if (!ZIPCODE_PATTERN.test(values.InputZipCode2.trim())) {
16    errors.InputZipCode2 = "Invalid ZIP Code format: example [12345]";
17  }
18
19  if (!values.InputCity1.trim()) {
20    errors.InputCity1 = "City is required";
21  } else if (!PATTERN.test(values.InputCity1.trim())) {
22    errors.InputCity1 = "Invalid City format";
23  }
24
25  if (!values.InputCity2.trim()) {
26    errors.InputCity2 = "City is required";
27  } else if (!PATTERN.test(values.InputCity2.trim())) {
28    errors.InputCity2 = "Invalid City format";
29  }
30}
```

^ - Oznacza początek ciągu.

[A-ZĄĆĘŁŃÓŚŻŁa-ąćęłńóśżł]+ - Określa zbiorczo, jakie znaki są akceptowane.

W tym przypadku:

A-Z: Duże litery od A do Z,

ĄĆĘŁŃÓŚŻŁ: Polskie znaki diakrytyczne,

a-z: Małe litery od a do z,

ąćęłńóśżł: Polskie znaki diakrytyczne,

\s: Białe znaki (spacje, tabulatory itp.).

+ - Oznacza, że poprzedzający element (cały ten zestaw znaków) może wystąpić jeden lub więcej razy.

\$ - Oznacza koniec ciągu.

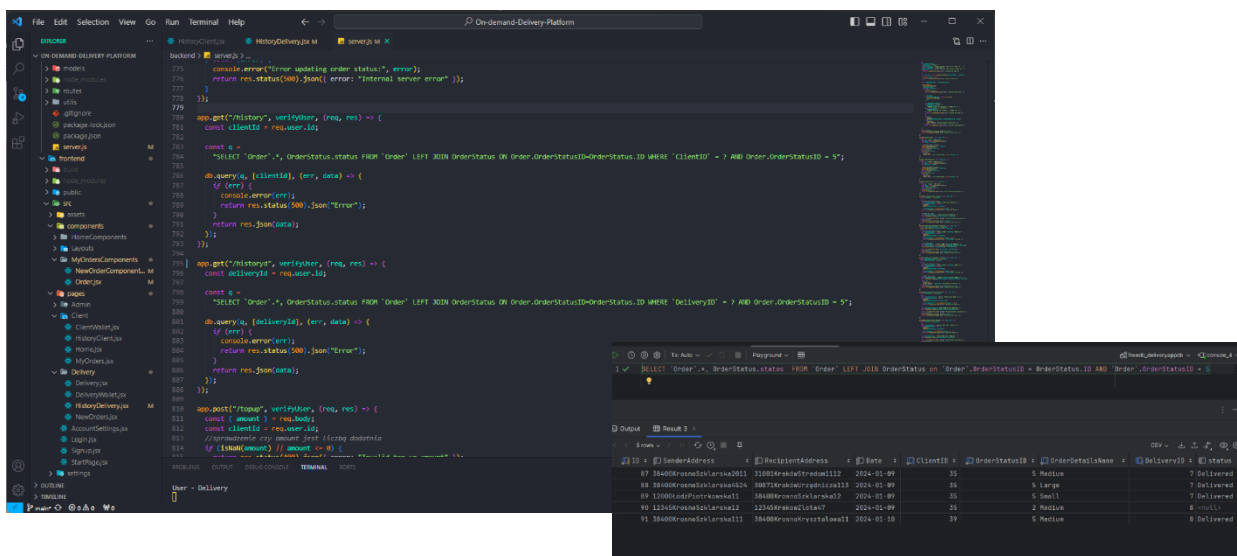
W rezultacie, wyrażenie to akceptuje ciągi znaków, które składają się wyłącznie z liter (zarówno dużej, jak i małej), polskich znaków diakrytycznych oraz białych znaków, nie dopuszczając innych znaków ani na początku, ani na końcu ciągu.

Problem, który może pojawić się podczas zmiany bazy danych i być związany z tym faktem, że nazwy kolumn zaczynają się od dużej litery.

Przykładowo, niektóre systemy baz danych traktują identyfikatory (np. nazwy tabel, kolumn) jako nieczułe na wielkość liter, co oznacza, że są one traktowane identycznie niezależnie od tego, czy są zapisane wielkimi czy małymi literami. Inne systemy baz danych zachowują wrażliwość na wielkość liter, co oznacza, że rozróżniają pomiędzy wielkimi i małymi literami w identyfikatorach.

Gdy zmieniamy bazę danych lub przenosimy schemat bazy danych między różnymi systemami, może dojść do sytuacji, gdzie jeden system traktuje nazwy kolumn jako nieczułe na wielkość liter, a inny jako wrażliwe na wielkość liter. W takim przypadku, jeśli pierwotna baza danych używała identyfikatorów zaczynających się od małych liter, a nowa baza danych wymaga identyfikatorów zaczynających się od dużych liter, może to prowadzić do konfliktów i błędów w odwołaniach do kolumn.

Rozwiązaniem tego problemu jest dostosowanie nazw kolumn w nowej bazie danych do wymagań dotyczących wielkości liter, co pozwala na spójność identyfikatorów i unika problemów związanych z różnicami w obszarze czułości na wielkość liter między systemami baz danych.



```
1000 GET /api/orders 200 OK
1001 Content-Type: application/json
1002 {
1003   "orders": [
1004     {
1005       "id": 1,
1006       "client_id": "client1",
1007       "status": "pending",
1008       "created_at": "2024-01-01T12:00:00Z",
1009       "updated_at": "2024-01-01T12:00:00Z"
1010     },
1011     {
1012       "id": 2,
1013       "client_id": "client2",
1014       "status": "delivered",
1015       "created_at": "2024-01-01T13:00:00Z",
1016       "updated_at": "2024-01-01T13:00:00Z"
1017     }
1018   ]
1019 }
1020
```

```
1001 SELECT "Order"."*", OrderStatus.status FROM "Order" LEFT JOIN OrderStatus ON Order.OrderStatusID=OrderStatus.ID WHERE "ClientID" = ? AND Order.OrderStatusID = 5;
```

Proces wdrożenia projektu

Proces wdrażania projektu strony internetowej obejmuje szereg kroków, które mają na celu przeniesienie projektu z etapu rozwoju do środowiska produkcyjnego, dostępnego dla użytkowników. Oto ogólny opis procesu wdrażania:

Przygotowanie na produkcję:

- Konfiguracja serwera: Skonfiguruj środowisko serwera, uwzględniając wszelkie zależności, takie jak serwer HTTP (np. Apache, Nginx), baza danych, serwer aplikacyjny itp.
- Bezpieczeństwo: Upewnij się, że wszelkie ustawienia zabezpieczeń są skonfigurowane, takie jak firewalle, certyfikaty SSL/TLS, zabezpieczenia dostępu do katalogów i plików.

Przeniesienie plików i baz danych:

- Transfer plików: Skopiuj pliki projektu na serwer produkcyjny. Może to obejmować pliki HTML, CSS, JavaScript, JSX obrazy i inne zasoby statyczne.
- Import bazy danych: Przenieś bazę danych z środowiska deweloperskiego na serwer produkcyjny.

Konfiguracja środowiska:

- Ustawienia serwera: Dostosuj ustawienia serwera do środowiska produkcyjnego, takie jak przekierowania, konfiguracja wirtualnych hostów itp.
- Zmienne środowiskowe: Ustaw wszelkie zmienne środowiskowe, takie jak klucze API, ścieżki plików, dane konfiguracyjne.

Testy na produkcji:

- Testy integrowane: Przeprowadź testy, aby upewnić się, że wszystkie komponenty współpracują ze sobą poprawnie w środowisku produkcyjnym.
- Monitorowanie: Uruchom narzędzia monitorujące, aby śledzić wydajność, obciążenie serwera i błędy na produkcji.

Certyfikacja bezpieczeństwa:

- Audyt bezpieczeństwa: Sprawdź projekt pod kątem potencjalnych luk bezpieczeństwa.
- Zabezpieczenia HTTP: Włącz odpowiednie zabezpieczenia HTTP, takie jak nagłówki bezpieczeństwa, unikaj narażania danych wrażliwych.

Backup i rollback:

- Backup: Przed wdrożeniem, zrób pełen backup plików i bazy danych w celu zminimalizowania ryzyka utraty danych.
- Rollback plan: Przygotuj plan cofania zmian w razie konieczności.

Informowanie zespołu i klienta:

- Komunikacja: Powiadom zespół deweloperski, administratorów i klienta o planowanym wdrożeniu.
- Dokumentacja: Zaktualizuj dokumentację projektu z informacjami dotyczącymi wersji i zmian.

Wdrożenie:

- Planowane okno czasowe: Wybierz odpowiednie okno czasowe na wdrożenie, minimalizując wpływ na użytkowników.
- Automatyzacja wdrożenia: Jeśli to możliwe, skorzystaj z narzędzi do automatyzacji wdrożeń, aby zminimalizować ryzyko ludzkich błędów.

Monitorowanie po wdrożeniu:

- Monitorowanie bieżące: Śledź wydajność i stabilność strony po wdrożeniu.
- Rozwiązywanie problemów: Monitoruj logi i reaguj na ewentualne problemy, które mogą się pojawić.

Proces wdrażania projektu strony internetowej wymaga skoordynowanego wysiłku zespołu deweloperskiego, administratorów systemów oraz innych interesariuszy. Ważne jest również zachowanie ostrożności i zabezpieczenie przed ewentualnymi problemami poprzez stosowanie praktyk bezpieczeństwa i dokładne testowanie przed przejściem do środowiska produkcyjnego.

Perspektywy rozwoju projektu

Perspektywy rozwoju projektu aplikacji dostawczej na żądanie (On-Demand Delivery App) mogą obejmować różnorodne obszary, zależnie od aktualnych potrzeb rynku, technologicznych trendów oraz specyfiki branży. Poniżej przedstawiam kilka potencjalnych perspektyw rozwoju:

Rozszerzenie Obszarów Działania

Nowe lokalizacje: Rozważanie ekspansji do nowych obszarów geograficznych, aby zwiększyć zasięg usług dostawczych.

Nowe branże: Poszerzanie usług dostaw na inne branże, takie jak apteki, sklepy spożywcze, restauracje, itp.

Udoskonalenia w Aplikacji

Interfejs użytkownika (UI) / Doświadczenie użytkownika (UX): Stałe doskonalenie interfejsu i doświadczenia użytkownika, co obejmuje prostsze i bardziej intuicyjne przeglądanie, płynne transakcje i łatwiejsze nawigowanie.

Personalizacja: Wprowadzenie funkcji personalizacji, tak aby aplikacja dostosowywała się lepiej do indywidualnych preferencji użytkowników.

Optymalizacja wydajności: Stałe usprawnianie wydajności aplikacji, zapewniając szybkość i responsywność.

Rozwój Technologiczny

Inteligentne algorytmy: Wykorzystanie sztucznej inteligencji i uczenia maszynowego do optymalizacji tras dostaw, przewidywania czasów dostawy, oraz personalizacji sugestii.

Rozwiązania IoT: Integracja z technologią Internetu Rzeczy (IoT) w celu monitorowania i śledzenia przesyłek w czasie rzeczywistym.

Płatności cyfrowe: Wprowadzenie nowoczesnych rozwiązań płatności, takich jak portfele cyfrowe, płatności zbliżeniowe itp.

Zróżnicowane Opcje Dostaw

Dostawy natychmiastowe: Ewentualne wprowadzenie opcji dostawy "natychmiastowej" z wykorzystaniem dostawców znajdujących się najbliżej miejsca zamówienia.

Dostawy grupowe: Wprowadzenie funkcji dostaw grupowych, gdzie różne zamówienia mogą być dostarczane jednocześnie do różnych lokalizacji.

Ekologia i Zrównoważony Rozwój

Zielona dostawa: Wprowadzenie zrównoważonych praktyk dostaw, takich jak dostawy na rowerach elektrycznych, samochodach elektrycznych, czy korzystanie z opakowań przyjaznych dla środowiska.

Programy recyklingu: Wprowadzenie programów recyklingu opakowań dostawczych.

Analiza Danych i Raportowanie

Narzędzia analizy danych: Implementacja rozbudowanych narzędzi analizy danych, umożliwiających lepsze zrozumienie zachowań klientów, trendów rynkowych i efektywności operacyjnej.

Raportowanie: Rozwinięcie funkcji raportowania dla przedsiębiorców i partnerów biznesowych, umożliwiające śledzenie kluczowych wskaźników wydajności (KPI).

Bezpieczeństwo i Prywatność

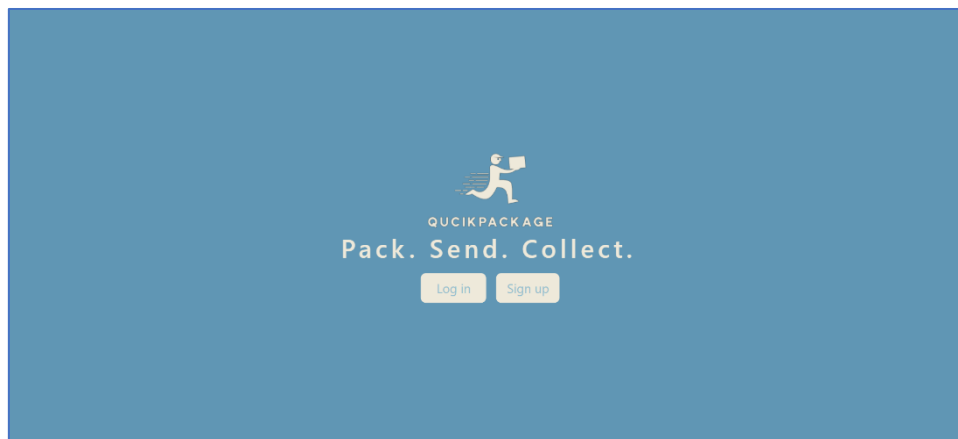
Rozwój funkcji bezpieczeństwa: Wprowadzenie nowoczesnych funkcji bezpieczeństwa, takich jak dwuskładnikowa autentykacja, monitorowanie działań użytkowników itp.

Zgodność z regulacjami: Regularne dostosowywanie aplikacji do aktualnych przepisów i regulacji dotyczących ochrony danych osobowych.

Rozwój aplikacji dostawczej na żądanie powinien być dynamiczny, elastyczny i odpowiedzieć na zmieniające się potrzeby rynku oraz oczekiwania użytkowników. Przygotowanie na ewentualne zmiany w technologii, trendach rynkowych i regulacjach może pomóc w utrzymaniu konkurencyjności na rynku.

Dokumentacja użytkowa

Widok klienta



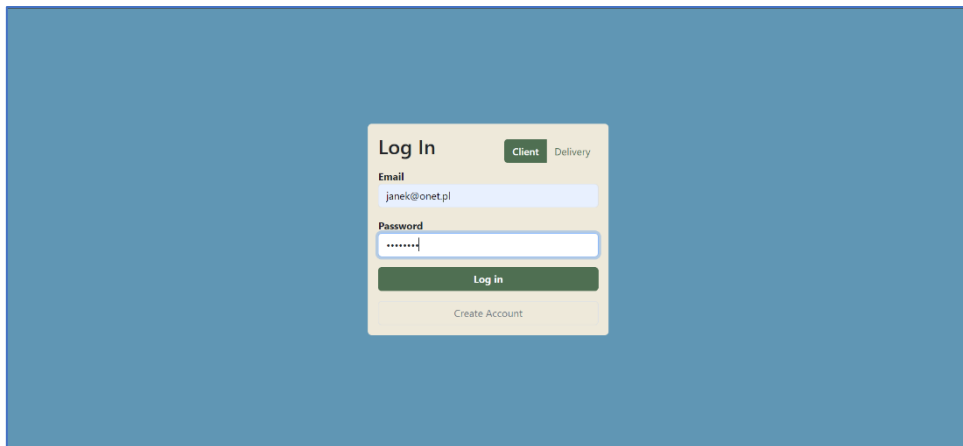
1. Strona startowa

- Klient po uruchomieniu aplikacji zostaje przekierowany na stronę startową, na której ma dwie możliwości: logowanie lub rejestracja.
- W celach demonstracyjnych założymy nowe konto klienta.

The image displays a 'Sign Up' form overlaid on the same dark blue background as the previous screenshot. The form is a light beige color and contains several input fields: 'Username' (with 'Janeczek12' entered), 'First Name' (with 'Jan' entered), 'Last Name' (with 'Kowalski' entered), 'Phone Number' (with '576 912 345' entered), and 'Email' (with 'janek@onet.pl' entered). There is a 'Password' field with masked characters (dots). Below these fields are two radio buttons: 'Client' (which is selected) and 'Delivery'. At the bottom of the form are two buttons: a green 'Sign up' button and a light blue 'Login' button.

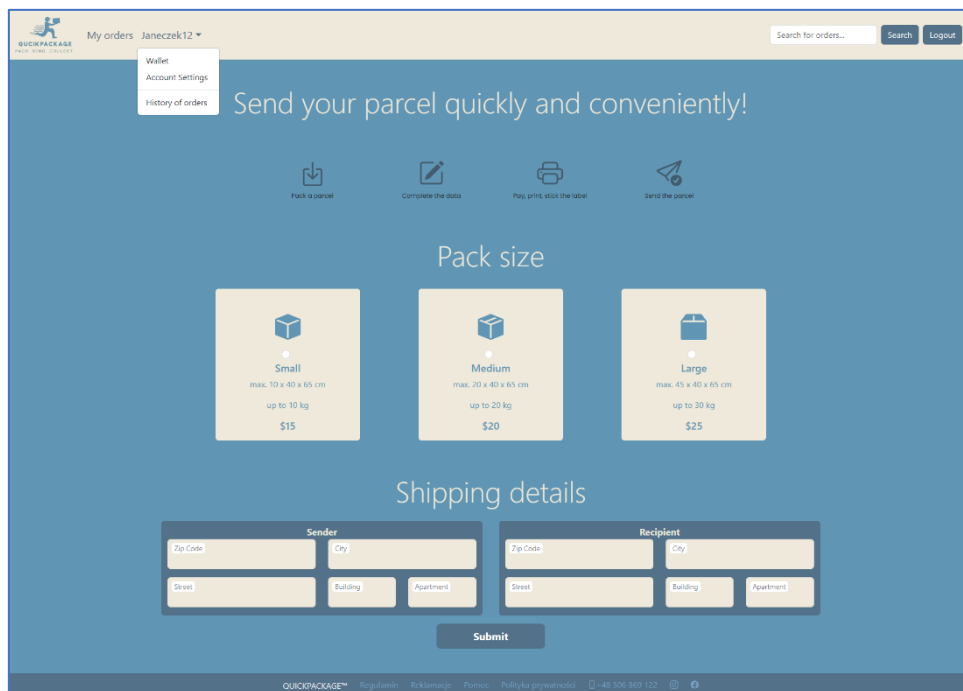
2. Rejestracja nowego użytkownika

- Zostaje wyświetlony formularz, w którym należy wprowadzić swoje dane oraz zaznaczyć typ tworzonego konta.
- Kluczowe pola posiadają walidacje zapobiegające wprowadzeniu źle sformatowanych danych:
 1. Hasło (8 znaków - co najmniej jedna duża litera i jedna cyfra)
 2. Mail (Ciąg znaków, bez spacji, wymagany znak @ oraz .)
 3. Telefon (Pole numeryczne - maksymalnie 9 znaków)



3. Logowanie użytkownika

- Po poprawnym wprowadzeniu danych, tworzymy konto użytkownika, które zostaje zapisane w bazie danych.



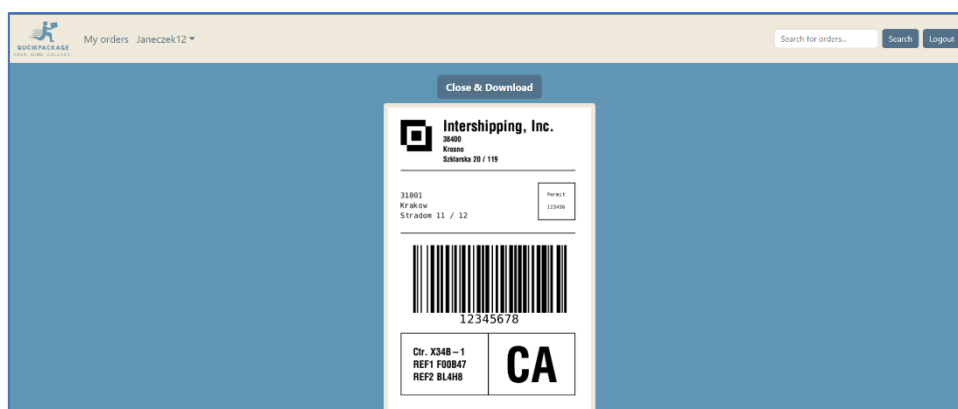
4. Strona główna

- Użytkownik zostaje przeniesiony na stronę główną, która umożliwia mu poruszanie się po aplikacji oraz stworzenie nowej przesyłki.
- W górnej części ekranu znajduje się pasek nawigacyjny za pomocą, którego klient może przejść do strony ze swoimi zamówieniami (My orders) oraz po naciśnięciu na przycisk rozwijany użytkownik może przejść do: swojego portfela (Wallet), ustawień profilu (Account Settings) i do historii swoich zamówień (History of orders).

5. Formularz nadawania paczek

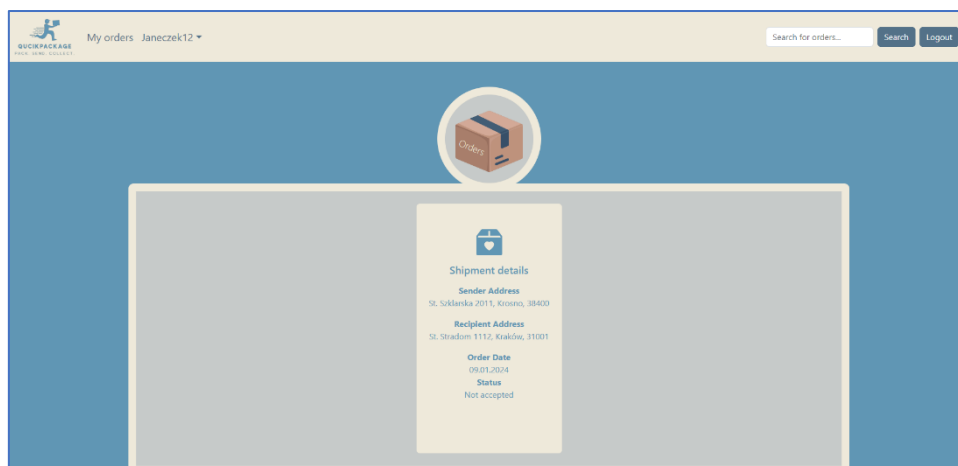
- W dolnej sekcji ekranu znajduje się formularz w którym użytkownik wybiera rozmiar paczki: mała, średnia, duża a następnie wprowadza dane nadawcy i odbiorcy: kod pocztowy, miasto, ulica, numer budynku, numer lokalu. Wszystkie te pola posiadają określone walidacje zapobiegające przed wprowadzeniem błędnych danych:

- Kod pocztowy (pole numeryczne, max 5 znaków, wymagane)
- Miasto, Ulica (pole tekstowe, może zawierać tylko litery, wymagane)
- Numer budynku, Numer lokalu (pole numeryczne, numer budynku wymagany, numer lokalu opcjonalny)



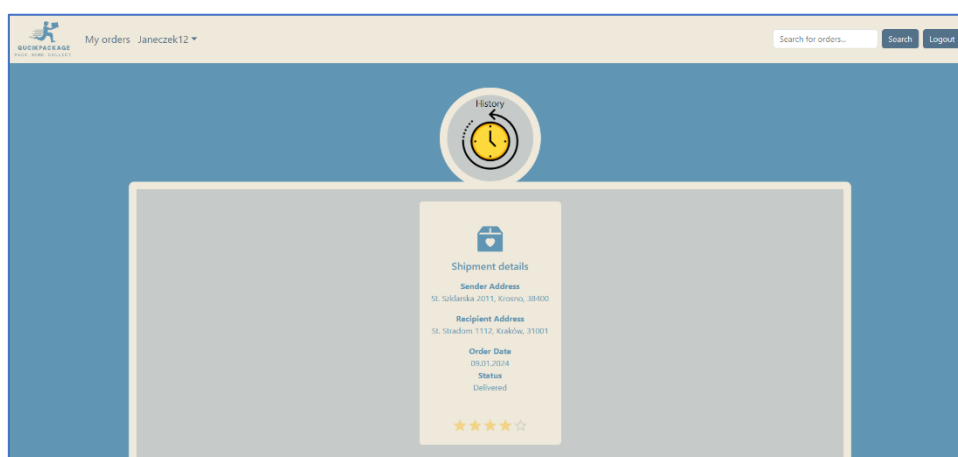
6. Wygenerowana etykieta ZPL

- Po naciśnięciu przycisku submit, dane zostają wysłane do bazy oraz zostaje wygenerowana etykieta ZPL z danymi wysyłki. Użytkownik może ją pobrać, wydrukować oraz nakleić na swoją wysyłkę.
- Każda paczka posiada swoją cenę, po wysłaniu zlecenia środki zostają pobrane z portfela umieszczonego na koncie klienta.



7. Strona z listą zamówień klienta

- Po zleceniu wysyłki zostaje ona zapisana na stronie moje zamówienia (My orders), gdzie użytkownikowi zostaje wyświetlona lista jego wysyłek. Każda z tych wysyłek posiada cechy status:
 - ❖ Not accepted – paczka nie została jeszcze zatwierdzona przez żadnego kuriera
 - ❖ Accepted – paczka została zaakceptowana przez kuriera
 - ❖ On way – paczka została odebrana przez kuriera i jest w drodze
 - ❖ Delivered – paczka została dostarczona. Wszystkie zlecenia z tym statusem zostają przeniesione do zakładki historia zamówień (History of orders)



8. Strona z historią zamówień

- W zakładce historia zamówień (History of orders) użytkownik może zobaczyć wszystkie zrealizowane zlecenia oraz wystawić opinię kurierowi – zaznacza gwiazdki w skali od 1 do 5

QUICKPACKAGE
PAID. WHEN COLLECT

My orders Janeczek12 ▾

Search for orders... Search Logout

Wallet

My balance : \$30

Top up your wallet Enter value

Submit

9. Portfel użytkownika

- Na tej stronie użytkownik może zobaczyć stan swojego portfela oraz doładować środki

QUICKPACKAGE
PAID. WHEN COLLECT

My orders Janeczek12 ▾

Search for orders... Search Logout

Account Settings

Account data

Username
Janeczek12

First Name
Jan

Last Name
Kowalski

Phone Number
576 912 345

Email
janek@onet.pl

Old Password
Enter Old Password

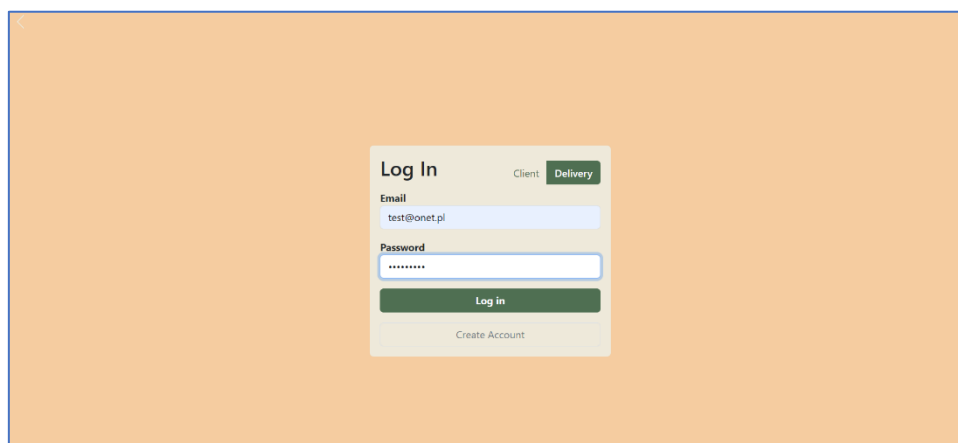
Password
Enter Password

Submit

10. Ustawienie konta użytkownika

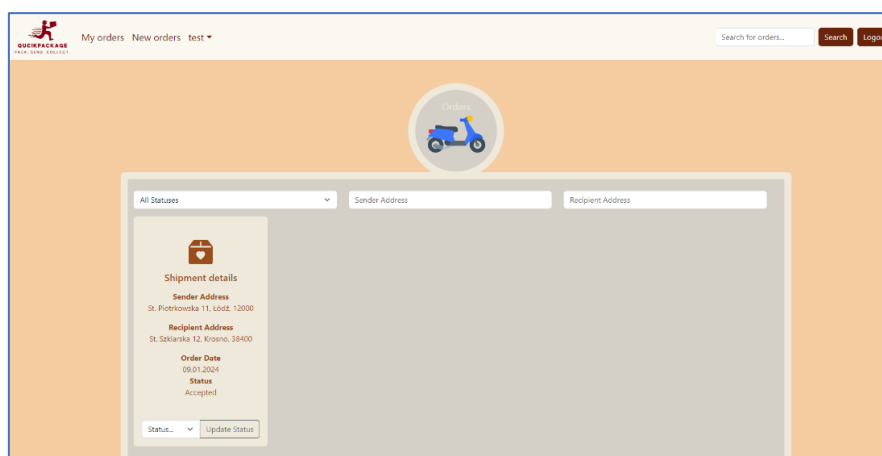
- W tej zakładce użytkownik może zmienić dane swojego konta. Po zmianie jakiegokolwiek pola klient zostaje automatycznie wylogowany w celu ponownego podania zaktualizowanych danych.

Widok kuriera



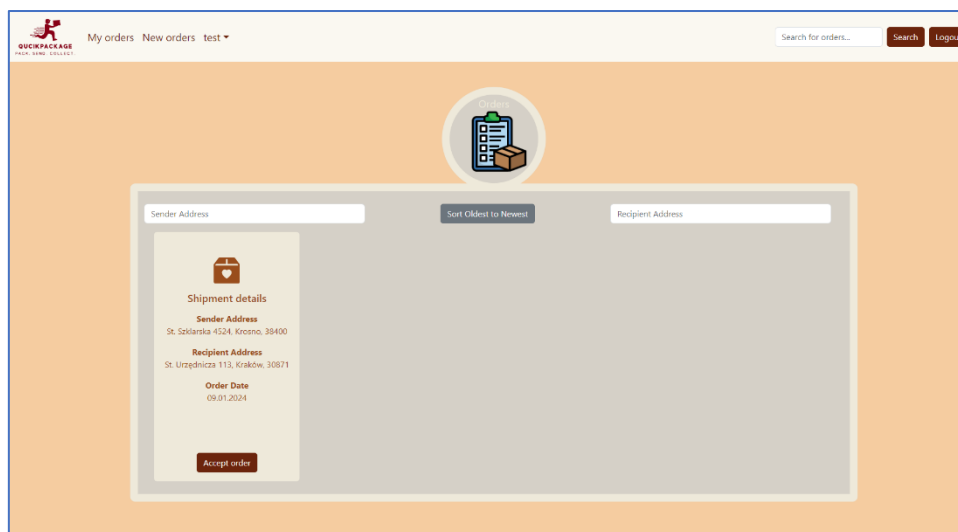
9. Widok logowania na konto kuriera

- Rejestracja konta kuriera to proces podobny do procesu rejestracji konta klienta z jedną różnicą - należy zaznaczyć typ konta: delivery (kurier)
- Po założeniu konta kurier loguje się do aplikacji za pomocą swoich danych.



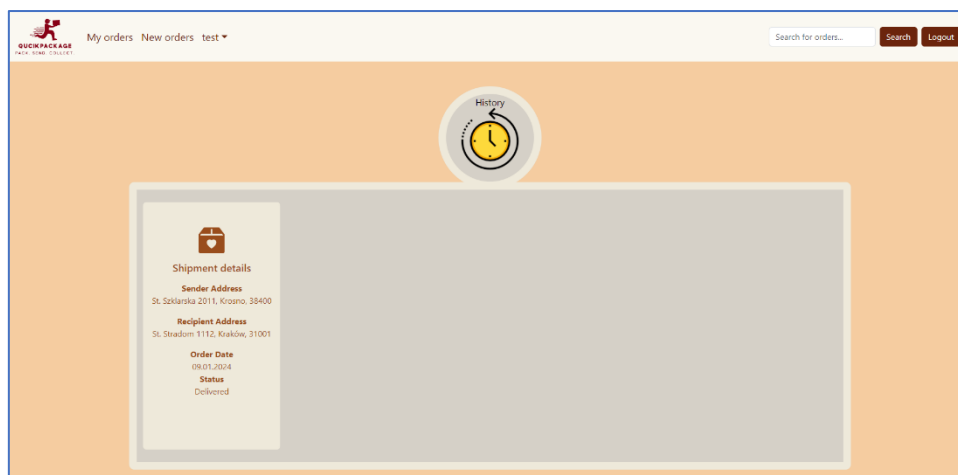
10. Strona główna kuriera

- Na stronie głównej zostaje wyświetlona lista zamówień przyjętych przez konkretnego kuriera.
- Kurier może wykonywać sortowanie swoich zamówień poprzez 3 atrybuty: status zamówienia, adres nadawcy lub adres odbiorcy.
- Kurier może również zmieniać status zamówienia w zależności od tego na jakim etapie usługi się znajduje. Wszystkie zmiany zostają wysłane do bazy i natychmiastowo zaktualizowane na kontach klientów.
- Jeśli status zamówienia zmieni się na dostarczony (delivered) to zlecenie zostanie automatycznie przeniesione do zakładki z historią oraz portfel zostanie doładowany odpowiednią kwotą.



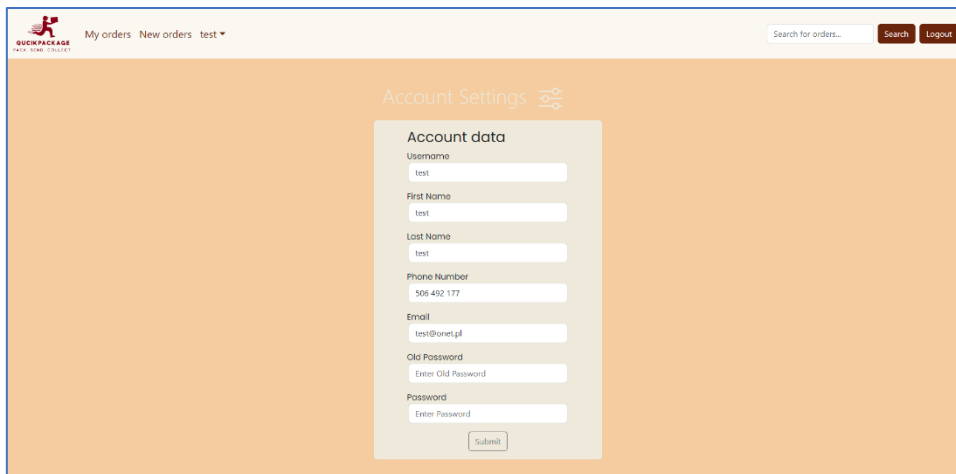
11. Strona z listą nowych zamówień dostępnych dla kuriera

- Na stronie z nowymi zamówieniami zostaje wyświetlona lista potencjalnych zleceń.
- Kurier ma możliwość filtrowania zleceń od najstarszych do najnowszych lub za pomocą dwóch atrybutów: adres nadawcy lub adres odbiorcy.
- Po naciśnięciu przycisku akceptuj zamówienie (Accept order), zamówienie zostaje przypisane konkretnemu kurierowi oraz przeniesione do zakładki moje zamówienia (My orders).



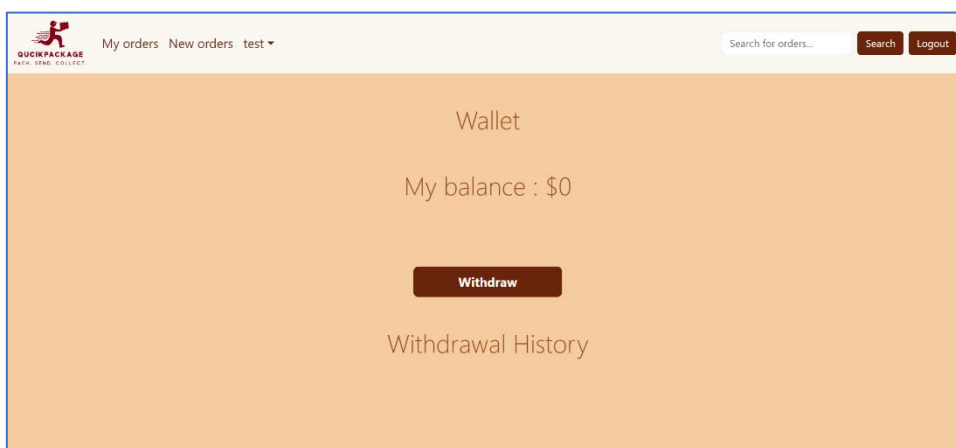
12. Strona z historią zamówień kuriera

- Na stronie zostaje wyświetlona historia zleceń zrealizowanych przez kuriera.



13. Strona z ustawieniami konta kuriera.

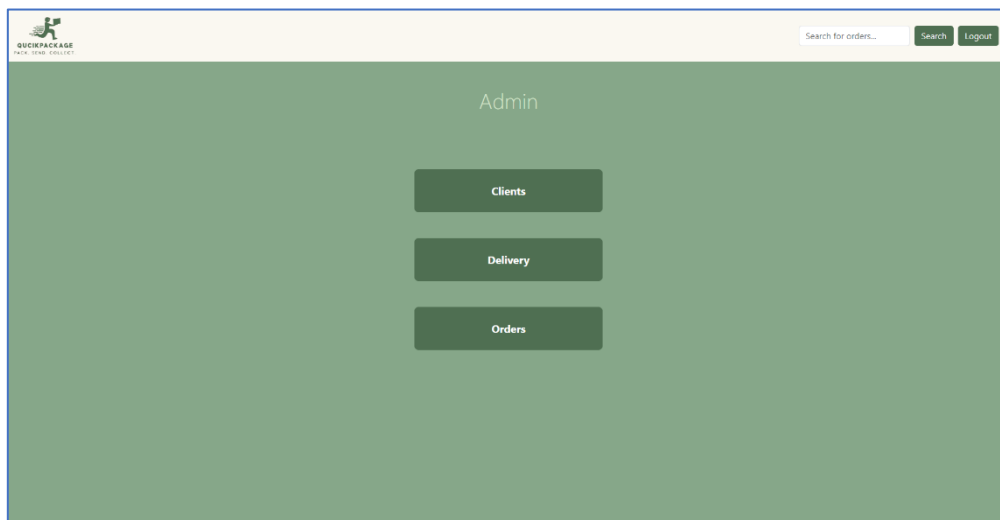
- W tej zakładce kurier może zmienić dane swojego konta. Po zmianie jakiegokolwiek pola kurier zostaje automatycznie wylogowany w celu ponownego podania zaktualizowanych danych.



14. Strona z portfelem przypisanym do konta kuriera.

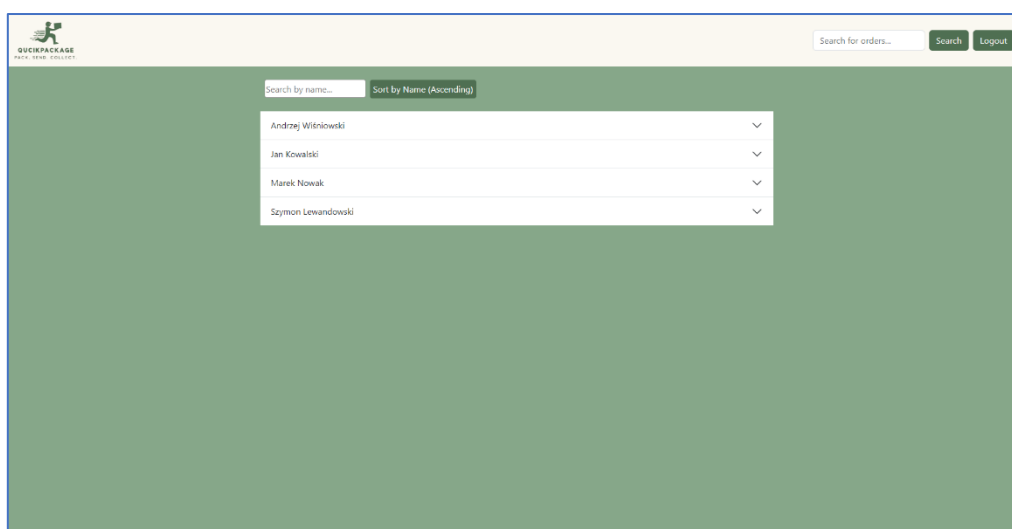
- Kurier ma możliwość zobaczyć stan swojego portfela oraz wykonać wypłatę na konto.
- W dolnej sekcji tej strony znajduje się historia przelewów, która jest aktualizowana wraz z każdym wyciągiem.

Widok administratora



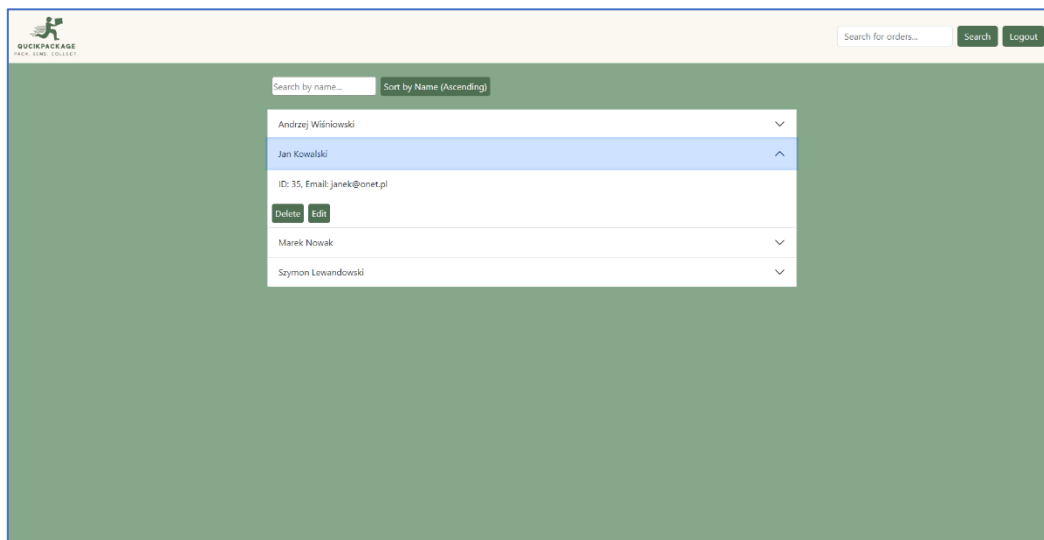
1. Strona startowa

- Administrator po wejściu w panel administratora aplikacji zostaje przekierowany na stronę główną, która umożliwia mu poruszanie się po aplikacji oraz wykonywanie odpowiednich czynności.
- W tej części znajdują się przyciski które administrator może wybrać aby przejść do zarządzania odpowiednio klientami, kurierami oraz zamówieniami



2. Zarządzanie klientami

- Na stronie zarządzania klientami znajduje się lista wszystkich klientów
- Administrator ma opcję wyszukania, sortowania klientów oraz wybrania konkretnego klienta do edycji



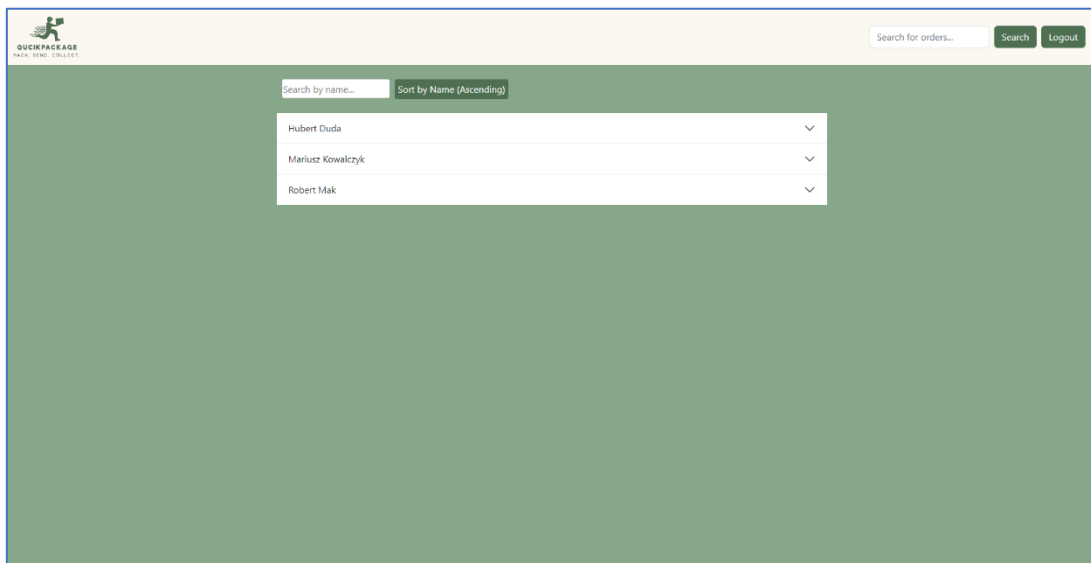
3. Edycja klienta

- Wyświetlane są podstawowe dane takie jak ID oraz Email
- Administrator ma możliwość usunięcia lub edycji klienta

The screenshot shows the 'Edit delivery Data' form. It has a title 'Edit delivery Data' and a section 'Current Data'. Below this, the current data is listed: 'First Name: Jan', 'Last Name: Kowalski', 'Phone Number: +48576912345', and 'Email: jarek@onet.pl'. There are input fields for each of these fields, with the current values pre-filled. At the bottom of the form, there is a 'Save Changes' button.

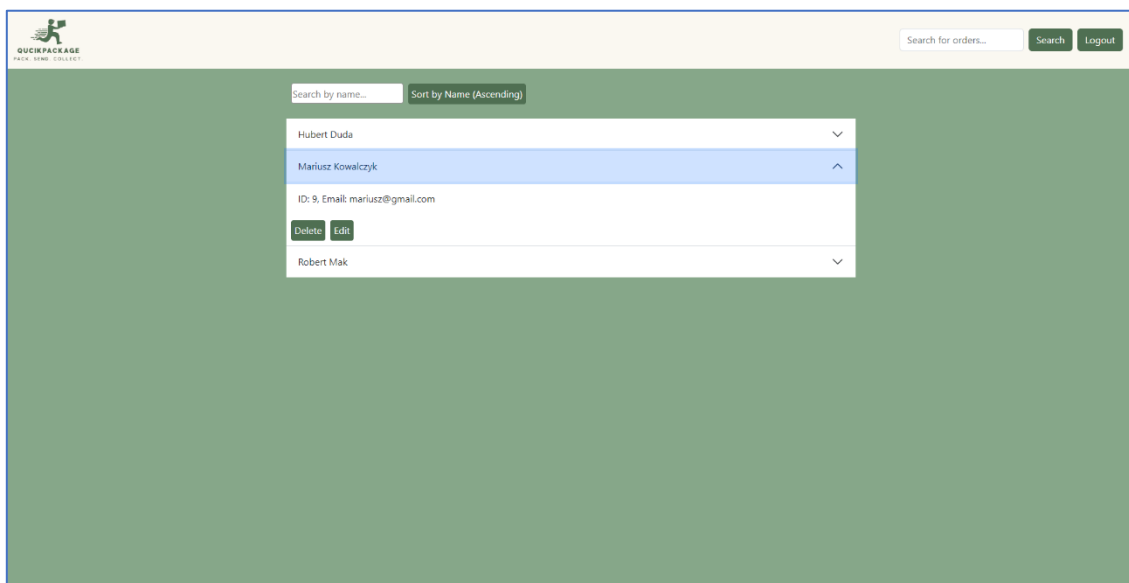
4. Edycja danych konta klienta

- Administrator widzi obecne dane, oraz jest w stanie je zmienić co akceptuje przyciskiem Save Changes



5. Zarządzanie kurierami

- Na stronie zarządzania kurierami znajduje się lista wszystkich kurierów
- Administrator ma opcję wyszukania, sortowania kurierów oraz wybrania konkretnego kuriera do edycji



6. Edycja zamówień

- Wyświetlane są podstawowe dane takie jak ID oraz Email
- Administrator ma możliwość usunięcia lub edycji kuriera

QUICKPACKAGE
PAKA. BIERA. COLLECT.

Search for orders... Search Logout

Edit delivery Data

Current Data

First Name Mariusz
Last Name Kowalczyk
Phone Number +48787567843
Email mariusz@gmail.com

First Name:

Last Name:

Phone Number:

Email:

7. Edycja zamówień

- Administrator widzi obecne dane, oraz jest w stanie je zmienić co akceptuje przyciskiem Save Changes

QUICKPACKAGE
PAKA. BIERA. COLLECT.

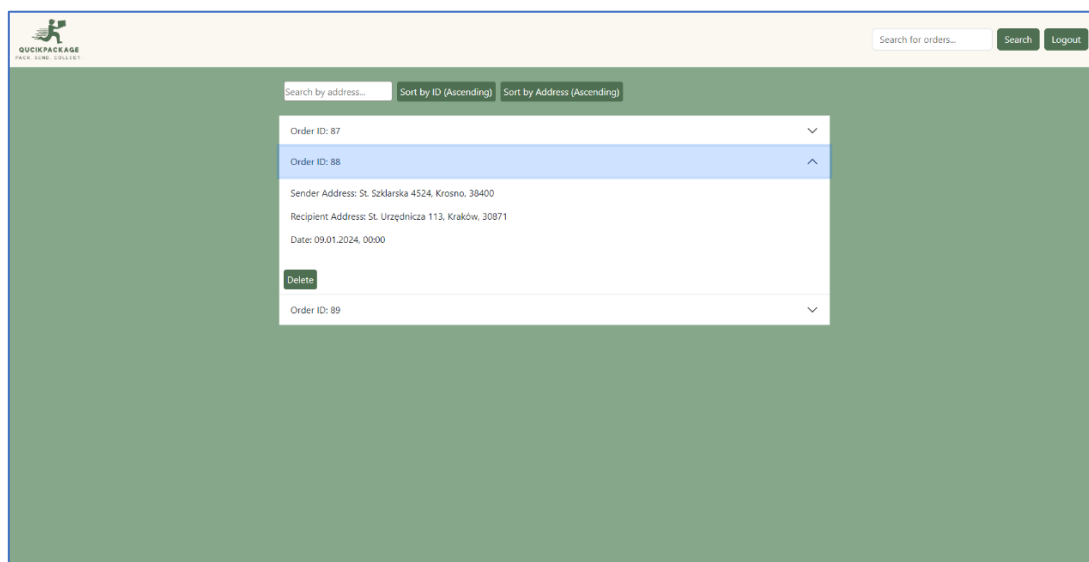
Search for orders... Search Logout

Search by address... Sort by ID (Ascending) Sort by Address (Ascending)

Order ID: 87	▼
Order ID: 88	▼
Order ID: 89	▼

8. Edycja zamówień

- Administrator ma podgląd na wszystkie zamówienia, jest w stanie je wyszukiwać oraz sortować



9. Usuwanie zamówienia

- Możliwy jest podgląd szczegółowych danych zamówienia oraz jego usunięcie