System Rezerwacji Wizyt u Fryzjera

Autorzy

Kamil Stecyk **stecykkamil@student.agh.edu.pl**
Bartłomiej Wajdzik
Filip Kubiś

Streszczenie systemu

Celem naszego systemu rezerwacji jest ułatwienie pracy fryzjerom, którzy pracują samodzielnie w swoim salonie. Każdy z nas regularnie odwiedza fryzjera i każdy z nas, aby odbyć taką wizytę musi najczęściej wykonać rozmowę telefoniczną. Dla nas jest to wygodne, jednak dla fryzjera niekoniecznie. Jeśli pracuje on sam i nie ma pomocnika to musi zwykle oderwać się od wykonywanej usługi albo oddzwonić później do klienta, jeśli o tym nie zapomni. Zarówno dla jednej, jak i drugiej strony może okazać się to czasami nieco frustrujące. Nasz system rezerwacji wizyt rozwiązuje taki problem. Cała procedura umówienia się do fryzjera może odbyć się bardzo sprawnie z poziomu strony internetowej, bez żadnej rozmowy telefonicznej, Wystarczy jedynie mieć zarejestrowane konto na tej stronie i można zacząć się cieszyć bardzo wygodnym sposobem na rezerwację wizyt w salonie fryzjerskim. Klient na ekranie ma możliwość wyboru dnia oraz godziny, jak i danego rodzaju usługi, którą chce wykonać. Bardzo wielkim atutem dla fryzjera jest to, że nie musi się w końcu odrywać od wykonywanej roboty, ani nie musi nic notować w związku z wizytami. Nasz system działa niczym prywatna sekretarka. Wszystkie umówione spotkania danego dnia zostają wyświetlanie na ekranie komputera w dedykowanym dla niego panelu admina. W kalendarzu wizyt panuje ład i spójność.

Lista wykorzystywanych technologii oraz podział roli

Front - end	HTML + CSS + JavaScript	- implementacja **Kamil Stecyk**
Back - end	Php	- implementacja **Bartłomiej Wajdzik**
Database	MySQL	- implementacja **Filip Kubiś**

Wybraliśmy właśnie te technologie, a nie inne głównie z powodu ograniczeń jakimi są umiejętności. HTML, CSS, JavaScript, Php i MySQL to technologie, z którymi ludzie z naszego zespołu mieli już do czynienia najwięcej, a więc przez to nie potrzeba dodatkowego

czasu na naukę nieznanych przez nas kompletnie języków, czy też frameworków oraz znacznie to ułatwia tworzenie systemu, jak i zmniejsza ilość potencjalnych błędów.

Opis Projektu

W ramach zajęć projektowych z przedmiotu Projekt Zespołowy Inżynierii Oprogramowania, chcielibyśmy zaprojektować system rezerwacji wizyt dla fryzjera. Nasz system zawierałby podstawowe funkcjonalności większości systemów rezerwacji takie jak:

- Założenie konta, oraz późniejsza możliwość logowania

- Wyszukiwanie wolnych slotów czasowych w wyznaczonym dniu dla danych usług

- Rezerwacja wizyty u fryzjera w wolnym terminie

- Wprowadzanie danych osobowych w celu kontaktu

- Dodawanie i usuwanie wizyt z poziomu fryzjera (admin)

- Panel użytkownika z możliwościa podglądu historii wizyt

- Panel fryzjera z możliwością podglądu stanu wizyt na dany dzień

Wymagania

1. **Wymagania Funkcjonalne**

ID wymagania	nazwa	opis
U001	Rejestracja użytkownika	Nowy użytkownik systemu powinien móc się zarejestrować w naszym serwisie i zostać dodanym do puli użytkowników
U002	Logowanie użytkownika	Użytkownik może zalogować się na swoje konto
U003	Logowanie Fryzjera	Fryzjer może się zalogować na swoje konto
U004	Pokazywanie wolnych terminów	Użytkownik powinien dostać listę wolnych terminów
U005	Wybór usługi	Użytkownik może wybrać dowolną usługę świadczoną przez salon fryzjerski
U006	Wybór dnia	Użytkownik może wybrać dogodny dla niego dzień
U007	Wybór godziny	Po wybraniu dnia użytkownik ma możliwość wybrania dogodnej godziny spośród podanych
U008	Rejestracja wizyty	Użytkownik po wybraniu dnia, godziny, oraz świadczonej usługi rejestruje swoją wizytę na stronie fryzjera
U009	Odczyt złożonych wizyt	Fryzjer może odczytać wizyty jakie ma zarejestrowane na dany dzień
U010	Sprawdzanie Historii	Użytkownik posiada możliwość sprawdzenia historii ostatnich wizyt
U011	Sprawdzenie instrukcji strony	Użytkownik ma możliwość zobaczenia jak strona działa i jak umówić na niej termin

2. **Wymaganie niefunkcjonalne**

- **Dostępność**

  Serwis dostępny 24/7/365
- **Wydajność**

  Czas odpowiedzi serwisu natychmiastowy, zdolny obsłużyć do kilku użytkowników jednocześnie, z dowolnej lokalizacji, przy użyciu minimalnej ilości zasobów dostępnego sprzętu
- **Wsparcie**

  Żaden system zgłaszania błędów nie jest dostępny na stronie, ewentualne błędy mogą być zgłaszane do współpracujący z nami fryzjerami. Czas naprawy błędów będzie zależeć od złożoności błędu oraz dostępnych zasobów ludzkich
- **Bezpieczeństwo**

  Serwis stosuje zabezpieczenie Captcha podczas logowania jak i rejestracji użytkowników i fryzjerów
- **Wdrożenie**

  Aplikacja wycelowana jest we wszystkich ludzi korzystających z usług fryzjerskich. Wdrożenie aplikacji polegać będzie do reklamowania go w sieci oraz salonach fryzjerskich zachęcając do przejścia na nasz model

Use-cases

Nasz system rezerwacji wizyt zakłada konieczność stworzenia konta w celu przeglądania , rezerwacji wizyty, wglądu do historii, oraz szybkiego i prostego anulowania wizyty. Gość nie będzie posiadał możliwości poruszania się po systemie. Wszystkie dostępne operacje w tej części systemu prezentowane są w poniższych tabelach.

Zakładamy również konieczność stworzenia konta admina przez zespół deweloperski w czasie wdrożenia programu do danego salonu fryzjerskiego - nie ma możliwości, aby admin utworzył swoje konto poprzez interfejs na stronie internetowej. Również godziny otwarcia danego salonu fryzjerskiego wraz z ofertą wykonywanych usług oraz długością ich trwania muszą zostać ustawione poprzez zespół wdrożeniowy.

Use 1	Rejestracja
Actor	Użytkownik/Fryzjer
Przebieg	Klient(lub fryzjer) przekierowywany jest do specjalnie przygotowanej strony do rejestracji w której podaje swoje imię, nazwisko, login, email, numer telefonu, hasło
Przebieg alternatywny	W przypadku podania błędnych danych (np. niespełnione wymagania co do hasła) użytkownik proszony jest o ponowne podanie danych

Use 2	Logowanie
Actor	Użytkownik/Fryzjer
Przebieg	Dany użytkownik(klient/fryzjer) podaje login oraz hasło. Jeśli podane dane są poprawne zostaje przekierowany na stronę główną
Przebieg alternatywny	W przypadku błędnych danych klient/fryzjer nie zostaje wpuszczony do serwisu i musi podać dane jeszcze raz

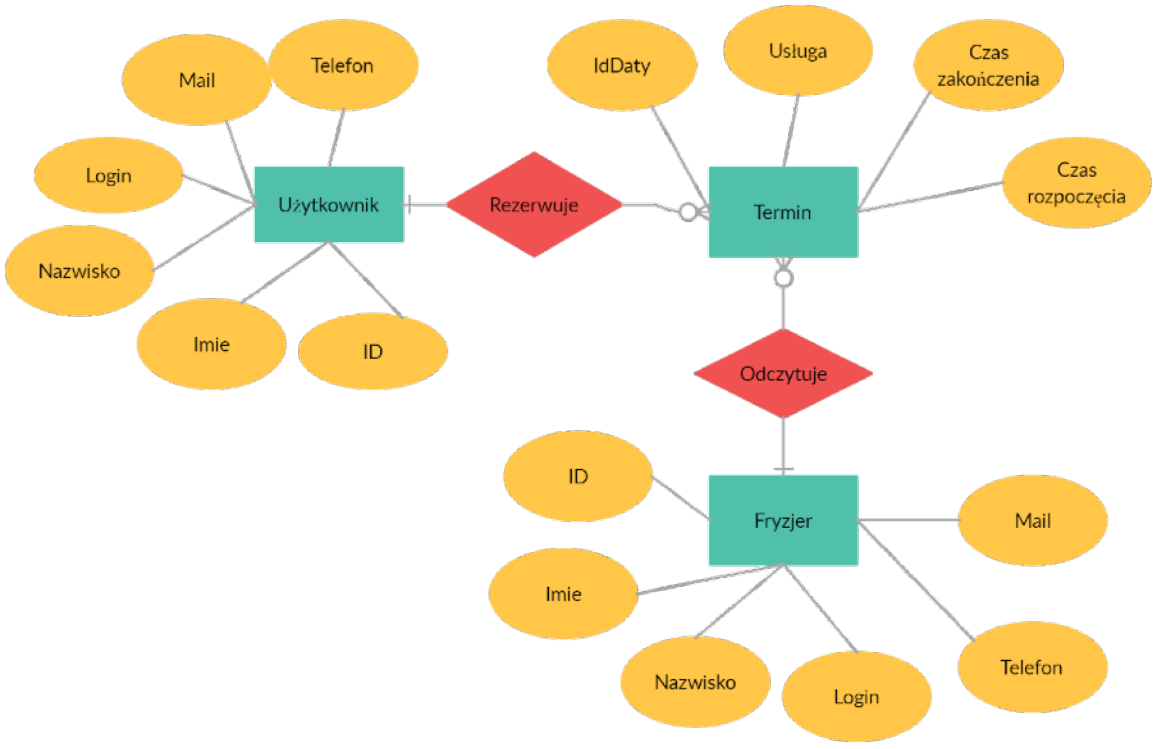
Use 3	Umawianie wizyty
Actor	Użytkownik
Przebieg	Użytkownik wybiera na kalendarzu dogodny dla niego dzień, po czym wybiera godzinę oraz usługę z listy dostępnych, a na samym końcu akceptuje rezerwacje terminu

Use 4	Przeglądanie wizyt
Actor	Fryzjer

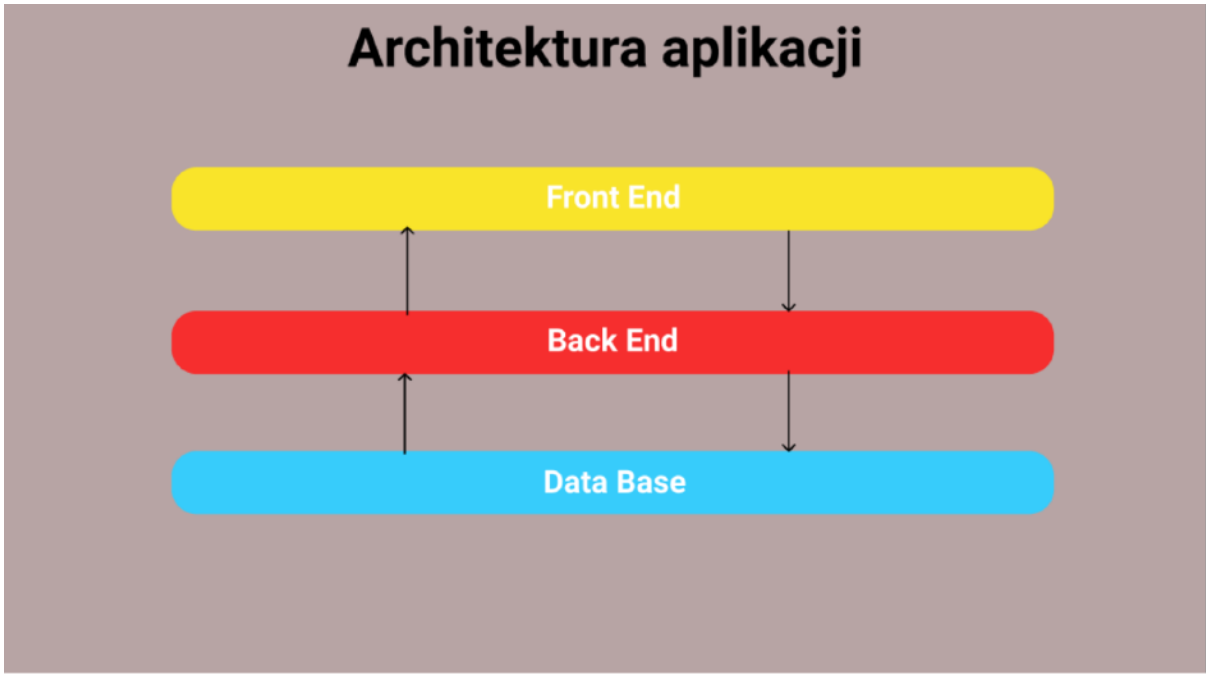
Przebieg	Fryzjer korzystając z serwisu dostaje listę terminów na dany dzień roboczy

Use 5	Przeglądanie historii
Actor	Użytkownik
Przebieg	Użytkownik chcący zobaczyć poprzednie odbyte wizyty ma możliwość przeglądnięcia swoich wizyt, na których są wyświetlone detale dotyczące odbytych spotkań

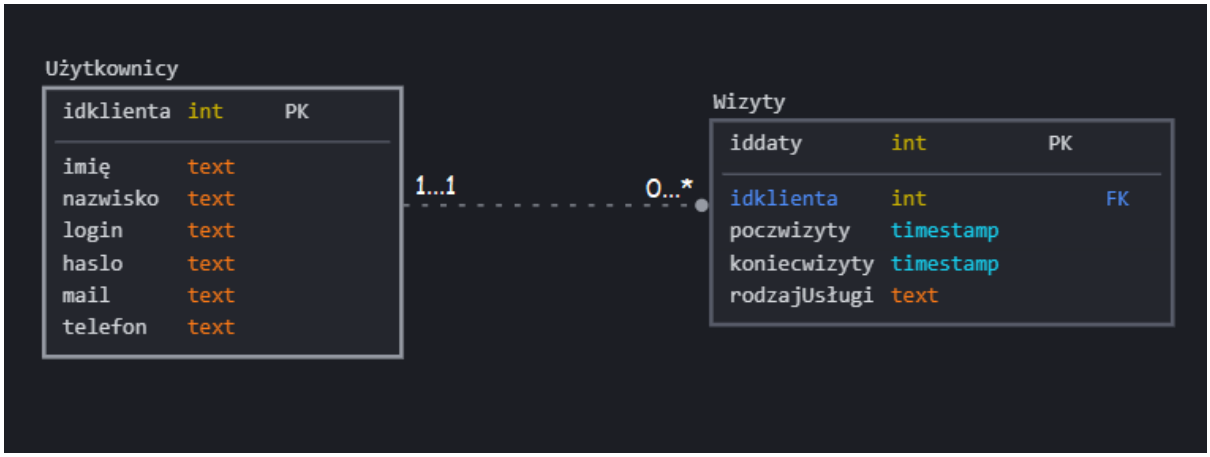
Diagram ER



Architektura systemu



Model architektury baz danych



Opis interfejsów

1. Rejestracja użytkownika w bazie danych systemu

Ta część systemu zapewnia tworzenie swojego konta. Będzie to możliwe dla każdego bez żadnych opłat. Sama rejestracja przebiega na specjalnie przygotowanej do tego stronie, wymagane będą dane wymienione poniżej. Dokładnie pokazuję to tabela 1.1

- Dane wejściowe: dane konkretnego klienta:

Imię : (String) length > 0

Nazwisko : (String) length > 0

Mail : (String) (REGEXP) ^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$. Login : (String) length > 0

Hasło : (String) length > 6 && length < 16, musi zawierać znak specjalny

Nr telefonu : (String) musi zawierać 9 cyfr następujących po sobie lub oddzielonych ” .-” Input (zarejestruj)

- Opis działania:

walidacja danych, a po jej pozytywnym przejściu połączeniu się systemu z bazą danych oraz dodanie nowego rekordu z danymi klienta do niej

- Dane wyjściowe:

otrzymanie potwierdzenia o pomyślnym zarejestrowaniu, a w przypadku błędnych danych zwrócenie informacji o błędnych danych

Nazwa przypadku użycia	Zakładanie konta
Aktorzy	Klient
Zdarzenie inicjujące	Naciśnięcie przez aktora przycisku “Rejestracja”
Warunek wstępny	Aktor jest klientem
Warunek sukcesu	Aktor jest klientem i nowy klient zostaje dodany do bazy danych
Scenariusz główny	1. Klient podaje wymagane dane do utworzenia konta 2. System sprawdza czy możliwe jest utworzenie konta (czy nie istnieje już użytkownik o takim samym mailu oraz czy dane są poprawne) 3. Wyświetlenie komunikatu o udanym założeniu konta 4. Przejście do logowania
Scenariusz poboczny 1	2.A Klient o takim samym email-u istnieje 2.A.1 Wyświetlenie komunikatu o istnieniu danego konta 2.A.2 Wprowadzenie ponownie danych przez klienta 2.A.3 Przejście do punktu 2
Scenariusz poboczny 2	2.B Dane wpisane przez użytkownika nie spełniają wymagań 2.B.1 Wyświetlenie komunikatu o potrzebie zmiany lub uzupełnienia danych 2.B.2 Gość wprowadza poprawione dane 2.B.3 przejście do punktu 2
Scenariusz wyjątku	Zdarzenie: Problemy z połączeniem z bazą danych 1. Wyświetlenie odpowiedniego komunikatu

tabela 1.1

W skrócie proces wymaga by użytkownik podał wszystkie swoje dane. Nie ma możliwości istnienia dwóch kont o takim samym email-u. Po podaniu danych konto zostanie utworzone. Cały graficznie przebieg zaprezentowany jest na diagramie 1.2

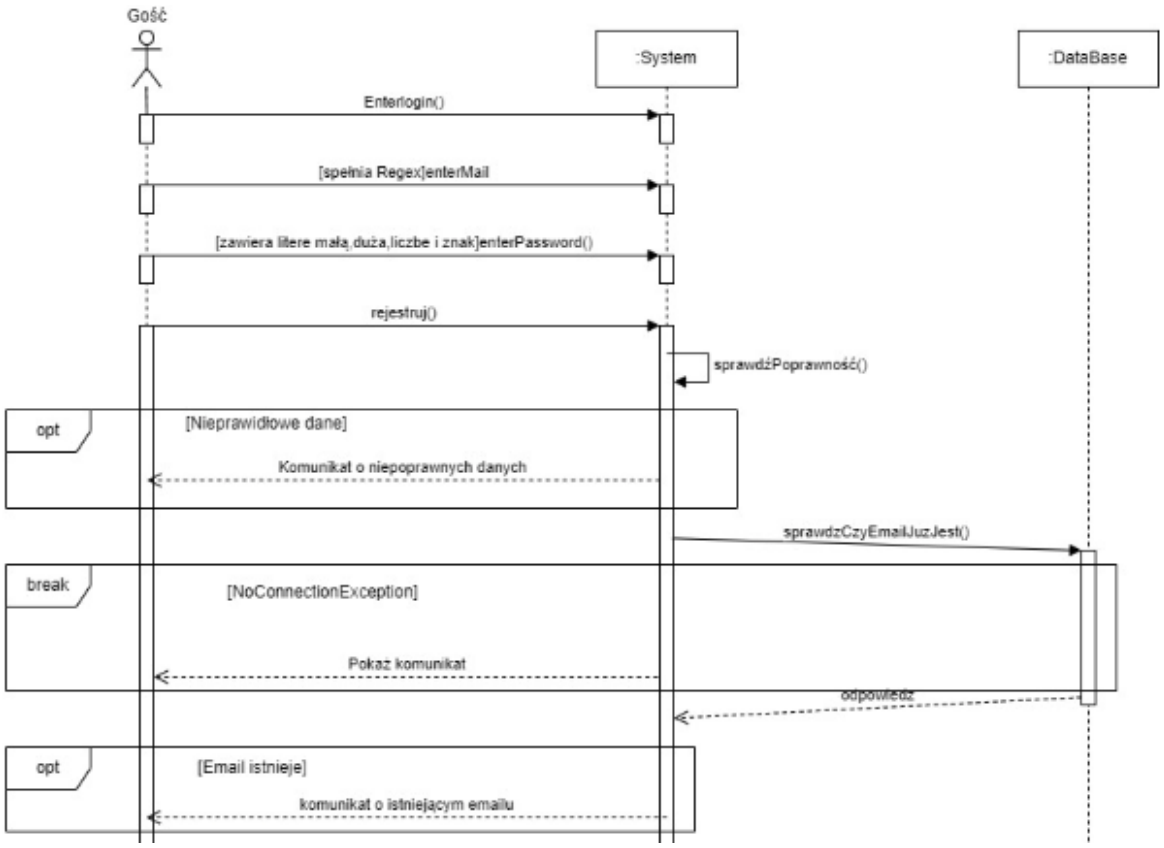


Diagram 1.2

Przewidziane są wyjątki w funkcjonowaniu przypadku użycia. Przewidziane przypadki to brak możliwości zapisania klienta w bazie danych. Zdarzenia takie niestety mogą wystąpić albo poprzez zawieszenie serwerów, albo poprzez zbyt duże obciążenie i z tymi sytuacjami się trzeba liczyć.

2. Logowanie użytkownika w systemie

Klient posiadający konto naturalnie powinien móc się zalogować. Nasz system umożliwia to poprzez zwykły login lub email oraz hasło. Wystarczy że dane zostają podane poprawnie a użytkownik dostaje dostęp do swojego konta. Dokładny opis zawiera tabela 2.1

- Dane wejściowe: dane konkretnego klienta:

Login : (String) Hasło : (String) Input (Zaloguj)

- Opis działania:

Wykonanie sanityzacji podanych przez użytkownika danych, a następnie wysłanie zapytania do bazy systemu o zwrócenie rekordu. Jeśli taki istnieje to przekierowanie użytkownika na właściwą stronę.

Jeśli wprowadzone przez klienta dane odpowiadają rekordowi w bazie danych admina to backend przekieruje na specjalną stronę dedykowanej dla niego

- Dane wyjściowe: w przypadku nie znalezienia danego rekordu w bazie danych, użytkownik otrzymuje informacje o tym.

Nazwa przypadku użycia	Logowanie
Aktorzy	Gość
Zdarzenie Inicjujące	Naciśnięcie przez aktora przycisku “zaloguj”
Warunek początkowy	Aktor jest gościem
Warunek sukcesu	Aktor jest klientem
Scenariusz główny	1. Gość wpisuje swoje dane logowania 2. System sprawdza poprawność danych 3. System przyznaje dostęp do konta
Scenariusz poboczny 1	3.A System nie przyznaje dostępu do konta 3.A.1 Wyświetlanie wiadomości o niepoprawności danych logowania 3.A.2 Użytkownik ponownie wprowadza dane 3.A.3 Przejście do punktu 2
Scenariusz wyjątku	Zdarzenie: System nie może połączyć się z bazą danych 1. Wyświetlenie komunikatu o niedostępności w danej chwili bazy danych

tabela 2.1

Graficzną prezentację przebiegu procesu logowania ilustruje diagram 2.2

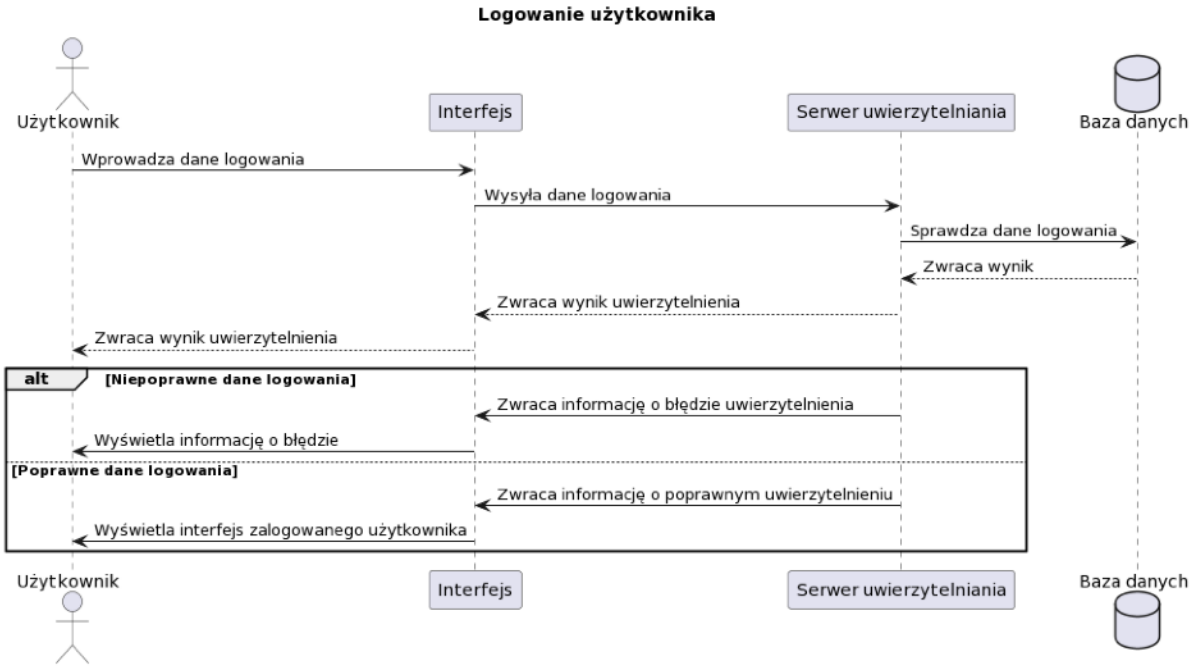


Diagram 2.2

3. Wybór terminu wizyty przez klienta

Ta część systemu odpowiada za wybór terminu wizyty przez zalogowanego klienta. Po wybraniu rodzaju usługi wykonywanej przez fryzjera i zaznaczeniu danego dnia miesiąca otrzymuje on możliwość umówienia się na wizytę w salonie fryzjerskim. Dokładnie pokazuję to tabela 3.1

- Dane wejściowe:

Input (dany dzień miesiąca) Input (rodzaj usługi)

Input (godziny wizyty)

- Opis działania:

Klienta wybiera konkretny dzień miesiąca, po dokonaniu wyboru wybiera usługę, jaka go interesuje. Na tej podstawie wysłane jest następnie zapytanie do bazy danych o zwrócenie wolnych godzin terminów w danym dniu, które użytkownik będzie mógł wybrać. Po dokonaniu wyboru wysyłane jest zapytanie do bazy danych o dodanie rekordu dotyczącego wizyty klienta.

- Dane wyjściowe:

przekierowanie przez backend na stronę z potwierdzeniem dla klienta pomyślnego zarezerwowania terminu wizyty

Nazwa przypadku użycia	Wybór terminu wizyty
Aktorzy	Klient
Zdarzenie Inicjujące	Naciśnięcie przez aktora przycisku “Wyszukaj terminów” po wcześniejszym wybraniu dnia oraz usługi
Warunek początkowy	Aktor jest klientem
Warunek sukcesu	Zostanie dodana do bazy wizyta klienta
Scenariusz główny	1. Klient wybiera dany dzień, który go interesuje, a następnie rodzaj usługi, po czym wciska przycisk “Wyszukaj terminów” 2. Klient wybiera jeden spośród wyświetlonych dostępnych terminów dla usługi 3. System sprawdza czy dany termin jest na pewno dalej dostępny 4. System dodaje rekord wizyty klienta do bazy danych 5. Zostaje wyświetlony alert z potwierdzeniem operacji 6. Zostaje wyświetlone potwierdzenie rezerwacji wizyty klientowi
Scenariusz poboczny 1	3.A Dana godzina usługi została nagle zarezerwowana 3.A.1 Wyświetlanie wiadomości o niedostępności danej godziny usługi 3.A.2 Zostaje odświeżona lista dostępnych terminów dla klienta 3.A.3 Przejście do punktu 2
Scenariusz poboczny 2	5.A Klient nie potwierdza operacji 5.A.1 Anulowanie operacji 5.A.2 Przejście do punktu 2
Scenariusz wyjątku	Zdarzenie: System nie może połączyć się z bazą danych 1. Wyświetlenie komunikatu o niedostępności w danej chwili bazy danych

Tabela 3.1

Graficzną prezentację przebiegu procesu wyboru terminu wizyty ilustruje diagram 3.2

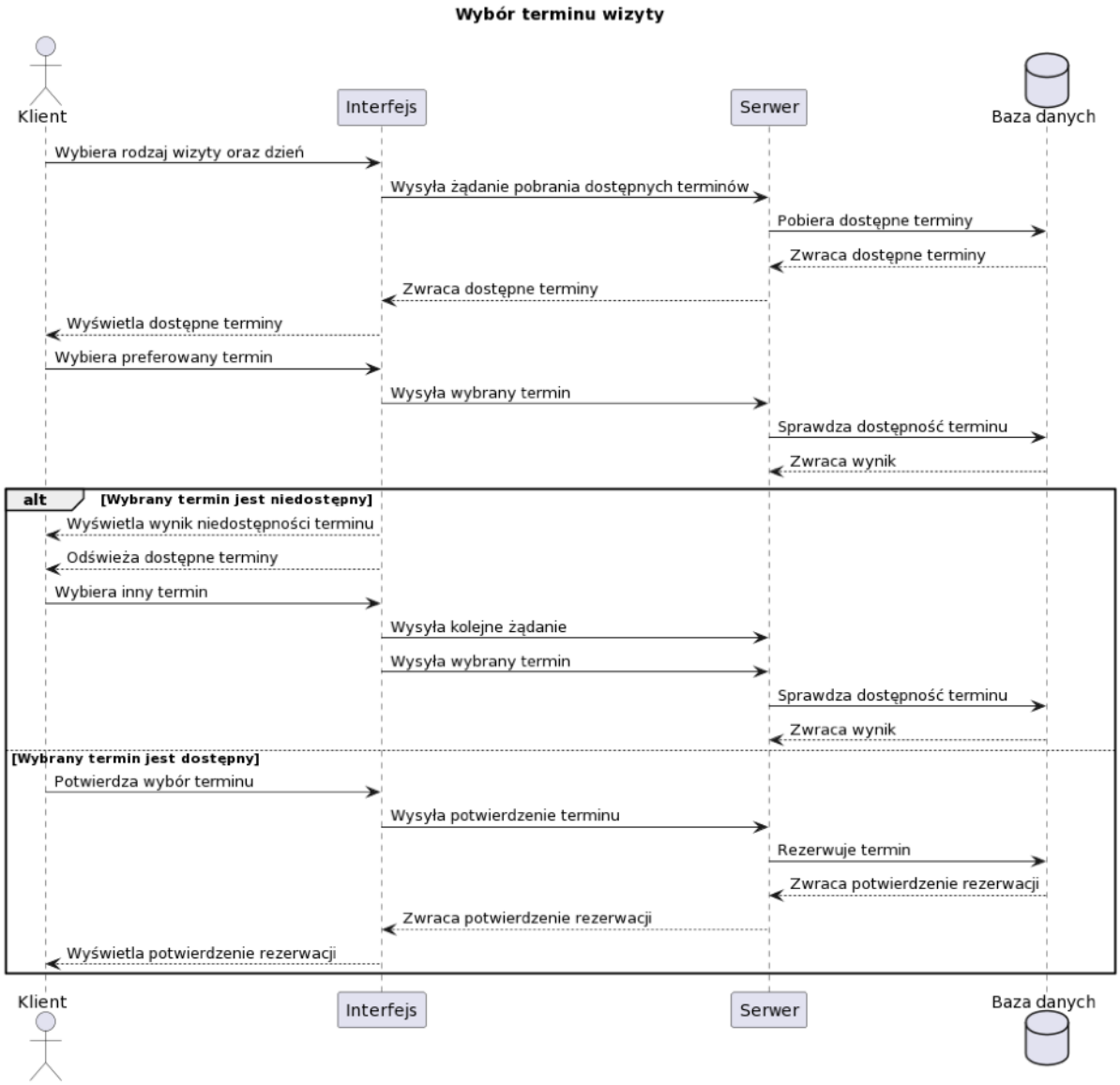


Diagram 3.2

4. Sprawdzenie historii wizyt przez klienta

Ta część systemu jest dość prosta. Klientowi zostaje zaprezentowana jego historia wizyt w salonie fryzjerskim. Błąd w tej części może wynikać głównie z powodu niepoprawnym połączeniem z bazą danych lub problemem z nią. Dokładny opis prezentuje tabela 4.1.

- Dane wejściowe:

Dane klienta (String)

- Opis działania:

Na podstawie danych klienta zostaje wysłane zapytanie do bazy danych systemu o zwrócenie jego wszystkich zarezerwowanych wizyt. Na tej podstawie zostaje mu wyświetlony wynik na stronie. Wizyty są posortowane według daty malejąco.

- Dane wyjściowe:

Informacje o wizytach danego klienta

Nazwa przypadku użycia	Logowanie
Aktorzy	Klient
Zdarzenie Inicjujące	Naciśnięcie w menu “Historia wizyt”
Warunek początkowy	Aktor jest Klientem
Scenariusz główny	1. System łączy się z bazą danych i prosi o otrzymanie jego historii wizyt 2. Zostaje wyświetlona klientowi jego historia wizyt
Scenariusz poboczny 1	2.A Klient nie posiada wizyt w bazie danych 3.A.1 Wyświetlanie wiadomości o braku wizyt w salonie fryzjerskim
Scenariusz wyjątku	Zdarzenie: System nie może połączyć się z bazą danych 1. Wyświetlenie komunikatu o niedostępności w danej chwili bazy danych

tabela 2.1

Graficzną prezentację przebiegu procesu wyboru terminu wizyty ilustruje diagram 4.2

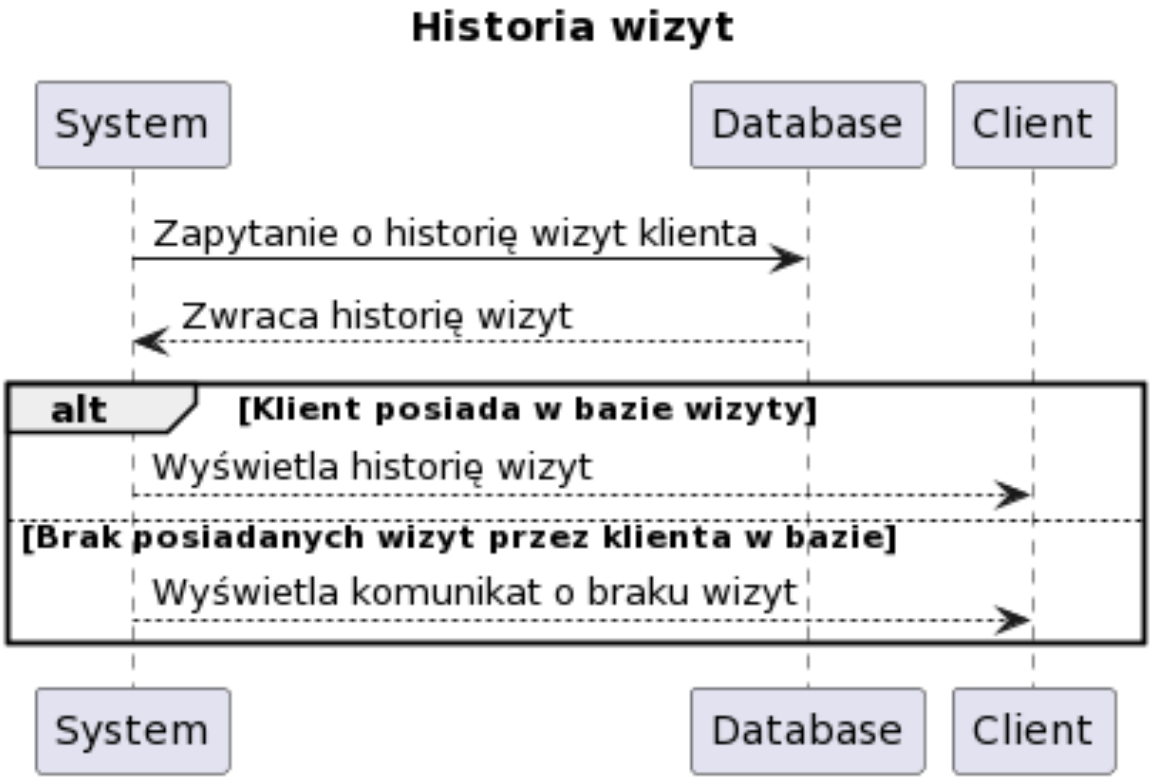


Diagram 4.2

5. Okno wizyt dla admina na dany dzień - Dane wejściowe:

W tej części systemu admin (fryzjer) otrzyma możliwość podglądu swojego grafiku opartego na zarezerwowanych już wizytach przez klientów. W domyślnym widoku zostaje zaprezentowany fryzjerowi dzień dzisiejszy, jednak ma on możliwość zmieniania dni na inne. Dokładny opis prezentuje tabela 5.1.

Data

- Opis działania:

Na podstawie obecnej daty zostają wygenerowane dla admina wszystkie zarezerwowane wizyty na dzień. Admin ma także możliwość zmiany daty na inny dzień.

- Dane wyjściowe:

Informacje o wizytach na dany dzień

Nazwa przypadku użycia	Logowanie
Aktorzy	Admin
Zdarzenie Inicjujące	Naciśnięcie przez aktora przycisku zmiany dnia
Warunek początkowy	Aktor jest adminem
Scenariusz główny	1. Admin naciska przycisk zmiany dnia na inny 2. System weryfikuje wizyty na wybrany przez admina dzień 3. Zostają wyświetlone zarezerwowane wizyty na dany dzień
Scenariusz poboczny 1	3.A Brak wizyt na dany dzień 3.A.1 Zostaje wyświetlona informacja o braku wizyt na dany dzień
Scenariusz wyjątku	Zdarzenie: System nie może połączyć się z bazą danych 1. Wyświetlenie komunikatu o niedostępności w danej chwili bazy danych

tabela 5.1

Graficzną prezentację przebiegu procesu wyboru terminu wizyty ilustruje diagram 5.2

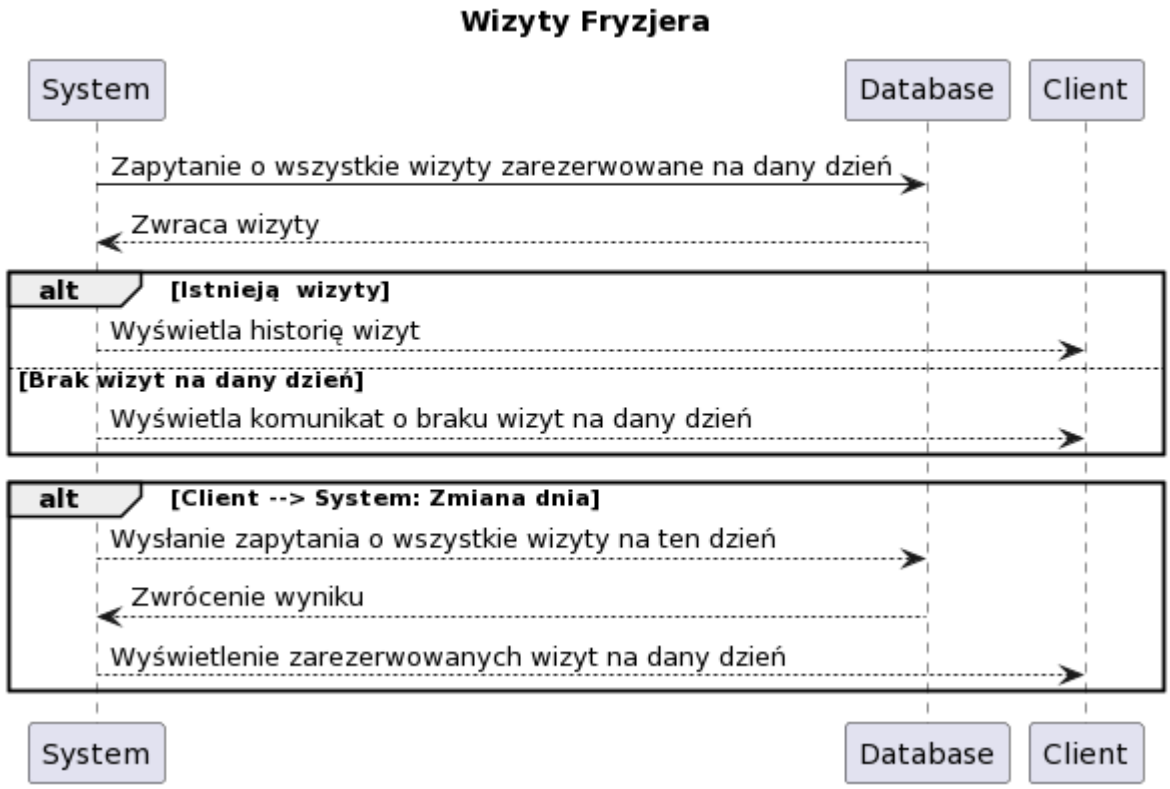


Diagram 5.2

Przebieg implementacji

Implementacja naszego systemu rezerwacji dla fryzjerów przebiegała bardzo sprawnie. Podział ról wymieniony na początku dokumentacji:

Front - end	implementacja **Kamil Stecyk**
Back - end	implementacja **Bartłomiej Wajdzik**
Database	implementacja **Filip Kubiś**

został zachowany, jednak czasami niektóre role się zazębiały oraz staraliśmy się sobie pomagać w razie napotykanych problemów. Cała aplikacja została napisana w taki sposób, aby implementacja poszczególnych części mogła być prowadzona w sposób niezależny od siebie. Tworzenie zapytań miało miejsce za pomocą skryptów kodów **PHP**, w których następowało połączenie z bazą danych z których następnie korzystał backend również napisany w tym języku. Na pierwszym **milestonie** posiadaliśmy aplikację z działającą większą częścią wymagań postawionych w dokumentacji. Główną funkcjonalnością, której wtedy nie mieliśmy była logika związana z rezerwacją wizyt po stronie klienta. Na szczęście, na drugim **milestonie** posiadaliśmy już całą gotową działająca aplikację webową wraz ze wspomnianą wcześniej logiką klienta. Była to zdecydowanie najcięższa dla nas do implementacji część tego systemu, tak jak wcześniej zakładaliśmy. Jednak wspólnymi siłami udało nam się osiągnąć sukces, nie korzystając z gotowych już algorytmów i innych implementacji. Cała aplikacja webowa została napisana zgodnie z tą dokumentacją i zostały w niej zaimplementowane wszystkie wyspecyfikowane wymagania funkcjonalne wraz z use casami.

Opis naszego algorytmu, który służy do rezerwacji wizyty przez klienta uwzględniający to, aby otrzymywane przez niego wolne terminy oczekiwanej wizyty, które zostaną wyświetlone użytkownikowi były optymalne i nie powodowały bezsensownych przerw z wizytami, które znajdują się już w bazie danych:

1) Najpierw kod sprawdza połączenie z bazą danych poprzez wywołanie **require_once 'database.php'** i przypisanie wyniku do zmiennej **$check_connection**. Jeśli połączenie nie powiedzie się, to wykonuje instrukcję **exit()**, która kończy wykonywanie skryptu.

2) Następnie kod tworzy tablicę **$timeOfServices**, która zawiera czasy trwania usług - są one zależne od danego salonu fryzjerskiego

3) Dalej następuje pobranie wartości zmiennej **$_COOKIE['indexs']** i przypisanie jej do zmiennej **$choosedIndex**. Jest to konieczne, aby program wiedział jaki rodzaj usługi został wybrany przez użytkownika w celu określenia długości czasu wizyty

4) Następnie kod dzieli czas trwania wybranej usługi na godziny i minuty poprzez wywołanie funkcji **explode()** i przypisanie wyniku do tablicy **$tableOfSplit**.

5) Kolejny krok to utworzenie zmiennej **$choosed_data**, która zawiera datę wybraną za pomocą ciasteczka **'year'**, **'month'** i **,day'** - jest to konieczne w celu ustalenia oczekiwanej daty wizyty przez klienta

6) Wykonywane jest zapytanie do bazy danych, które pobiera wszystkie rekordy z tabeli **'wizyty'**, gdzie data w kolumnie **'poczwizyty'** jest równa **$choosed_data** i jest większa od obecnej daty. Wynik zapytania jest przypisywany do zmiennej **$result**, a liczba wierszy jest przechowywana w zmiennej **$rows**. Od tej zmiennej zależy jakie instrukcje programu zostaną dalej wykonane. Jeśli **$rows == 0**, oznacza to, że w bazie danych dla danego dnia nie ma żadnej wizyty, więc generacja możliwych wizyt jest łatwiejsza - nie mamy wizyt pomiędzy. W przypadku gdy **$rows > 0**, algorytm musi wygenerować wolne wizyty w ten sposób , aby nie nastąpiła kolizja w bazie danych oraz niepotrzebne przerwy miedzy wizytami, które będą nieoptymalne.

Przypadek **$rows == 0:**

6.1.1) Algorytm w zależności od danej usługi, która została wybrana przez klienta i czasu jej trwania, która jest zależna od salonu, iteruje po godzinach, zaczynając od godziny otwarcia danego punktu fryzjerskiego do jego zamknięcia. Generuje potencjalne do rezerwacji czasy wizyt w danym dniu np. „10 - 10:30” - 0.5h to interwał, który jest czasem trwania danej usługi. Co ważne musi uwzględniać to, aby czas ostatniej wygenerowanej wizyty nie przekraczał godzin otwarcia danego salonu. Na przykład zakładając, że strzyżenie męskie trwa 0.5h, a czas otwarcia salonu fryzjerskiego to 9:00 - 18:00 to ostatnia możliwa wizyta w salonie z tą usługą to 17:30.

6.1.2) Wygenerowane terminy poprzez algorytm są wyświetlane na stronie za pomocą elementu **<input type=„submit">**.

Przypadek **$rows > 0:**

6.2.1) Jeśli **$rows** jest większe od 0, oznacza to, że istnieją już zarezerwowane terminy spotkań w bazie danych dla wybranej daty. W takim przypadku funkcja pomocnicza **availableHours()** zostanie wywołana, aby wygenerować dostępne terminy, które nie kolidują z już istniejącymi rezerwacjami. Będzie ona uwzględniać godziny otwarcia i zamknięcia salonu oraz godziny zajęte z bazy danych, aby wygenerować dostępne terminy spotkań. Będzie to miało na celu uniknięcie kolizji terminów i zapewnienie, że generowane terminy będą wolne. Opis jej działania znajduję się poniżej.

6.2.2) Wygenerowane dostępne terminy spotkań zostaną wyświetlone na stronie za pomocą elementu **<input type="submit">**, tak jak w przypadku, gdy **$rows** jest równe 0. Użytkownik będzie mógł wybrać jeden z tych terminów do zarezerwowania spotkania.

Wyjaśnienie krok po kroku, co robi funkcja **availableHours()** zastosowana w przypadku, gdy **$rows > 0**:

1. Na początku funkcja otrzymuje następujące argumenty: **openingHours** (godziny rozpoczęcia wizyt, które są już w bazie danych na dany dzień), **closingHours** (godziny zakończenia wizyt, które są już w bazie danych na dany dzień), **startOfBarber** (godzina rozpoczęcia pracy fryzjera), **endOfBarber** (godzina zakończenia pracy fryzjera) oraz **duration** (czas trwania usługi wybranej przez klienta usługi).
2. Funkcja rozpoczyna od przekształcenia czasu trwania usługi i godzin otwarcia/zamknięcia na tablice z wartościami godzin i minut za pomocą funkcji pomocniczej **ParseDateToInts()**.

3. Tworzone są zmienne pomocnicze, takie jak **currentTimestamp** (bieżący znacznik czasu) i **possibleMeetings** (tablica możliwych spotkań).
4. Następnie godziny otwarcia i zamknięcia salonu są przekształcane na tablice z wartościami godzin i minut za pomocą funkcji pomocniczej **ParseDateToInts()**.
5. Inicjalizowany jest wskaźnik (**pointer**) na 0. Jest on tworzony w celu orientacji ile zarezerwowanych wizyt zostało nam jeszcze do sprawdzenia w czasie całej iteracji.
6. Rozpoczyna się pętla, która będzie trwać, dopóki są dostępne terminy spotkań:
   a. Obliczane są godzina zakończenia aktualnego spotkania na podstawie bieżącego znacznika czasu i czasu trwania usługi. Jeśli minuty przekraczają 60, następuje inkrementacja godziny i odpowiednie dostosowanie minut.
   b. Sprawdzane jest, czy aktualny wskaźnik (**pointer**) jest mniejszy od liczby elementów w tablicy **openingHoursTimeStamps**, a także czy godzina zakończenia aktualnego spotkania jest większa niż godzina otwarcia wskazana przez ten wskaźnik. Jeśli tak, oznacza to, że istnieje istniejące spotkanie w tym czasie, więc aktualizowane są godzina i minuta na godzinę i minutę danej wizyty, która znajduje się już w bazie danych wskazanej przez ten wskaźnik, a następnie wskaźnik jest inkrementowany.
   c. Jeśli godzina zakończenia aktualnego spotkania przekracza godzinę zamknięcia salonu wskazaną przez **endOfBarber**, to oznacza, że przekroczono godzinę zamknięcia i pętla jest przerywana.
   d. W przeciwnym razie dodawane jest nowe możliwe spotkanie do tablicy **possibleMeetings**. Aktualizowane są także minuty bieżącego znacznika czasu poprzez dodanie 30 minut, a jeśli przekraczają 60 minut, następuje inkrementacja godziny i odpowiednie dostosowanie minut.
7. Po zakończeniu pętli funkcja zwraca tablicę **possibleMeetings** z dostępnymi godzinami wizyt dla danej usługi wybranej przez użytkownika.

Projekt testów gotowej aplikacji

Nasze testy opierały się na testach manualnych. Uzasadnienie testowania naszej aplikacji webowej za pomocą testów manualnych wynika z kilku czynników. Oto kilka głównych powodów:

- Elastyczność: Testowanie manualne daje nam testerom elastyczność w eksploracji aplikacji, interakcji z nią i dostosowaniu testów do różnych scenariuszy. Możemy dostosowywać swoje działania do zmieniających się wymagań lub oczekiwań użytkowników, co może być trudne do osiągnięcia w przypadku testów automatycznych.

- Interfejs użytkownika: Testowanie manualne umożliwia ocenę interfejsu użytkownika pod kątem użyteczności, estetyki i łatwości nawigacji. Możemy sprawdzić, czy interfejs jest intuicyjny i czy spełnia oczekiwania użytkowników.

- Scenariusze nieprzewidziane: Testowanie manualne pozwala na eksplorację aplikacji w celu znalezienia scenariuszy nieprzewidzianych lub niestandardowych. Testerzy mogą wykrywać potencjalne luki w funkcjonalności lub błędy, które mogą pojawić się w rzeczywistych warunkach użytkowania.

- Złożoność interakcji: Aplikacje webowe często mają złożone interakcje i scenariusze użytkowania. Testowanie manualne pozwala testerom na bezpośrednie eksplorowanie różnych funkcji, zachowań i przypadków użycia, co umożliwia wykrycie subtelnych błędów, które mogą być trudne do wykrycia automatycznie.

- Wdrożenie i czas: W naszym przypadku dużo łatwiej wprowadzić testy manualne, ze względu, że nie wiążą się one z pisaniem kodu - w tym przypadku każdy z nas musi znać dany język oraz bibliotekę lub środowisko. Przez to, że po przygotowanych scenariuszach testów można praktycznie od razu przystąpić do testowania aplikacji możemy zaoszczędzić czas, jak i również każdy z nas ma możliwość, aby wykonać dane testy w łatwy sposób i zanotować swoje wyniki.

Po ponownym zapoznaniu się przez nas z niniejszą dokumentacją projektu, przygotowaliśmy zestaw scenariuszy testów, które obejmują funkcjonalności naszej aplikacji. Każdemu z nas został przydzielony dany scenariusz, który ręcznie miał przetestować na podstawie naszej działającej aplikacji. Wykonane przez nas testy to **funkcjonalne**, **eksploracyjne** oraz **interfejsu użytkownika** na ekranach desktopowych, na których głownie skupiliśmy się pisząc tą aplikację webową. Zależało nam, aby kluczowe funkcjonalności systemu były wolne od jakichkolwiek błędów, które mogłyby wpłynąć na błędne działanie oraz samo użytkowanie było dla klienta dość intuicyjne i proste. Testy podzieliliśmy na dane moduły, które odpowiadają danym **use casom**. Wszystkie informacje, opisy scenariuszy oraz wyniki zebraliśmy w arkusze kalkulacyjne wykonane w **excelu**, aby usprawnić sam proces oraz zapewnić należytą czytelność.

Testowanie aplikacji

W folderze „testy” znajduje się 5 arkuszy excelowych odpowiadających konkretnym modułom i ich testom naszej aplikacji. Każdy moduł jest dokładnie opisany - kto go stworzył, kto go oceniał oraz kiedy to miało miejsce. Z kolei dane testy zawierają informację o scenariuszu jaki wykonują, wstępnych warunkach potrzebnych do wykonania testu, kolejnych krokach, które należy przeprowadzić oraz oczekiwanym i otrzymanym rezultacie.

Rownież istnieje wzmianka o tym kto przeprowadzał dany test oraz kolorem zaznaczone jest czy przeszliśmy go pozytywnie czy nie. Modułami, w których znajdują się te testy są:

- Rejestracja (10 testów - wszystkie zdane)
- Logowanie (7 testów - wszystkie zdane)
- Historia wizyt klienta (2 testy - 1 niezdany)
- Rezerwacja wizyt (14 testów - 3 niezdane)
- Panel admina (8 testów - wszystkie zdane)

Łączna liczba testów we wszystkich wymienionych modułach to 41 testów. Łączna liczba testów, których nie przeszliśmy pozytywnie to 4 co stanowi niespełna 9,76%, przy czym jeden błąd to zwykła literówka. Opis tych testów:

- Moduł: Historia wizyt klienta, Błąd: Literówka w przypadku komunikatu o braku zarezerwowanych wizyt
- Moduł: Rezerwacja wizyt, Błąd: System pozwala na wybranie sobót i niedzieli
- Moduł: Rezerwacja wizyt, Przy braku zaznaczenia w combo boxie wartości system znacząco spowalnia, po czym następuje zacięcie aplikacji webowej i pozostaje opcja jej zamknięcia
- Moduł: Rezerwacja wizyt, Po zarezerwowaniu wizyty przez klienta w przypadku ponownej chęci wybrania wolnego terminu, terminy nie są wyczyszczone na starcie, lecz zapamiętane z poprzedniej rezerwacji i zaaktualizowane o ten jeden, którego nie ma, bo już zarezerwowaliśmy.

Dla wszystkich błędów znaleźliśmy potencjalne rozwiązanie i zdiagnozowaliśmy problem przez, który błąd może występować. Znajduje się ono w komentarzu przy danym teście w przedostatniej kolumnie. Ogólnie tylko jeden z tych błędów można by uznać za faktycznie niepokojący i należałoby go jak najszybciej naprawić, ponieważ może realnie utrudnić korzystanie z aplikacji i powodować niezadowolenie klienta końcowego.

Konfiguracja aplikacji

W wersji produkcyjnej naszą stronę możemy hostować na danym serwerze do którego posiadamy dostęp. Zakładamy, że nasz zespół wdrożeniowy wykonuje to po zakupie programu przez klienta. Możemy również uruchomić naszą aplikację webową na localhost - tak testowaliśmy naszą aplikację z jej funkcjonalnościami. Aby to zrobić należy wykonać następujące kroki:

- Do postawienia serwera Apache potrzebny będzie nam program Xampp który można pobrać ze strony https://www.apachefriends.org/pl/index.html . Umożliwi on nam hostowanie naszej testowej aplikacji, jak i również korzystanie z usług bazodanowych MySql.

- Po zakończonej instalacji folder z naszym kodem źródłowym aplikacji wrzucamy do folderu Xampp/htdocs. Następnie uruchamiamy uruchamiamy usługi Xammp. (Apache serwer oraz MySql)

- Dodatkowo potrzebna nam będzie baza danych stworzona na http://localhost/phpmyadmin/. W zakładce bazy danych na tej stronie tworzymy nową bazę. Następnie klikamy w import i importujemy do bazy plik „wizyty.sql”. Powtarzamy dodawanie dla „uzytkownicy.sql” (pliki te zawierają tabele, które potrzebne są do poprawnego funkcjonowania aplikacji oraz przykładowe dane)

- Plik „config.php” w kodzie źródłowym musi zawierać poprawną nazwę użytkownika (domyślnie „root”) oraz hasło (jeśli jest ustawione) oraz nazwę danej bazy do której zaimportowaliśmy dane. Host w naszym przypadku musi mieć wartość „localhost”

- Po wykonaniu z sukcesem powyższych operacji, jedyne co nam pozostało to uruchomienie aplikacji webowej w przeglądarce pod adresem: „localhost/[nazwa folderu z projektem]”

Przed rozpoczęciem działań na naszej stronie zachęcamy do zapoznania się z **instrukcją użytkowania aplikacji,** która została przygotowana w prosty i czytelny sposób poprzez wykorzystanie zrzutów ekranów z opisami.