Bazy danych – projekt

Tomasz Kajda
Przemysław Kociuba

System sprzedaży biletów kolejowych

Wymagania aplikacji

- Możliwość wyszukania najszybszego połączenia pomiędzy wybranymi miastami (bez uwzględnienia przesiadek)
- Możliwość zakupu biletu przy uwzględnieniu rodzaju wagonu (przedziałowy/bez przedziałów)
- Możliwość rejestracji użytkownika, jak i zakupu bez rejestracji
- W przypadku rejestracji, możliwość wglądu w historię swoich biletów

Architektura i proponowane technologie

- Relacyjna baza danych przechowująca dane o użytkownikach (PostgreSQL)
- Dokumentowa baza danych przechowująca dane o trasach pociągów, miejscach i zakupionych biletach (MongoDB)
- Zamodelowany obiektowo backend aplikacji, pośredniczący pomiędzy frontendem i obiema bazami (Java (Spring) i Hibernate do ułatwienia obsługi danych z relacyjnej bazy)
- Zamodelowany obiektowo frontend aplikacji (React)

Typowe wykorzystanie bazy TrainsInfo

Najczęściej użytkownicy będą wyszukiwać interesujące ich połączenia. Z tego powodu ta operacja musi być odpowiednio zoptymalizowana. Na drugim miejscu jest kupno biletów.

Struktura bazy TrainsInfo (MongoDB)

Kolekcja *TrainRoute* – pola dokumentów (identyfikowanych trainRouteID)

- *TravelDate:String*
- *TrainStops: Array – elementami są dokumenty tyupu {stationName:String, departureTime: String, arrivalTime}*
- Stations : {compartmentSeats:Integer, nonCompartmentSeats: Integer, numer:String}

Kolekcja ticket – pola dokumentów (identyfikowanych ticketID)

- *RouteID : Str*
- *StartingStation : Str*
- *EndingStation : Str*
- *Discount : {Type:String, Value:Decimal}*
- *Price: Decimal*
- *TravelerName: String*
- *TravelerSurname: String*
- *seatNo: Integer*
- *compartmentSeat: Boolean*
- *TravelerCountry: String*
- *TravelerCity: String*
- *TravelerZip: String*

- *TravelerAddress: String*
- *travelerEmail: String*
- *(opcjonalne) id_użytkownika : Int*

Baza użytkowników (PostgreSQL)

Złożona z tylko jednej tabeli

Users:

- User_ID Int
- Login Str
- Password Str
- Name Str
- Surname Str
- E-mail Str
- City Str
- Country Str
- Zip-Code Str
- Address Str

Front-end (React)

6 podstron – strona główna, formularz rejestracji, formularz logowania, formularz wyszukiwania połączenia, historia biletów, Strona wprowadzenia danych pasażera podczas zakupu biletu

Strona główna zawiera odnośniki do formularza rejestracji, logowania, zakupu biletu

Formularz rejestracji pobiera wszystkie dane opisane w tabeli Users

Formularz logowania pobiera login i hasło

Formularz wyszukiwania połączenia zawiera:

- Stację początkową
- Stację końcową
- Opcje wyszukiwania (brak przesiadek)
- Datę wyjazdu

Po wypełnieniu formularza i otrzymaniu odpowiedzi z backendu z listą proponowanych połączeń, użytkownik wybiera jedno z nich (o ile istnieje). Po wybraniu wypełnia dane (imię, nazwisko, email) lub się loguje. Wypełnienie tych pól (lub zalogowanie się) potwierdza rezerwację biletu.

Strona historii biletów (dostępna tylko dla zalogowanych użytkowników).

Po otrzymaniu odpowiedzi z backendu listuje wszystkie bilety:

- Id biletu
- Stację początkową
- Stację końcową
- Datę wyjazdu
- Numer miejsca

- Cenę biletu
- Informację o typie miejsca (w wagonie przedziałowym/bezprzedziałowym)

Proponowany back-end (Java)

Oferuje API do:

- *Rejestracji użytkownika (zwraca informację o powodzeniu)*
- *Zalogowania użytkownika (zwraca informację o powodzeniu)*
- *Zapytania dotyczącego trasy (zwraca listę proponowanych połączeń uwzględniając dostępność biletów)*
- *Akceptacji proponowanego po wcześniejszym zapytaniu biletu (zwraca informację o powodzeniu)*

Endpointy

/register/submit (POST)

Wymagane body zapytania w postaci JSONa postaci

{
username:String,
password:String,
firstName:String,
lastName:String,
email:String,
city:String,
country:String,
zip:String,
address:String
}

Działanie: zapis użytkownika w bazie.

Statusy:
200 – użytkownik dodany poprawnie
580 – użytkownik z daną nazwą już istnieje

/routes/find (GET)

Parametry:
firstStation:String
lastStation:String
departureTime:String
arrivalTime:String
travelDate:String

Działanie: wyszukiwanie połączeń w danym dniu po godzinie zawartej w departureTime

Zwraca: Liste obiektów JSON postaci

{

routeID:String
firstStation:String,
lastStation:String,
departureTime:String,
arrivalTime:String
travelDate:String
}

/routes/add (POST)

Wymagane body zapytania – JSON postaci

{
travelDate:String
trainStops:Array obiektów zagnieżdżonych typu TrainStop
train: Obiekt typu train
}

trainStop – obiekt:
{
stationName:String
arrivalTime:String
departureTime:string
compartmentSeats:Array (Integer)
nonCompartmentSeats: Array (Integer)
distanceFromBeginning:Integer
}

train – obiekt:

{
compartmentSeats:Integer
nonCompartmentSeats:Integer
numer:String
}

Działanie: Dodaje opisaną trasę

/routes/tickets (GET)

Parametry:
UserID

Działanie: Zwraca listę obiektów typu ticket osoby o danym userID

Ticket – obiekt

{
routeID:String
startingStation:String
endingStation:String
discount : obiekt typu discount
price: decimal

ticketDate:String
travelerName:String
travelerSurname:String
travelerEmail:String
userId:Integer
seatNo:Integer
compartmentSeat:Boolean
travelerCountry:String
travelerCity:String
travelerZip:String
travelerAddress:String
}

/routes/ticket (POST)

Wymagane body postaci

{
routeID:String
startingStation:String
endingStation:String
discount:String
travelerName:String
travelerSurname:String
travelerEmail:String
compartmentSeat:Boolean
travelerCountry:String
travelerCity:String
travelerZip:String
travelerAddress:String
}

Działanie: Dodaje bilet, aktualizuje wolne miejsca w trasach, wyznacza cene biletu