Lekcja 22: Modelowanie danych

Wyobraź sobie witrynę internetową, która zawiera listę różnych wydarzeń technicznych:

1. Konferencje techniczne: ludzie spotykają się w określonym miejscu i wysłuchują kilku prelekcji. Konferencje zwykle coś kosztują, więc mają swoją cenę.

2. Meetupy: spotkania o mniejszej skali, z punktu widzenia danych, są podobne do konferencji. Odbywają się również w określonym miejscu z szeregiem rozmów, ale w porównaniu z konferencjami technicznymi są zwykle bezpłatne. Cóż, przynajmniej w naszym przykładzie tak jest.

3. Webinary: zamiast ludzi uczęszczających na fizyczną przestrzeń, webinary odbywają się online. Nie potrzebują lokalizacji, ale adresu URL, pod którym ludzie mogą oglądać webinar w przeglądarce. Mogą mieć swoją cenę, ale mogą też być bezpłatne. W porównaniu z pozostałymi dwoma typami wydarzeń webinary obejmują tylko jedną rozmowę.

Wszystkie wydarzenia techniczne mają wspólne właściwości, takie jak data, opis, maksymalna liczba uczestników i liczba RSVP. Mamy również identyfikator typu string w rodzaju właściwości, w którym możemy rozróżnić konferencje, seminaria internetowe i spotkania. W naszej aplikacji dużo pracujemy z tego rodzaju danymi. Pobieramy listę zdarzeń technicznych jako JSON z zaplecza, a także kiedy dodajemy nowe zdarzenia do listy lub chcemy pobrać ich właściwości, aby wyświetlić je w interfejsie użytkownika. Aby życie było łatwiejsze - i znacznie mniej podatne na błędy - chcemy poświęcić trochę czasu na modelowanie tych danych jako typów TypeScript. Dzięki temu otrzymujemy nie tylko odpowiednie oprzyrządowanie, ale także czerwone faliste linie, jeśli o czymś zapomnimy. Zacznijmy od łatwej części. Każde wydarzenie technologiczne ma jakiś temat, może kilka. Przemówienie ma tytuł, streszczenie i mówcę. Na razie mówimy prosto i reprezentujemy je za pomocą prostego ciągu znaków. Typ rozmowy wygląda następująco:

```
type Talk = {

title: string,

abstract: string,

speaker: string

}
```

Mając to na miejscu, możemy opracować typ konferencji:

```
type Conference = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

kind: string,

location: string,

price: number,
```

talks: Talk[]

}

… typ spotkań, w których cena to ciąg znaków („bezpłatnie”) zamiast liczby:

type Meetup = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

kind: string,

location: string,

price: string,

talks: Talk[]

}

… I rodzaj webinarów, w których prowadzimy tylko jedną rozmowę i nie mamy fizycznej lokalizacji, ale adres URL do organizacji wydarzenia:

type Webinar = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

kind: string,

url: string,

price?: number,

talks: Talk

}

Widzisz również, że typy są opcjonalne. Mając te cztery typy na miejscu, modelowaliśmy już dużą część możliwych danych, które możemy uzyskać z zaplecza. Niektóre części mają wspólny kształt we wszystkich trzech typach wydarzeń, a inne są subtelnie lub całkowicie różne.

Typy przecięć

Pierwszą rzeczą, jaką sobie uświadamiamy, jest to, że istnieje wiele podobnych właściwości; właściwości, które również powinny pozostać takie same, podstawowy kształt TechEvent. Dzięki TypeScript jesteśmy w stanie wyodrębnić ten kształt i połączyć go z właściwościami specyficznymi dla naszych konkretnych pojedynczych typów. Najpierw utwórzmy typ TechEventBase, który będzie zawierał wszystkie właściwości, które są takie same we wszystkich trzech typach zdarzeń.

```
type TechEventBase = {

title: string,

description: string

date: Date,

capacity: number,

rsvp: number,

kind: string

}
```

Następnie refaktoryzujmy oryginalne trzy typy, aby połączyć TechEventBase z określonymi właściwościami każdego typu.

```
type Conference = TechEventBase & {

location: string,

price: number,

talks: Talk[]

}

type Meetup = TechEventBase & {

location: string,

price: string,

talks: Talk[]

}

type Webinar = TechEventBase & {

url: string,

price?: number,

talks: Talk

}
```

Nazywamy to pojęcie typami skrzyżowań. Czytamy operator & jako i. Łączymy właściwości jednego typu A z właściwościami innego typu B, podobnie jak rozszerzanie klas. Rezultatem jest nowy typ z właściwościami typu A i typu B. Natychmiastowa korzyść polega na tym, że możemy modelować typowe właściwości w jednym miejscu, co znacznie ułatwia aktualizacje i zmiany. Co więcej,

rzeczywista różnica między typami staje się znacznie wyraźniejsza i łatwiejsza do odczytania. Każdy podtyp ma tylko kilka właściwości, którymi musimy się zająć, zamiast pełnej listy.

Typy związków

Ale co się stanie, jeśli otrzymamy listę wydarzeń technicznych, w których każde zgłoszenie może być seminarium internetowym, konferencją lub spotkaniem? Gdzie nie wiemy dokładnie, jakie otrzymujemy wpisy, tylko że należą one do jednego z trzech typów zdarzeń. W takich sytuacjach możemy użyć pojęcia zwanego typami związków. Dzięki typom związkowym możemy modelować dokładnie następujący scenariusz: zdefiniowanie typu TechEvent, który może być seminarium internetowym, konferencją lub spotkaniem. Lub w kodzie:

type TechEvent = Webinar | Conference | Meetup;

Czytamy operatora potoku | jak lub. Otrzymujemy nowy typ, typ, który próbuje objąć wszystkie możliwe właściwości dostępne z typów, które ustawiliśmy w union. Nowy typ ma dostęp do następujących właściwości:

• title, description, date, capacity, rsvp, kindj - cechy wspólne dla wszystkich trzech typów z pierwotnym typem pierwotnym. To właśnie daje nam kształt TechEventBase.

• talks. Ta właściwość może być pojedynczą Talk lub tablicą Talk[]. Jego nowy typ to Talk | Talk[].

• prie. Właściwość price jest również dostępna we wszystkich trzech oryginalnych typach obiektów, ale jej własny typ jest inny. price może być string lub number, a po Webinar może być opcjonalna. Aby bezpiecznie pracować z ceną, musimy przeprowadzić pewne kontrole w naszym kodzie: musimy sprawdzić, czy jest dostępna, a następnie musimy sprawdzić typeof, aby zobaczyć, czy mamy do czynienia z number, czy string

Praca z price i talks może wyglądać mniej więcej tak:

function printEvent(event: TechEvent) {

if(event.price) {

// Price exists!

if(typeof event.price === 'number') {

// We know that price is a number

console.log('Price in EUR: ', event.price)

} else {

// We know that price is a string, so the

// event is free!

console.log('It is free!')

}

}

if(Array.isArray(event.talks)) {

// talks is an array

```
event.talks.forEach(talk => {

console.log(talk.title)

})

} else {

// It's just a single talk

console.log(event.talks.title)

}

}
```

Czy ta struktura coś ci przypomina? Wcześniej dowiedzieliśmy się o koncepcji przepływu kontroli i zawężaniu typów za pomocą strażników typów. To jest dokładnie to, co się tutaj dzieje. Ponieważ typ może przybierać różne kształty, możemy użyć ochrony typów (instrukcji if), aby zawęzić typ unii do jednego typu. Należy pamiętać, że poruszamy się między typami związków w odpowiednich cenach nieruchomości i rozmowach. Wszystkie inne informacje z oryginalnych typów seminariów internetowych, konferencji i spotkań, których nie można ujednolicić (takie jak lokalizacja i adres URL), są usuwane z kształtu związku. Potrzebujemy więcej informacji, aby zawęzić zakres do oryginalnych kształtów obiektów.