Consuming a REST API from React

1. Sample REST API
2. Invoking the REST API from React

1. Sample REST API

- Overview
- Reviewing the database schema
- Defining 1:many entity relationships
- Defining a CRUD repository interface
- Defining a REST API

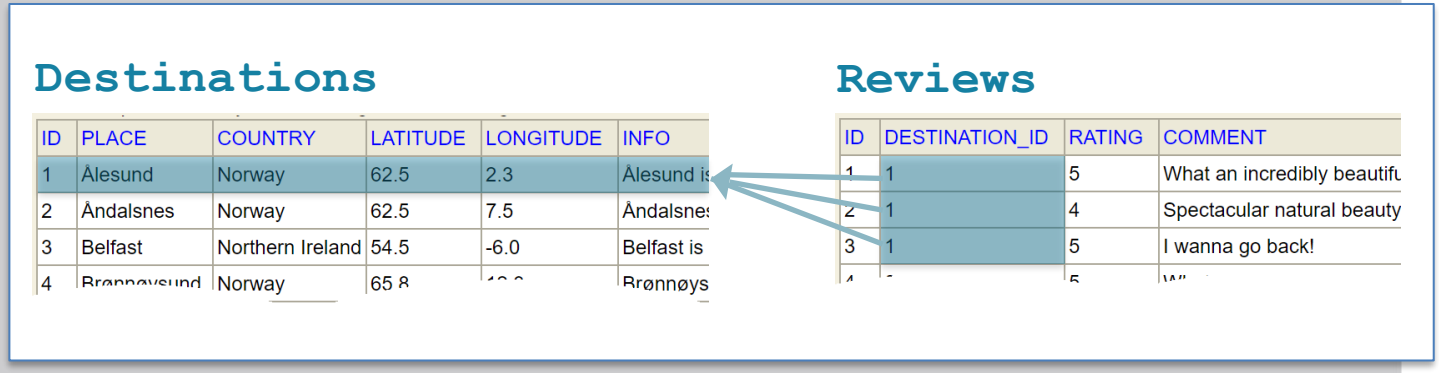
Demo Spring module: demo-full-stack-app

Overview

- We've implemented a Spring Boot app with a REST API
  - In IntelliJ, see module `demo-full-stack-app`
- We'll review:
  - The database schema
  - Entities and repositories
  - The REST API that provides a callable façade

Reviewing the Database Schema

- The database schema has 2 tables:



- Note there are many reviews per destination
  - Via the `DESTINATION_ID` foreign key

Defining 1:Many Entity Relationships

- One destination has many reviews:

```
@Entity
@Table(name="DESTINATIONS")
public class Destination {
    …
    @OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
    @JoinColumn(name="DESTINATION_ID")
    private List<Review> reviews;
}
```

Destination.java

```
@Entity
@Table(name="REVIEWS")
public class Review {
    …
}
```

Review.java

Defining a CRUD Repository Interface

- We've defined a CRUD repository interface to simplify persistence of destination entities

```
public interface DestinationRepository extends CrudRepository<Destination, Long> {}
```

DestinationRepository.java

- Any changes to a destination entity will automatically cascade to review entities, due to the "cascade" option

```
@Entity @Table(name="DESTINATIONS")
public class Destination {
    …
    @OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
    @JoinColumn(name="DESTINATION_ID")
    private List<Review> reviews;
}
```

Destination.java

Defining a REST API

- We've defined a REST API for destinations
  - See `DestinationController.java`
- You can `GET` these endpoints:
  - `destinations`
  - `destinations/1`
- You can `PUT` this endpoint:
  - `destinations/addReviewForDestination/1`

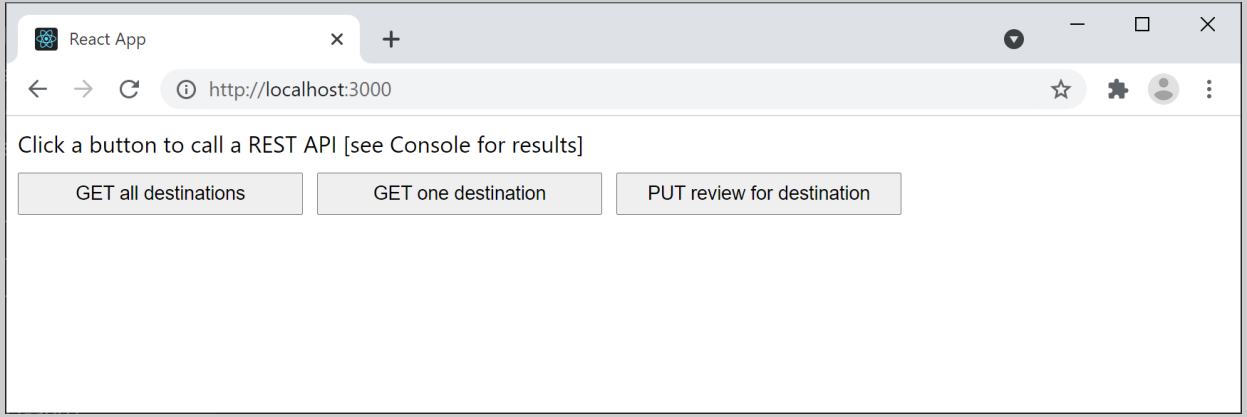
2. Calling the REST API from React

- Overview
- Organizing your client code
- How to call REST endpoints via fetch()
- Getting all destinations
- Getting one destination
- Adding a review for a destination

```
React demo: demo-react-app3
To install: npm install
To run:     npm start
```

Overview

- We're going to see a React app that calls the REST API
  - Run `demo-react-app3`
  - Each button pings one of the REST endpoints
  - Results are displayed in the Console



Organizing your Client Code

- It's good practice to put all your REST client code in a dedicated class (or loose functions)
  - See `RestClient.tsx`

```
export default class RestClient {

    static baseUrl = "http://localhost:8080"

    // Define methods here, to encapsulate calls to REST endpoints...

}
```

RestClient.tsx

How to Call REST Endpoints via `fetch()`

- The easiest way to call a REST endpoint is via `fetch()`
  - Standard JavaScript function
  - Takes a URL parameter, plus optional other info
- `fetch()` invokes the REST endpoint asynchronously
  - Returns a `Promise` immediately
  - The `Promise` will eventually hold the response
  - Call `then()` to access the response, when it's ready

Getting All Destinations

- Let's call the REST endpoint to get all destinations:

```
static getDestinations() : Promise<any> {

    const url = `${RestClient.baseUrl}/destinations`
    const promise1 = fetch(url)    // Call REST endpoint (asynchronously).

    const promise2 = promise1.then(response => {
        return response.json()     // Extract JSON from response body (asynchronously).
    })
    return promise2
}
```

RestClient.tsx

- We can utilize the above function in our UI code:

```
function demo1() {
    const promise = RestClient.getDestinations()
    promise.then(data => console.log(`All destinations: ${JSON.stringify(data)}`))
}
```

App.tsx

Getting One Destination

- Let's call the REST endpoint to get one destination
  - `await` keyword subscribes to `Promise` (implicit `then`)
  - `async` keyword allows us to use `await`

```
static async getDestination(id: number) : Promise<any> {
    const url = `${RestClient.baseUrl}/destinations/${id}`
    const response = await fetch(url)
    return response.json()
}
```

RestClient.tsx

- We can utilize the above function in our UI code:

```
async function demo2() {
  const data = await RestClient.getDestination(1)
  console.log(`Destination 1: ${JSON.stringify(data)}`)
}
```

App.tsx

Adding a Review for a Destination

- Let's call the REST endpoint to add (`PUT`) a review

```
static addReview(id: number, review: any) : Promise<any> {
    const url = `${RestClient.baseUrl}/destinations/addReviewForDestination/${id}`
    return fetch(
              url,
              {
                  method: 'PUT',
                  headers: { 'Content-Type': 'application/json' },
                  body: JSON.stringify(review)
              }
    )
}
```

RestClient.tsx

- We can utilize the above function in our UI code:

```
async function demo3() {
    const aReview = {rating: 5, comment: ' FANTASTIC', by: 'Andy'}
    const addResponse = await RestClient.addReview(1, aReview)
    console.log(addResponse)
}
```

App.tsx

Summary

- Sample REST API
- Invoking the REST API from React