

Travel Project

Tuesday, November 2, 2021 7:37 AM

Display Destination details

The screenshot shows a web browser window titled "Traveler's Handbook". The URL in the address bar is "http://localhost:3000/destination/1". The page content is for the destination "Ålesund". The header includes links for "Home", "Destinations", and "About". Below the header, the title "Ålesund" is displayed. A table provides basic information: Country (Norway), Latitude (62.5), Longitude (2.3), and Info (Ålesund is a sea port on the west coast of Norway, noted for its beautiful Art Nouveau architecture and spectacular mountain scenery). A large, scenic photograph of Ålesund is shown, featuring the town built on several islands in a fjord, surrounded by snow-capped mountains.

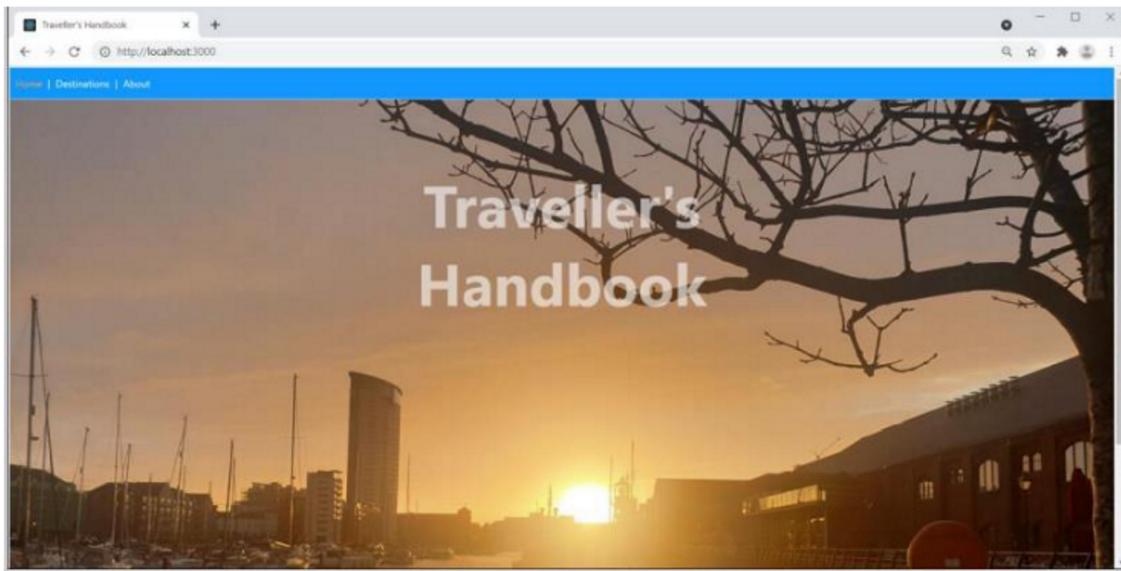
Add a Review for a destination

The screenshot shows a web form for adding a review. The form is divided into two sections: "Reviews" and "Add Review". The "Reviews" section displays three reviews with small profile icons and names: "What an incredibly beautiful place [J. Smith]", "Spectacular natural beauty, with snow-peaked mountains and crystal sea [O. Nordmann]", and "I wanna go back! [A. Olsen]". The "Add Review" section contains fields for "Rating" (with a dropdown menu), "Comment" (a text area), and "Your name" (a text input field). At the bottom is a "Add review" button.

Travel-backend

Travel Handbook server side

This is an all-purpose travel directory, providing essential information and advice for planning any kind of trip anywhere in the world. It has top tips from famous travelers, distinguished experts and insiders in the travel industry.



- Home
 - Display home page.
- Destination
 - Display available destinations

A screenshot of a web browser window titled "Traveller's Handbook". The address bar shows "http://localhost:3000/destinations". The page has a blue header with the word "Destinations" in white. Below the header is a table with four columns and five rows of destination names. The first four rows have four cells each, and the fifth row has two cells, with the last three cells being empty.

Ålesund	Åndalsnes	Belfast	Brønnøysund
Budapest	Cape Town	Copenhagen	Cork
Grimstad	Johannesburg	Oslo	Rome
Singapore	Swansea		

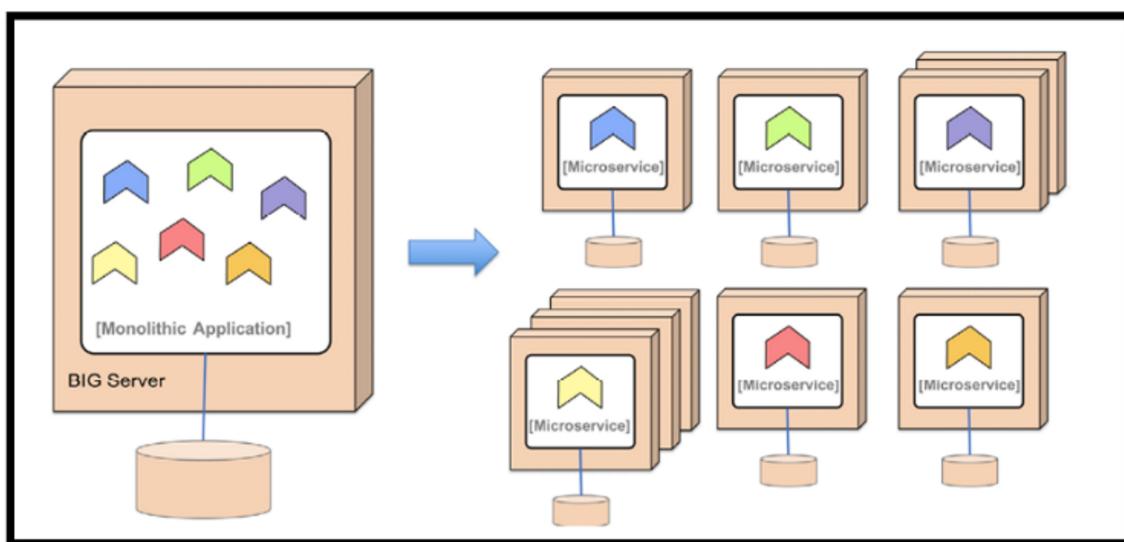
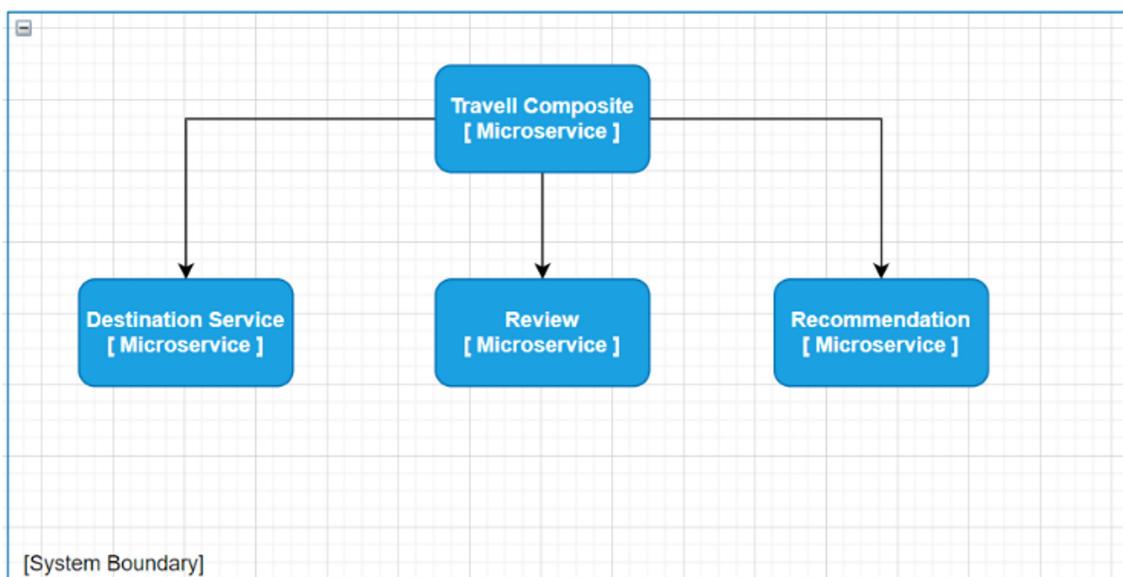
My way of Microservice

An autonomous software component that is independently upgradeable, replicable and scalable True up to some extent but not completely due to changing real-time scenarios.

This is a very small system landscape of cooperating microservices.

The surrounding support services that we will add might look overwhelmingly complex for these few microservices.

But keep in mind that the solutions presented here aim to support a much larger system landscape.



There is plan for document complete team formation and managing Microservice application.

Microservices which we would be working is **Destination, Review and Recommend**. These microservices deals with one type of resources and a composite microservice called **Travel Composite** which aggregates the information from other core service, and this would be best for Backend-FOR-Frontend

- Destination Service The destination service manages destination information.
 - destId
 - place
 - country
 - latitude
 - longitude
 - info
 - image
- Review Service
 - destId
 - reviewId
 - author
 - subject
 - content
- Recommend
 - destId
 - recommendationId
 - author
 - rate
 - content
- Travel Composite the Composite service gets the information from core microservices and presents information about the destination.
 - A Destination information as described in Destination service.
 - A list of destination reviews for the specific destination, described in review service.
 - A list of travel recommendation for the specific destination, described in recommend service.

- It will be good to add service address attribute in each microservice resources to know which container is responding.
- My way of microservices design is to try to have single responsible services as much possible.

```
microservice/
|---TravelCompositeService
|---DestinationService
|---ReviewService
|---RecommendService
```

NOTE:

It is debatable whether it is good practice to have **API** in a common module or not.

It is good practice for microservices that are part of the same delivery organization, that is, same whose release is governed by same organization. Again, From DevOps perspective it is preferable to build each project in its own build pipeline

Sample Project Structure

```
microservices/
|---TravelCompositeService
|---DestinationService
|---ReviewService
|---RecommendService
api/
util/
```

Thinking Solution Building Block

These Instructions will get you a copy of the project up and running on local machine. We will develop microservice that contain business logic based on Spring Beans and expose REST API using **Spring WEB-FLUX**.

The API will be documented based on **OpenAPI** specification using **springdoc-openapi**

Prerequisites

What things you need to install the software and how to install them?

It is recommended you have the Chocolatey Package Manager installed for windows to ease up the setup process. Please install using instructions [here](#).

Once you have that please proceed with rest of the steps:

```
choco install maven  
choco install docker-desktop  
choco install istioctl  
choco install kubernetes-helm  
choco install kubernetes-cli  
choco install cygwin
```

Let's use **spring-init** tool for easier way of creating microservices.

For each microservice , we will create a Spring boot project that does the following

- Uses **gradle** build tool.
- Generates Java code for **Version 8** .
- Packages the project as **Fat Jar** .
- Brings in dependencies for the **Actuator** and **Web-Flux** .
- Is Based on **Spring Latest (V2.5.2)**

Setting Up multi-project build in gradle

1. Create a **settings.gradle** file.
2. Copy the gradle executable files that were generated from one of the projects so we can reuse.

```
cp -r microservices/destination-service/gradle .
cp microservices/destination-service/gradlew .
cp microservices/destination-service/gradlew.bat .
```

3. We no longer need the **Gradle** executable in each project. We can remove with below command.

```
find microservices -depth -name "gradle" -exec rm -rfv "{}" \;
find microservices -depth -name "gradlew*" -exec rm -fv "{}" \;
```

4. Now build the microservice project with one command.

```
./gradlew build
```

- `java -jar microservices/destination-service/build/libs/*.jar &`

Initial development phase should use **RestTemplate** helper class.
We will replace later with Reactive HTTP client **Webclient**.

```
java -jar microservices/destination-composite-service/build/libs/*.jar &
java -jar microservices/destination-service/build/libs/*.jar &
java -jar microservices/recommendation-service/build/libs/*.jar &
java -jar microservices/review-service/build/libs/*.jar &
```

```
curl http://localhost:7000/dest-composite/1 -s | jq
```

```
kill $(jobs -p)
```

