*Eclipse shortcuts:*

*Open class:*

* *Ctrl + Shift + T* on a PC or
* *Cmd + Shift + T* on a Mac.

*Import dependency:*

* *Ctrl + Shift + O* on a PC or
* *Cmd + Shift + O* on Mac

*Paleistą projektą atsidarome šiuo adresu:*

[*http://localhost:8081/*](http://localhost:8081/)

*Pradinis failas:*

[*https://tinyurl.com/yx4habwb*](https://tinyurl.com/yx4habwb)

## Užkrauname oro uosto pavadinimą iš JSON failo:

#### Pridedame kintamuosius Airport objektui ir metodus jiems gauti/priskirti

public class Airport {

private int waitTimeMinutes;

private String name;

private String iataCode;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getIataCode() {

return iataCode;

}

public void setIataCode(String iataCode) {

this.iataCode = iataCode;

}

...

}

#### Pildome AirportFileService klasę. Parsiname JSON. Ieškome oro uosto JSON faile, grąžiname Airport objekto pavidalu.

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

List<JsonAirport> airports = ResourceUtils

.loadDataFromJson(RESOURCE\_PATH, JsonAirport.class);

final String finalAirportCode = airportCode.trim().toUpperCase();

Optional<JsonAirport> optAirport = Objects.requireNonNull(airports).stream()

.filter(o -> o.getIataCode().equals(finalAirportCode)).findFirst();

JsonAirport jsonAirport = optAirport.get();

Airport airport = new Airport();

airport.setIataCode(jsonAirport.getIataCode());

airport.setName(jsonAirport.getName());

return airport;

}

#### Atnaujiname *TripServiceImpl.* Naudojame *AirportFileService* oro uostų info gauti.

public class TripServiceImpl implements TripService {

@Autowired

private AirportService airportService;

public Trip getTrip(String fromAirport, String toAirport) {

Trip trip = new Trip();

Airport from = airportService.getAirport(fromAirport);

Airport to = airportService.getAirport(toAirport);

trip.setFromAirport(from);

trip.setToAirport(to);

return trip;

}

}

#### Pridedam trūkstamus metodus į *Trip* klasę

public class Trip {

...

private Airport fromAirport;

private Airport toAirport;

public Airport getFromAirport() {

return fromAirport;

}

public void setFromAirport(Airport fromAirport) {

this.fromAirport = fromAirport;

}

public Airport getToAirport() {

return toAirport;

}

public void setToAirport(Airport toAirport) {

this.toAirport = toAirport;

}

...

}

## Add CityFileService (get city objects from JSON)

#### Kuriame servisą *CityFileService* oro uosto miestui gauti:

public class CityFileService implements CityService {

private static final String RESOURCE\_PATH = "static/cities.json";

@Override

public City getCityByAirport(String iataCode) {

List<JsonCity> cities = ResourceUtils.loadDataFromJson(RESOURCE\_PATH, JsonCity.class);

Optional<JsonCity> optCity = Objects.requireNonNull(cities).stream()

.filter(o -> o.getAirport().equals(iataCode)).findFirst();

if (optCity.isPresent()) {

JsonCity jsonCity = optCity.get();

City city = new City();

city.setName(jsonCity.getName());

return city;

}

return null;

}

}

#### Klasė, į kurią sudėsime City info, gautą iš JSON failo

@JsonInclude(JsonInclude.Include.NON\_NULL)

public class JsonCity {

@JsonProperty("city\_name")

private String name;

@JsonProperty("iata\_code")

private String airport;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAirport() {

return airport;

}

public void setAirport(String airport) {

this.airport = airport;

}

}

#### Papildome City objektą (klasė, kuria operuoja mūsų sukurtas *CityFileService)*

public class City {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

#### Atnaujiname *TripServiceImpl*

public class TripServiceImpl implements TripService {

@Autowired

private AirportService airportService;

@Autowired

@Qualifier("city-file")

private CityService cityService;

public Trip getTrip(String fromAirport, String toAirport) {

Trip trip = new Trip();

Airport from = airportService.getAirport(fromAirport);

from.setCity(cityService.getCityByAirport(from.getIataCode()));

Airport to = airportService.getAirport(toAirport);

to.setCity(cityService.getCityByAirport(to.getIataCode()));

trip.setFromAirport(from);

trip.setToAirport(to);

return trip;

}

}

#### Pridedam trūkstamą *City* kintamąjį

public class Airport {

private int waitTimeMinutes;

private String name;

private String iataCode;

private City city;

public City getCity() {

return city;

}

public void setCity(City city) {

this.city = city;

}

...

}

*City name should be visible*

## Užkrauname oro uosto šalies info

#### Papildome Airport klasę

public class Airport {

private String name;

private String iataCode;

private String countryCode;

private String region;

private City city;

...

public String getCountryCode() {

return countryCode;

}

public void setCountryCode(String countryCode) {

this.countryCode = countryCode;

}

public String getRegion() {

return region;

}

public void setRegion(String region) {

if (region.contains("-")) {

this.region = region.substring(region.indexOf("-")+1, region.length());

} else {

this.region = region;

}

}

}

#### Pridedame country code ir region (taip pat jie bus reikalingi scraperio request URLui sukurti)

@Service

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

...

Airport airport = new Airport();

airport.setIataCode(jsonAirport.getIataCode());

airport.setName(jsonAirport.getName());

airport.setCountryCode(jsonAirport.getIsoCountry());

airport.setRegion(jsonAirport.getIsoRegion());

return airport;

}

*Country should be visible*

## Kuriame web scrapinimo servisą.

#### Pridedame funkciją request URL kurti. Taip pat ir funkciją kuri parsiunčia web puslapį kaip Document objektą.

@Service

public class WeatherScraperService implements WeatherService {

private static final String REQUEST\_URL = "https://www.wunderground.com/weather/";

@Override

public Weather getAirportWeather(Airport airport) {

try {

Document document = getDocument(getRequestUrl(airport));

} catch (IOException e) {

e.printStackTrace();

}

return new Weather();

}

protected Document getDocument(String scrapingUrl) throws IOException {

return Jsoup.connect(scrapingUrl).get();

}

private String getRequestUrl(Airport airport) {

String requestUrl = REQUEST\_URL + airport.getCountryCode().toLowerCase();

if (airport.getRegion() != null && airport.getCountryCode().equalsIgnoreCase("US")) {

requestUrl = requestUrl + "/"

+ airport.getRegion().toLowerCase();

}

return requestUrl + "/" + airport.getCity().getName().toLowerCase();

}

}

#### Pildome Weather objektą

public class Weather {

private Double temperature;

private String temperatureUnit;

private Double windSpeed;

private String windSpeedUnit;

public Double getTemperature() {

return temperature;

}

public void setTemperature(Double temperature) {

this.temperature = temperature;

}

public String getTemperatureUnit() {

return temperatureUnit;

}

public void setTemperatureUnit(String temperatureUnit) {

this.temperatureUnit = temperatureUnit;

}

public Double getWindSpeed() {

return windSpeed;

}

public void setWindSpeed(Double windSpeed) {

this.windSpeed = windSpeed;

}

public String getWindSpeedUnit() {

return windSpeedUnit;

}

public void setWindSpeedUnit(String windSpeedUnit) {

this.windSpeedUnit = windSpeedUnit;

}

}

#### Parsiname HTML turinį į Weather objektą

@Service

public class WeatherScraperService implements WeatherService {

private static final String REQUEST\_URL = "https://www.wunderground.com/weather/";

@Override

public Weather getAirportWeather(Airport airport) {

Weather weather = new Weather();

try {

Document document = getDocument(getRequestUrl(airport));

weather.setTemperature(parseTemperature(document));

weather.setTemperatureUnit(parseTempUnit(document));

weather.setWindSpeed(parseWindSpeed(document));

weather.setWindSpeedUnit(parseWindSpeedUnit(document));

} catch (IOException e) {

e.printStackTrace();

}

return weather;

}

protected Double parseTemperature(Document document) {

Element temperature = document.select(".conditions-circle .wu-unit-temperature .wu-value.wu-value-to")

.first();

if (temperature != null) {

return Double.parseDouble(temperature.text());

}

return null;

}

protected String parseTempUnit(Document document) {

Element tempUnit = document.select(".conditions-circle .wu-unit-temperature .ng-star-inserted")

.first();

if (tempUnit != null) {

return tempUnit.text();

}

return null;

}

protected Double parseWindSpeed(Document document) {

Element windSpeed = document.select(".condition-wind .wu-unit-speed .wu-value.wu-value-to")

.first();

if (windSpeed != null) {

return Double.parseDouble(windSpeed.text());

}

return null;

}

protected String parseWindSpeedUnit(Document document) {

Element windSpeedUnit = document.select(".condition-wind .wu-unit-speed .ng-star-inserted")

.first();

if (windSpeedUnit != null) {

return windSpeedUnit.text();

}

return null;

}

protected Document getDocument(String scrapingUrl) throws IOException {

return Jsoup.connect(scrapingUrl).get();

}

private String getRequestUrl(Airport airport) {

String requestUrl = REQUEST\_URL + airport.getCountryCode().toLowerCase();

if (airport.getRegion() != null) {

requestUrl = requestUrl + "/"

+ airport.getRegion().toLowerCase();

}

return requestUrl + "/" + airport.getCity().getName().toLowerCase();

}

}

#### Pridedame orų servisą į TripServiceImpl (kitaip nematysime oro info frontende)

@Service

public class TripServiceImpl implements TripService {

@Autowired

private AirportService airportService;

@Autowired

@Qualifier("city-file")

private CityService cityService;

@Autowired

@Qualifier("weather")

private WeatherService weatherService;

public Trip getTrip(String fromAirport, String toAirport) {

Trip trip = new Trip();

...

trip.setFromAirport(from);

trip.setToAirport(to);

Weather airportWeather = weatherService.getAirportWeather(to);

trip.setWeather(airportWeather);

return trip;

}

}

#### Pridedame trūkstamą kintamąjį į *Trip* klasę

public class Trip {

private Weather weather;

public Weather getWeather() {

return weather;

}

public void setWeather(Weather weather) {

this.weather = weather;

}

}

*Temperature and wind speed should be visible. Although status 404 may be encountered*

## Ištraukiame oro uosto koordinates ir iš jų skaičiuojame skrydžio atstumą.

#### Papildomą Airport objektą ilgumos ir platumos kintamaisiais

public class Airport {

private String name;

private String iataCode;

private String countryCode;

private Double latitude;

private Double longitude;

private City city;

...

public Double getLatitude() {

return latitude;

}

public void setLatitude(Double latitude) {

this.latitude = latitude;

}

public Double getLongitude() {

return longitude;

}

public void setLongitude(Double longitude) {

this.longitude = longitude;

}

}

#### Išparsiname ilgumą ir platumą į JsonAirport POJO

@Service

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

...

Airport airport = new Airport();

airport.setIataCode(jsonAirport.getIataCode());

airport.setName(jsonAirport.getName());

airport.setCountryCode(jsonAirport.getIsoCountry());

airport.setLatitude(Double.parseDouble(jsonAirport.getLatitudeDeg()));

airport.setLongitude(Double.parseDouble(jsonAirport.getLongitudeDeg()));

return airport;

}

#### Apskaičiuojame atstumą tarp oro uostų pagal ilgumą ir platumą. Rezultatą priskiriame Trip objektui

@Service

public class TripServiceImpl implements TripService {

...

public Trip getTrip(String fromAirport, String toAirport) {

...

Weather airportWeather = weatherService.getAirportWeather(to.getCountryCode(), to.getCity().getName());

trip.setWeather(airportWeather);

double distance = GeoUtils.distance(

from.getLatitude(), from.getLongitude(),

to.getLatitude(), to.getLongitude());

trip.setDistance(distance);

return trip;

}

}

*Turi rodyti validų atstumą*

## Krauname transporto informaciją:

#### Pildome Airport modelį

public class Airport {

private String name;

private String iataCode;

private String countryCode;

private Double latitude;

private Double longitude;

private Set<String> transportation;

private City city;

….

public Set<String> getTransportation() {

return transportation;

}

public void setTransportation(Set<String> transportation) {

this.transportation = transportation;

}

}

#### Priskiriame transportavimo info Airport objektui iš JSON failo

@Service

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

...

Airport airport = new Airport();

airport.setIataCode(jsonAirport.getIataCode());

airport.setName(jsonAirport.getName());

airport.setCountryCode(jsonAirport.getIsoCountry());

airport.setLatitude(Double.parseDouble(jsonAirport.getLatitudeDeg()));

airport.setLongitude(Double.parseDouble(jsonAirport.getLongitudeDeg()));

airport.setTransportation(jsonAirport.getTransportation());

return airport;

}

*Transportation should be visible*

## Krauname vidutinį laukimo laiką oro uoste:

#### Traukiame šią info iš JSON

@Service

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

...

Airport airport = new Airport();

...

airport.setWaitTimeMinutes(jsonAirport.getWaitTimeMinutes());

return airport;

}

Wait time should be visible

## Add short summary to Airport

#### Traukiame summary iš JSON

@Service

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

...

Airport airport = new Airport();

...

airport.setWaitTimeMinutes(jsonAirport.getWaitTimeMinutes()); airport.setSummary(jsonAirport.getSummary());

return airport;

}

#### Pridedame trūkstamą fieldą.

public class Airport {

...

private String summary;

public String getSummary() {

return summary;

}

public void setSummary(String summary) {

this.summary = summary;

}

...

}

*Summary should be visible*

*1st milestone - all cells of table should be filled*

## Suhandliname errorus:

#### Check klasėje bus atliekami input tikrinimai ir invokinami exception’ai

package com.wcc.backend.travelerassistant.validation;

import java.util.Optional;

public class Check {

public static void notBlank(String str, String message, Object... args) {

if (str == null || str.trim().length() == 0) {

throw new RestException(message, args);

}

}

public static void hasOptional(Optional optional, String message, Object... args) {

if (optional.isEmpty()) {

throw new RestException(message, args);

}

}

}

#### Kuriame savo exception klasę

package com.wcc.backend.travelerassistant.validation;

public class RestException extends RuntimeException {

private String message;

private Object[] args;

public RestException(String message, Object[] args) {

this.message = message;

this.args = args;

}

@Override

public String getMessage() {

return message;

}

public Object[] getArgs() {

return args;

}

}

* 1. Rašome tikrinimo funkcijas *TripController* klasėje ir ir apvelkame *Trip* klasę į *ResponseEntity*

@RestController

public class TripController {

@Autowired

private TripService tripService;

@GetMapping("/trip")

public ResponseEntity<Trip> tripInfo(@RequestParam(value="from") String fromAirport,

@RequestParam(value="to") String toAirport) {

fromAirport = fromAirport.trim();

toAirport = toAirport.trim();

Check.notBlank(fromAirport, Message.MSG\_NO\_FROM\_AIRPORT);

Check.notBlank(toAirport, Message.MSG\_NO\_TO\_AIRPORT);

return new ResponseEntity<>(tripService.getTrip(fromAirport, toAirport), HttpStatus.OK);

}

}

#### Tikriname, ar buvo rastas oro uostas JSON faile

public class AirportFileService implements AirportService {

private static final String RESOURCE\_PATH = "static/airports.json";

@Override

public Airport getAirport(String airportCode) {

List<JsonAirport> airports = ResourceUtils

.loadDataFromJson(RESOURCE\_PATH, JsonAirport.class);

final String finalAirportCode = airportCode.trim().toUpperCase();

Optional<JsonAirport> optAirport = Objects.requireNonNull(airports).stream()

.filter(o -> o.getIataCode().equals(finalAirportCode)).findFirst();

Check.hasOptional(optAirport, Message.MSG\_AIRPORT\_NOT\_FOUND, airportCode);

JsonAirport jsonAirport = optAirport.get();

Airport airport = new Airport();

airport.setIataCode(jsonAirport.getIataCode());

airport.setName(jsonAirport.getName());

return airport;

}

#### Kuriame klasę, skirtą exceptionų apdorojimui

@ControllerAdvice

public class RestExceptionHandler {

@Autowired

private MessageSource messageSource;

@ExceptionHandler(RestException.class)

public ResponseEntity<ErrorMessage> handleIllegalArgument(RestException ex, Locale locale) {

String errorMessage = messageSource.getMessage(ex.getMessage(), ex.getArgs(), locale);

return new ResponseEntity<>(new ErrorMessage(errorMessage), HttpStatus.BAD\_REQUEST);

}

@ExceptionHandler(Exception.class)

public ResponseEntity<ErrorMessage> handleExceptions(Exception ex, Locale locale) {

ex.printStackTrace();

String errorMessage = messageSource.getMessage(Message.MSG\_UNEXPECTED, null, locale);

return new ResponseEntity<>(new ErrorMessage(errorMessage), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

#### Apvelkame žinutę, kuri bus rodoma vartotojui klaidos metu

package com.wcc.backend.travelerassistant.validation;

public class ErrorMessage {

private String errorMessage;

public ErrorMessage(String message) {

this.errorMessage = message;

}

public String getErrorMessage() {

return errorMessage;

}

}

#### Klasė žinučių tipams grupuoti

package com.wcc.backend.travelerassistant.validation;

public class Message {

public static final String MSG\_AIRPORT\_NOT\_FOUND = "message.exception.notFound.airport";

public static final String MSG\_NO\_FROM\_AIRPORT = "message.exception.field.empty.fromAirport";

public static final String MSG\_NO\_TO\_AIRPORT = "message.exception.field.empty.toAirport";

public static final String MSG\_UNEXPECTED = "message.exception.unexpected";

}

#### Sudedame žinučių reikšmes į configą

*traveler-assistant/src/main/resources/messages.properties*

message.exception.notFound.airport=Airport {0} not found.

message.exception.unexpected=Unexpected error. Check system log file.

message.exception.field.empty.fromAirport=Departure airport cannot be empty.

message.exception.field.empty.toAirport=Destination airport cannot be empty.

*3-4 types of error messages should be displayed*

## Update Check and Message classes

#### Funkcija tikrinanti ar IATA kodas yra validus

public class Check {

...

public static void iata(String iataCode, String message, Object... args) {

if (iataCode.length() != 3 || !iataCode.chars().allMatch(Character::isLetter)) {

throw new RestException(message, args);

}

}

}

#### Pridedam IATA kodo tikrinimą į *TripController*

@RestController

public class TripController {

@Autowired

private TripService tripService;

@GetMapping("/trip")

public ResponseEntity<Trip> tripInfo(@RequestParam(value="from") String fromAirport,

@RequestParam(value="to") String toAirport) {

fromAirport = fromAirport.trim();

toAirport = toAirport.trim();

Check.notBlank(fromAirport, Message.MSG\_NO\_FROM\_AIRPORT);

Check.notBlank(toAirport, Message.MSG\_NO\_TO\_AIRPORT);

Check.iata(fromAirport, Message.MSG\_INVALID\_IATA, fromAirport);

Check.iata(toAirport, Message.MSG\_INVALID\_IATA, toAirport);

return new ResponseEntity<>(tripService.getTrip(fromAirport, toAirport), HttpStatus.OK);

}

}

#### Įdedame papildomą žinutės tipą

public class Message {

...

public static final String MSG\_NO\_TO\_AIRPORT = "message.exception.field.empty.toAirport";

public static final String MSG\_UNEXPECTED = "message.exception.unexpected";

public static final String MSG\_INVALID\_IATA = "message.exception.invalid.iata";

}

#### Įdedame žinutės reikšmę į configą

*traveler-assistant/src/main/resources/messages.properties*

...

message.exception.field.empty.toAirport=Destination airport cannot be empty.

message.exception.invalid.iata=Invalid IATA code {0}.

*Another error message should be visible*

1. Iškeliame hardcodą į configą

#### Pridedame trūkstamas configo reikšmės miestų ir oro servisams į *application.properties*

server.port=8081

service.airport.resource.path=static/airports.json

service.city.resource.path=static/cities.json

service.weather.request.url.template=https://www.wunderground.com/weather/

#### AirportFileService

@Service

public class AirportFileService implements AirportService {

@Value("${service.airport.resource.path}")

private String resourcePath;

@Override

public Airport getAirport(String airportCode) {

List<JsonAirport> airports = ResourceUtils

.loadDataFromJson(resourcePath, JsonAirport.class);

...

}

}

#### CityFileService

@Service("city-file")

public class CityFileService implements CityService {

@Value("${service.city.resource.path}")

protected String resourcePath;

@Override

public City getCityByAirport(String iataCode) {

List<JsonCity> cities = ResourceUtils.loadDataFromJson(resourcePath, JsonCity.class);

...

}

#### WeatherScraperService

@Service

public class WeatherScraperService implements WeatherService {

@Value("${service.weather.request.url.template}")

private String requestUrlTemplate;

@Override

public Weather getAirportWeather(String countryCode, String city) {

...

}

...

private String getRequestUrl(Airport airport) {

String requestUrl = requestUrlTemplate + airport.getCountryCode().toLowerCase();

if (airport.getRegion() != null) {

requestUrl = requestUrl + "/"

+ airport.getRegion().toLowerCase();

}

return requestUrl + "/" + airport.getCity().getName().toLowerCase();

}

}

## Kuriame lokalų miestų informacijos API

#### Sukuriame request URL template confige *application.properties*

server.port=8081

service.airport.resource.path=static/airports.json

service.city.resource.path=static/cities.json

service.city.request.url=http://localhost:8081/city

service.weather.request.url.template=https://www.wunderground.com/weather/{countryCode}/{city}

#### Kuriame atskirą servisą CityExternalService

@Service("city-external")

public class CityExternalService implements CityService {

private static final RestTemplate REST\_TEMPLATE = new RestTemplate();

@Value("${service.city.request.url}")

private String requestUrl;

@Override

public City getCityByAirport(String iataCode) {

UriComponentsBuilder builder = UriComponentsBuilder

.fromUriString(requestUrl)

.queryParam("iataCode", iataCode);

try {

JsonCity response = REST\_TEMPLATE.getForObject(builder.toUriString(), JsonCity.class);

if (response == null) {

return null;

}

City city = new City();

city.setName(response.getName());

return city;

} catch (Exception e) {

return null;

}

}

}

#### Pakeičiame TripServiceImpl taip, kad naudotų naujai sukurtą miestų servisą

@Service

public class TripServiceImpl implements TripService {

@Autowired

private AirportService airportService;

@Autowired

@Qualifier("city-external")

private CityService cityService;

@Autowired

@Qualifier("weather")

private WeatherService weatherService;

public Trip getTrip(String fromAirport, String toAirport) {

...

trip.setFromAirport(from);

trip.setToAirport(to);

if (to.getCity() != null) {

Weather airportWeather = weatherService.getAirportWeather(to);

trip.setWeather(airportWeather);

}

double distance = GeoUtils.distance(

from.getLatitude(), from.getLongitude(),

to.getLatitude(), to.getLongitude());

trip.setDistance(distance);

return trip;

}

}

#### Kuriame atskirą kontrolerį - imituojame išorinį servisą

@RestController

public class CityExternalMockController {

@Autowired

@Qualifier("city-file")

private CityService cityService;

@GetMapping("/city")

public ResponseEntity<JsonCity> getCityForAirport(@RequestParam("iataCode") String iataCode) {

final String finalIataCode = iataCode.trim().toUpperCase();

City city = cityService.getCityByAirport(finalIataCode);

Check.notNull(city, Message.MSG\_CITY\_FOR\_IATA\_NOT\_FOUND, iataCode);

JsonCity jsonCity = new JsonCity();

jsonCity.setName(city.getName());

jsonCity.setAirport(finalIataCode);

return new ResponseEntity<>(jsonCity, HttpStatus.OK);

}

}

#### Įdedame šio serviso klaidos žinutės tipą

public class Message {

...

public static final String MSG\_CITY\_FOR\_IATA\_NOT\_FOUND = "message.exception.notFound.cityForIata";

}

#### Rašome žinutės tekstą į *messages.properties*

...

message.exception.field.empty.toAirport=Destination airport cannot be empty.

message.exception.invalid.iata=Invalid IATA code {0}.

message.exception.notFound.cityForIata=City for IATA code {0} not found.

* 1. Funkcija tikrinanti ar grąžintas *CityExternalService* objektas (šiuo atveju *City*) yra validus

public class Check {

...

public static void notNull(Object obj, String message, Object... args) {

if (obj == null) {

throw new RestException(message, args);

}

}

}

## Handliname errorą, atsitinkantį tuomet, kai nurodomas tas pats oro uostas kaip kelionės pradžia ir pabaiga

#### Tikriname šią sąlygą *TripController*

@RestController

public class TripController {

@Autowired

private TripService tripService;

@GetMapping("/trip")

public ResponseEntity<Trip> tripInfo(@RequestParam(value="from") String fromAirport,

@RequestParam(value="to") String toAirport) {

...

Check.notEqual(fromAirport, toAirport, Message.MSG\_SAME\_IATAS);

return new ResponseEntity<>(tripService.getTrip(fromAirport, toAirport), HttpStatus.OK);

}

}

#### Įdedame funkcijos implementaciją

public class Check {

...

public static void iata(String iataCode, String message, Object... args) {

if (iataCode.length() != 3 || !iataCode.chars().allMatch(Character::isLetter)) {

throw new RestException(message, args);

}

}

public static void notEqual(Object obj1, Object obj2, String message, Object... args) {

if (obj1.equals(obj2)) {

throw new RestException(message, args);

}

}

}

#### Sukuriame error’o žinutės kintamąjį *Message* klasėj

...

public static final String MSG\_INVALID\_IATA = "message.exception.invalid.iata";

public static final String MSG\_SAME\_IATAS = "message.exception.same.iata";

}

#### Dedame žinutės turinį į configą *messages.properties*

...

message.exception.field.empty.toAirport=Destination airport cannot be empty.

message.exception.invalid.iata=Invalid IATA code {0}.

message.exception.same.iata=Airport codes must be different.

## Kuriame kitą web scrapinimo servisą - jis bus kviečiamas tuomet, kai susidursime su error response iš orų API

#### Extendiname esamą orų servisą:

@Service

@Qualifier("weather-extended")

public class ExtendedWeatherService extends WeatherScraperService {

private static final String QUERY\_PROPERTY\_PREFIX = "service.weather.nearbyCities.query.";

private static final RestTemplate REST\_TEMPLATE = new RestTemplate();

@Value("${service.weather.nearbyCities.request.url}")

private String requestUrl;

@Autowired

private Environment env;

@Override

public Weather getAirportWeather(Airport airport) {

try {

return parseWeatherData(airport);

} catch (IOException e) {

JsonNearbyLocation nearbyLocation = parseNearbyLocations(airport);

if (nearbyDataExist(nearbyLocation)) {

try {

return parseWeatherData(nearbyLocation, airport.getRegion()

);

} catch (IOException ex) {

// Ignore exception

}

}

}

return new Weather();

}

private JsonNearbyLocation parseNearbyLocations(Airport airport) {

return REST\_TEMPLATE.getForObject(getNearbyCityRequestUrl(airport.getLatitude(), airport.getLongitude()), JsonNearbyLocation.class);

}

private String getNearbyCityRequestUrl(Double latitude, Double longitude) {

return UriComponentsBuilder.fromUriString(requestUrl)

.queryParam("apiKey", env.getProperty(QUERY\_PROPERTY\_PREFIX + "apiKey"))

.queryParam("language", env.getProperty(QUERY\_PROPERTY\_PREFIX + "language"))

.queryParam("locationType", env.getProperty(QUERY\_PROPERTY\_PREFIX + "locationType"))

.queryParam("format", env.getProperty(QUERY\_PROPERTY\_PREFIX + "format"))

.queryParam("query", latitude + "," + longitude)

.toUriString();

}

private boolean nearbyDataExist(JsonNearbyLocation nearbyLocation) {

return nearbyLocation != null

&& nearbyLocation.getLocation() != null

&& nearbyLocation.getLocation().getCities().size() != 0

&& nearbyLocation.getLocation().getCountryCodes().size() != 0;

}

protected Weather parseWeatherData(JsonNearbyLocation nearbyLocation, String region) throws IOException {

Document document = getDocument(getRequestUrl(nearbyLocation, region));

Weather weather = new Weather();

weather.setTemperature(parseTemperature(document));

weather.setTemperatureUnit(parseTempUnit(document));

weather.setWindSpeed(parseWindSpeed(document));

weather.setWindSpeedUnit(parseWindSpeedUnit(document));

return weather;

}

private String getRequestUrl(JsonNearbyLocation nearbyLocation, String region) {

String requestUrl = requestUrlTemplate + nearbyLocation.getLocation().getCountryCodes().get(0);

if (region != null) {

requestUrl = requestUrl + "/" + region.toLowerCase();

}

return requestUrl + "/" + nearbyLocation.getLocation().getCities().get(0);

}

}

#### Atnaujiname *TripServiceImpl*, kad naudotų naują servisą *ExtendedWeatherService*

private CityService cityService;

@Autowired

@Qualifier("weather-extended")

private WeatherService weatherService;

public Trip getTrip(String fromAirport, String toAirport) {

#### Parefactoriname *WeatherScraperService*

@Service

@Qualifier("weather")

public class WeatherScraperService implements WeatherService {

@Value("${service.weather.request.url.template}")

protected String requestUrlTemplate;

@Override

public Weather getAirportWeather(Airport airport) {

try {

return parseWeatherData(airport);

} catch (IOException e) {

// Ignore exception

}

return new Weather();

}

protected Weather parseWeatherData(Airport airport) throws IOException {

Document document = getDocument(getRequestUrl(airport));

Weather weather = new Weather();

weather.setTemperature(parseTemperature(document));

weather.setTemperatureUnit(parseTempUnit(document));

weather.setWindSpeed(parseWindSpeed(document));

weather.setWindSpeedUnit(parseWindSpeedUnit(document));

return weather;

}

protected Double parseTemperature(Document document) {

Element temperature = document.select(".conditions-circle .wu-unit-temperature .wu-value.wu-value-to")

.first();

}

#### POJO nearby location išparsinti

@JsonInclude(JsonInclude.Include.NON\_NULL)

public class JsonNearbyLocation {

@JsonProperty("location")

private JsonNearbyCity location;

public JsonNearbyCity getLocation() {

return location;

}

public void setLocation(JsonNearbyCity location) {

this.location = location;

}

}

#### POJO nearby city išparsinti

@JsonInclude(JsonInclude.Include.NON\_NULL)

public class JsonNearbyCity {

@JsonProperty("countryCode")

private List<String> countryCodes;

@JsonProperty("city")

private List<String> cities;

public List<String> getCountryCodes() {

return countryCodes;

}

public void setCountryCodes(List<String> countryCodes) {

this.countryCodes = countryCodes;

}

public List<String> getCities() {

return cities;

}

public void setCities(List<String> cities) {

this.cities = cities;

}

}

#### Configo reikšmės, naudojamos *ExtendedWeatherService* klasėj (*application.properties*)

server.port=8081

service.airport.resource.path=static/airports.json

service.city.resource.path=static/cities.json

service.city.request.url=http://localhost:8081/city

service.weather.request.url.template=https://www.wunderground.com/weather/{countryCode}/{city}

service.weather.nearbyCities.request.url=https://api.weather.com/v3/location/search

service.weather.nearbyCities.query.apiKey=6532d6454b8aa370768e63d6ba5a832e

service.weather.nearbyCities.query.language=en-US

service.weather.nearbyCities.query.locationType=city

service.weather.nearbyCities.query.format=json

Jei jau spėjai padaryti visas aukščiau išvardintas užduotis, siūlome atlikti nors vieną bonus task’ą iš šių:

* Caching + expire time
* Create security layer
* Scrape more info
* Write tests
* Implement lazy loading
* Store search history to DB
* Logging
* Parallelize services
* Refactor code
* Integrate OpenTable API

<https://opentable.herokuapp.com/>

<https://opentable.herokuapp.com/api/restaurants?city=Chicago>

Kilus klausimams, prašome drąsiai kreiptis į mentorius :)