package cloud.nativ.flamewars;

import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

import javax.json.JsonObject;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.MediaType;

@RegisterRestClient

@Path("/data/2.5/weather")

public interface OpenWeatherMap {

@GET

@Consumes(MediaType.APPLICATION\_JSON)

JsonObject getWeather(@QueryParam("q") String city, @QueryParam("APPID") String appid);

}

package cloud.nativ.flamewars;

import org.eclipse.microprofile.config.inject.ConfigProperty;

import javax.enterprise.context.ApplicationScoped;

import javax.inject.Inject;

@ApplicationScoped

public class OpenWeatherMapConfiguration {

@Inject

@ConfigProperty(name = "weather.appid", defaultValue = "5b3f51e527ba4ee2ba87940ce9705cb5")

private String weatherAppId;

@Inject

@ConfigProperty(name = "weather.uri", defaultValue = "https://api.openweathermap.org")

private String weatherUri;

public String getWeatherAppId() {

return weatherAppId;

}

public String getWeatherUri() {

return weatherUri;

}

}

package cloud.nativ.flamewars;

import org.eclipse.microprofile.faulttolerance.Fallback;

import org.eclipse.microprofile.faulttolerance.Retry;

import org.eclipse.microprofile.faulttolerance.Timeout;

import org.eclipse.microprofile.rest.client.RestClientBuilder;

import javax.annotation.PostConstruct;

import javax.enterprise.context.ApplicationScoped;

import javax.inject.Inject;

import javax.json.Json;

import javax.json.JsonNumber;

import javax.json.JsonObject;

import javax.json.JsonPointer;

import javax.json.JsonString;

import java.net.URI;

import java.net.URISyntaxException;

import java.time.temporal.ChronoUnit;

@ApplicationScoped

public class OpenWeatherMapConnector {

private static final double KELVIN\_2\_CELSIUS = 273.15;

@Inject

private OpenWeatherMapConfiguration configuration;

private OpenWeatherMap openWeatherMap;

@PostConstruct

void initialize() {

try {

openWeatherMap = RestClientBuilder.newBuilder()

.baseUri(new URI(configuration.getWeatherUri()))

.build(OpenWeatherMap.class);

} catch (URISyntaxException e) {

throw new IllegalArgumentException(e);

}

}

@Timeout(value = 5L, unit = ChronoUnit.SECONDS)

@Retry(delay = 500L, maxRetries = 1)

@Fallback(fallbackMethod = "defaultWeather")

public PayaraMicroWeather getWeather(String city) {

JsonObject response = openWeatherMap.getWeather(city, configuration.getWeatherAppId());

JsonPointer pointer;

pointer = Json.createPointer("/weather/0/main");

String weather = ((JsonString) pointer.getValue(response)).getString();

pointer = Json.createPointer("/main/temp");

double temperature = ((JsonNumber) pointer.getValue(response)).doubleValue() - KELVIN\_2\_CELSIUS;

return new PayaraMicroWeather(city, weather, temperature);

}

public PayaraMicroWeather defaultWeather(String city) {

return new PayaraMicroWeather(city, "Unknown", 0.0);

}

}

package cloud.nativ.flamewars;

import javax.json.bind.annotation.JsonbTransient;

import javax.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "current\_weather")

@NamedQuery(name = "findCurrentWeatherByCity", query = "SELECT w FROM PayaraMicroWeather w WHERE w.city = :city AND w.nextUpdate > :now")

public class PayaraMicroWeather {

PayaraMicroWeather() {

}

public PayaraMicroWeather(final String city, final String weather, final double temperature) {

this.city = city;

this.weather = weather;

this.temperature = temperature;

}

@Id

@Column(name = "city", unique = true, nullable = false)

private String city;

@Column(name = "weather", nullable = false)

private String weather;

@Column(name = "temperature", nullable = false)

private double temperature;

@Column(name = "next\_update", columnDefinition = "TIMESTAMP")

@JsonbTransient

private LocalDateTime nextUpdate;

public String getCity() {

return city;

}

public String getWeather() {

return weather;

}

public double getTemperature() {

return temperature;

}

public void setTemperature(double temperature) {

this.temperature = temperature;

}

public LocalDateTime getNextUpdate() {

return nextUpdate;

}

public PayaraMicroWeather touch(int offset) {

this.nextUpdate = LocalDateTime.now().plusHours(offset);

return this;

}

public void setCity(String city) {

this.city = city;

}

public void setWeather(String weather) {

this.weather = weather;

}

}

package cloud.nativ.flamewars;

import javax.ws.rs.ApplicationPath;

import javax.ws.rs.core.Application;

@ApplicationPath("/api")

public class PayaraMicroWeatherApplication extends Application {

}

package cloud.nativ.flamewars;

import org.eclipse.microprofile.health.HealthCheck;

import org.eclipse.microprofile.health.HealthCheckResponse;

import org.eclipse.microprofile.health.Liveness;

import javax.enterprise.context.ApplicationScoped;

@Liveness

@ApplicationScoped

public class PayaraMicroWeatherLivenessCheck implements HealthCheck {

@Override

public HealthCheckResponse call() {

return HealthCheckResponse.named(PayaraMicroWeatherLivenessCheck.class.getSimpleName()).withData("live", true).up().build();

}

}

package cloud.nativ.flamewars;

import org.eclipse.microprofile.health.HealthCheck;

import org.eclipse.microprofile.health.HealthCheckResponse;

import org.eclipse.microprofile.health.Readiness;

import javax.enterprise.context.ApplicationScoped;

@Readiness

@ApplicationScoped

public class PayaraMicroWeatherReadinessCheck implements HealthCheck {

@Override

public HealthCheckResponse call() {

return HealthCheckResponse.named(PayaraMicroWeatherReadinessCheck.class.getSimpleName()).withData("ready", true).up().build();

}

}

package cloud.nativ.flamewars;

import javax.ejb.Stateless;

import javax.ejb.TransactionAttribute;

import javax.ejb.TransactionAttributeType;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.TypedQuery;

import javax.transaction.Transactional;

import java.time.LocalDateTime;

import java.util.List;

import java.util.Optional;

@Stateless

@TransactionAttribute(TransactionAttributeType.REQUIRED)

@Transactional

public class PayaraMicroWeatherRepository {

@PersistenceContext

private EntityManager em;

public PayaraMicroWeather save(PayaraMicroWeather currentWeather) {

return em.merge(currentWeather);

}

public Optional<PayaraMicroWeather> findCurrentWeatherByCity(String city) {

TypedQuery<PayaraMicroWeather> query = em.createNamedQuery("findCurrentWeatherByCity", PayaraMicroWeather.class);

query.setParameter("city", city);

query.setParameter("now", LocalDateTime.now());

List<PayaraMicroWeather> results = query.getResultList();

if (results.isEmpty()) {

return Optional.empty();

} else {

return Optional.of(results.get(0));

}

}

}

package cloud.nativ.flamewars;

import org.eclipse.microprofile.metrics.annotation.Timed;

import javax.enterprise.context.ApplicationScoped;

import javax.inject.Inject;

import javax.validation.constraints.NotBlank;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

@Path("/weather")

@ApplicationScoped

public class PayaraMicroWeatherResource {

@Inject

private PayaraMicroWeatherService service;

@GET

@Produces(MediaType.APPLICATION\_JSON)

@Timed(name = "getWeather")

public Response getWeather(@QueryParam("city") @NotBlank String city) {

return Response.ok(service.getWeatherForCity(city)).build();

}

}

package cloud.nativ.flamewars;

import org.eclipse.microprofile.config.inject.ConfigProperty;

import javax.enterprise.context.ApplicationScoped;

import javax.inject.Inject;

import java.util.Optional;

@ApplicationScoped

public class PayaraMicroWeatherService {

@Inject

private PayaraMicroWeatherRepository repository;

@Inject

@ConfigProperty(name = "next.update.offset", defaultValue = "1")

private int nextUpdateOffset;

@Inject

private OpenWeatherMapConnector connector;

public PayaraMicroWeather getWeatherForCity(String city) {

Optional<PayaraMicroWeather> currentWeather = repository.findCurrentWeatherByCity(city);

return currentWeather.orElseGet(() -> {

PayaraMicroWeather current = connector.getWeather(city);

return repository.save(current.touch(nextUpdateOffset));

});

}

}