

List of contents

1. [Making REST API Requests](#)
2. [DOCKER](#)
3. [gRPC](#)

REST API

Making REST API Requests

With *WebClient* object:

GET

```
public void getAllPersons(CustomerSortingOptions sortingOptions, Consumer<List<Customer>> customerConsumer) {
    webClient.get()
        .uri(uriBuilder -> uriBuilder
            .path("customers")
            .queryParams("sortField", sortingOptions.getSortField())
            .queryParams("sortingOrder", sortingOptions.getSortingOrder())
            .build())
        .retrieve()
        .bodyToMono(new ParameterizedTypeReference<List<Customer>>() {
        })
        .onErrorStop()
        .subscribe(newPersons -> {
            customers.clear();
            customers.addAll(newPersons);
            customerConsumer.accept(customers);
        });
}
```

POST

```
public void addPerson(Customer customer, Consumer<List<Customer>> customerConsumer) {
    // TODO Part 2: Complete the http post request to the server
    webClient.post()
        .uri("customers")
        .bodyValue(customer)
        .retrieve()
        .bodyToMono(Customer.class)
        .onErrorStop()
        .subscribe(newCustomer -> {
            customers.add(newCustomer);
            customerConsumer.accept(customers);
        });
}
```

PUT

```
public void updatePerson(Customer customer, Consumer<List<Customer>> customerConsumer) {
    // TODO Part 2: Complete the http put request to the server
    webClient.put()
        .uri("customers/" + customer.getId())
        .bodyValue(customer)
        .retrieve()
        .bodyToMono(Customer.class)
        .onErrorStop()
        .subscribe(newCustomer -> {
            customers.replaceAll(x -> x.getId().equals(customer.getId()) ? newCustomer : x);
            customerConsumer.accept(customers);
        });
}
```

DELETE

```
public void deletePerson(Customer customer, Consumer<List<Customer>> customerConsumer) {
    // TODO Part 2: Make an http delete request to the server
    webClient.delete()
        .uri("customers/" + customer.getId())
        .retrieve()
        .toBodilessEntity()
        .onErrorStop()
        .subscribe(deletedCustomer -> {
            customers.remove(customer);
            customerConsumer.accept(customers);
        });
}
```

Just in case: webClient Constructor

```
public CustomerController() {
    this.webClient = WebClient.builder()
        .baseUrl("http://localhost:8080/")
        .defaultHeader(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
        .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
        .build();
    this.customers = new ArrayList<>();
}
```

With restTemplate:

```
@GetMapping("/getTimeLine/{id}")
public List<Tweet> getTimeLine(@PathVariable("id") int userID) {
    headers.setContentType(MediaType.APPLICATION_JSON);

    ResponseEntity<List<Tweet>> response = restTemplate.exchange(
        getActivityURL + userID, // Set the uri
        HttpMethod.GET, // HTTP Method to use
        null, // requestEntity = null as there is no body
        new ParameterizedTypeReference<List<Tweet>>(){} // response body type
    );
    return response.getBody();
}
```

REST API Endpoints

Example Java class:

```
@RestController
@RequestMapping("/activity")
public class ActivityController {
    private Map<Integer, List<Tweet>> userActivityMap = new HashMap<>();
    private Map<Integer, List<User>> userFollowedMap = new HashMap<>();

    @GetMapping("/getActivity/{id}")
    public List<Tweet> getActivity(@PathVariable("id") int userID) {
        List<Tweet> userTweets = userActivityMap.get(userID);

        if(userTweets == null) return new ArrayList<>();

        return userTweets;
    }

    @PostMapping("/addActivity")
    public void addActivity(@RequestBody Tweet tweet) {
        userActivityMap.computeIfAbsent(tweet.getUser().getUserID(), x -> new LinkedList<>());
        userActivityMap.get(tweet.getUser().getUserID()).add(tweet);
    }

    @PostMapping("/addFollower")
    public void addFollower(@RequestBody FollowRequest followRequest) {
        User follower = followRequest.getFollower();
        User followed = followRequest.getFollowed();

        userFollowedMap.computeIfAbsent(follower.getUserID(), x -> new LinkedList<>())
            .add(followed);
    }

    @DeleteMapping("/deleteActivity")
    public void deleteActivity(@RequestBody Tweet tweet) {
        List<Tweet> userTweets = userActivityMap.get(tweet.getUser().getUserID());
        if(userTweets == null) return;

        userTweets.remove(tweet);
    }

    @DeleteMapping("/deleteFollower")
    public void deleteFollower(@RequestBody FollowRequest followRequest) {
        User follower = followRequest.getFollower();
        User followed = followRequest.getFollowed();

        List<User> followedUsers = userFollowedMap.get(follower.getUserID());
        if(followedUsers == null) return;

        followedUsers.remove(followed);
    }

    @GetMapping("/getFollowedList/{id}")
    public List<User> getFollowedList(@PathVariable("id") int userID) {
        // TODO: this method is also not complete, so you have to implement it accordingly :)
        List<User> followed = userFollowedMap.get(userID);
        if(followed == null) return new ArrayList<>();
        return followed;
    }
}
```

DOCKER

Docker reference: <https://docs.docker.com/>

Dockerfiles

<https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/>

<https://docs.docker.com/reference/dockerfile/>

```
FROM python:3.13
WORKDIR /usr/local/app

# Install the application dependencies
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy in the source code
COPY src ./src
EXPOSE 8080

# Setup an app user so the container doesn't run as the root user
RUN useradd app
USER app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Building docker

```
docker build -t <tag>
```

Docker processes

```
docker ps
```

Exercises

- [L09P01 - Registry Office Service Deployment](#)
- [L09P02 - Garching Best Software](#)
- [L08PE04 - Spaceship](#)

gRPC

gRPC reference: <https://grpc.io/docs/>

gRPC Java basics: <https://grpc.io/docs/languages/java/basics/>

Defining classes:

Important tips:

- Array in protobuf means `repeated` field

```
message Date {  
    int32 day = 1; // Number specifies order  
    int32 month = 2;  
    int32 year = 3;  
}  
  
message Location {  
    string city = 1;  
    string country = 2;  
}  
  
message LocationDate {  
    Location location = 1;  
    Date date = 2;  
}
```

Defining services:

```
service WeatherReporter {  
  
    //Returning one object  
    rpc GetCityWeatherSingleDay(LocationDate) returns (CityWeatherData) {}  
  
    // Returning a stream of objects  
    rpc GetCityWeatherMultipleDays(LocationDatePeriod) returns (stream CityWeatherData) {}  
}
```

Example of services and objects with repeated fields

This will be utilized by some of the client side examples underneath

```
service MetadataService {
  rpc getTrackMetadata(EntitiesRequest) returns (Tracks) {}

  rpc getAlbumMetadata(EntitiesRequest) returns (Albums) {}
}

message EntitiesRequest {
  repeated uint32 id = 1;
}

message Track {
  uint64 id = 1;
  string name = 2;
  string album = 3;
  string artist = 4;
  uint32 duration = 5;
}

message Tracks {
  repeated Track track = 1;
}

message Album {
  uint64 id = 1;
  string name = 2;
  string artist = 3;
  repeated uint32 tracks = 4;
}

message Albums {
  repeated Album album = 1;
}
```

Generating classes from *.proto* files

1. Gradle tab in IDE and press "Tasks > other > generateProto")
2. The generated classes from the proto file should be in build/generated/source/proto/main folder

Server side

In practice can be done with streams or with simple iteration

```
public void getCityWeatherSingleDay(LocationDate request, StreamObserver<CityWeatherData> responseObserver) {
    var data = allWeatherData.stream().filter(cityWeatherData ->
cityWeatherData.getLocationDate().equals(request)).findFirst().orElse(null);
    if(data != null)responseObserver.onNext(data);
    responseObserver.onCompleted();
}
```

For returning a *stream* of data just call onNext multiple times

With iteration:

```
public void getCityWeatherMultipleDays(LocationDatePeriod request, StreamObserver<CityWeatherData> responseObserver) {
    for(CityWeatherData data : allWeatherData) {
        if(data.getLocationDate().getLocation().equals(request.getLocation())) {
            if(compareDate(data.getLocationDate().getDate(), request.getStartDate()) >= 0 &&
                compareDate(data.getLocationDate().getDate(), request.getEndDate()) <= 0)
            {
                responseObserver.onNext(data);
            }
        }
    }
    responseObserver.onCompleted();
}
```

With streams:

```
public void getCityWeatherMultipleDays(LocationDatePeriod request, StreamObserver<CityWeatherData> responseObserver) {
    var startDate = request.getStartDate();
    var endDate = request.getEndDate();
    allWeatherData.stream()
        .filter(cityWeatherData -> cityWeatherData.getLocationDate().getLocation().equals(request.getLocation()))
        .filter(cityWeatherData -> compareDate(cityWeatherData.getLocationDate().getDate(), startDate) >= 0)
        .filter(cityWeatherData -> compareDate(cityWeatherData.getLocationDate().getDate(), endDate) <= 0)
        .forEach(responseObserver::onNext);
    responseObserver.onCompleted();
}
```

For returning an object which contains array of other objects (stupid???)

```
public void getAlbumMetadata(EntitiesRequest request, StreamObserver<Albums> responseObserver) {
    Albums.Builder albumsBuilder = Albums.newBuilder();

    for(int i = 0; i < request.getIdCount(); i++) {
        // Find the album by its id
        Album album = database.findAlbumById(request.getId(i));

        if(album != null) {
            albumsBuilder.addAlbum(album);
        }
    }

    // In case that an album isn't found builder builds empty list
    Albums albums = albumsBuilder.build();
    responseObserver.onNext(albums);
    responseObserver.onCompleted();
}
```

Client Side

Setup

The **blocking stub** is a placeholder on which the functions are called. It needs to be received from the gRPC object in some way.

```
public WeatherClient(Channel channel) {
    // Get blocking stub from the gRPC object
    blockingStub = WeatherReporterGrpc.newBlockingStub(channel);
}
```

or

```
public SpotifyClient(Channel channel) {  
    // spotifyConnection is a wrapper defined in the exercise  
    // this is just an example that there can be other ways to get the stub  
    this.spotifyConnection = new SpotifyConnection(channel);  
    this.blockingStub = spotifyConnection.getStub();  
}
```

Calling method with *single object* return value

```
public void getCityWeatherSingleDay(int day, int month, int year, String city, String country) {  
    // Need to use builders to create the objects for the request  
    Date date = Date.newBuilder().setDay(day).setMonth(month).setYear(year).build();  
    Location location = Location.newBuilder().setCity(city).setCountry(country).build();  
  
    // Building the locationDate which will be passed as a parameter  
    LocationDate locationDate = LocationDate.newBuilder().setDate(date).setLocation(location).build();  
  
    // Getting the returned object from calling the method on the stub  
    CityWeatherData cityWeatherData = blockingStub.getCityWeatherSingleDay(locationDate);  
  
    System.out.println(cityWeatherData.toString());  
}
```

Calling method which returns *stream*

```
public void getCityWeatherMultipleDays(int startDay, int startMonth, int startYear, int endDay, int endMonth, int endYear, String  
city, String country) {  
    // Building objects for the request  
    Date startDate = Date.newBuilder()  
        .setDay(startDay)  
        .setMonth(startMonth)  
        .setYear(startYear)  
        .build();  
    Date endDate = Date.newBuilder()  
        .setDay(endDay)  
        .setMonth(endMonth)  
        .setYear(endYear)  
        .build();  
    Location location = Location.newBuilder().setCity(city).setCountry(country).build();  
  
    // Building LocationDatePeriod object to be passed as parameter  
    LocationDatePeriod locationDatePeriod = LocationDatePeriod.newBuilder()  
        .setStartDate(startDate)  
        .setEndDate(endDate)  
        .setLocation(location)  
        .build();  
  
    // Method call returns Iterator<CityWeatherData> for the stream of CityWeatherData  
    Iterator<CityWeatherData> iterator = blockingStub.getCityWeatherMultipleDays(locationDatePeriod);  
  
    // Iterating over the objects  
    while (iterator.hasNext()) {  
        CityWeatherData cityWeatherData = iterator.next();  
        System.out.println(cityWeatherData.toString());  
    }  
}
```

Calling method which has *parameter with repeated field*

```
public String displayPlaylist(int[] playlist) throws SpotifyClientException {
    StringBuilder stringBuilder = new StringBuilder();

    EntitiesRequest.Builder builder = EntitiesRequest.newBuilder();

    // IMPORTANT for REPEATED fields
    // Add all id's to request
    for(int id : playlist) {
        builder.addId(id);
    }

    EntitiesRequest request = builder.build();

    // Unlike in earlier examplese returns OBJECT which contains multiple Tracks
    Tracks tracks;

    tracks = blockingStub.getTrackMetadata(request);
    if(tracks.getTrackCount() != playlist.length) throw new SpotifyClientException("Failed to get track metadata");

    for(int i = 0; i < tracks.getTrackCount();i++) {
        Track track = tracks.getTrack(i);

        int durationInSeconds = track.getDuration();
        int durationInMinutes = durationInSeconds / 60;
        durationInSeconds = durationInSeconds % 60;

        stringBuilder.append(String.format("%d. %s - %s (%d:%02d)%n", i+1, track.getName(), track.getArtist(), durationInMinutes,
        durationInSeconds));
    }

    return stringBuilder.toString();
}
```

ALL RESOURCE LINKS:

- Lecture slides (PDF)
- Your own homework solutions (cloned)
- Cheat-sheets/self-written notes (printed/on paper/PDF)
- C reference: <https://en.cppreference.com/w/c/language>
- Python reference: <https://docs.python.org/3/reference/index.html>
- Java reference: <https://www.baeldung.com/get-started-with-java-series>
- Java reference: <https://docs.oracle.com/en/java/javase/22/>
- gRPC reference: <https://grpc.io/docs/>
- Spotbugs: <https://spotbugs.readthedocs.io>
- AddressSanitizer: <https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html>
- Spring Boot: <https://www.baeldung.com/spring-boot>
- Spring Boot: <https://spring.io/projects/spring-boot>
- Docker: <https://docs.docker.com/>
- JavaFX: <https://openjfx.io/openjfx-docs>
- Gradle: <https://docs.gradle.org/current/userguide/userguide.html>
- Git: <https://git-scm.com/docs>
- JUnit: <https://junit.org/junit5/docs/current/user-guide/>
- Mockito: <https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html>
- EasyMock: <https://easymock.org/user-guide.html>
- man pages (command line, not online)

