

# Distributed Systems Programming

A.Y. 2020/21

## Laboratory 2

In this laboratory activity, you are invited to practice with gRPC, a modern open source high performance framework implementing the remote procedure call (*RPC*) paradigm.

The activity is comprehensive of the following tasks:

- definition of a .proto file (Protocol Buffer file), describing the data structures (called messages) and services involved in the communication;
- implementation of a gRPC client (in node.js) and its integration within the *ToDoManager* service developed in Laboratory 1;
- implementation of a gRPC server (in Java).

The tools that are recommended for the development of the solution are:

- *Visual Studio Code* (<https://code.visualstudio.com/>) for the extension of the *ToDoManager* service by implementing a gRPC client;
- *vscode-proto3*, extension of Visual Studio Code, (<https://marketplace.visualstudio.com/items?itemName=zxh404.vscode-proto3>), for the definition of Protocol Buffer files;
- *Eclipse IDE for Enterprise Java Developers* for the development of the gRPC server;
- *PostMan* (<https://www.postman.com/>) for testing the extended version of the *ToDoManager* service;
- *DB Browser for SQLite* (<https://sqlitebrowser.org/>) for the management of the database for the *ToDoManager* service.

Even though completing all these tasks by yourself would represent the best way to benefit from this laboratory session, this is not required, because of the limited time available. In the reminder of this document, after a description of the context where this activity is carried out, additional information is provided about how you can experience this activity in the way that is most suitable for you.

## Context of the activity

The *ToDoManager* service is extended with a new functionality with respect to the service version developed for Laboratory 1, i.e., it offers the possibility to associate a set of *images* to each task. Each image can have one of the following three image media types: PNG (.png), JPEG (.jpg), GIF (.gif). No other image media types are allowed by the service.

The *ToDoManager* service exposes three new REST APIs, related to the association of images to the tasks, for the authenticated users:

- A user can associate an image to a task, by uploading the image to the *ToDoManager* service. Multiple images can be associated with a single task. When the service receives an image with an accepted image media type, it locally stores the image with its media type, and saves the related information (i.e., name of the image, and association of the image to the task) in the database;
- A user can retrieve an image associated to a task. In this operation, the value of the *Accept* header of the HTTP request can be used to specify the requested media type for the image. Three media types are supported: image/png, image/jpg, and image/gif. In case the user requests another media type, the operation fails;
- A user can delete an image associated to a task.

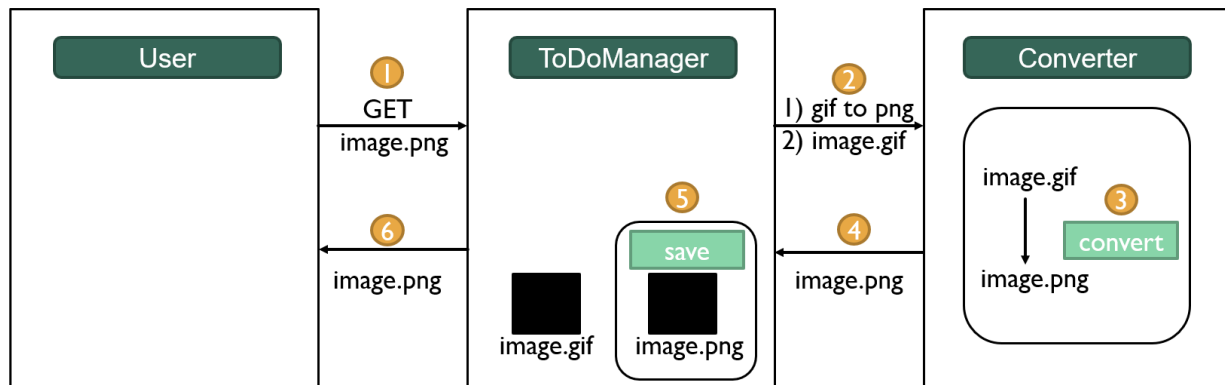
When a user requests an image with a certain media type and the *ToDoManager* service has that image already locally saved with that media type, it immediately sends back the image to the user.

Instead, if the image is locally saved with another media type, then the *ToDoManager* service interacts with another service, called *Converter* service, delegating the operation of media type conversion. In greater detail, the *ToDoManager* service creates a gRPC channel towards the *Converter* service (i.e., the *ToDoManager* service covers the role of gRPC client, the *Converter* service covers the role of gRPC server). Their communication is organized in the following way:

- the *ToDoManager* service sends a first messages describing the media type of the image to be converted, and the destination media type. Then, it sends the image, divided into chunks.
- the *Converter* service converts the received image into the requested media type, and sends it back to the *ToDoManager* service, if no problem arises in the conversion. Otherwise, it notifies why the conversion failed. The *Converter* service does not save either the received image or the result or the conversion.

After the *ToDoManager* receives the chunks of the converted image and locally saves the file with this new media type, it can finally send it back to the user who requested it.

The following picture illustrates an example of a client requesting an image (e.g., image.png) whose media type is not available server-side (e.g., only image.gif is available):



## How to experience this laboratory activity

The full development of the extension involves the following tasks:

1. definition of the new REST APIs exposed by the *ToDoManager* service;
2. definition of the .proto file on which the gRPC service is based;
3. extension of the node.js *ToDoManager* service implementation, for the management of images (REST APIs implementation) and of the gRPC communication;
4. implementation of the gRPC *Converter* service in Java;

However, alongside this document, we provide you with:

- a .proto file specifying a possible organization of the messages and services involved in the gRPC *Converter* service;
- a Java-based gRPC server stub for the *Converter* service, derived from the previous .proto file, where only the method in charge of file conversion and gRPC communication must be completed;
- a full implementation of the Java-based gRPC server stub for the *Converter* service.

Hence, the Lab work consists of developing steps 1. and 3. However, if you would like to have a more extended development experience, you can also develop (part of) the other steps by yourself. Even if you do not develop the *Converter* server implementation by

yourself, you are strongly invited to have a careful look at it, and to understand it. The reason is that in the final exam you may be asked to extend it.

## Useful tips

Here are some recommendations provided with the aim to help you in completing the tasks required by this Lab session activity:

- you can use the node.js *multer* (<https://www.npmjs.com/package/multer>) module to handle the reception and saving of images in the *ToDoManager* service. This module provides a middleware for handling *multipart/form-data*, which is the principal way used for uploading files;
- you can use the node.js *grpc* module (<https://www.npmjs.com/package/grpc>) for implementing the functionality of gRPC client in the *ToDoManager* service;
- you can use the node.js *@grpc/proto-loader* module (<https://www.npmjs.com/package/@grpc/proto-loader>) for loading .proto files to use with gRPC;
- you can use the Java *imageIO* package (<https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html>) for the image conversion.

Here is some useful information related to the Java gRPC server:

- The .proto files must be put in /src/main/proto folder of the Maven project.
- In order to automatically generate the required classes from the .proto file, in Eclipse you must synchronize the configuration of the project with the setting of the pom.xml. To do so, follow these instructions: Right click on the Eclipse project name → Maven → Update Project... → OK.
- If you decide to develop your own server, you are invited to use the pom.xml file you can find together with the provided complete implementation for your own Maven project. This pom.xml file includes all the dependencies that are needed to use gRPC in Java.
- The compliance of the compiler must be set to 1.6 (or higher). The reason is that the classes that are automatically generated from .proto files implement interface methods, which in Java 1.6 can be annotated with `@Override` (in Java 1.5, instead, the `@Override` annotation can only be applied to methods overriding a superclass method). The pom.xml file you can find together with the provided implementation of the server already includes the definition of the necessary properties to set the correct compliance level of the compiler.