Distributed Systems (H0N08A)

Report: Remote communication

Jules Verbessem *(r0957436)*
Gert-Jan Gillis *(r0674083)*

October 18, 2023

What is the role of stub and skeleton in Java RMI? Does REST have similar concepts? Stub in Java RMI wordt aangesproken als de client een call doet naar het remote object. Dan wordt de call ontvangen door de Stub en deze geeft die door. De stub is een object dat je in de front-end gebruikt om de methodes te invoken van het skeleton object in de back-end. Je bind de stub in het registry aan een bepaalde naam als een key-value pair. Skeleton in Java RMI vertaalt remote request in methode calls. Het skeleton object is deel van het class diagram van de backend. Er is geen stub in REST dat het skeleton object moet voorstellen. Maar er is wel een soortgelijk object dat requests fetcht van de backend. Er is echter wel een soortgelijk skeleton object. Het skeleton object heeft zowel de functionaliteit van de REST Controller als de Business Service. Het skeleton object zal remote calls krijgen zoals de REST Controller en ook deze afhandelen zoals de business service.

What is the difference between serializable and remote? In your Java RMI assignment, which classes were made serializable and which classes were made remote? Motivate your choice. De serializable klassen zijn de klassen die gebruikt worden als DTO (Data Transfer Object). Deze worden doorgegeven van de back-end naar de front-end en terug. Bv in de assignment "BookingDetail". De Remote interface wordt overgeërfd door een zelf gemaakte interface. Dan zal uw skeleton deze interface implementeren. Bv in de assignment "BookingManager". Dit object wordt gebruikt om remote requests af te handelen.

What role does the RMI registry play? Why is there no registry in REST? De registry wordt gebruikt om remote objecten op te slaan onder een naam als een key-value pair. Zo kan de front-end aan het remote object. REST heeft geen registry omdat deze niet rechtstreeks aan het remote object moet kunnen. Hier wordt het HTTP protocol gebruikt om data door te geven. Als je de Java RMI registry wilt gebruiken moeten de 2 systemen die met elkaar communiceren allebei Java gebruiken. Bij REST is dit niet het geval.

How do you make sure that your Java RMI implementation is thread-safe? Moreover, which methods were required and selected to be thread-safe? Can you motivate your decisions? Thread-safe code manipuleert alleen gedeelde data. Het serializable is niet gedeeld in memory tussen de 2 systemen omdat de 2 systemen in praktijk op verschillende machines draaien. De "BookingManager" (het skeleton object) moet een interface implementeren. Deze interface laat toe om met threads te werken. Iedere methoden van de "BookingManager" moet dan per call op een eigen thread werken. Zo is er geen interference tussen de methoden.

Level 3 RESTful APIs are hypermedia-driven. How does this affect the evolution of your software, and more specifically your APIs in terms of coupling and future upgrades? Bij level 3 REST API zal in de response staan welke end-point je moet gebruiken voor de volgende actie. Bij alle lagere levels moet je de end-point weten en manueel coderen. Maar bij level 3 geeft de back-end aan welk de end-point zijn voor deze acties. Nu moet je bij de client

zeggen "gebruik de end-point gekregen van de back-end" voor de actie uit te voeren. Dit zorgt ervoor dat als er besloten wordt om de end-points in de backend te wijzigen dat dit in de front-end ook automatisch gebeurt. Dit maakt toekomstige upgrades makkelijker.

During the REST session, you used a code generator to generate the server side of the application. What other components could you generate from the OpenAPI specification? In de openapi.yaml staat alles wat er automatisch gegenereerd moet worden. Hier zien we dat er 2 schema's gegenereerd worden: meal en mealupdaterequest. Mealupdaterequest is het zelfde als meal maar zonder id. Dit is nodig voor de put request omdat de id in de link wordt meegegeven. Deze klassen worden geïnjecteerd in de controller en repository klassen. Ook zal het API endpoints genereren om met de controller te werken. Als laatste genereert het een UI om met de API's te kunnen werken.

Your OpenAPI specification indicates that "Name" is a mandatory field for a meal. Your client received the specification, can you be sure "Name" will always be present when you receive a request from that client? When you generate server code from this specification, will your application check if "Name" is always present? Als er een Meal wordt toegevoegd zonder het veld "name" dan wordt een nieuw Meal aangemaakt waarin het veld "name" op null wordt gezet. Name zal dus altijd aanwezig zijn, maar niet per definitie data bevatten. Je krijgt dus geen foutmelding als het mandatory field niet wordt meegegeven.

What is the advantage of using code generation, e.g. using OpenAPI, over a language-integrated solution such as Java RMI? What are the downsides of using code generation from an implementation perspective? Je moet minder code zelf schrijven bij code generation. Dit kan ook in verschillende talen. Het nadeel van code generation is dat je het systeem moet specificieren zoals de componenten, path en de securityschemes voordat je alles kan testen. Het laten genereren zorgt ervoor dat er veel complexe processen automatisch gebeuren die niet altijd nodig zijn. Dit maakt het voor beginnende developers ook moeilijker om te begrijpen.

How do Java RMI and a RESTful service compare in terms of flexibility (e.g., at platform level), extensibility (in particular when working with third parties), and susceptibility to protocol errors? REST is meer flexibel dan Java RMI want bij REST kunnen de verschillende systemen die communiceren op verschillende platformen draaien. Terwijl bij Java RMI moeten de systemen allebei Java gebruiken. REST is ook meer uitbereidbaar. Als "third parties" een ander platform gebruiken kan je via REST er toch met communiceren. Bij Java RMI gaat dit niet. REST is meer vatbaar voor protocol errors. Omdat als het ene systeem met HTTP1.0 werkt en het andere met HTTP1.1, zal dit fouten geven als ze dit niet onderhandelen. Dit probleem bestaat niet bij Java RMI.

Suppose that you are tasked with developing the following applications, which of the two remote communication technologies (Java RMI, REST) would you use to realize them? What is your motivation for choosing a specific technology? If you choose REST, is it beneficial to also use OpenAPI?

A public web API that can be used by applications all over the world. Hier is het gebruik van REST de enige optie. Niet heel de wereld gebruikt Java. Maar met REST kan heel de wereld je public web API gebruiken. Ook is het handig als je een public OpenAPI hebt open staan. Dit zorgt ervoor dat developers je API kunnen gebruiken op een gebruiksvriendelijke manier.

The internal communication of a high-performance SaaS application that consists of multiple distributed components, written in various programming languages Omdat er verschillende programmeertalen zijn in de verdeelde componenten moet je REST gebruiken. Het opzetten van OpenAPI is enkel handig voor testen van de API maar niet een must.

An internal distributed application of a large company, written in Java only. Hier kan je kiezen voor REST en Java RMI. Omdat zowel client en server side Java gebruiken is het makkelijker om in Java te blijven werken. En geen hele REST API op te stellen.