Problem

Monday, 2 October 2023 22:10

Before you start thinking about the structure of your programs, the frameworks, languages and paradigms is to understand what problem it is that your project is trying to solve

Talk about a problem you had and how you solved that with code - relatable to other people

Then ask the big questions:

- What tech stack am I gonna use
- What database
- What backend structure am I gonna implement
- What is my frontend going to look like
- What are the main components that my project is going to need

"Often when you're building a web application or an app of any sort you're going to have to do some type of long running process - the idea behind this app is to have an AI generated starting plan which you can follow "

Coder's bucket list

Tuesday, 26 September 2023 22:06

Main idea overview:

- Develop a series of services that enable a fellow programmer to store spontaneous coding ideas, assign them a category and then retrieve one at random from a particular category when kickstarting a new personal project

Main Services:

Project Service= to create and view project categories (acts like a project catalogue)

Fetch service= can fetch/retrieve a project idea under a certain category

Inventory Service= can check if project from a particular category is available or not

Notification Service= stateless service with no database that sends notifications to users

Whenever a "Retrieve" command is placed, the Fetch Service is going to synchronously communicate with the Inventory Service to check whether a project idea from the category requested is available or not

After the inventory has been checked the Fetch Service is going to asynchronously communicate with the Notification Service to deliver the project skeleton to the coder

Solution Architecture

Tuesday, 26 September 2023 22:38

Project Service talks to MongoDB
Fetch Service talks to MySQL
Inventory Service talks to MySQL

Model View Controller

Tuesday, 26 September 2023 23:08

MVC DESIGN PATTERN

- It helps organising the code in a scalable and maintainable way

REPOSITORY

- Role= encapsulates the storage, retrieval and search behaviour which emulates a collection of objects
- Responsibilities= communicates with the database and performs CRUD operations, acts as middleman between database and service layer
- Class annotation (Spring): @Repository

SERVICE

- Role= hosts the business logic
- Responsibilities= Controls the transaction of data between repo and controller, communicates with other services and repositories
- Class annotation (Spring): @Service

CONTROLLER:

- Controller= handles incoming HTTP requests, invoking appropriate action and returning the HTTP response
- Responsibilities= Interacts with service layer to perform the operations and send back the response, delegates the business logic responsibility to service layer
- I.e. intercepts incoming requests, modifies them to the internal structure of the data, sends the data to the Service layer for further processing
- Class annotation (Spring): @RestController

DTO (Data Transfer Object):

- Role: Object that caries data between processes - passes data form the client to the server and vice versa
- Responsibility: Holds the data, does not contain any business logic- it is used for transferring data between layers and can be serialized for persistence or message passing

Flow of control:

1. Controller receives the HTTP request and processes it
2. Controller calls appropriate method in Service layer
3. Service layer handles the business logic and may communicate with Repository layer for CRUD operations
4. Data retrieved from the database is often put in DTOs and sent back through the layers
5. Controller send the response (usually a DTO back to the client)

Microservices

Tuesday, 26 September 2023 22:22

What are Microservices?

Basics

- A blueprint that improves a codebases' scalability
- Depending on the application it can improve a monolith architecture by breaking down the codebase into multiple components (can be easier to maintain and manage)

Spring Framework

Tuesday, 26 September 2023 22:23

= application development framework for Java

Spring Framework => Dependency injection container with a couple of convenience layers such as db access, proxies, Aspect-Oriented Programming, RPC and web mvc framework

Spring Boot = built on top of Spring Framework -> provides an easy way to configure and run web applications

ApplicationContext - central interface of a Spring framework application that is used for providing configuration information to the application (ex. Bean factory methods for accessing application components)

Beans - objects that form the backbone of application - object that is instantiated and managed by Spring IoC container (Inversion of Control- dependencies are defined without creating them)

Profiles- a way to group and activate a set of configuration with a single profile parameter

Maven

Tuesday, 26 September 2023 22:23

Essential directories:

- src = contains all the source code and resources (Java files, configuration files, property files, XML files etc.), as well as source code and resources for tests
- target = Maven places the output of the build - compiled bytecode, JAR files, WAR files, documentation etc. -it can be regenerated by building the project again
- pom.xml (project object model)= it is an XML file that contains info about the project and configuration details used by Maven to build the project

It contains:

- Project Coordinates
- Dependencies
- Plugins
- Properties= contains properties that can be referenced within the POM file
- Profiles= specifies build profiles for different build environments
- Repositories= specifies if Maven should search for dependencies
- Build= build configuration

Synchronous and Asynchronous Comm

Tuesday, 26 September 2023 22:37

Synch= make an HTTP request to the Fetch Service and then the Fetch Service makes an HTTP request to the Inventory Service - one service awaits for the others response

Async= files the request but doesn't wait for it - this can be enabled by the event driven architecture - when fetch performed successfully triggers an event (FetchSuccessEvent)- we can place this event object as a message inside the Kafka broker and our Notification Service which is the consumer will consume this message and process this message accordingly

Docker

Tuesday, 26 September 2023 22:39

In order to see how they all work together - we need to dockerize all the services a tool called docker compose

Service Discovery

Friday, 29 September 2023 20:32

In a cloud environment there can't be dedicated IP addresses

There can be multiple instances of the same microservice and each instance can have a dynamic IP address- how will an external microservice that wishes to communicate with the developed microservice know which of those instances to call

Service Discovery Pattern entails using a server (called Discovery Server)- which stores all the information about our microservices- the service name as well as all the IP addresses of the instances

When using the discovery server the microservices will at first try to register with the Discovery Server by making a request. Upon receiving these requests the Discovery server will add the entries of these services into its local copy (registry).

Example

Microservice 1 and 2 have registered all their instances with the Discovery Server. When registering an instance the client receives a copy of the registry to be stored- if for some reason the Discovery server is not available the client will have its own local copy.

M1 will first make a call to the Discovery Server asking where to find M2. The Discovery Server will then reply with the IP address of M2 and then M1 will know where to find M2.

This way one ca avoid hardcoding the url of the M2 service.

API Gateway

Sunday, 1 October 2023 14:21

Before implementing the API gateway:

- If user wants to access one of the microservices it tries to call an arbitrary port where the microservice is running on- this would be fine in a local dev environment but would not work in a production code

Microservices can have multiple instances and application can run on different ports- we can't hardcode the ports

Solution is to introduce a component at the start of the architectural landscape called an API gateway which is responsible for routing the client requests to the corresponding services

API gateway acts like an entry point- user calls API gateway without based on Request Headers - the gateway can be configured to do the routing accordingly

API gateway can address some additional concerns- Authentication, Security, Load Balancing, SSL Termination

Authentication= if you want to make sure that all requests are authenticated you can configure that in API gateway - it can contact the authorisation server (if that is the authorisation method of choice)

Load balancer= if you have multiple instances of microservices and the user makes a request to the product service - it will choose the instance to fulfil it and it will send the response back to the user

SSL termination= if user makes a call to an external service-HTTPS scheme by following TLS protocol- so if user calls the API gateway from the outside with the TLS protocol, the API gateway which is already part of the microservices network doesn't require HTTPS when calling the other microservices (internal communication) so it can perform HTTP communication- SSL connection will be terminated at the API gateway level

Circuit Breaker

Sunday, 1 October 2023 20:54

Used when we want to have resilient communication between our services

Synchronous communication can introduce problems

- The called service may be unavailable
- Remote service calls can be slow - if something goes wrong in the called service- a performance issue or a database issue- that can slow your API calls

When encountering these issue we want a resilient system- we don't want our request to terminate abruptly

Circuit breaker pattern is a set of states:

- By default if communication between services is working fine - CIRCUIT BREAKER CLOSED
- If called microservice has an issue - CIRCUIT BREAKER OPEN (communication will not be allowed for a certain amount of time)- during this period it can either throw an error message or it can execute some fallback logic
- After a certain amount of time of CB being in OPEN state- the state will change to HALF OPEN- it will check if requests are going through or not and change state accordingly

Resilience4J can help with implementation- alternative to Netflix Hystrix

Distributed Tracing

Tuesday, 3 October 2023 20:06

How can we track down issue- one solution is logs but in a production ready application where services receive thousands of logs it is not plausible to understand these problems through logs

Distributed tracing helps us track a request from start to finish- if request failed at any point of time wwe can understand why it failed

User wishes to fetch an idea from his/her/their stash- this request first reaches the API gateway and after that the API gateway will proxy the request to the Fetch service makes a request to the Inventory service

To be able to track the request all the way to the Inventory service we need some kind of mechanism to trace it - traceID and spanID

traceID= unique identifier which identifies the request that comes into the system

spanID= the number of trips the request is going to take inside the system - for each "stop" there is a unique identifier

By having both traces and spans we can trace the whole request lifecycle in our services and we can also understand if at all the service is responding slowly or if it's having some performance issues

For a spring project- use Spring Cloud Sleuth = distributed tracing framework to generate the span id and trace id whenever we receive a request

In order to visualize the journey of the request- use Zipkin

Because a circuit breaker is used the request as a whole is fragmented - so different thread ids for what is seemingly a single request

Dependency Injection

Sunday, 5 November 2023 14:30

1. When do we do it?

Class A
Class B

Upon creation of class A object (i.e. in the constructor) we instantiate an object of class B.

2. Why do we do it?

When testing the application if we were to create an instance of class A in a test, it would also create an instance of class B. If a class B method would go to a DB to get some data and then retrieve - if the db is down then we couldn't test that. This can make our code very difficult to test.

Mockito- which allows us to mock objects- we wouldn't be able to use it to mock the class A object in this format.

We are also creating a lot of objects which are going to be stored in the heap and you don't necessarily know when they will be garbage collected.

ADDTIONALLY- we might want the class B objects to be Singletons.

3. How do we do it?

We rely on other frameworks for object creation instead of creating them ourselves (using "new" keyword/operator).

Example of frameworks: Spring, Guice, Dagger.

The frameworks take care of managing these objects and allow us to inject them in our code.

In Spring if we annotate a class as @Service the object is automatically created as a Singleton. This means that if we choose to inject it in multiple classes it would simply use the same instance.

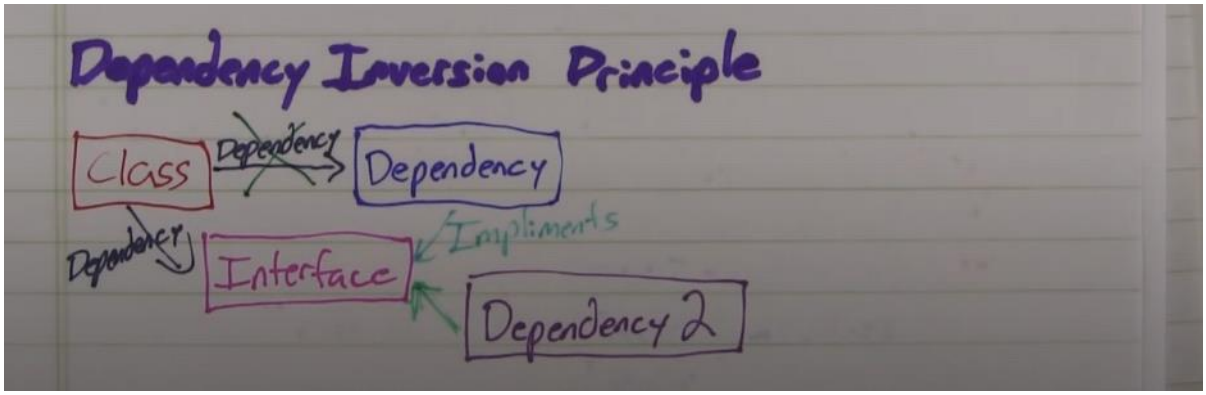
4. More on the terminology and additional aspects of Dependency Injection

Dependency = another object that your class needs to function

Injection = dependency is pushed into the class form the outside (i.e. you shouldn’t instantiate the dependency using the "new" operator form inside of the class) - you treat the dependency as a constructor parameter

Dependency injection decouples your classes construction from the construction of its dependencies.

Dependency Inversion Principle= code should depend upon abstractions



It makes the code cleaner, easier to modify and reuse.

Testing (TDD)

Monday, 18 December 2023 13:43

TDD= software paradigm
Requirements are reworked into specific test cases to ensure full functionality - you write the tests first and then code in the requirement

Unit testing
= used to verify the smallest piece of testable software

It is important to constantly question the value a unit test provides versus the cost it has in maintenance or the amount it constrains your implementation.

Component testing
= used to verify that the different parts of a service work as intended by isolating third-party code and other services

We use mocks to replace external services and in memory db to replace external datastore

Components in microservices are the services themselves

Integration testing

= used to verify communication paths between services (mainly used to catch interface defects) or data stores

Ex.
Gateway Testing - verifies if the interface to the external microservices is fully functional
Persistence Testing - verifies if the external datastore works (if you can connect to it, retrieve/ save the data, and if the schema returned is the same as what we expect)

JUNIT- allows us to do automated testing (rather than using Postman or the console)

H2- in memory Java db

Mockito - testing framework used for creation of mock objects

Mock objects- objects that mimic the behaviour of real objects in controlled ways (integrates well with Spring)

JsonPath - query language for JSON and lets you extract specific bits of a JSON document (use it to verify responses from REST API)

AssertJ - assertion library for assert statements (tests a predicate if it's true or false based on the output of program)

Hamcrest - assertion framework for writing "matcher" objects

WireMock - mimics the behavior of an HTTP API and capture the HTTP requests sent to that API - allows us to stay productive when an API we depend on doesn't exist or isn't complete

Testcontainers- Java library that supports Junit tests provides throwaway instances of anything that can be run in a docker container

Grafana and Prometheus

Sunday, 28 January 2024 15:49

Spring Boot App will make use of the Actuator endpoint which will expose all the metrics of our application

Prometheus software will poll the SBA for every X seconds and store it inside the in-memory db

Prometheus will act like a datasource for Grafana which provides a UI dashboard- Grafana also polls Prometheus