DEVCONF.cz

How to enjoy E2E testing

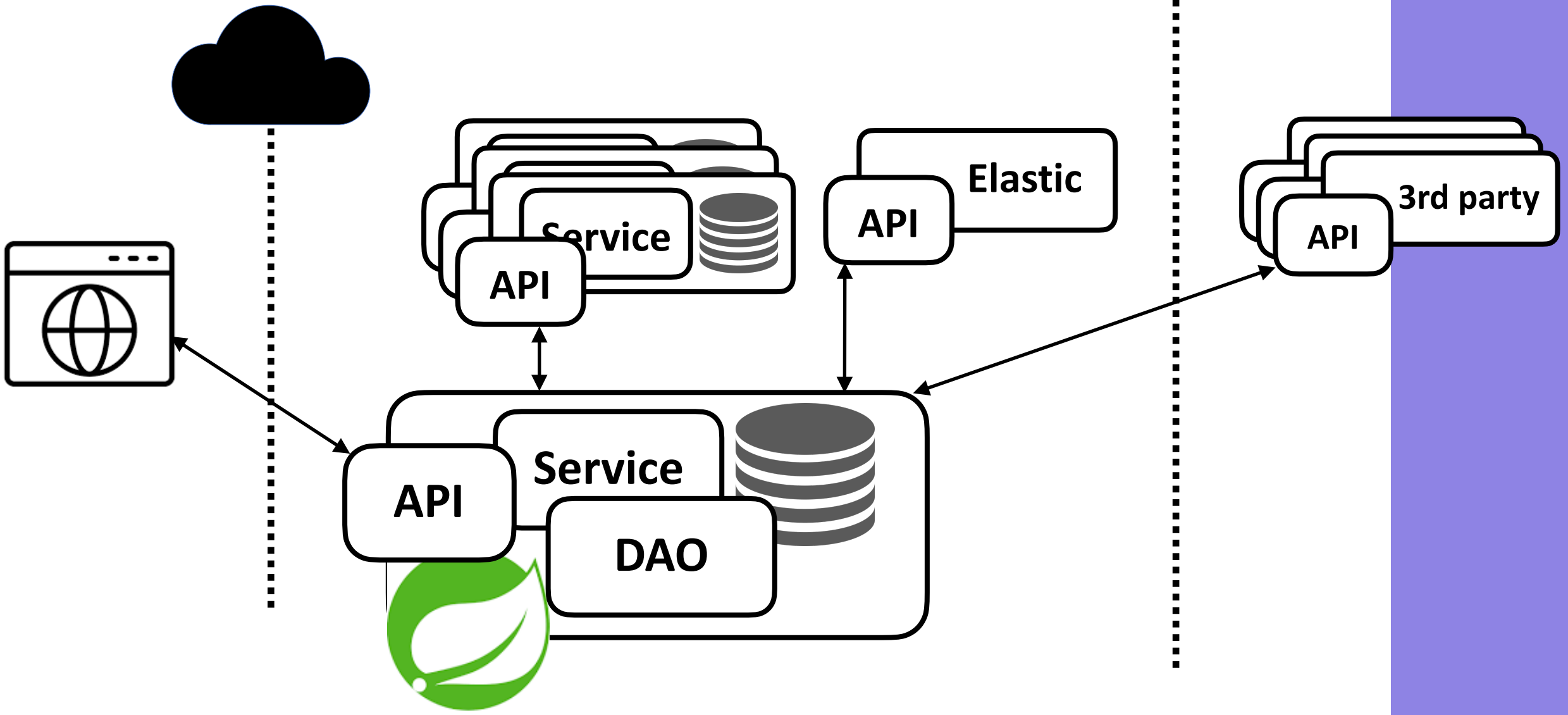
in Kotlin / Spring

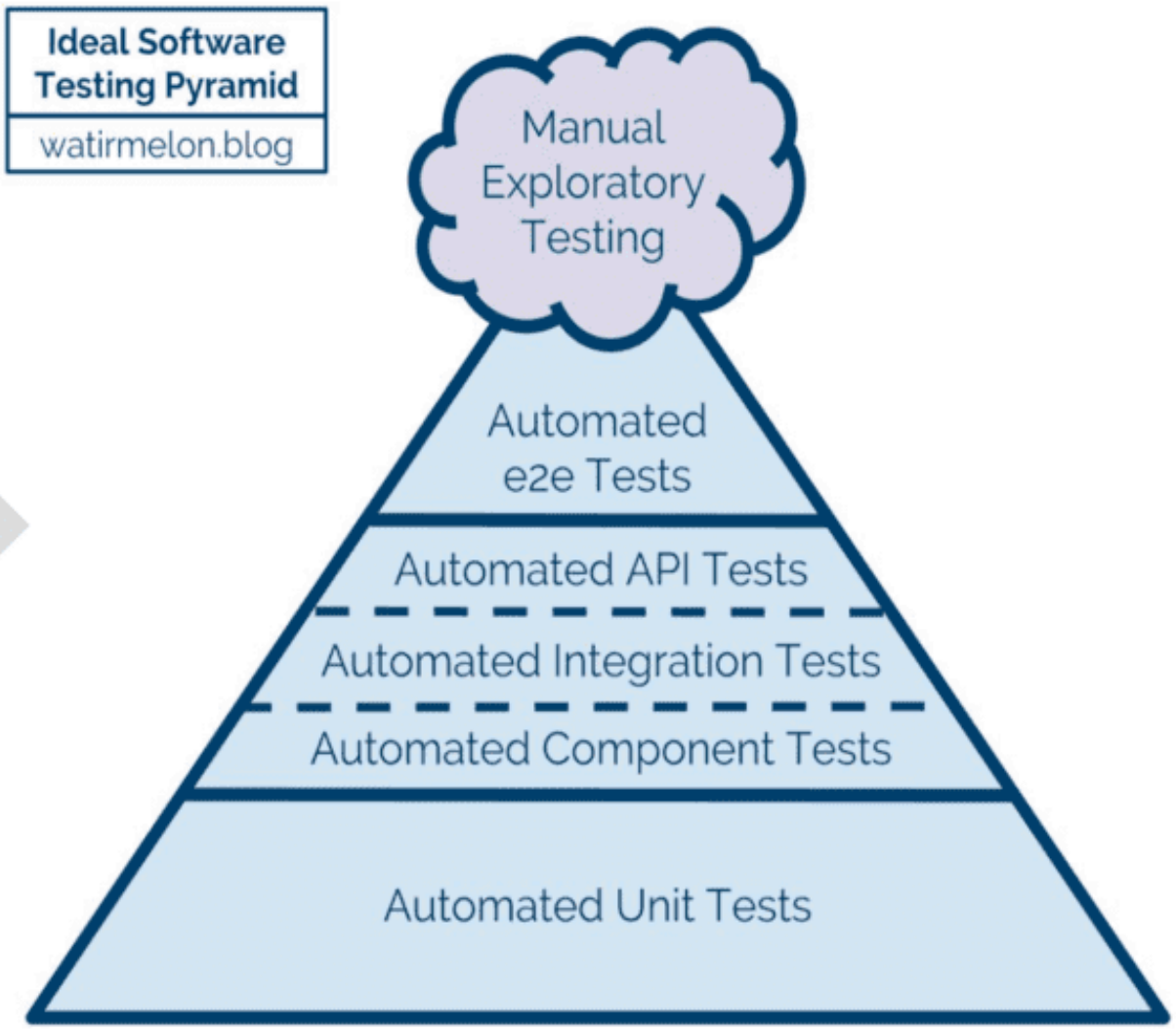
Víťa Plšek

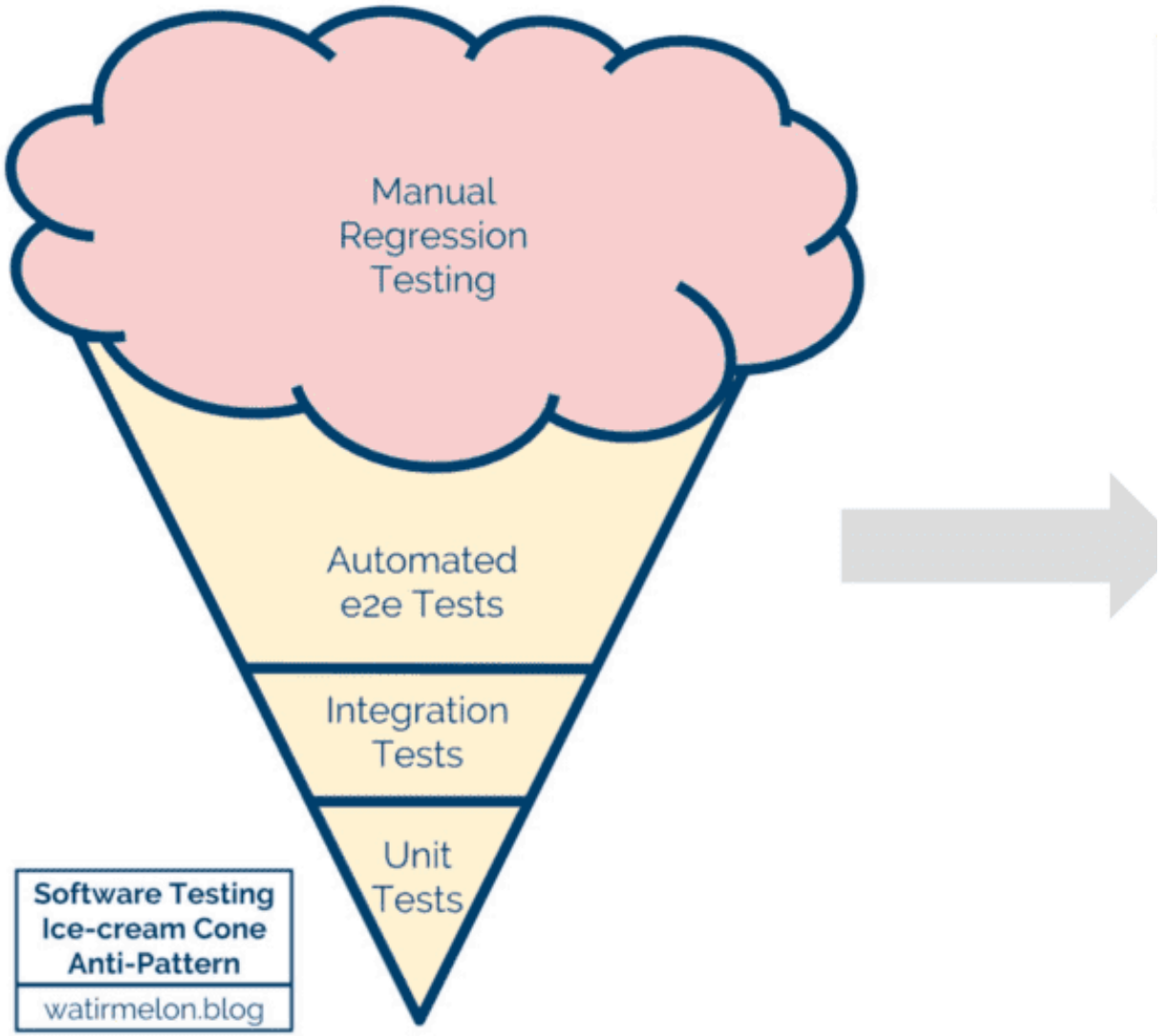
@vitaplsek

MOROSYSTEMS

How does a typical web application look ?




Ideal Software Testing Pyramid
watirmelon.blog
Manual Exploratory Testing
Automated e2e Tests
Automated API Tests
Automated Integration Tests
Automated Component Tests
Automated Unit Tests


Manual
Regression
Testing
Automated
e2e Tests
Integration
Tests
Unit
Tests
Software Testing
Ice-cream Cone
Anti-Pattern
watirmelon.blog

What do you usually test ?

What I usually test

Unit tests
validators, algorithms

What I usually test

Integration tests

database queries

communication with services

Unit tests

validators, algorithms

Acceptance criteria - from analyst

- It is possible to add, edit and delete a customer's branch
- Deactivating a customer will deactivate his branches
- Cannot delete a customer who has offices
- Listing is sorted alphabetically and can be paginated

What I usually test

E2E

rest api - acceptance criteria

Integration tests

database queries

communication with services

Unit tests

validators, algorithms

What I usualy test

E2E

rest api - acceptance criteria

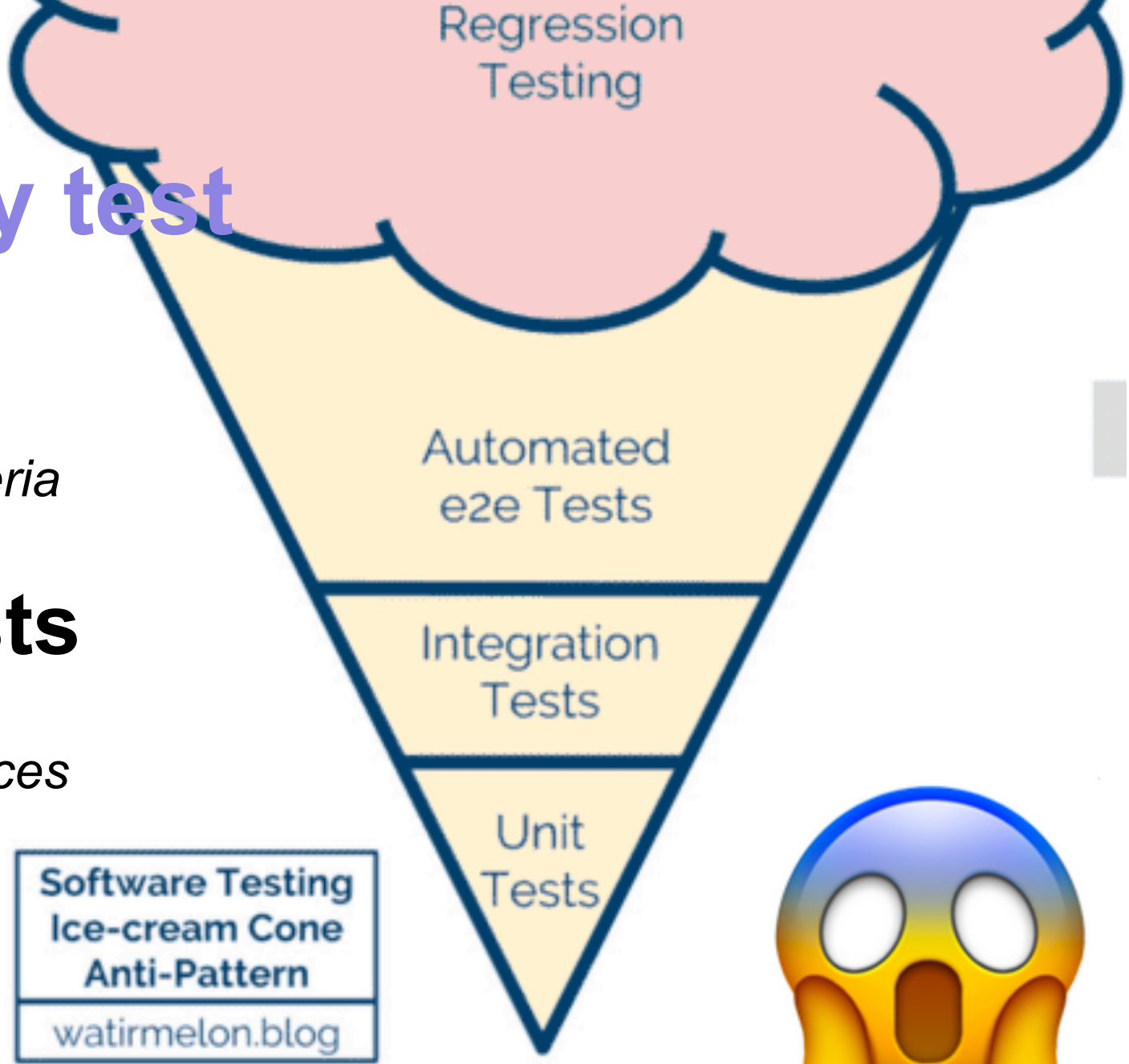
Integration tests

database queries

communication with services

Unit tests

validators, algorithms



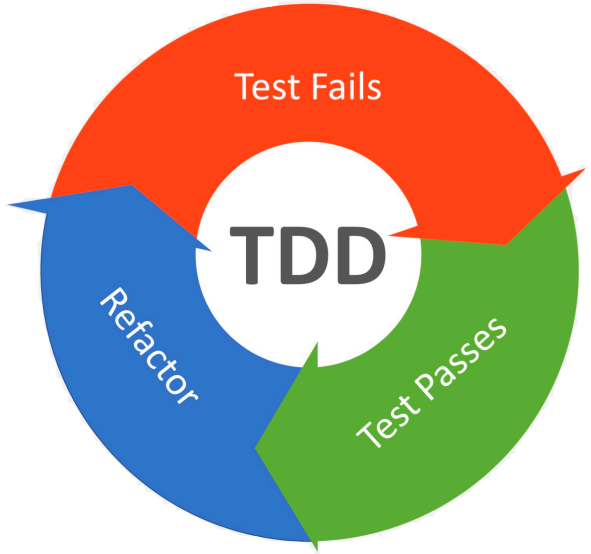
DEV CONF .CZ

"If something has worked, do it all the time.”

Kent Beck

Extreme programming

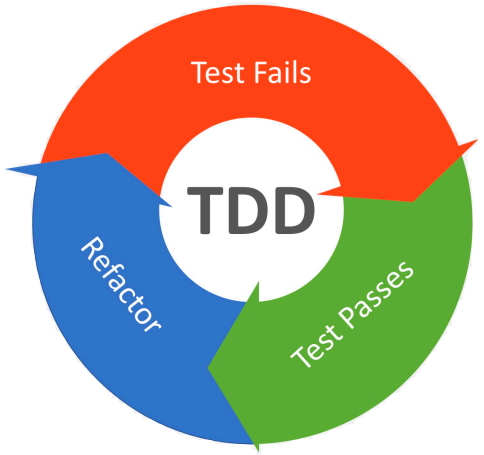
TDD - Test driven development



The development of BE as I used to do it

- Database design
- Service / dao / logic <- lots of space for "this may come in handy"
- REST API
- Continuous debugging with Curl / Postman
- Write tests if there is some time left …

How to develop API according TDD



- Acceptance criteria given by analysts
- Tests
- REST API
- Business logic
- Service / Dao / Database

How to do that, and also enjoy it ?


LEGO
NEXO KNIGHTS

Kotest

DEV CONF .CZ

JUnit

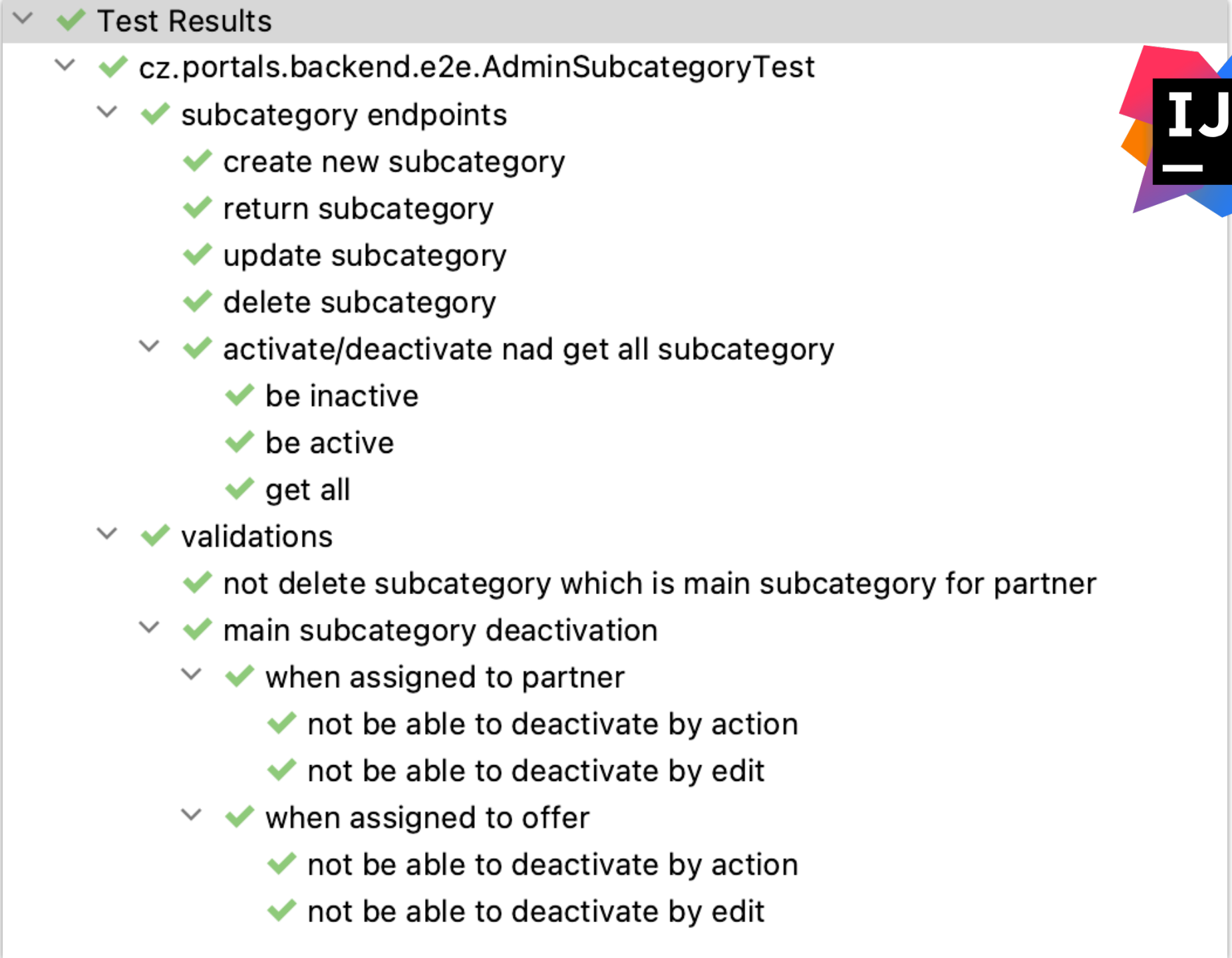
Kotest

DEV CONF .CZ

```
class AlgorithmTests {
    @Test
    fun `count the result of algorithm`() {
        // asserts here
    }
}
```

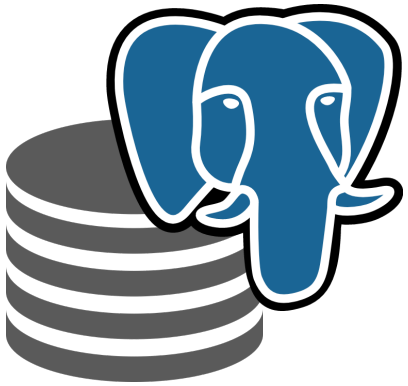
```
class AlgorithmTests : ShouldSpec({
    should("count the result of algorithm") {
        // asserts here
    }
})
```

```
class SubcategoryTest() : ShouldSpec() {

    context("subcategory endpoints") {

        // common data preparation
        val subcategoryForCreate = createSubcategoryDto()

        context("when creating new subcategory") {

            // another data preparation / action call
            val createdSubcategory = restTemplate.createSubcategory(subcategoryForCreate)

            should("create basic info") {
                // tests, asserts
                createdSubcategory.id shouldNotBe null
                createdSubcategory.name shouldBe subcategoryForCreate.name
                createdSubcategory.active shouldBe true
            }

            should("contain default image") { ... }

            context("validations") {
                should("not allow empty name") { ... }
                ...
            }
        }
```


Test Results
cz.portals.backend.e2e.AdminSubcategoryTest
subcategory endpoints
create new subcategory
return subcategory
update subcategory
delete subcategory
activate/deactivate nad get all subcategory
be inactive
be active
get all
validations
not delete subcategory which is main subcategory for partner
main subcategory deactivation
when assigned to partner
not be able to deactivate by action
not be able to deactivate by edit
when assigned to offer
not be able to deactivate by action
not be able to deactivate by edit

I appreciate tests over real database

- DAO
  - Mapping nested objects into DTO
- Queries to db
  - "Get all listings that currently have one active subcategory"
  - “Delete all logs older than 30 days that have not been opened”



Starting the container

```
private const val POSTGRES_IMAGE_NAME = "postgres:13.8"

val postgresSQLContainer = PostgreSQLContainer<Nothing>("postgres:13.8")
postgresSQLContainer.start()
```

```
spring:
    datasource:
        driver-class-name: "org.testcontainers.jdbc.ContainerDatabaseDriver"
        url: "jdbc:tc:postgresql:13.8://localhost:45432/postgres"
        username: postgres
        password: postgres
```

E2E Tests of REST API ?

E2E tests Support in spring

```
@SpringBootTest
class E2ETest(
    val mockMvc: MockMvc,
    val restTemplate: TestRestTemplate,
    val webTestClient: WebTestClient
) : ShouldSpec() {

    ...
}
```

WebTestClient

```
webTestClient
    .post()
    .uri("/admin/partner")
    .bodyValue(partnerCreateDto)
    .exchange()
    .expectStatus()
    .is2xxSuccessful
    .expectBody(PartnerDto::class.java)
    .returnResult()
    .responseBody!!
```

Encapsulate your API Calls

```
webTestClient
    .post()
    .uri("/admin/partner")
    .bodyValue(partnerCreateDto)
    .exchange()
    .expectStatus()
    .is2xxSuccessful
    .expectBody(PartnerDto::class.java)
    .returnResult()
    .responseBody!!
```

Encapsulate your API Calls

```
webTestClient
    .post()
    .uri("/admin/partner")
    .bodyValue(partnerCreateDto)
    .is2xxSuccessful
    .returnBody(PartnerDto::class.java)
```

Encapsulate your API Calls


```
val createdPartner = webTestClient.createPartner(partnerCreateDto)
```

Encapsulate your API Calls


```
val createdPartner = webTestClient.createPartner(partnerCreateDto)
```

```
fun WebTestClient.createPartner(partnerCreateDto : CreatePartnerDto) =
    webTestClient
        .post()
        .uri("/admin/partner")
        .bodyValue(partnerCreateDto)
        .is2xxSuccessful
        .returnBody(PartnerDto::class.java)
```

Encapsulate also data creation

```
fun newCreatePartnerDto(name = "Partner_" + Random.nextInt()) =
    CreatePartnerDto(
        name = name,
        active = true,
        lastModified = OffsetDateTime.now()
)
```

Prepare data just for your test

- **How ?**
  - Using REST calls
  - Directly into the DB
- **When?**
  - ~~Once before all the tests~~
  - Before the test class
  - **Before each test**

Prepare data directly into the DB

```
val customer = dslContext.createCustomer { name = "Customer 1" }
val office = dslContext.createOffice(customer)
```

Prepare data directly into the DB

```
val customer = dslContext.createCustomer { name = "Customer 1" }
val office = dslContext.createOffice(customer)
```

```
fun DSLContext.createCustomer(applyFunction: CustomerRecord.() → Unit = {}) =
    newRecord(Tables.PARTNER)
        .apply {
            name = "customer" + Random.nextInt()
            active = true
            lastModified = OffsetDateTime.now()
            apply(applyFunction)
            store() ^apply
        }
```

What about speed ?

Do not start containers when you do TDD


```
spring:
  datasource:
    driver-class-name: "org.testcontainers.jdbc.ContainerDatabaseDriver"
    url: "jdbc:tc:postgresql:13.8://localhost:45432/postgres?TC_REUSABLE=true"
```

```
val postgresSQLContainer = PostgreSQLContainer<Nothing>(POSTGRES_IMAGE_NAME)
    .apply { this: PostgreSQLContainer<Nothing>
        withReuse( reusable: true)
        start()
    }
```

Speed up start and builds

```
./gradlew testClasses
    --exclude-task generateJooq
    --exclude-task update

    -PturnOffSlowAction=true
```

- Modified test build
- Run configuration template for Kotest

- Log level = INFO

Let’s do some livecoding …

Summary

- Give TDD a chance
- Do not hesitate to use WebTestClient
- Use real database
- Speed up your build

- https://kotest.io/
- https://www.testcontainers.org/
- https://wiremock.org/ / https://github.com/Ninja-Squad/springmockk

Source: https://github.com/VitaPlsek/DevConf2023

Contact: vita.plsek@morosystems.cz / www.linkedin.com/in/vitaplsek

DEVCONF.cz

Thank you for your attention.

Ask me anything!