



# How to enjoy E2E testing in Kotlin / Spring



Vítá Plšek



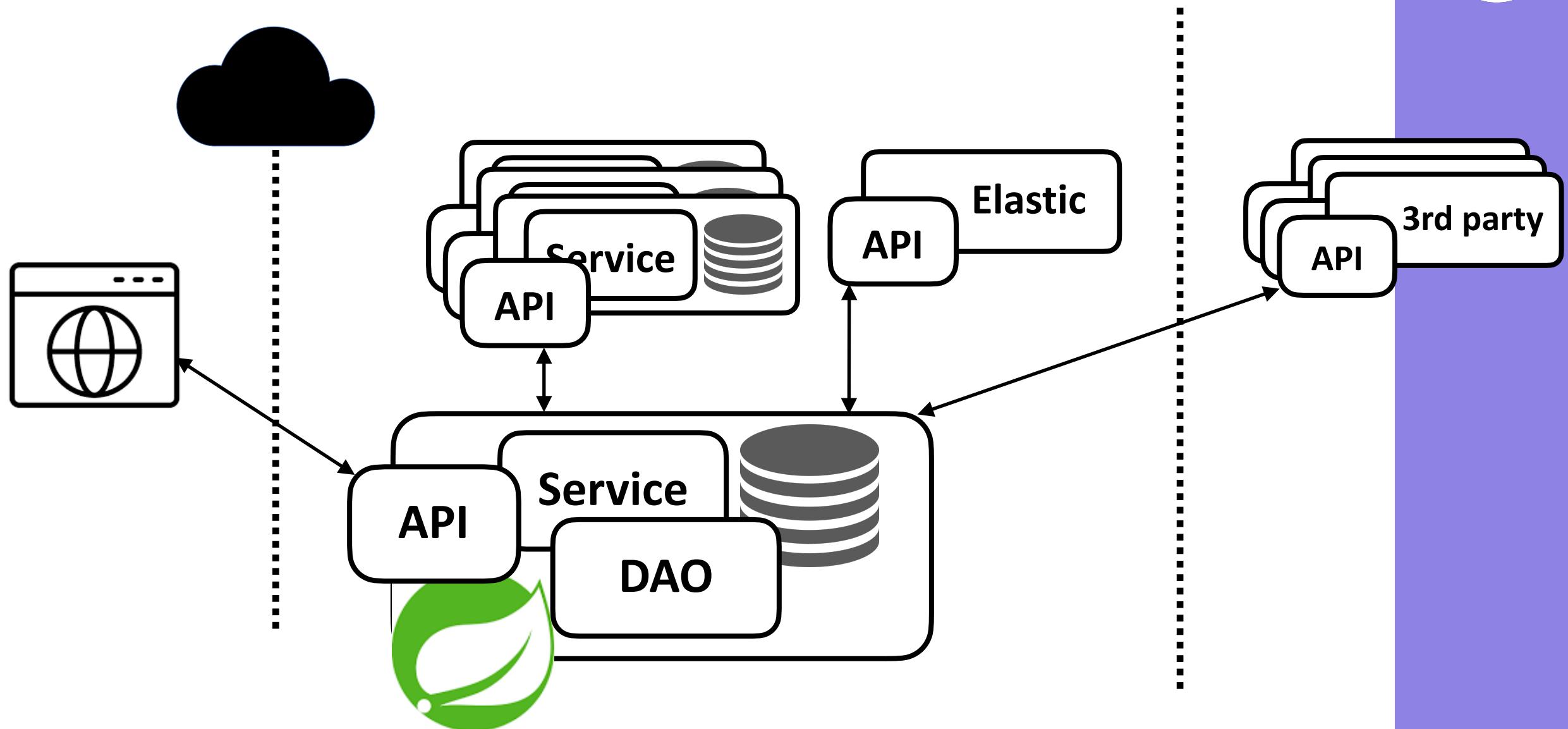
@vitaplsek

Software Engineer

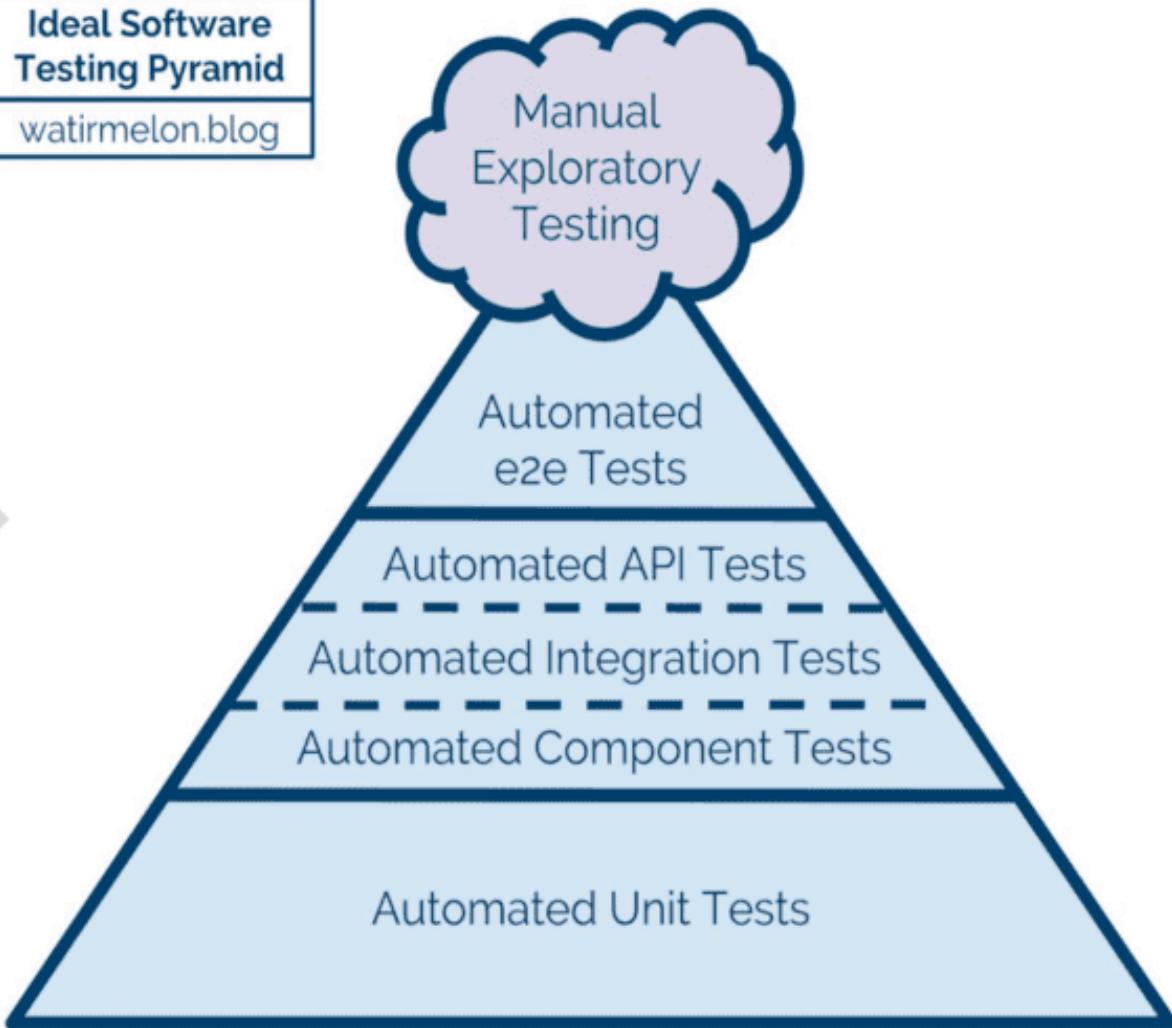
**MOROSYSTEMS**

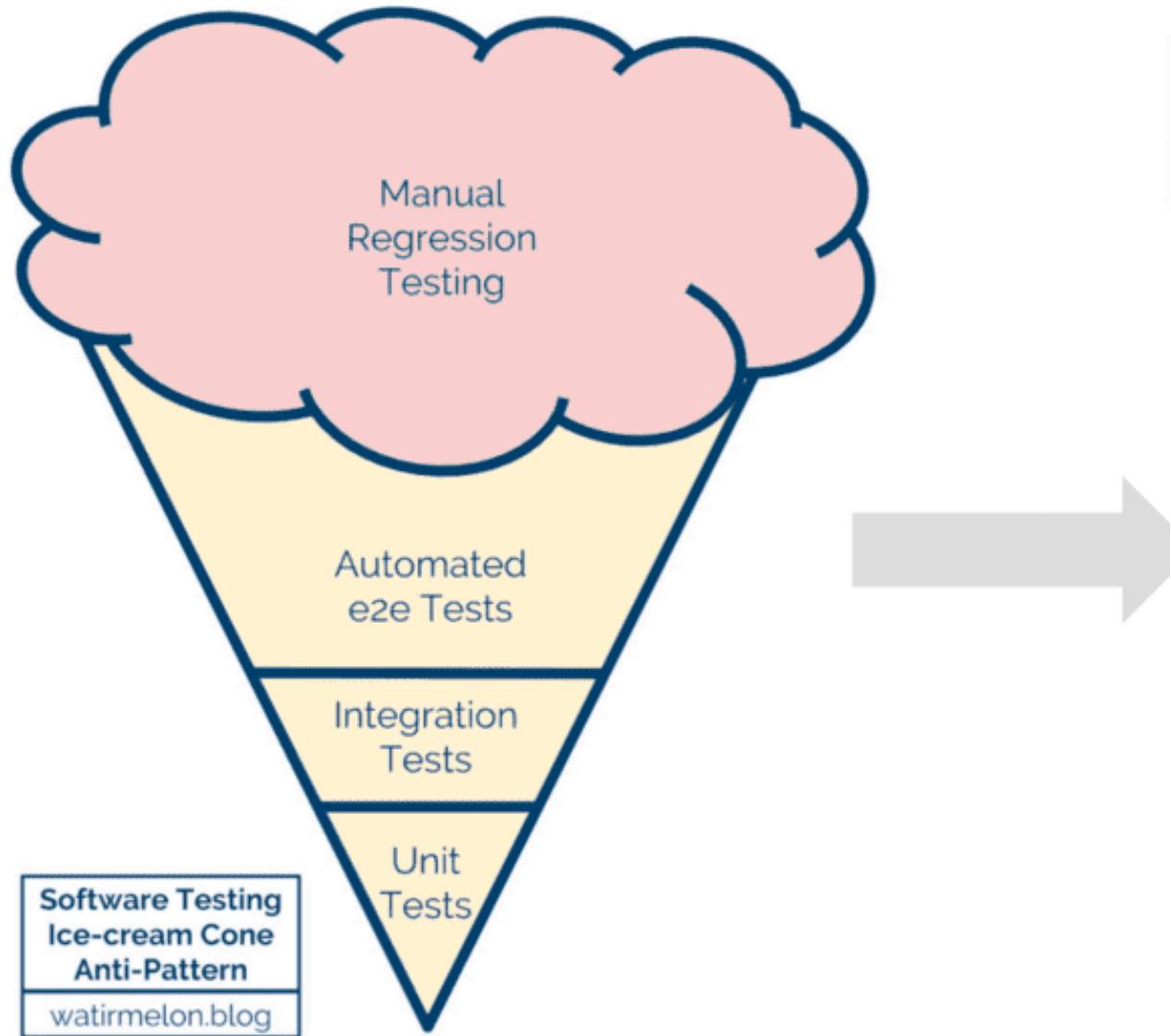


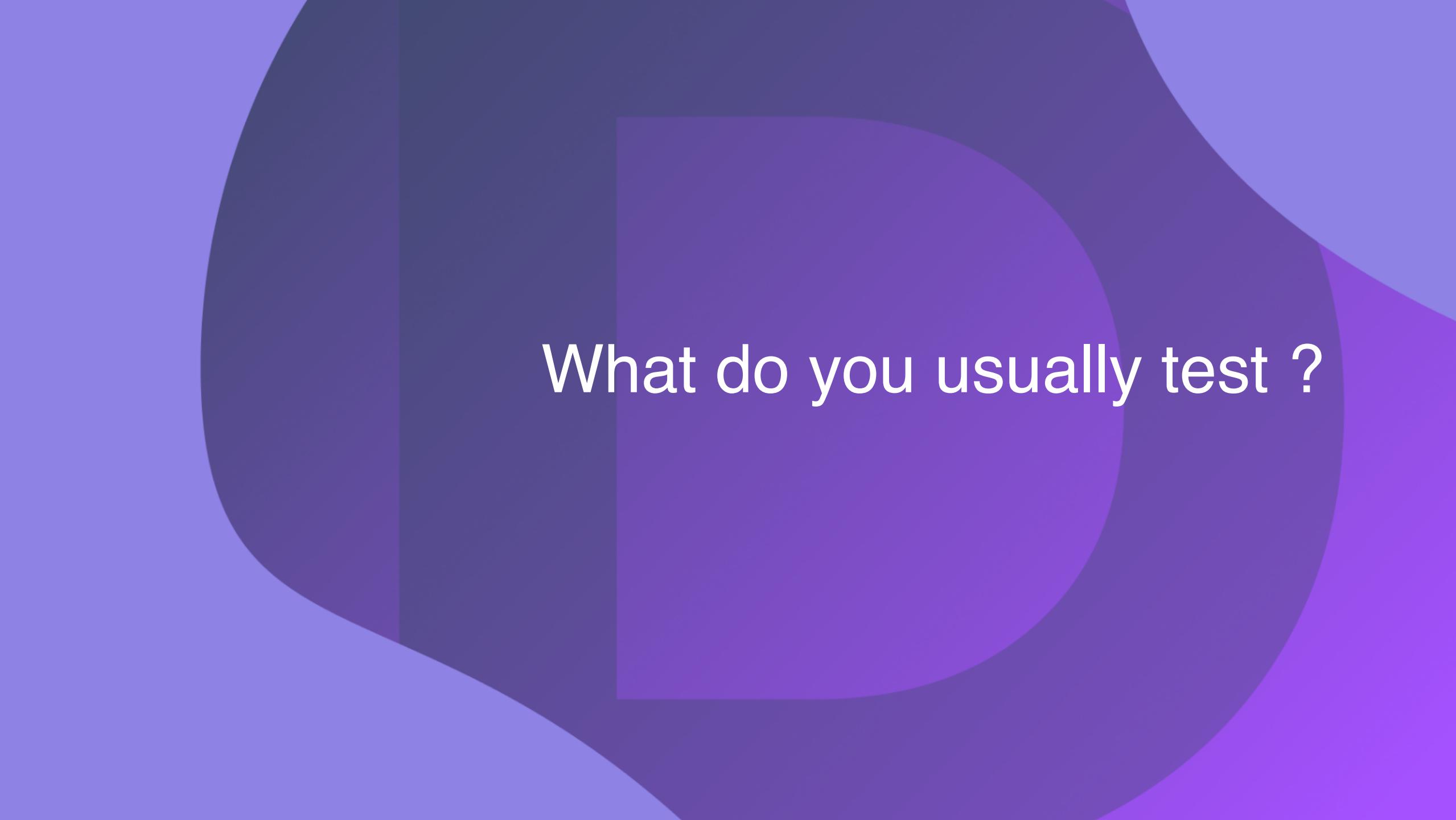
# How does a typical web application look ?



Ideal Software  
Testing Pyramid  
[watirmelon.blog](http://watirmelon.blog)





The background of the slide features a complex, abstract design composed of several overlapping circles in various shades of purple. The circles overlap in a non-uniform manner, creating a sense of depth and movement. The overall effect is modern and minimalist.

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 usually test

## E2E

*rest api - acceptance criteria*

## Integration tests

*database queries*

*communication with services*

## Unit tests

*validators, algorithms*

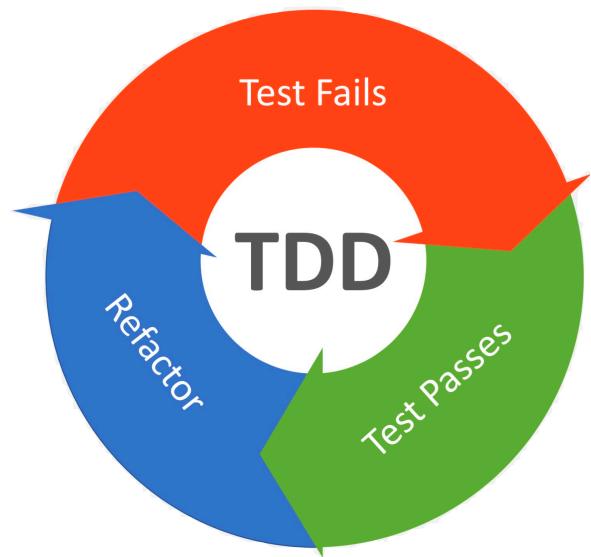


"If something has worked, do it all the time."

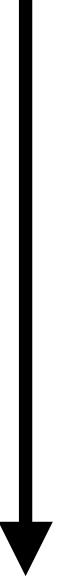
Kent Beck  
*Extreme programming*



# TDD - Test driven development



# The development of BE as I used to do it

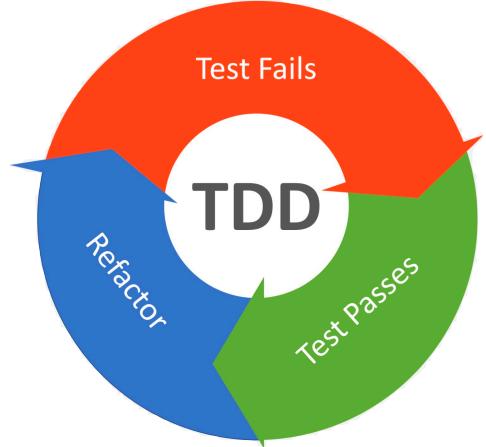
- 
- Database design
  - Service / dao / logic <- lots of space for "this may come in handy"
  - REST API
  - Continuous debugging with Curl / Postman
  - Hmm now I will create / fix the tests

# The development of BE and I used to do it

- Database design
- Service design / logic
- REST API
- Continuous debugging
- Hmm now I will create "this makes me in handy"

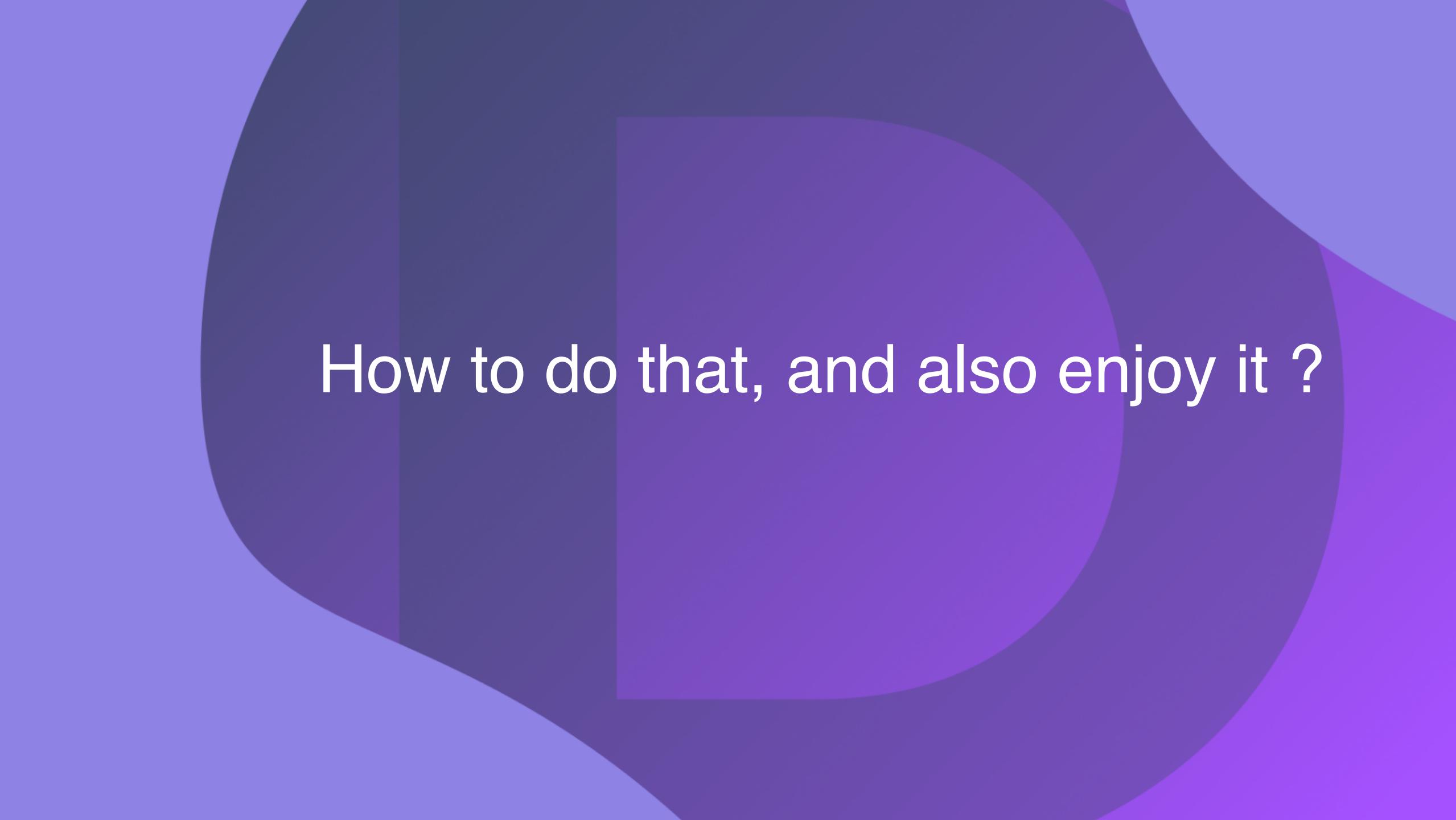


# How to develop API according TDD



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



The background of the slide features a complex, abstract design composed of several overlapping circles in various shades of purple. The circles overlap in a non-uniform manner, creating a sense of depth and movement. The overall effect is organic and modern.

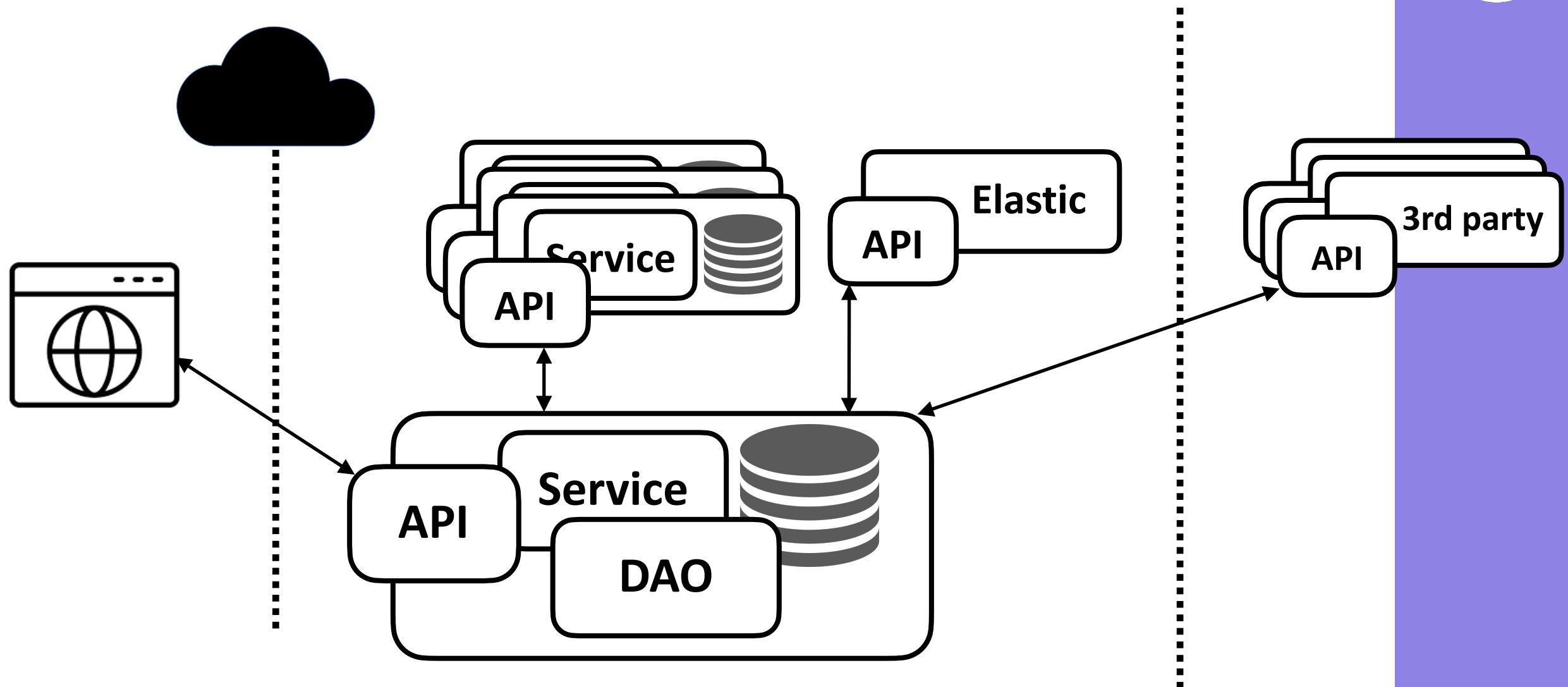
How to do that, and also enjoy it ?



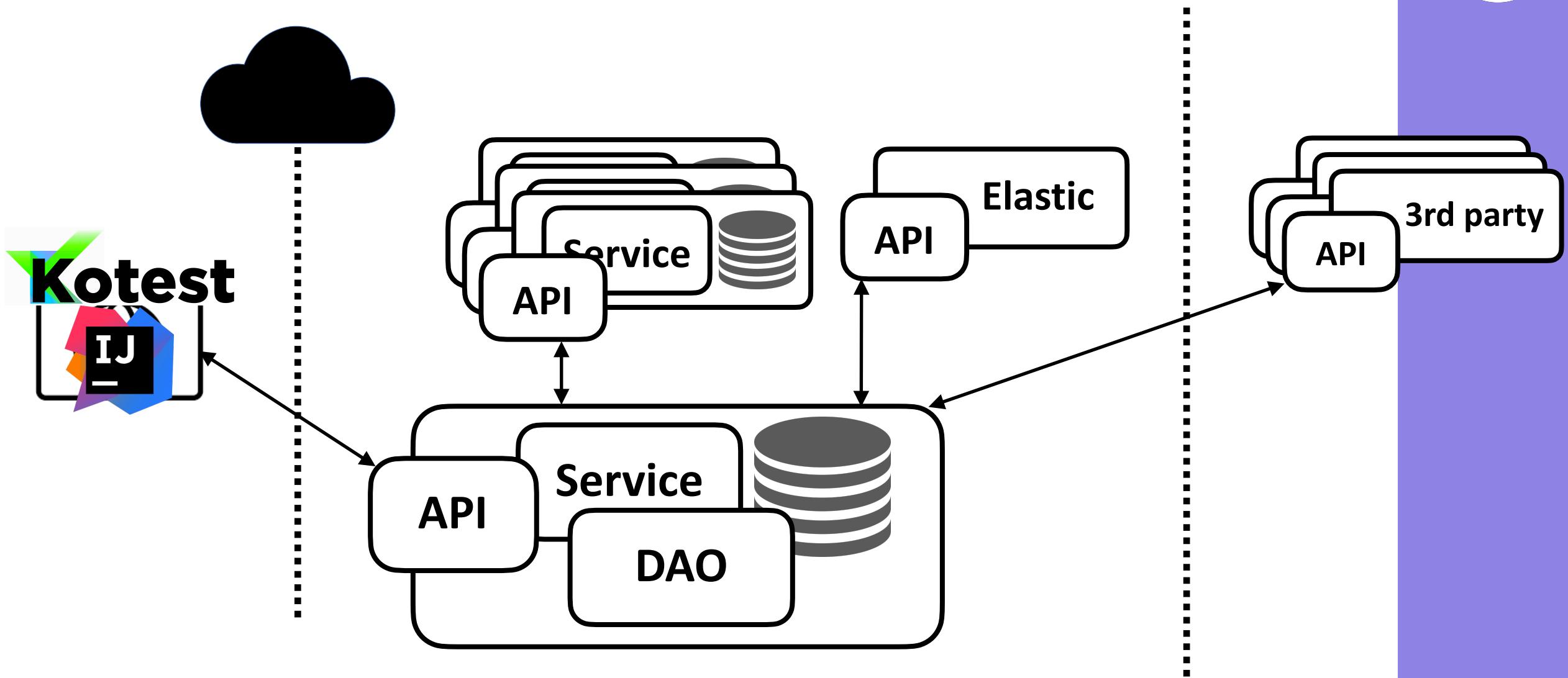




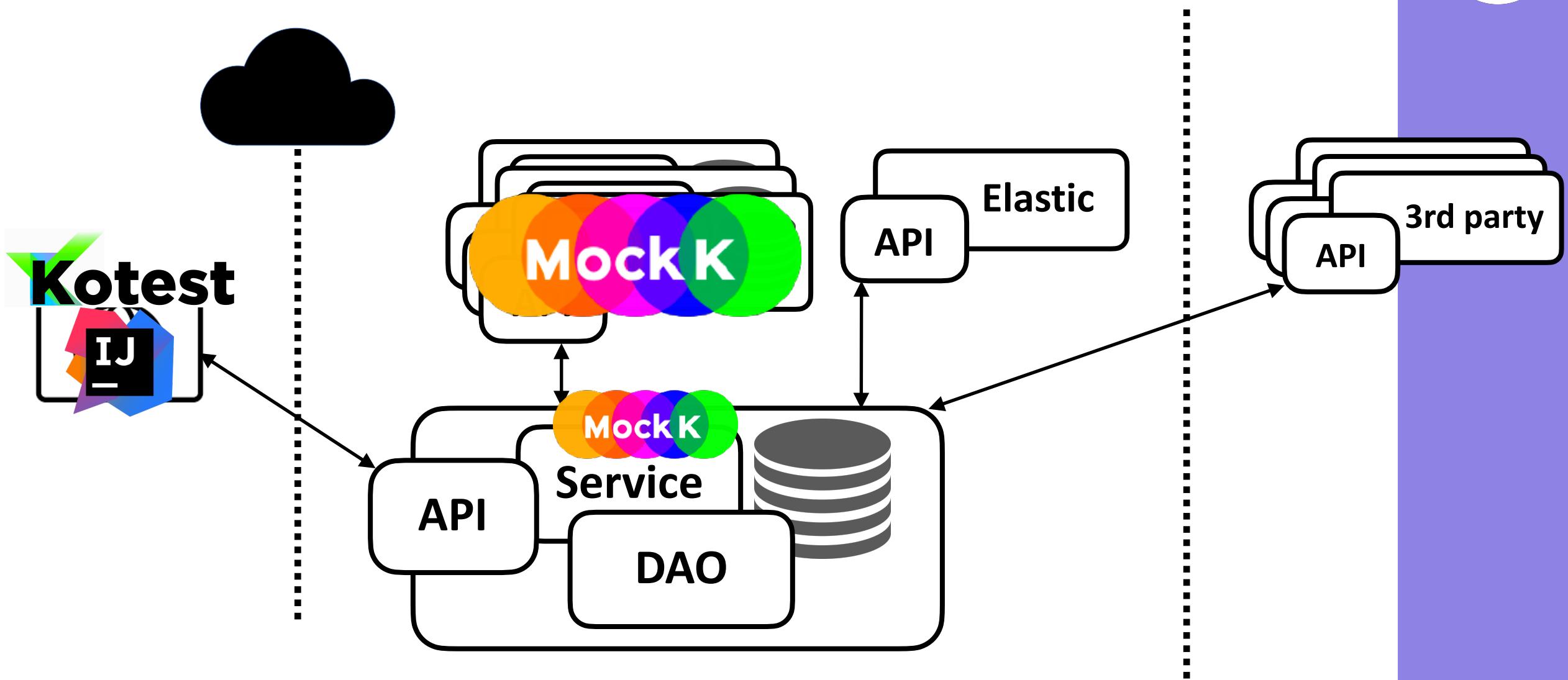
# How can right tools help you



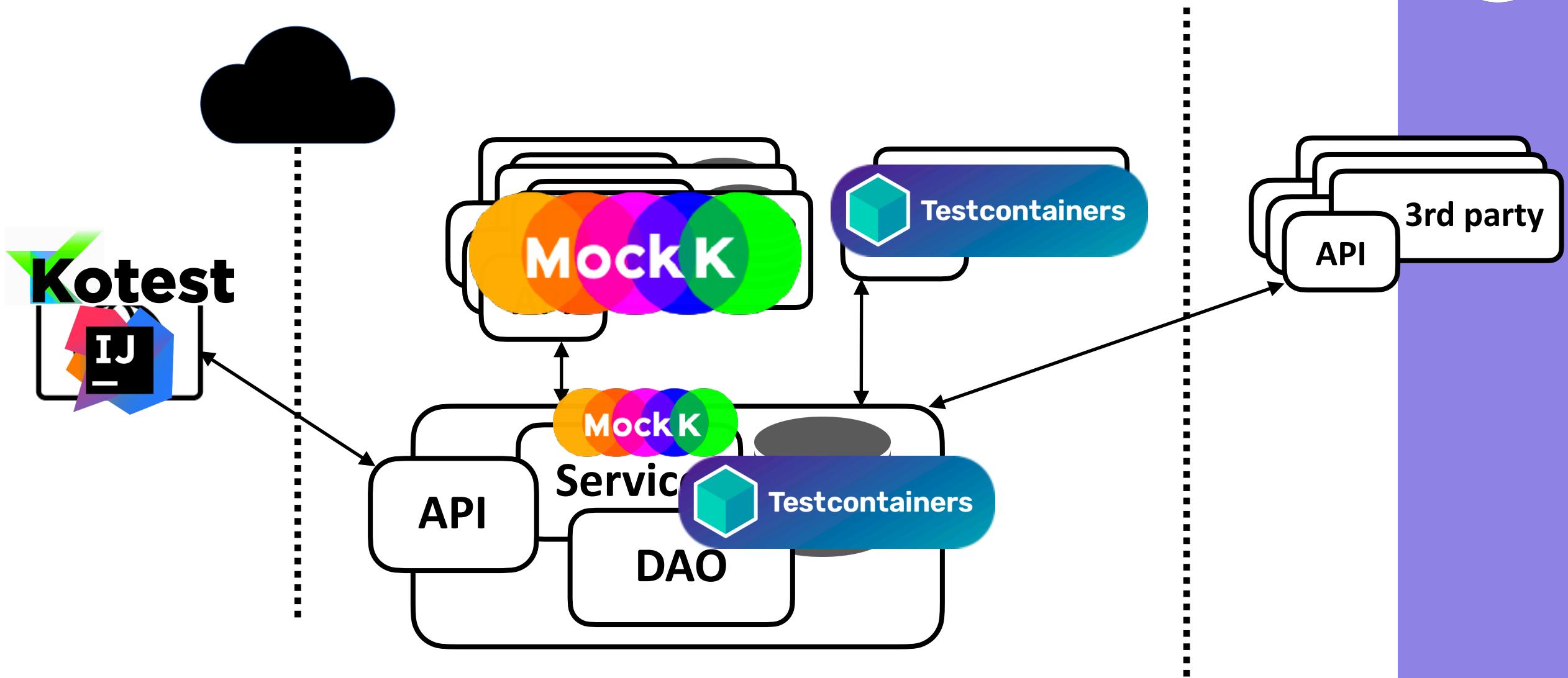
# How can right tools help you



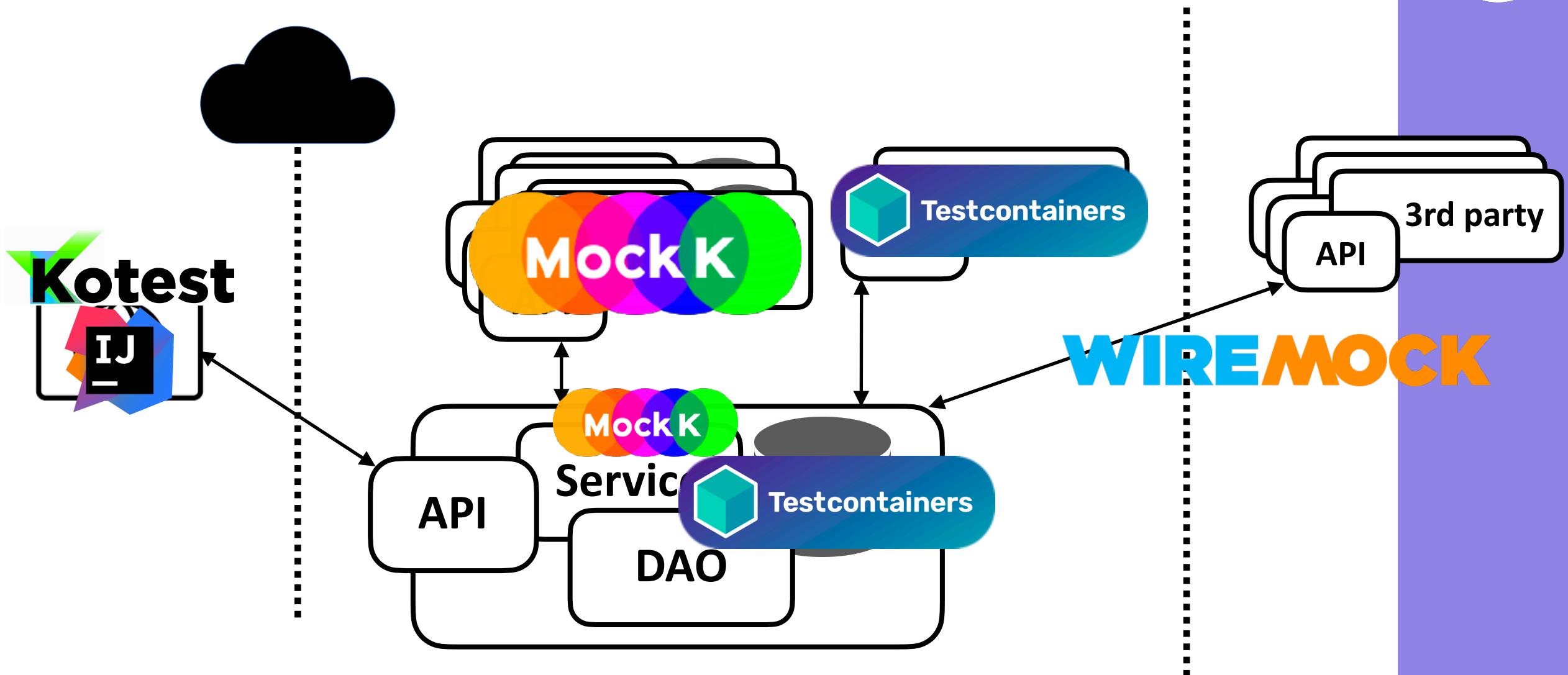
# How can right tools help you



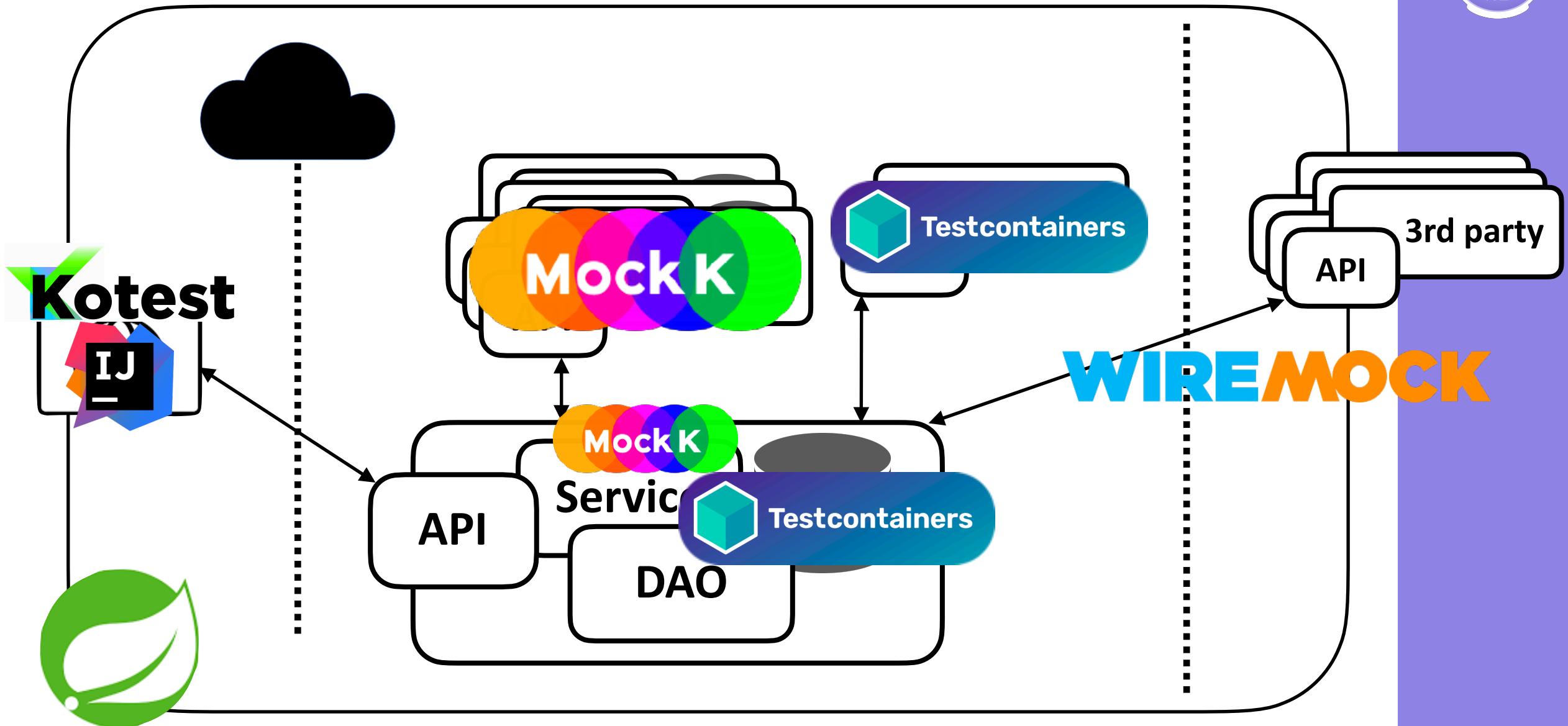
# How can right tools help you



# How can right tools help you



# How can right tools help you





**Kotest**

The logo consists of the word "Kotest" in a large, bold, black sans-serif font. To the left of the "K", there is a graphic element: a white square containing a diagonal band with a green-to-blue gradient. The "K" itself is partially overlaid on this graphic, with its vertical stroke matching the blue part of the gradient.

# JUnit

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



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

```
14 >
15 > class SubcategoryTest() : ShouldSpec() {
16
17 >     context("subcategory endpoints") {
18
19         // common data preparation
20         val subcategoryForCreate = createSubcategoryDto()
21
22 >     context("when creating new subcategory") {
23
24         // another data preparation / action call
25         val createdSubcategory = restTemplate.createSubcategory(subcategoryForCreate)
26
27 >     should("create basic info") {
28         // tests, asserts
29         createdSubcategory.id shouldNotBe null
30         createdSubcategory.name shouldBe subcategoryForCreate.name
31         createdSubcategory.active shouldBe true
32     }
33
34 >     should("contain default image") { ... }
35
36 >     context("validations") {
37 >         should("not allow empty name") { ... }
38 >         ...
39     }
40 }
```



- ✓ 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



# Matchers



```
createdSubcategory.id shouldNotBe(null)  
createdSubcategory.id shouldNotBe null
```

```
createdSubcategory.name shouldBe subcategoryForCreate.name  
createdSubcategory.active shouldBe true  
createdSubcategory.labels shouldContain (Labels.NEW)
```

```
partners shouldHaveSize 1  
partners.shouldNotContainDuplicates()  
partners shouldExist { it.id = partner.id }
```

The Kotest logo, which consists of the word "Kotest" in a bold, black, sans-serif font. To the left of the text is a stylized graphic element composed of overlapping triangles in green, blue, and white.

# Matchers

DEV

```
infix fun <T> Collection<T>.shouldHaveSize(size: Int): Collection<T> {  
    this should haveSize(size = size)  
    return this  
}
```

```
createdSubcategory.id.shouldNotBe(null)
```

```
createdSubcategory.id shouldNotBe null
```

```
createdSubcategory.name shouldBe subcategoryForCreate.name
```

```
createdSubcategory.active shouldBe true
```

```
createdSubcategory.labels shouldContain (Labels.NEW)
```

```
partners shouldHaveSize 1
```

```
partners.shouldNotContainDuplicates()
```

```
partners shouldExist { it.id = partner.id }
```

 Kotest

# Spring support



```
class MyTests(val myService: MyService) : ShouldSpec() {  
    override fun extensions() = listOf(SpringExtension)  
  
    @Autowired  
    lateinit var myService2: MySecondService  
  
    init {  
        should("do something") {  
            myService.answer(myService2.question) shouldBe 42  
        }  
    }  
}
```





# I refuse to write copy of implementation

```
val pdfService: PdfService = mockk()  
val emailService: EmailService = mockk()  
val invoiceService = InvoiceService(pdfService, emailService)  
  
val invoice = createTestInvoice(email = "jan@novak.cz")  
val pdf = mockk<Pdf>()  
  
every { pdfService.generate(invoice) } returns pdf  
  
invoiceService.sendInvoice(invoice)  
  
verify { emailService.send("jan@novak.cz", pdf) }
```

```
class InvoiceService(  
    val pdfService: PdfService,  
    val emailService: EmailService,  
) {  
    fun sendInvoice(invoice: Invoice) {  
        val pdf = pdfService.generate(invoice)  
        emailService.send(invoice.customerEmail, pdf)  
    }  
}
```



# Mocking of services

- @MockkBean - SpringMockK

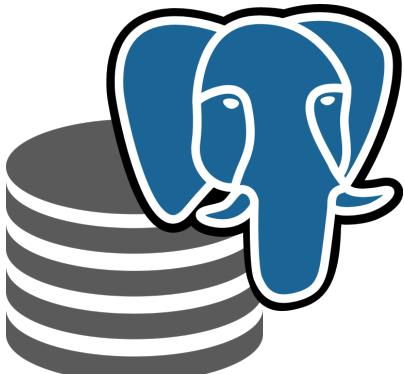
```
@MockkBean  
private lateinit var greetingService: GreetingService  
...  
every { greetingService.greet("John") } returns "Hi John"
```



- You can create test double on http level

# 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”





# Testcontainers



# Starting the container

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

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

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

The background of the slide features a minimalist design with abstract, overlapping circles in various shades of purple. These circles are positioned in the upper half of the slide, creating a sense of depth and focus on the central text.

# 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**

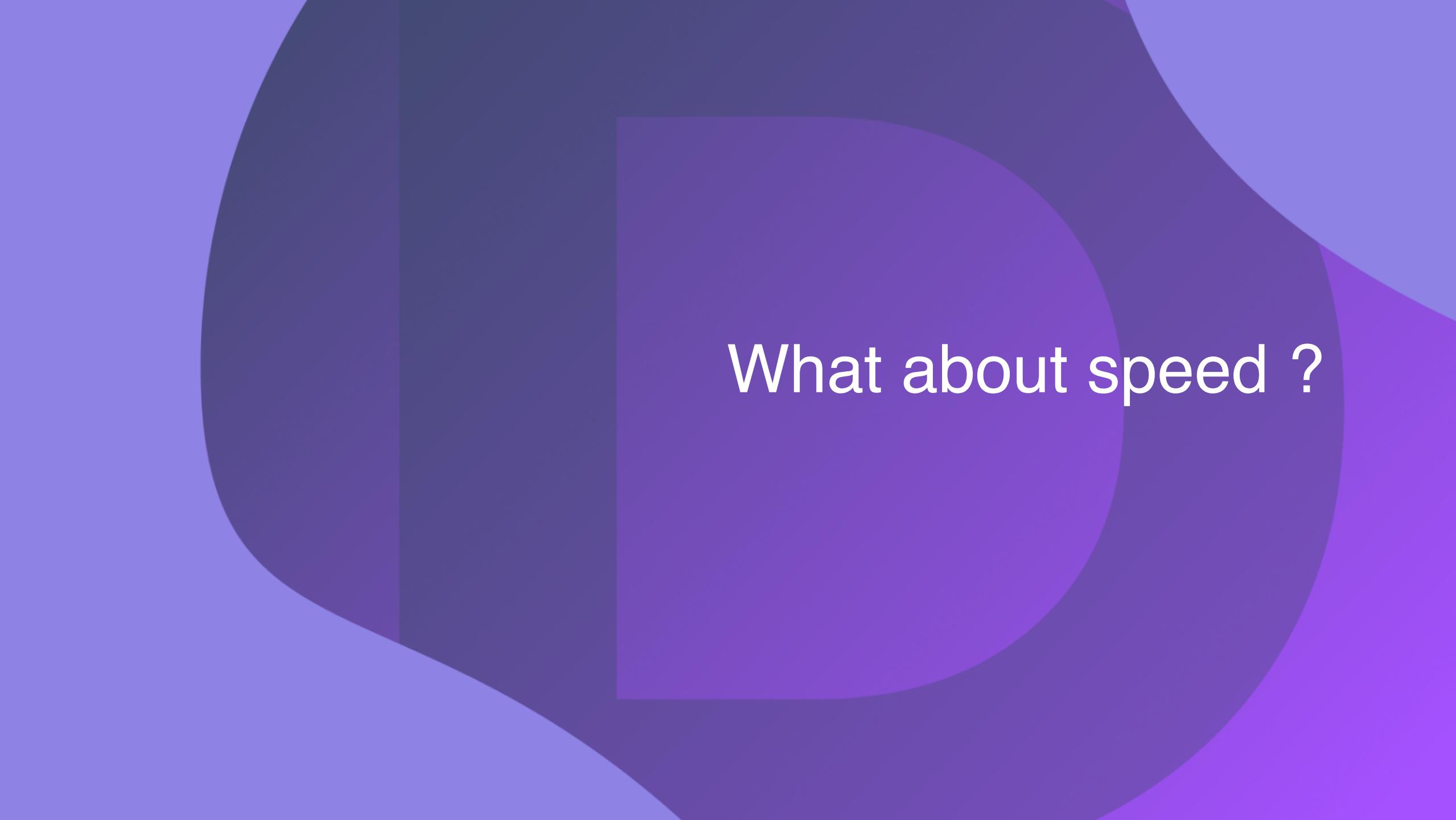
# Prepare data directly into the DB

```
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
        }
```

The background of the slide features a minimalist design with abstract, overlapping circles in various shades of purple. A large, semi-transparent circle in a medium shade of purple covers the central area, while darker and lighter circles are scattered around it, creating a sense of depth and motion.

What about speed ?

# Do not start containers when you do TDD

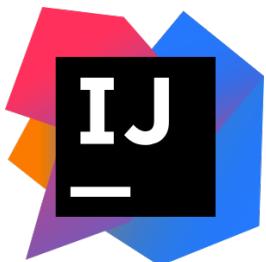


```
spring:  
  datasource:  
    driver-class-name: "org.testcontainers.jdbc.ContainerDatabaseDriver"  
    url: "jdbc:tc:postgresql:13.8://localhost:45432/postgres?TC_REUSEABLE=true"
```

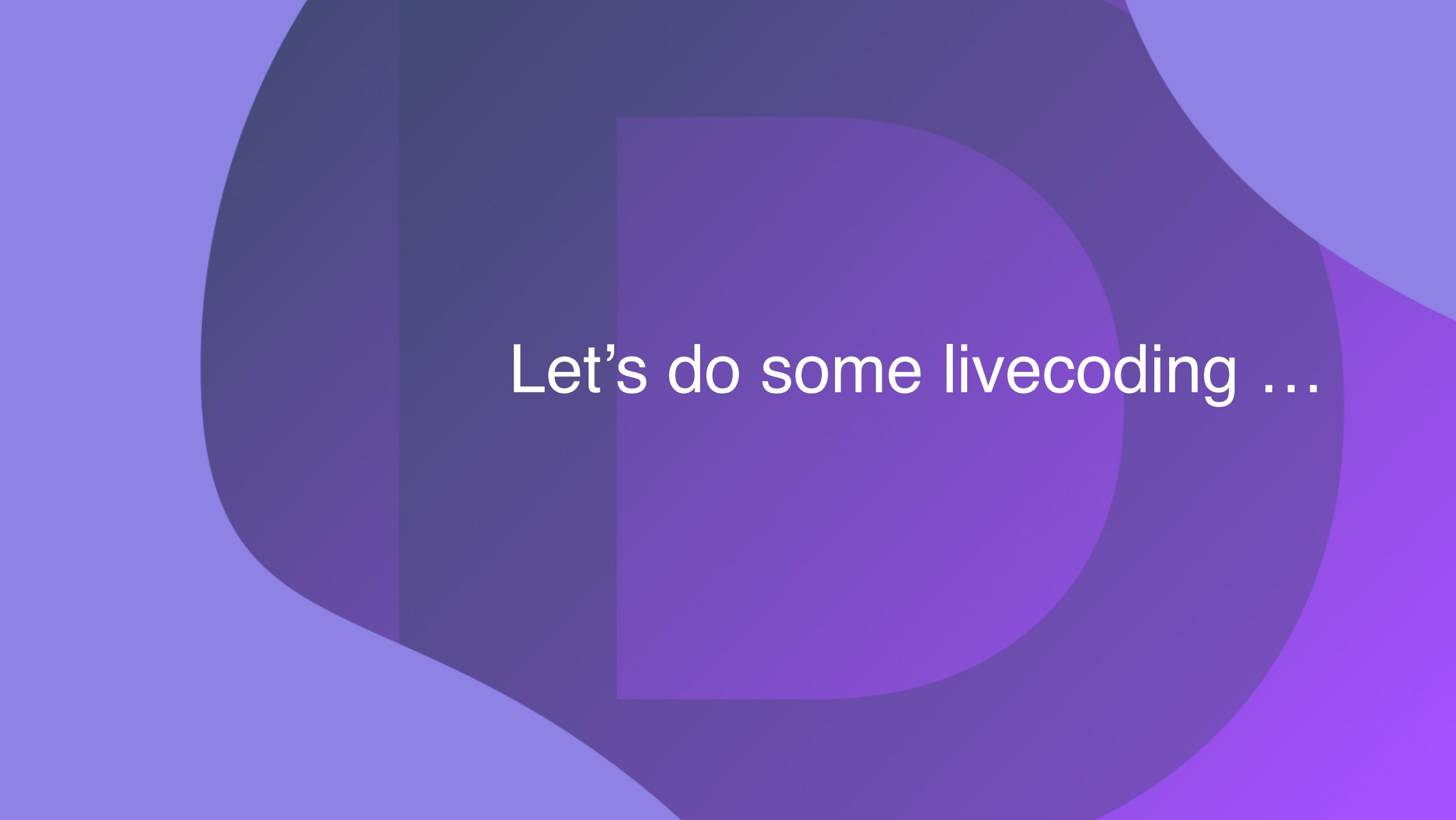
```
val postgresSQLContainer = PostgreSQLContainer<Nothing>(POSTGRES_IMAGE_NAME)  
  .apply { this: PostgreSQLContainer<Nothing>  
    withReuse( reusable: true )  
    start()  
  }
```

# Speed up start and builds

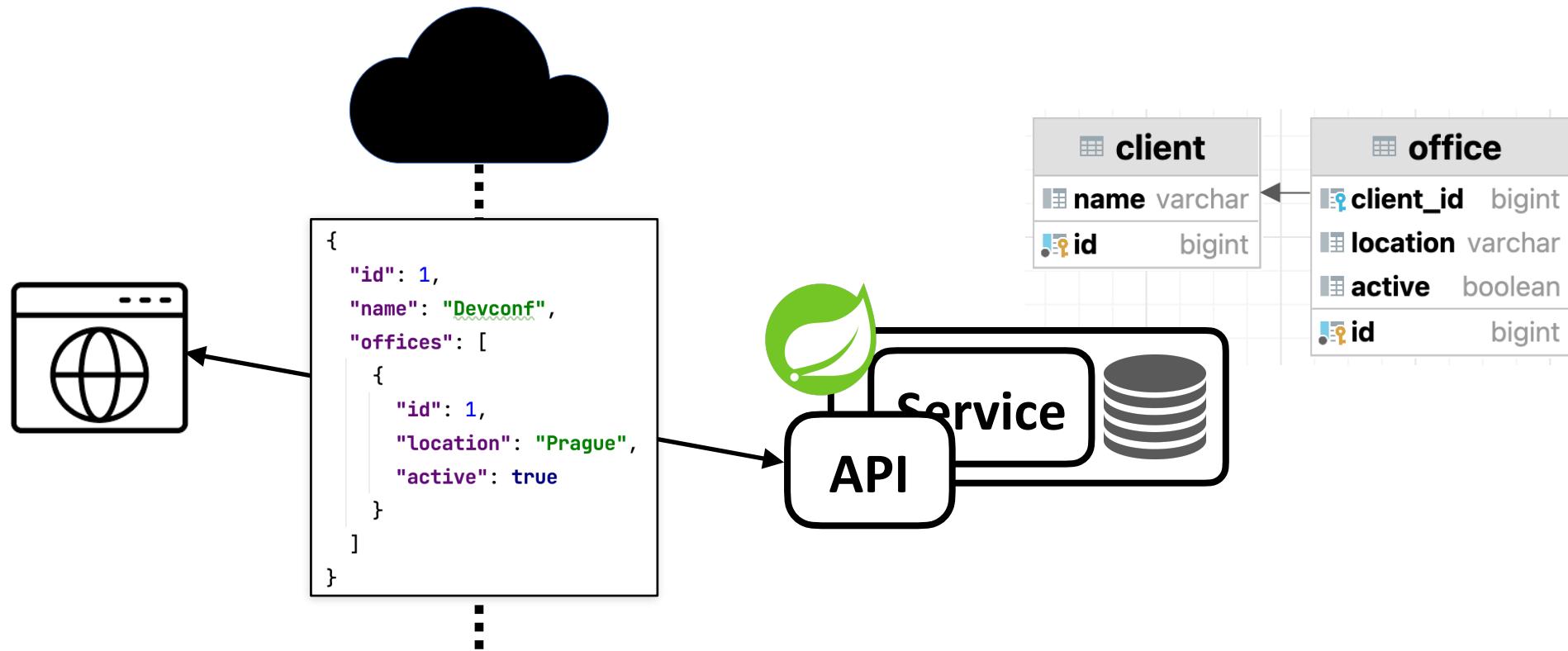
```
./gradlew testClasses  
    --exclude-task generateJooq  
    --exclude-task update  
  
    -PturnOffSlowAction=true
```



- Modified test build
- Run configuration template for Kotest
- Log level = INFO

The background of the slide features a complex, abstract design composed of several overlapping circles in various shades of purple. The circles overlap in a non-uniform manner, creating a sense of depth and movement. The overall effect is modern and minimalist.

Let's do some livecoding ...



## Acceptance criteria from analyst

- I can get existing client
- When office is created, it can be read with client
- When office is deactivated, it is removed from clients' list

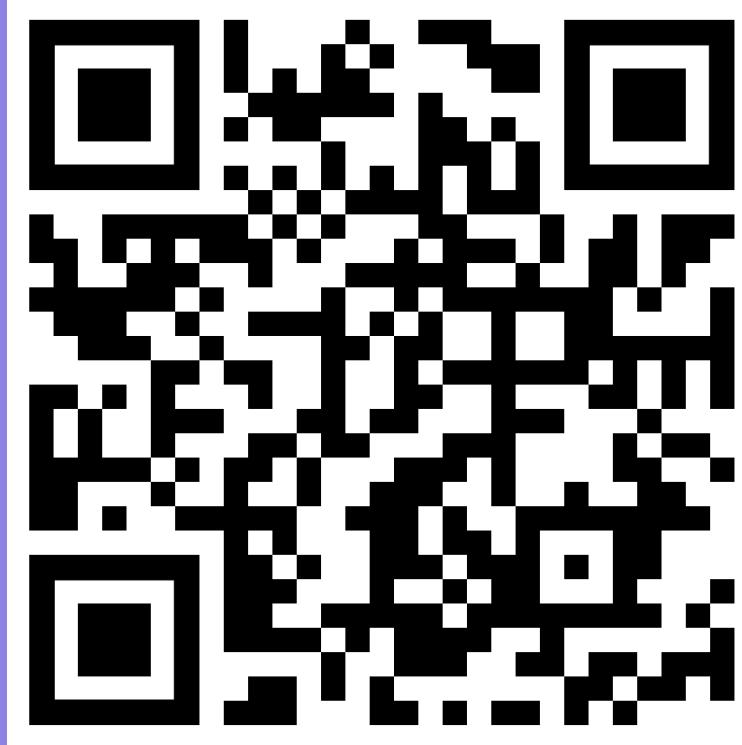
# Summary

- Give TDD a chance in API Tests
  - Encapsulate API calls
  - Use real database
  - Speed up your build
  - Enjoy :)
- 
- <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](mailto:vita.plsek@morosystems.cz) / [www.linkedin.com/in/vitaplsek](https://www.linkedin.com/in/vitaplsek)

# DEVCONF.cz



Thank you for your attention.

Ask me anything!