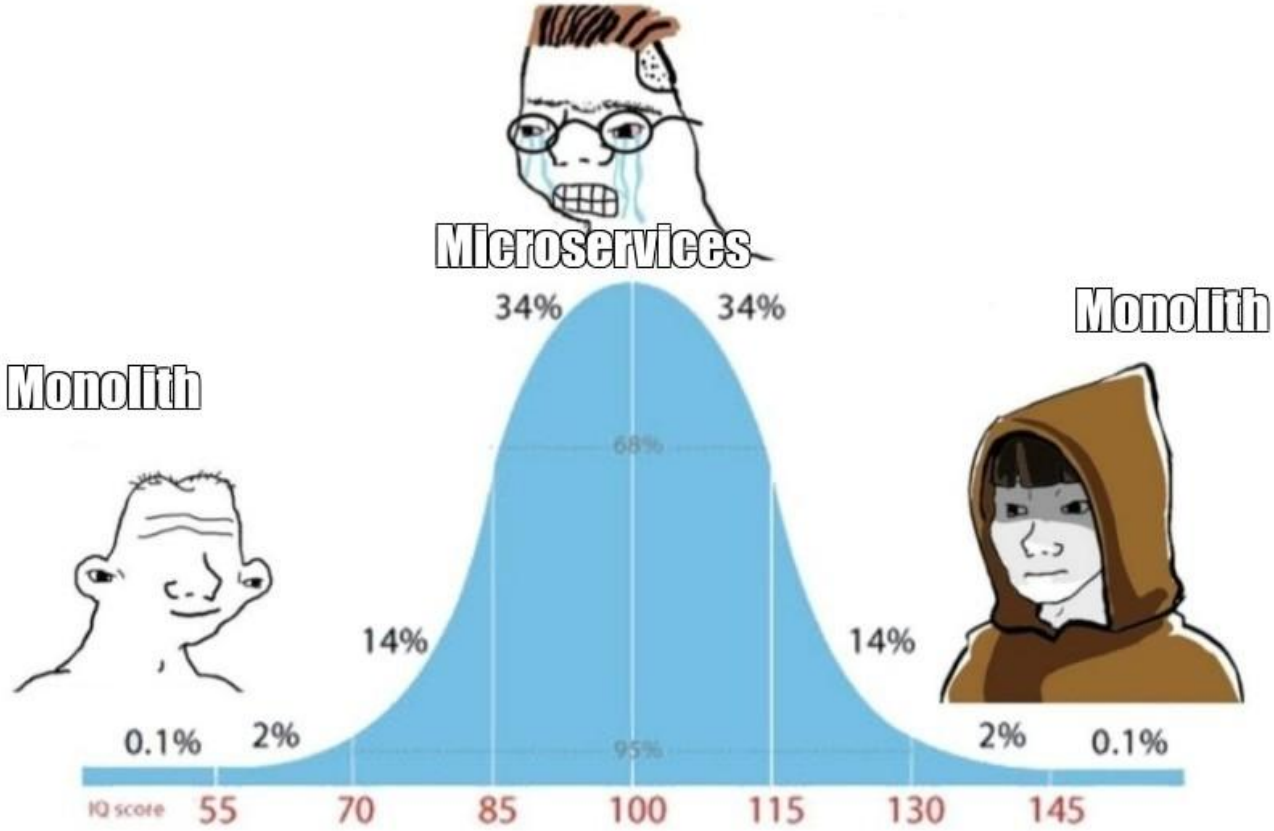
Вадим Бубликов
Газпромбанк

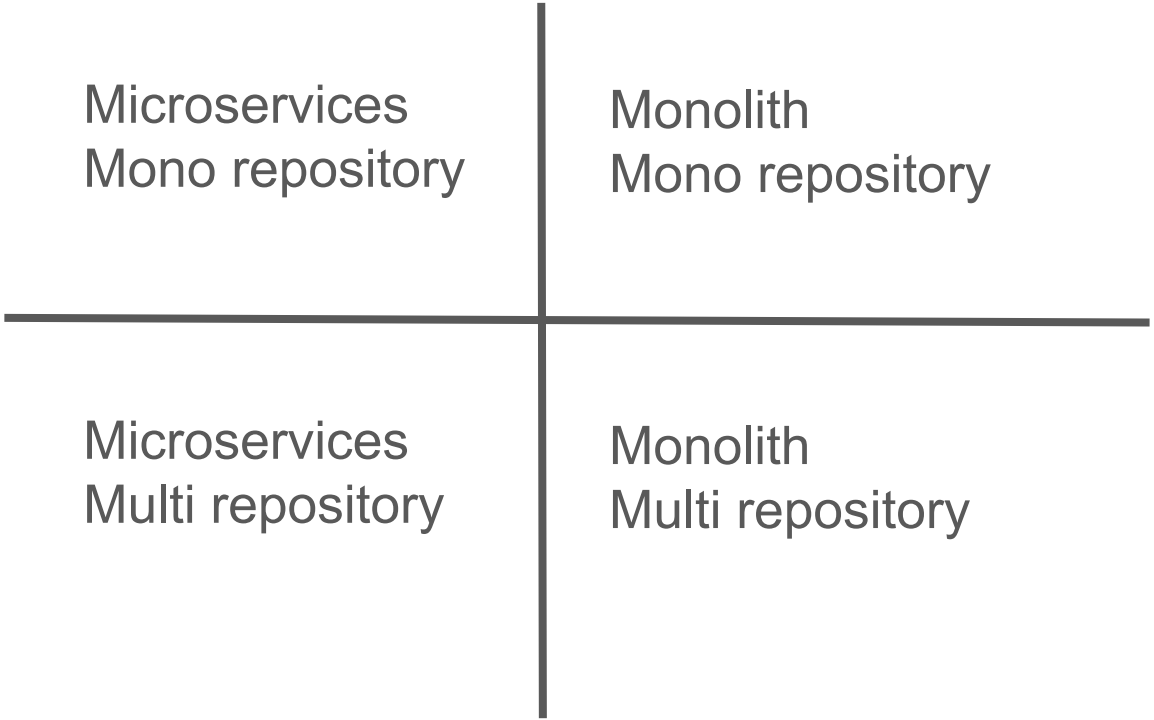
Разрабатываем модульный бэкенд, используя стандартные возможности Spring Boot


НЕТ!
Microservices

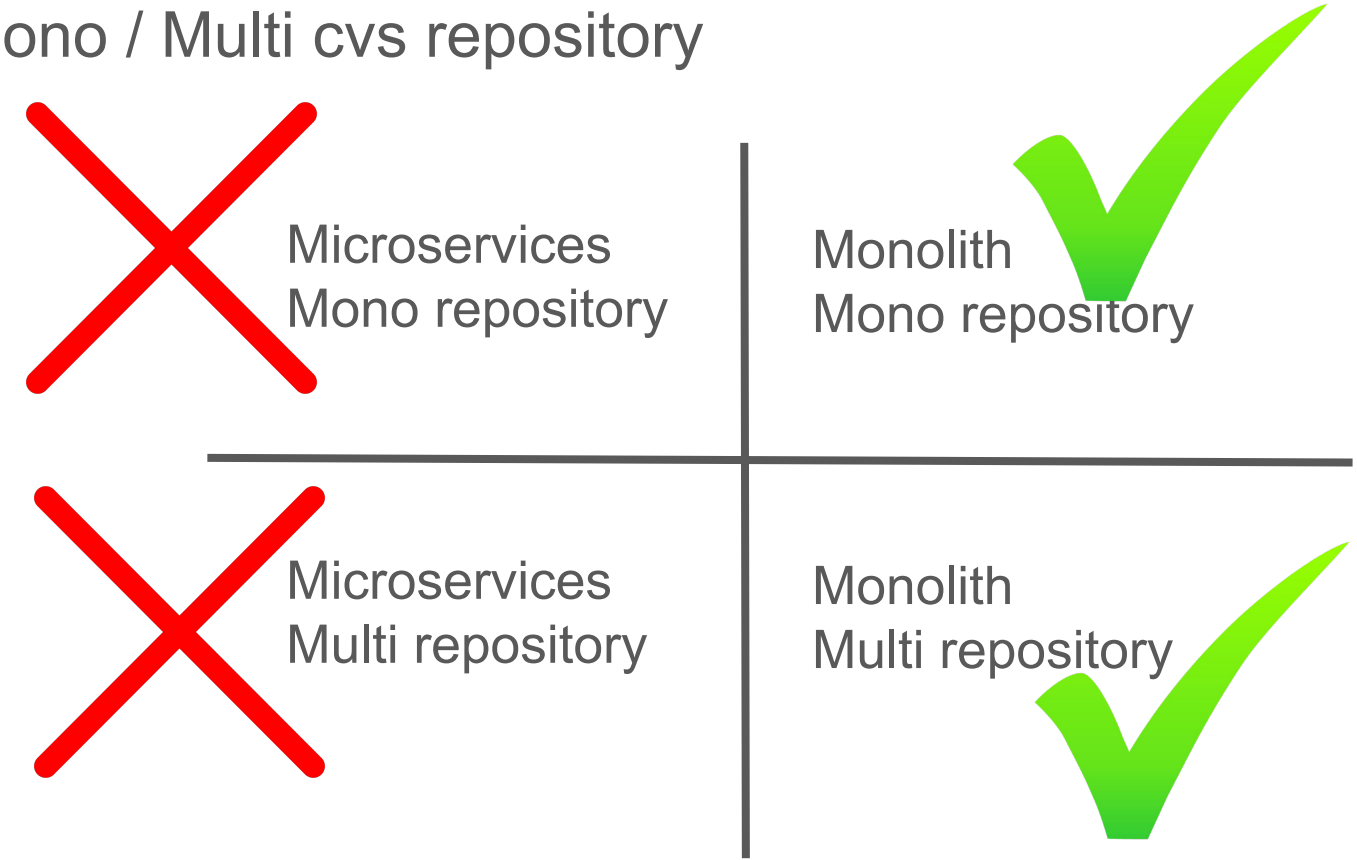

Microservices
Monolith
Monolith
34%
34%
14%
14%
0.1%
2%
2%
0.1%
68%
95%
IQ score
55
70
85
100
115
130
145

Monolith / Microservices architecture
Mono / Multi cvs repository

Monolith / Microservices architecture
Mono / Multi cvs repository



Monolith / Microservices architecture
Mono / Multi cvs repository



План

1. Мотивация
2. Проблемы
3. Classpath модули
4. Spring Modulith

Мотивация выбора в пользу монолитного приложения

1. Мало пользователей, много бизнес-логики

Мотивация выбора в пользу монолитного приложения

1. Мало пользователей, много бизнес-логики

   ➢ корпоративные, бэкофисные приложения
   - учётные системы, ERP, CRM
   - документооборот
   - внутренние порталы

Мотивация выбора в пользу монолитного приложения

1. Мало пользователей, много бизнес-логики
   - ➢ корпоративные, бэкофисные приложения
     - учётные системы, ERP, CRM
     - документооборот
     - внутренние порталы
2. Mvp, startup

Недостатки монолитной архитектуры приложения

Высокая связность компонентов, сложно вносить изменения	
Сложность независимой разработки, длительные релизы, команды разработки блокируются	
Медленная сборка при большом проекте	
Сложность масштабирования СУБД	
Риск недоступности всего приложения из-за сбоя в одном компоненте, например OutOfMemory	

Недостатки монолитной архитектуры приложения

Высокая связность компонентов, сложно вносить изменения	
Сложность независимой разработки, длительные релизы, команды разработки блокируются	
Медленная сборка при большом проекте	
Сложность масштабирования СУБД	
Риск недоступности всего приложения из-за сбоя в одном компоненте, например OutOfMemory	

Недостатки монолитной архитектуры приложения

Высокая связность компонентов, сложно вносить изменения	Декомпозируйте предметную область, применяйте модульную структуру
Сложность независимой разработки, длительные релизы, команды разработки блокируются	Модули как отдельные проекты, мульти репозитории
Медленная сборка при большом проекте	Модули как отдельные проекты
Сложность масштабирования СУБД	
Риск недоступности всего приложения из-за сбоя в одном компоненте, например OutOfMemory	

Недостатки монолитной архитектуры приложения

Высокая связность компонентов, сложно вносить изменения	Декомпозируйте предметную область, применяйте модульную структуру
Сложность независимой разработки, длительные релизы, команды разработки блокируются	Модули как отдельные проекты, мульти репозитории
Медленная сборка при большом проекте	Модули как отдельные проекты, мульти репозитории
Сложность масштабирования СУБД	Кеширование, индексирование, шардирование, партиционирование, анализ статистики; возможность увеличения аппаратных ресурсов. Рассмотреть применение распределенных СУБД, например YDB.
Риск недоступности всего приложения из-за сбоя в одном компоненте, например OutOfMemory	Работа с несколькими репликами, горизонтальное масштабирование

Модульное монолитное приложение

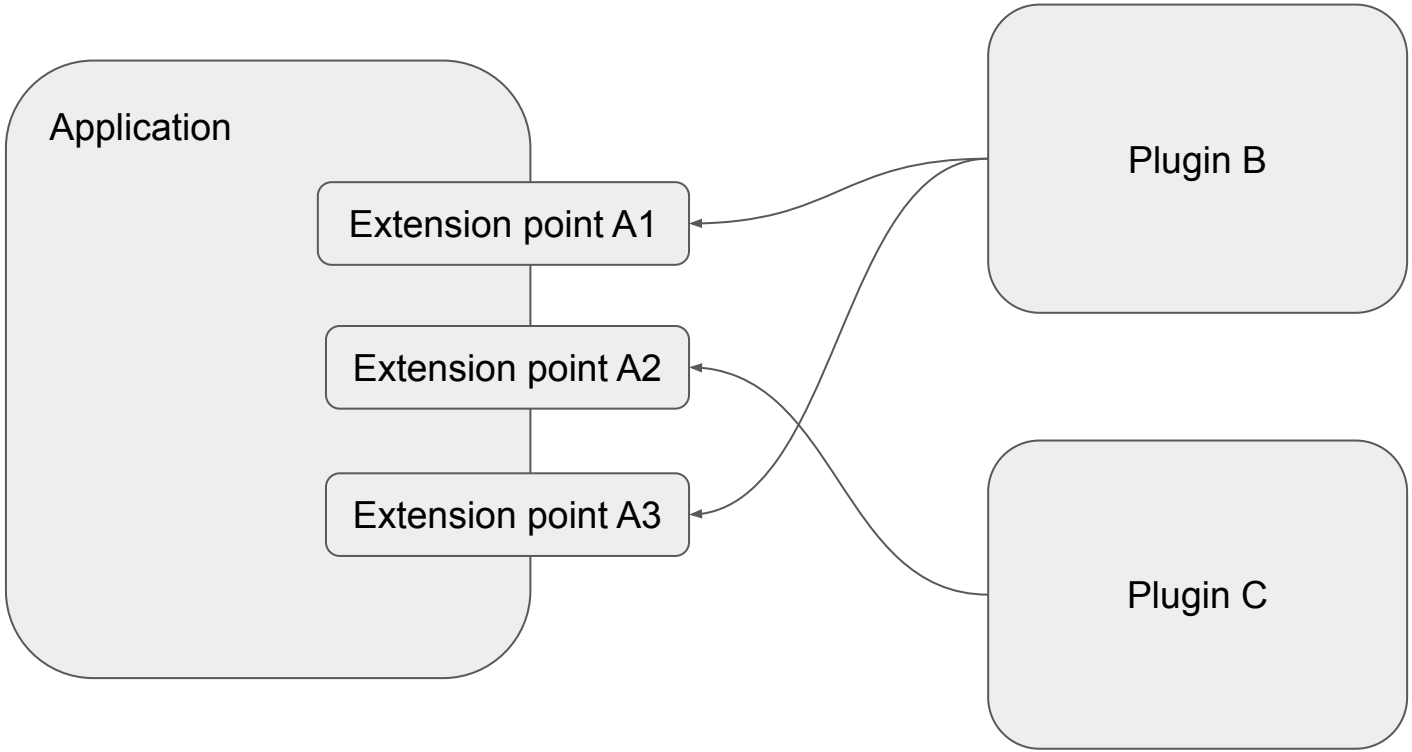
Что есть модуль?

Что есть модуль?

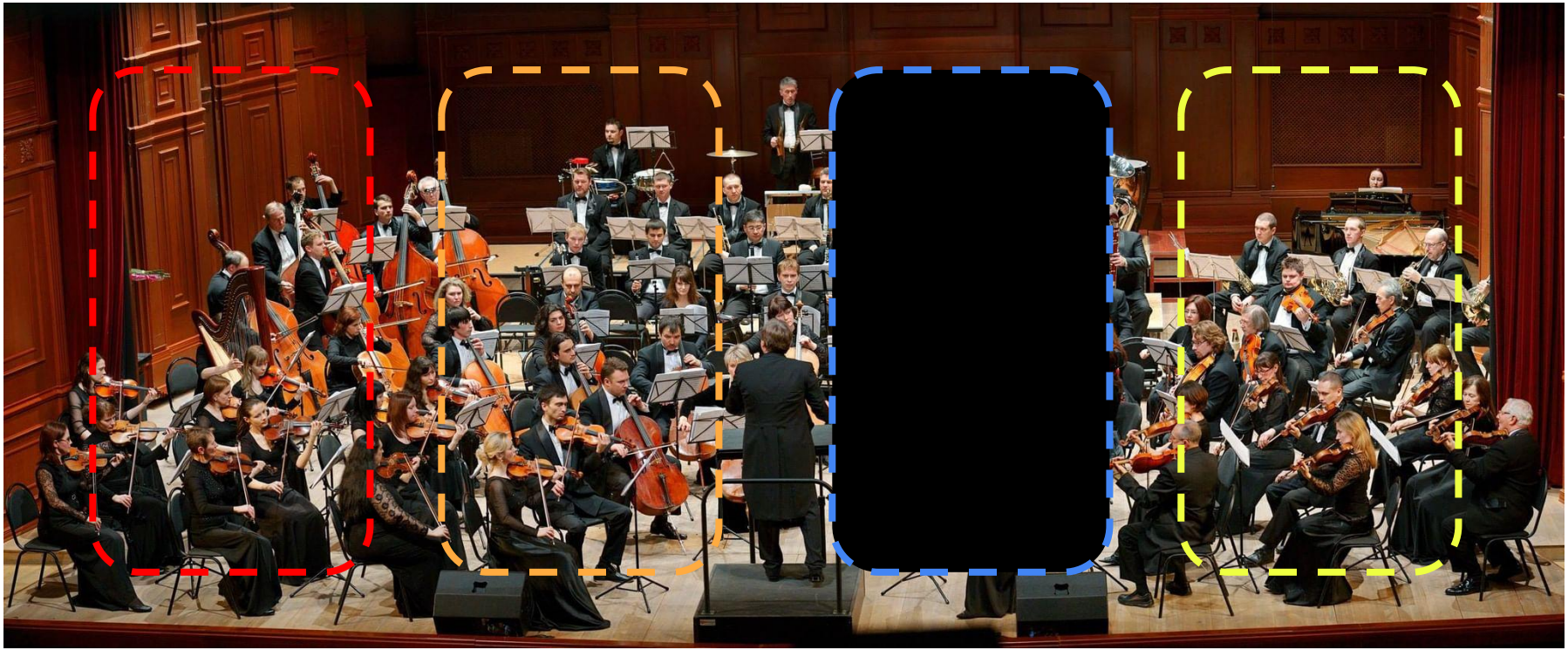
MODULE

VS

PLUGIN


Application
Extension point A1
Extension point A2
Extension point A3
Plugin B
Plugin C

Что есть модуль?



MODULE

VS

PLUGIN

- Classpath modules
- Spring modulith
- Gradle subprojects
  Maven modules

- Spring boot starters
- Java modules (JPMS)
- Jakarta/JEE application

- PF4J
- OSGI
- Service provider interface (SPI)

- Classpath modules
- Spring modulith
- Gradle subprojects
  Maven modules

- Spring boot starters
- Java modules (JPMS)
- Jakarta/JEE application

- PF4J
- OSGI
- Service provider interface (SPI)

Classpath modules

2021 год

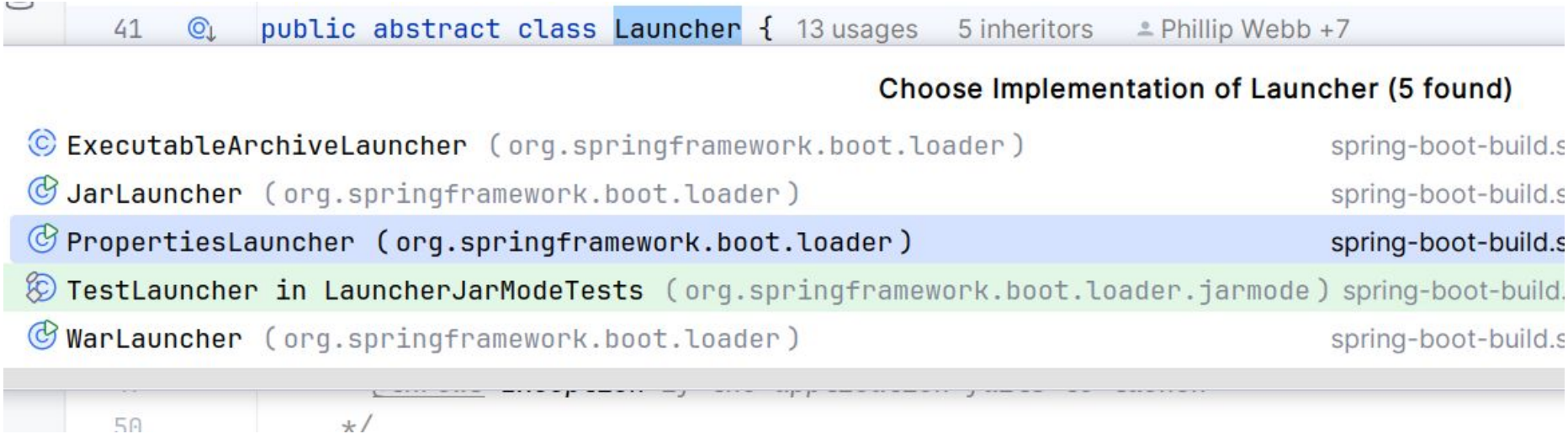
Как запускается spring boot приложение

```
1.  java -jar spring-app.jar
```

Как запускается spring boot приложение

1. `java -jar spring-app.jar`

2. `java -cp spring-app.jar o.sf.b.l.JarLauncher`


41
public abstract class Launcher { 13 usages 5 inheritors Phillip Webb +7
Choose Implementation of Launcher (5 found)
ExecutableArchiveLauncher (org.springframework.boot.loader) spring-boot-build.s
JarLauncher (org.springframework.boot.loader) spring-boot-build.s
PropertiesLauncher (org.springframework.boot.loader) spring-boot-build.s
TestLauncher in LauncherJarModeTests (org.springframework.boot.loader.jarmode) spring-boot-build.
WarLauncher (org.springframework.boot.loader) spring-boot-build.s
50
*/

Как запускается spring boot приложение

1. `java -jar spring-app.jar`

2. `java -cp spring-app.jar o.sf.b.l.JarLauncher`

   `java -cp spring-app.jar o.sf.b.l.PropertiesLauncher`

PropertiesLauncher Features

`PropertiesLauncher` has a few special features that can be enabled with external properties (System properties, environment variables, manifest entries, or `loader.properties`). The following table describes these properties:

Key	Purpose
`loader.path`	Comma-separated Classpath, such as `lib,${HOME}/app/lib`. Earlier entries take precedence, like a regular `-classpath` on the `javac` command line.
`loader.home`	Used to resolve relative paths in `loader.path`. For example, given `loader.path=lib`, then `${loader.home}/lib` is a classpath location (along with all jar files in that directory). This property is also used to locate a `loader.properties` file, as in the following example `/opt/app` It defaults to `${user.dir}`.
`loader.args`	Default arguments for the main method (space separated).
`loader.main`	Name of main class to launch (for example, `com.app.Application`).
`loader.config.name`	Name of properties file (for example, `launcher`). It defaults to `loader`.
`loader.config.location`	Path to properties file (for example, `classpath:loader.properties`). It defaults to `loader.properties`.
`loader.system`	Boolean flag to indicate that all properties should be added to System properties. It defaults to `false`.

Варианты сформировать classpath на этапе старта spring boot приложения

```
1.  java -cp spring-app.jar:spring-lib.jar
    o.sf.b.l.JarLauncher
```

Варианты сформировать classpath на этапе старта spring boot приложения

1. `java -cp spring-app.jar:spring-lib.jar o.sf.b.l.JarLauncher`

2. `java -Dloader.path=spring-lib.jar -cp spring-app.jar o.sf.b.l.PropertiesLauncher`

Сделаем библиотеку, содержащую bean

```
dependencies {
    compileOnly("org.springframework:spring-context")
    compileOnly("jakarta.annotation:jakarta.annotation-api")
}
```

```
@Component //org.springframework:spring-context
public class SomeBean {

    @PostConstruct //jakarta.annotation:jakarta.annotation-api
    void init() {
        System.out.println("Bean from lib");
    }

}
```

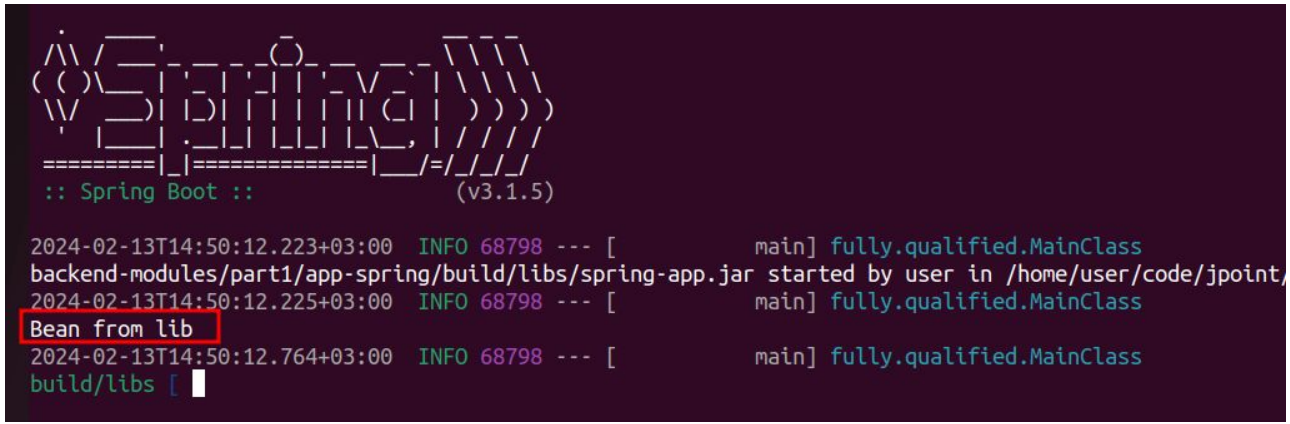
Варианты сформировать classpath на этапе старта spring boot приложения

1. `java -cp spring-app.jar:spring-lib.jar o.sf.b.l.JarLauncher` bean не инициализирован

Варианты сформировать classpath на этапе старта spring boot приложения

1. `java -cp spring-app.jar:spring-lib.jar o.sf.b.l.JarLauncher` bean не инициализирован

2. `java -Dloader.path=spring-lib.jar -cp spring-app.jar o.sf.b.l.PropertiesLauncher` bean инициализирован



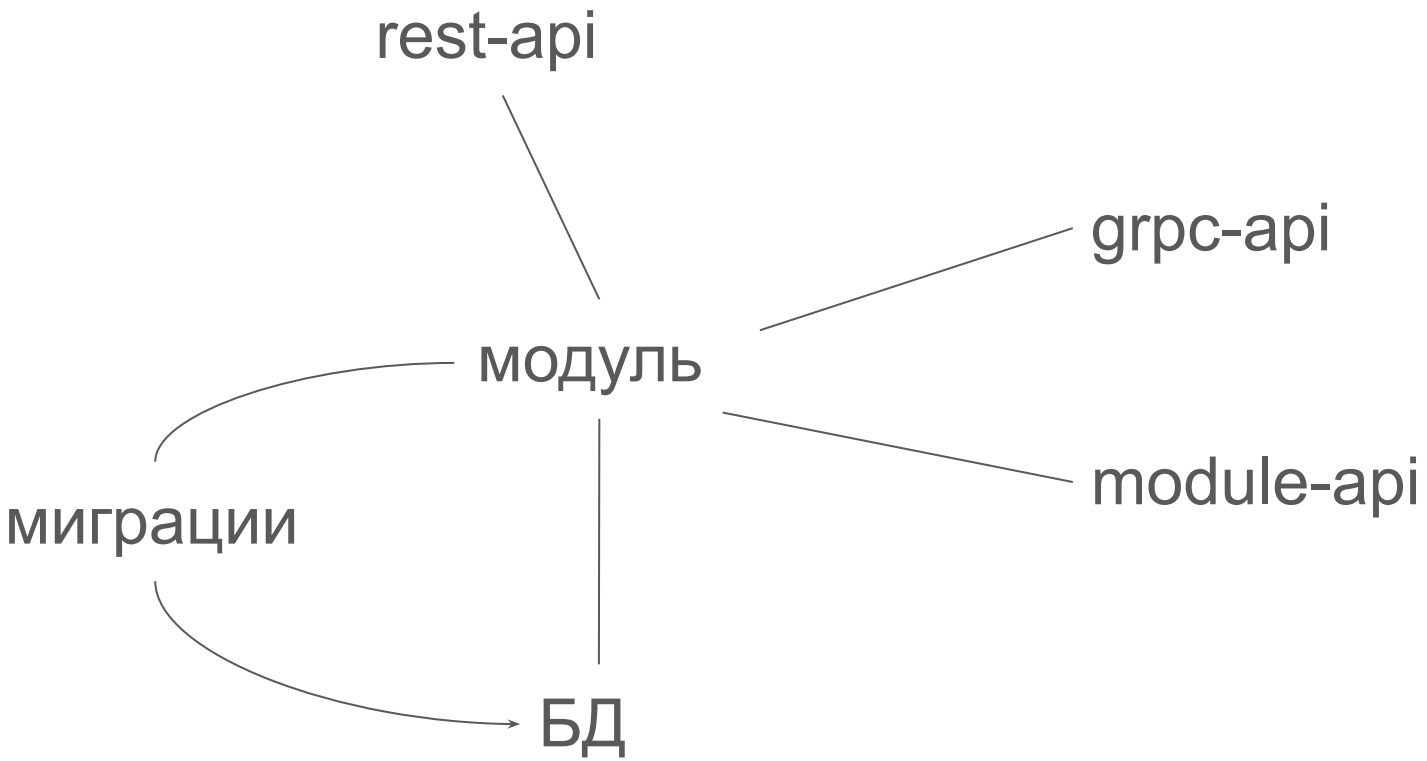
Варианты сформировать classpath на этапе старта spring boot приложения

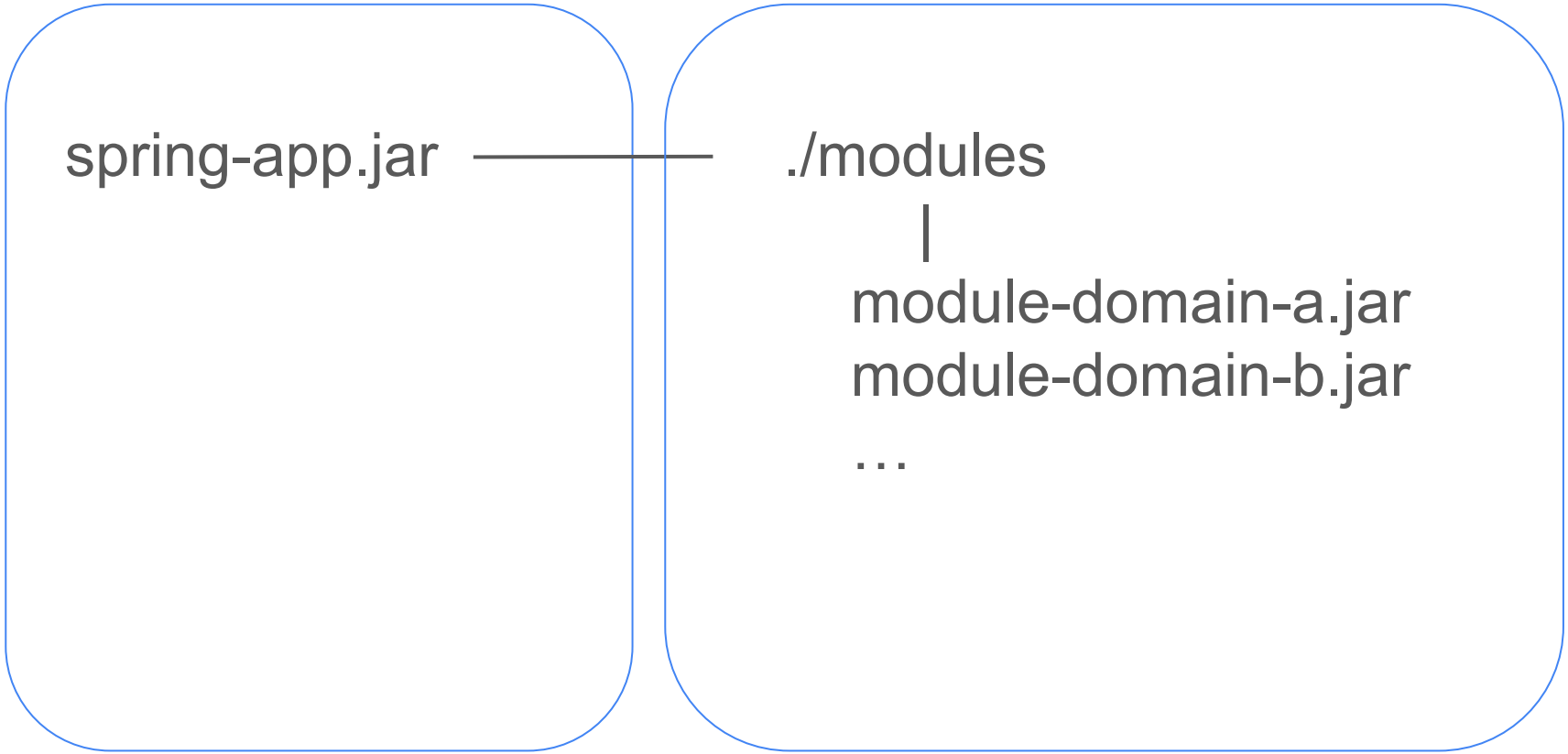
```
java -Dloader.path=spring-lib.jar -cp spring-app.jar
o.sf.b.l.PropertiesLauncher
```

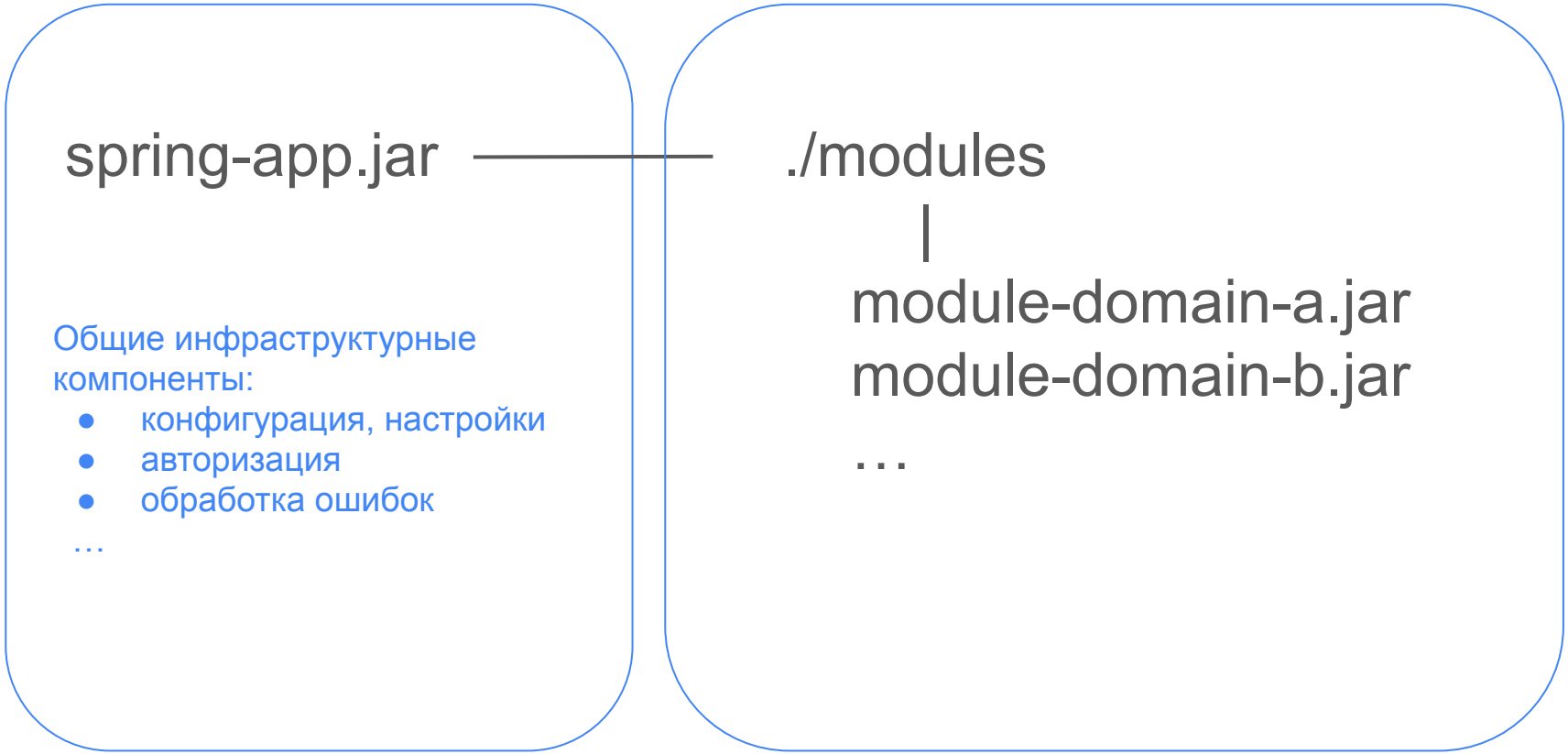
Варианты сформировать classpath на этапе старта spring boot приложения

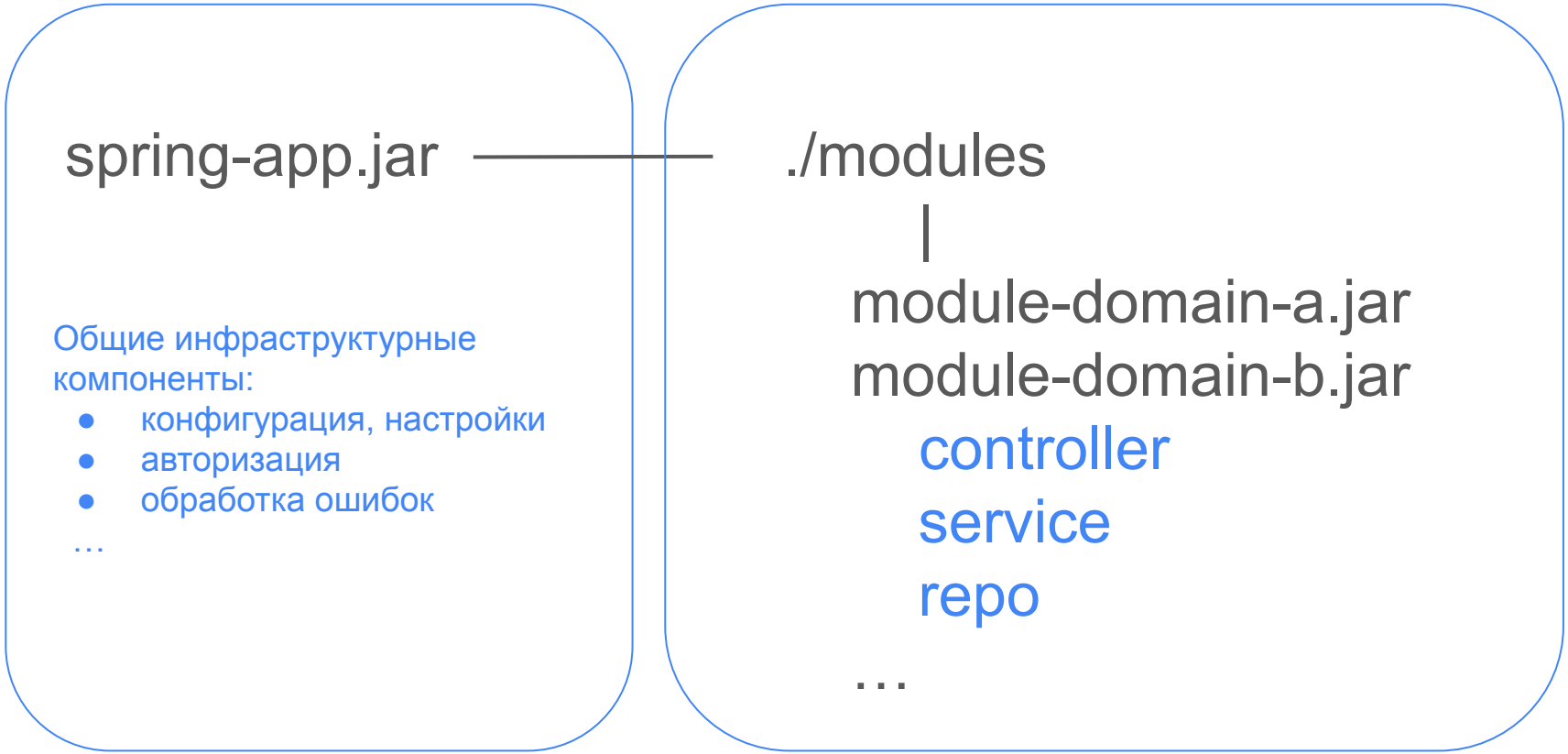
```
java -Dloader.path=spring-lib.jar -cp spring-app.jar
o.sf.b.l.PropertiesLauncher
```

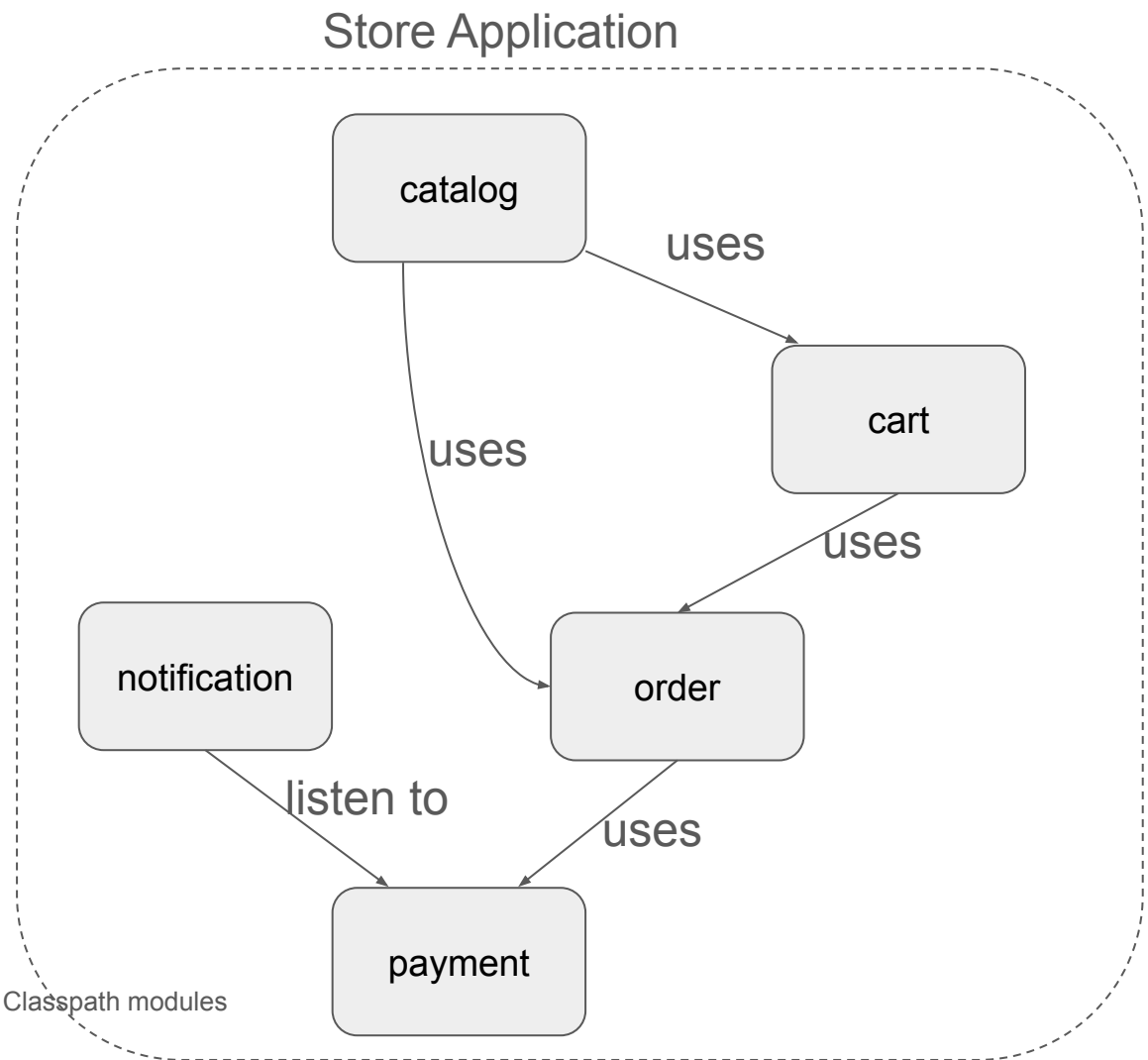
```
java -Dloader.path=./modules/ -cp spring-app.jar
o.sf.b.l.PropertiesLauncher
```


rest-api
grpc-api
модуль
module-api
миграции
БД

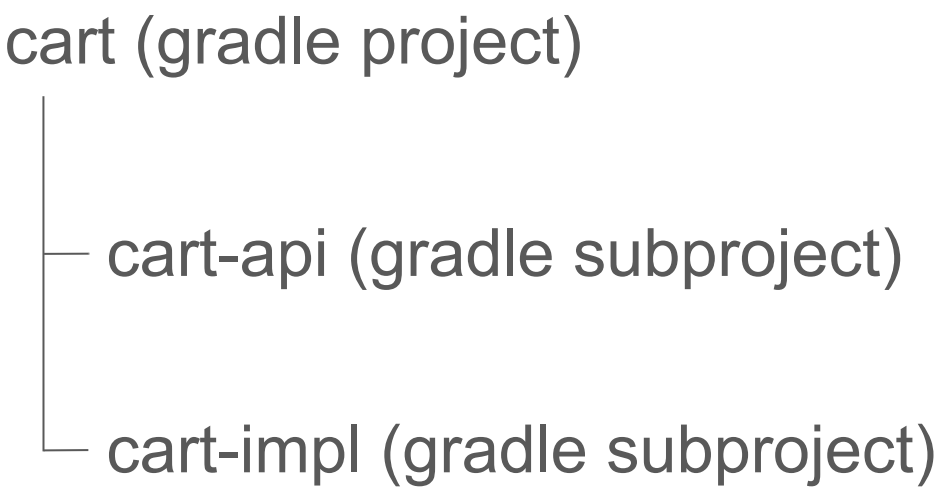

spring-app.jar
./modules
|
module-domain-a.jar
module-domain-b.jar
…

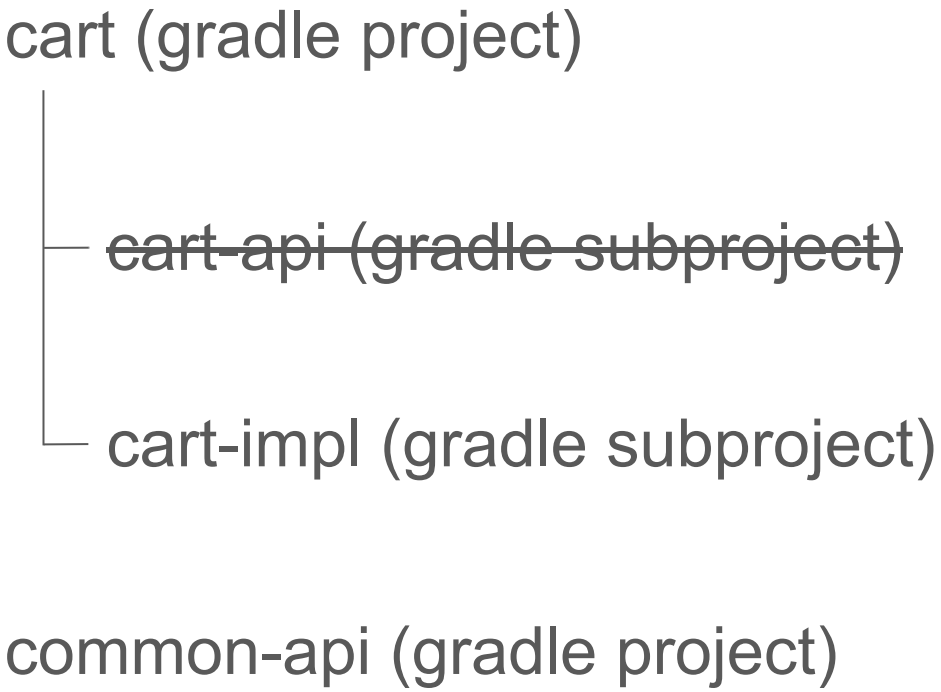

spring-app.jar
./modules
|
module-domain-a.jar
module-domain-b.jar
…
Общие инфраструктурные компоненты:
конфигурация, настройки
авторизация
обработка ошибок
…

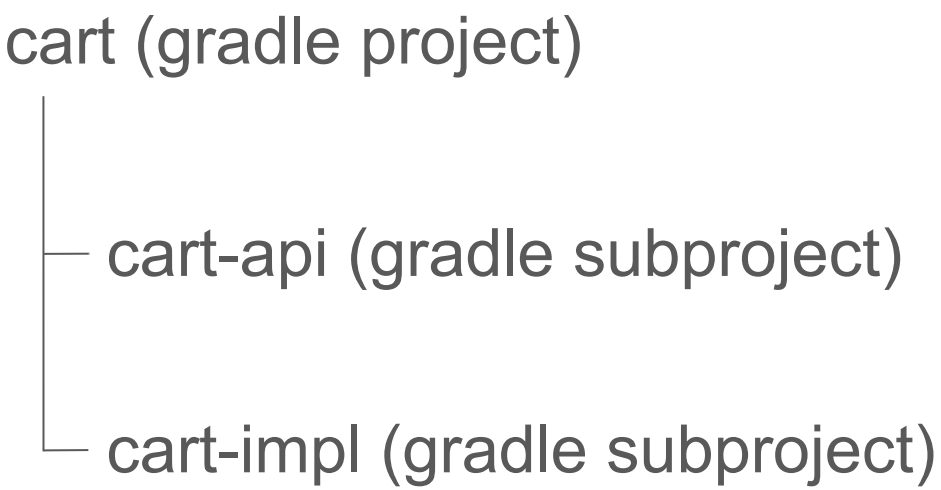

spring-app.jar
Общие инфраструктурные компоненты:
конфигурация, настройки
авторизация
обработка ошибок
…
./modules
|
module-domain-a.jar
module-domain-b.jar
controller
service
repo
…

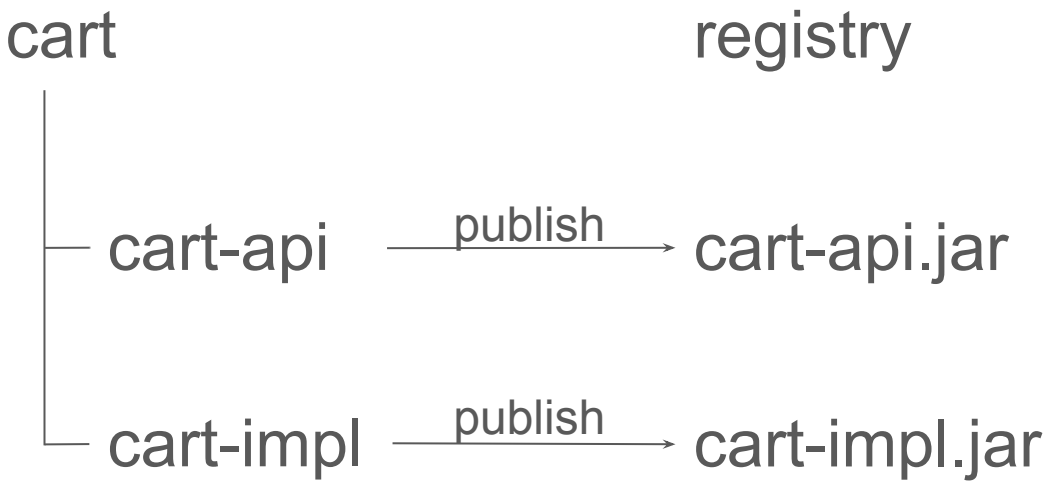


https://github.com/vadimblikov/backend-modules


cart (gradle project)
cart-api (gradle subproject)
cart-impl (gradle subproject)


cart (gradle project)
cart-api (gradle subproject)
cart-impl (gradle subproject)
common-api (gradle project)


cart (gradle project)
cart-api (gradle subproject)
cart-impl (gradle subproject)


cart
registry
cart-api
publish
cart-api.jar
cart-impl
publish
cart-impl.jar

registry

cart-impl.jar

deploy to
a server

registry

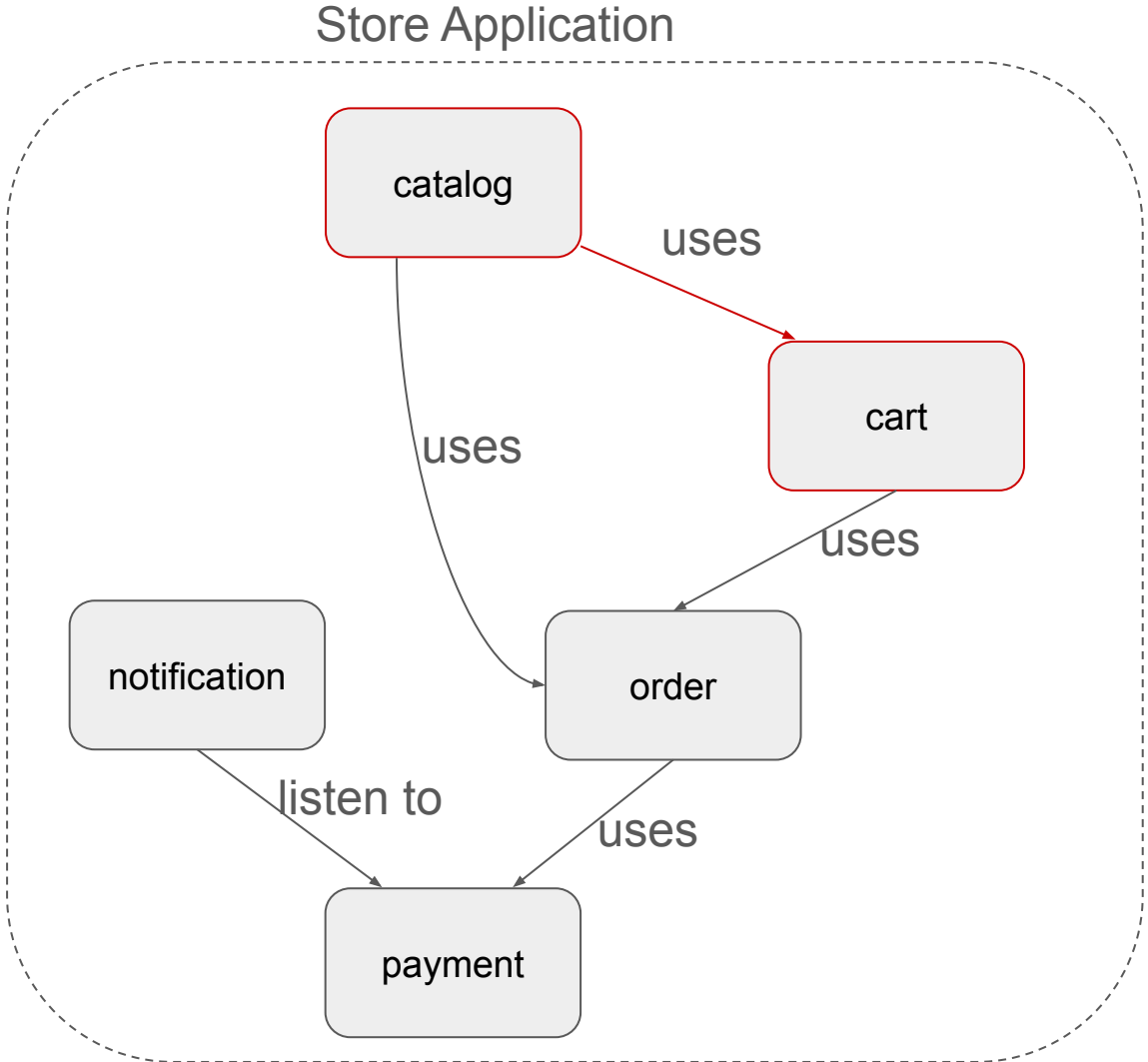
cart-impl.jar

deploy to
a server

cart-api.jar

catalog-impl.jar

```
dependencies {

    compileOnly("tech.cmodule.store:cart-api")

}
```


Store Application
catalog
uses
cart
uses
uses
notification
order
listen to
uses
payment

registry

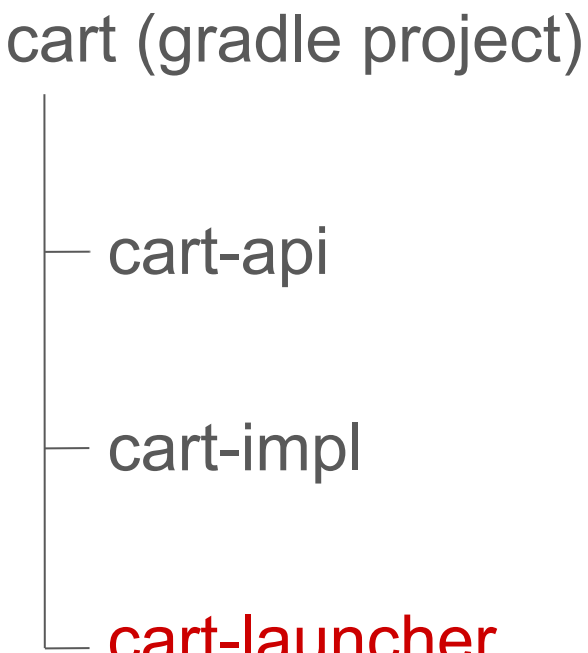
cart-impl.jar

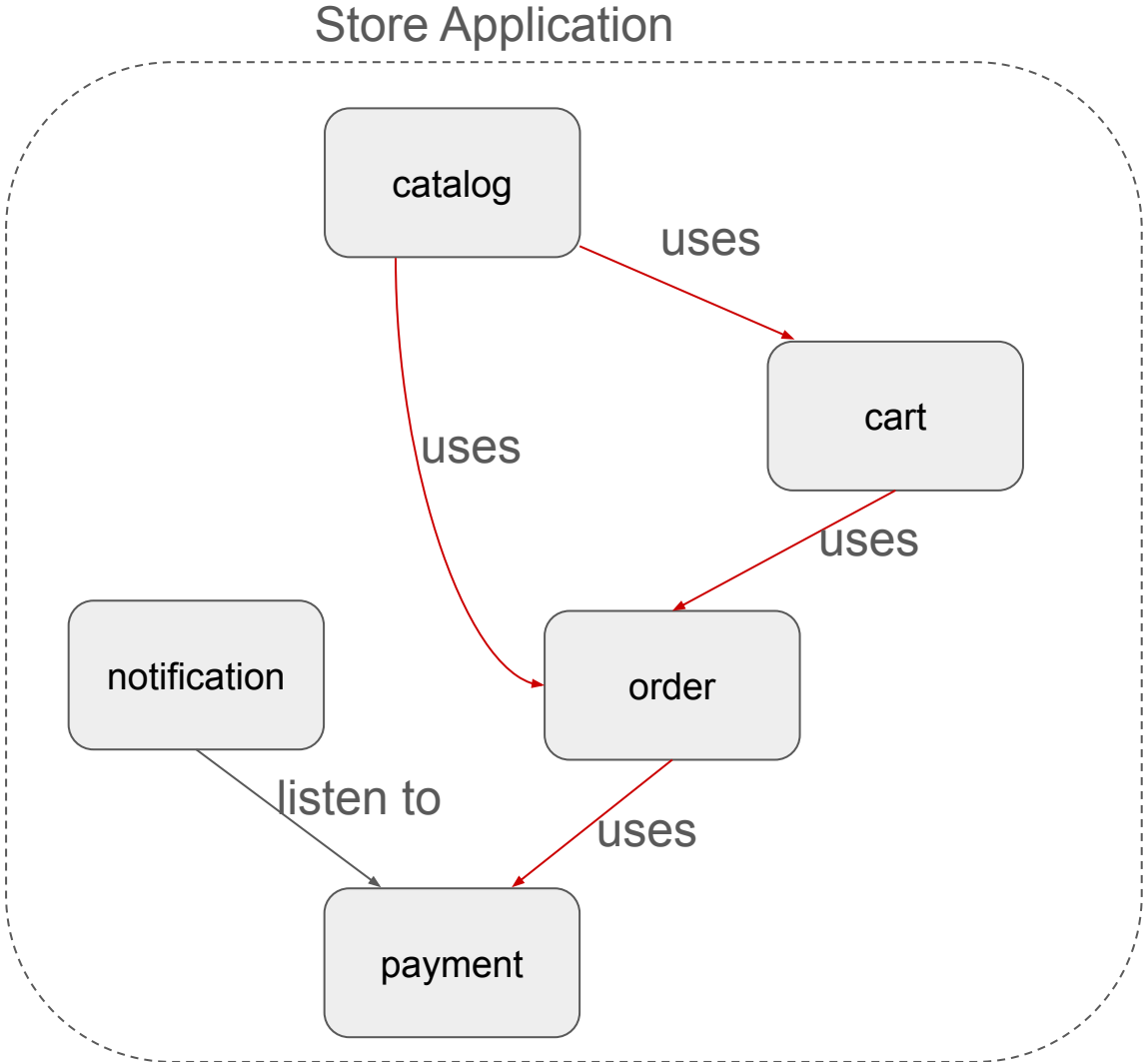
cart-api.jar

use

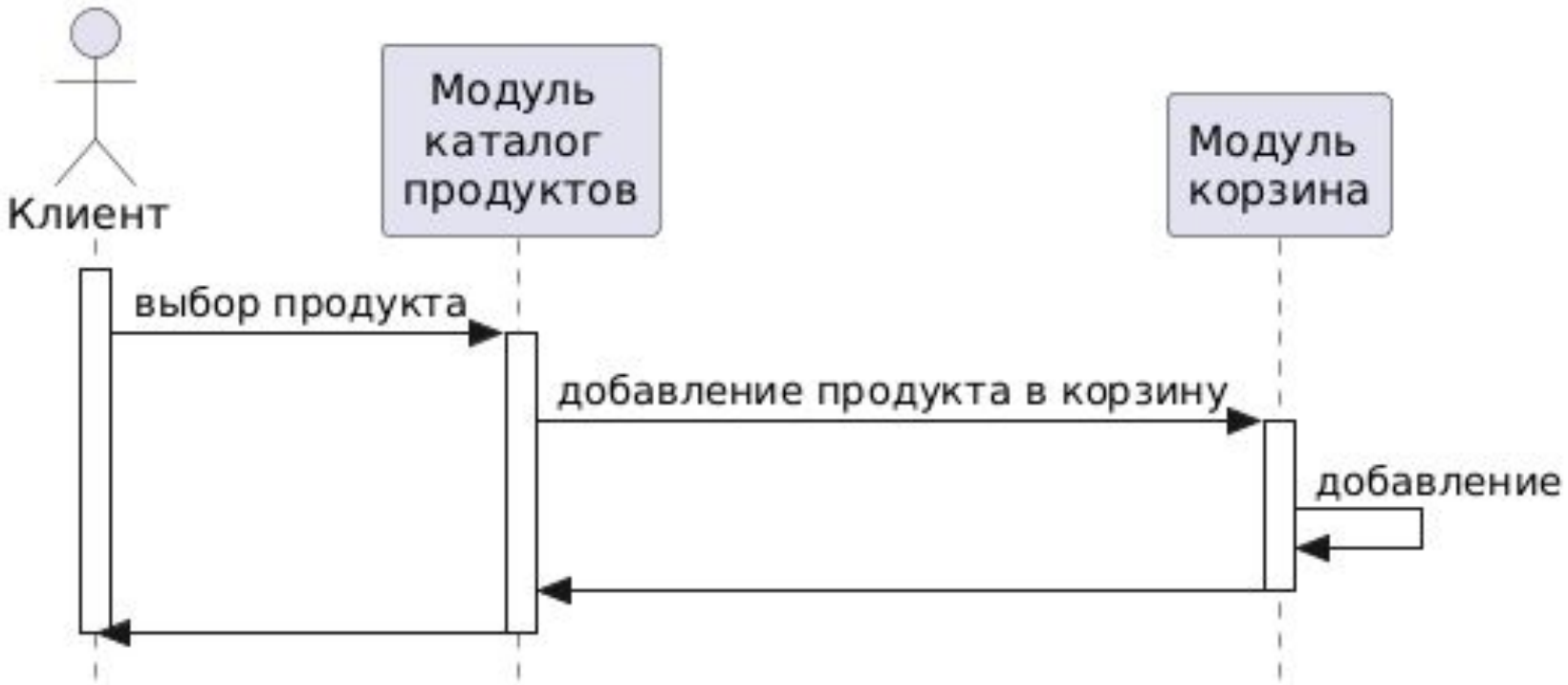
catalog-impl.jar


```
dependencies {

    compileOnly("tech.cmodule.store:cart-api")

}
```


cart (gradle project)
cart-api
cart-impl
cart-launcher


Store Application
catalog
uses
cart
uses
uses
notification
order
listen to
uses
payment

Пример межмодульного вызова



Взаимодействие модулей - синхронные вызовы


```
@Service
@Transactional
class CatalogService(
    private val cart: ICartFacade
) {

    fun productToCart(productId: String): String {
        cart.createItem(CreateItemRq(productId))
        ...
    }

}
```

Взаимодействие модулей - обратная совместимость

```
interface CartFacade {
    fun productToCart(productId: String)
}



class CartService: CartFacade {
    override fun productToCart(productId: String) {

    }
}
```

Взаимодействие модулей - обратная совместимость

```
interface CartFacade {
    fun productToCart(productId: String, quantity: Int)
}



class CartService: CartFacade {
    override fun productToCart(productId: String, quantity: Int) {

    }
}
```

Взаимодействие модулей - обратная совместимость

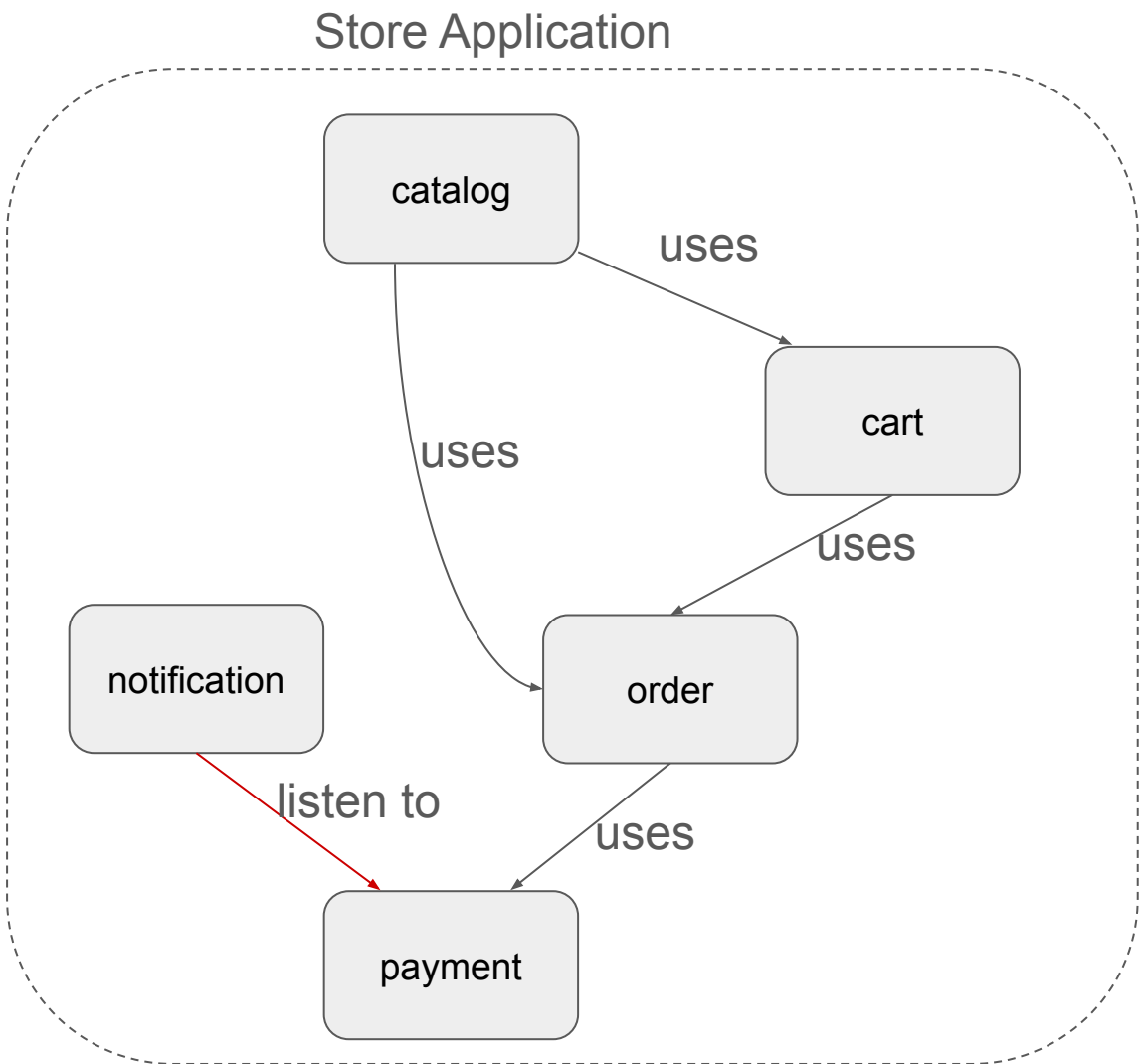
```
interface CartFacade {
    fun productToCart(productId: String, quantity: Int = 1)
}



class CartService: CartFacade {
    override fun productToCart(productId: String, quantity: Int) {

    }
}
```

Взаимодействие модулей - обратная совместимость

```
interface CartFacade {
    @JvmOverloads
    fun productToCart(productId: String, quantity: Int = 1)
}

class CartService: CartFacade {
    @JvmOverloads
    override fun productToCart(productId: String, quantity: Int) {

    }
}
```

Взаимодействие модулей - обратная совместимость

```
interface CartFacade {
    fun productToCart(rq: CreateItemRq)
}

class CartService: CartFacade {
    override fun productToCart(rq: CreateItemRq) {

    }
}

data class CreateItemRq @JvmOverloads constructor(
    val productId: String,
    val quantity: Int = 1
)
```



Classpath modules

Spring Events

```
@SpringBootApplication
class StoreApplication

fun main(args: Array<String>) {
    runApplication<StoreApplication>(*args)
}

@Component
class ReadyListener : ApplicationListener<ApplicationReadyEvent> {

    override fun onApplicationEvent(event: ApplicationReadyEvent) {
        println("Application ready event")
    }

}
```

```
@SpringBootApplication
class StoreApplication

fun main(args: Array<String>) {
    runApplication<StoreApplication>(*args)
}
```

```
@Component
class ReadyListener {

    @EventListener
    fun applicatonReady(event: ApplicationReadyEvent) {
        println("Application ready event")
    }

}
```

```
@SpringBootApplication
class StoreApplication

fun main(args: Array<String>) {
    runApplication<StoreApplication>(*args)
}

@Component
class ReadyListener(val publisher: ApplicationEventPublisher) {

    @EventListener
    fun applicatonReady(event: ApplicationReadyEvent) {
        publisher.publishEvent( MyEvent( text = "Application ready event") )
    }

}

data class MyEvent (
    val text: String
)
```

```
dependencies {

    implementation("org.springframework.boot:spring-boot-starter-jdbc" )

}
```

```
@Component
class MyListener {

    @TransactionalEventListener (phase = TransactionPhase.AFTER_COMMIT)
    fun onEvent(event: MyEvent) {
        println("after commit event")
    }

}
```

```
dependencies {

    implementation("org.springframework.boot:spring-boot-starter-jdbc" )

}
```

```
@Component
class MyListener {

    @TransactionalEventListener (phase = TransactionPhase.AFTER_COMMIT)
    @Async
    fun onEvent(event: MyEvent) {
        println("after commit event")
    }

}
```

```
dependencies {

    implementation("org.springframework.modulith:spring-modulith-events-core" )
    implementation("org.springframework.modulith:spring-modulith-events-jdbc" )
    implementation("org.springframework.modulith:spring-modulith-events-jack" )

}
```

application.properties

```
spring.modulith.events.jdbc.schema-initialization.enabled=true

spring.modulith.republish-outstanding-events-on-restart=true
```

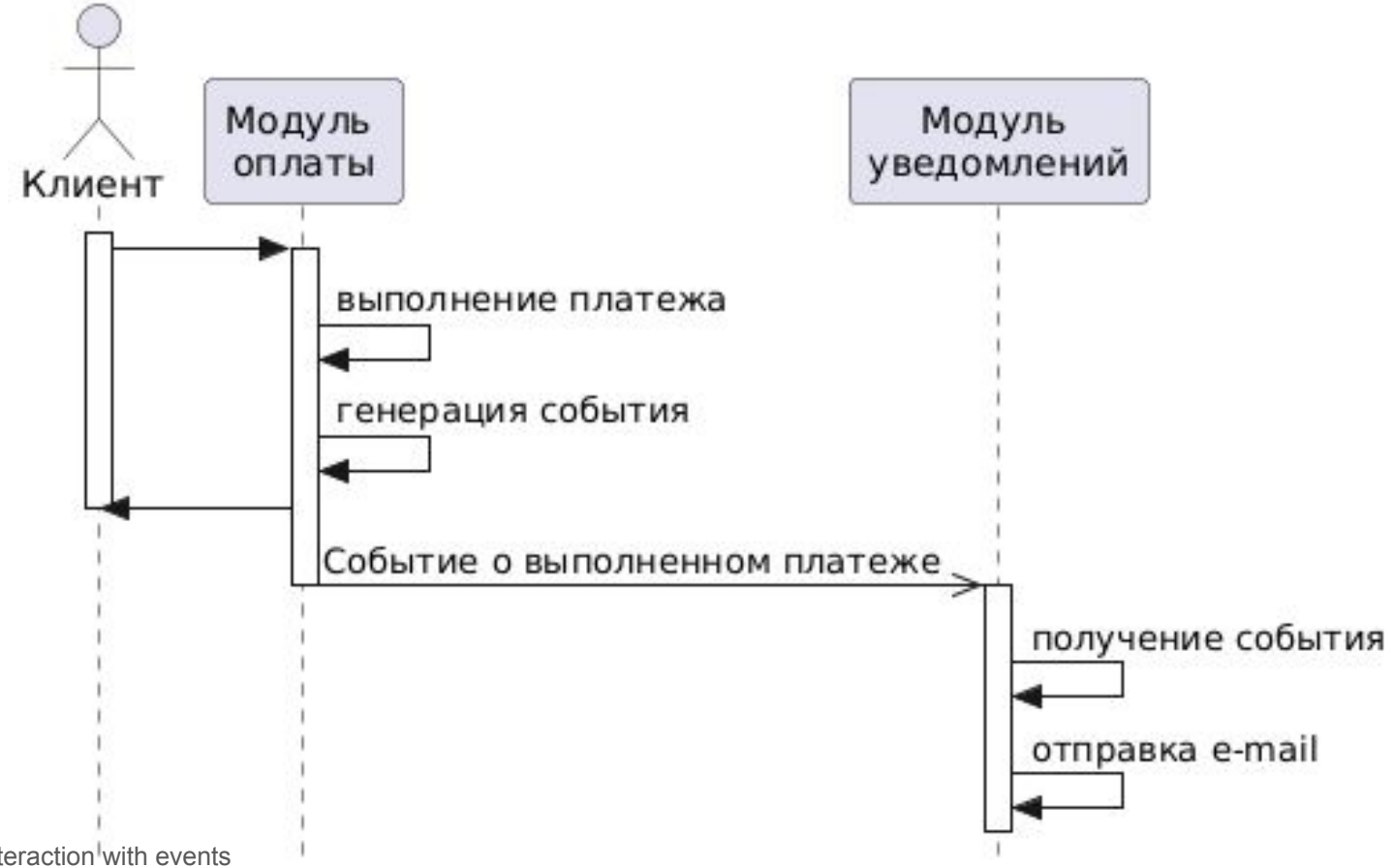
```
dependencies {

    implementation("org.springframework.modulith:spring-modulith-events-core" )
    implementation("org.springframework.modulith:spring-modulith-events-jdbc" )
    implementation("org.springframework.modulith:spring-modulith-events-jack" )

}
```

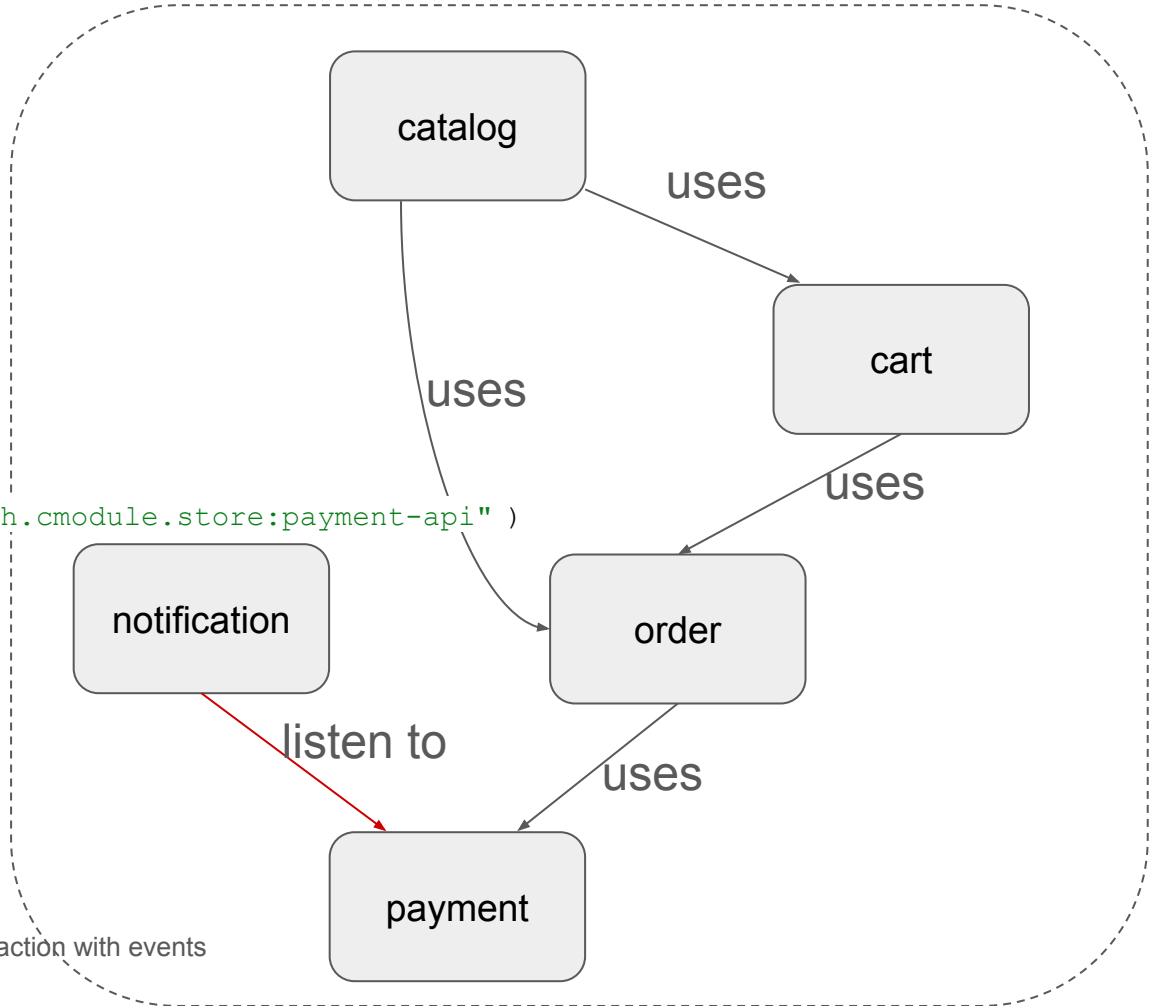
application.properties

```
spring.modulith.events.jdbc.schema-initialization.enabled=true

spring.modulith.republish-outstanding-events-on-restart=true
```

	id	listener_id	event_type	serialized_event	publication_date	completion_date
1	797ded16-f6ef-4d43-bc67-a5cca9ff7190	tech.cmodule.stor	tech.cmodule.store.payment.api.PaymentProcessed	{"paymentId":"3a2c4fbf-325c-4b40-b2f0-07a583110	18 сент. 2024 г., 22:11:45	18 сент. 2024 г., 22:11:45

Пример сценария работы с событием






```
dependencies {
    compileOnly("tech.cmodule.store:payment-api" )
}
```

Взаимодействие модулей - события
Источник события - payment module

```
01    @Service
02    class PaymentService(
03       private val events: ApplicationEventPublisher
04    ) : IPaymentFacade {
05
06       @Transactional
07       override fun processPayment(rq: PaymentProcessRq): PaymentProcessRs
{
08
09          val paymentId = UUID.randomUUID().toString()
10          val event = PaymentProcessed(paymentId, RUB ,rq.amount)
11          events.publishEvent(event)
12          return PaymentProcessRs(paymentId = paymentId)
13
14       }
15    }
```

Взаимодействие модулей - события
Класс события

```
/**
* Event, about the payment has processed
*/
data class PaymentProcessed(
    val paymentId: String,
    val currency: Currency,
    val amount: Amount
)
```

Взаимодействие модулей - события
Получатель события - notification module


```
dependencies {

    compileOnly("tech.cmodule.store:payment-api: $paymentVersion"
)

}
```

Взаимодействие модулей - события
Получатель события


```
@Service
class NotificationService {

    @TransactionalEventListener
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @Async
    fun notifyAboutPayment(event: PaymentProcessed) {
        sendEmail("text")
    }

}
```

Взаимодействие модулей - события
Получатель события


```
@Service
class NotificationService {

    @TransactionalEventListener
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @Async
    fun notifyAboutPayment(event: PaymentProcessed) {
        sendEmail("text")
    }

}
```

Взаимодействие модулей - события
Получатель события

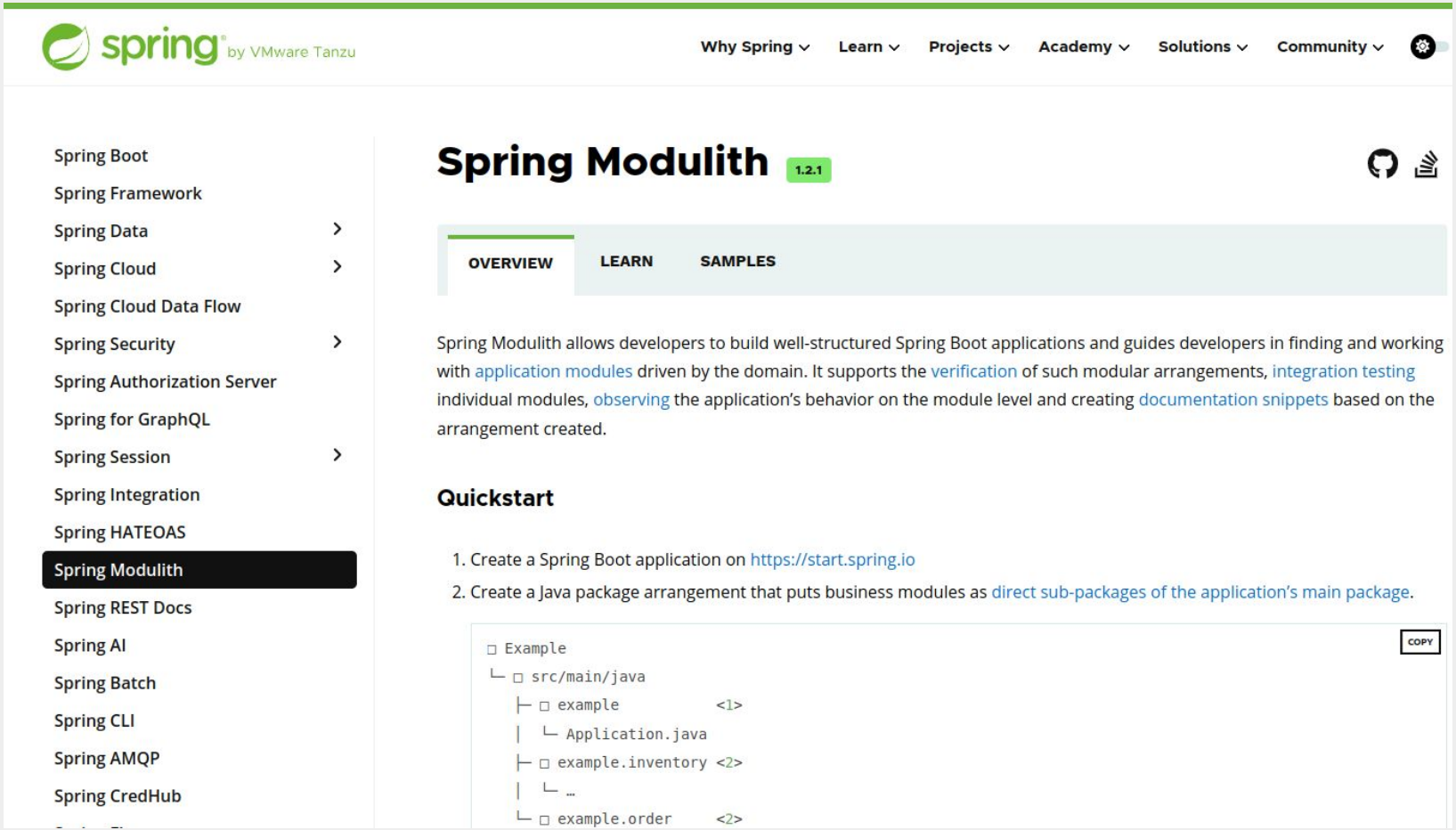

```
@Service
class NotificationService {

    @TransactionalEventListener @EventListener
    @Transactional
    @Async
    fun notifyAboutPayment(event: PaymentProcessed) {
        sendEmail("text")
    }

}
```

Classpath modules

2021 год

Spring Modulith

2023 год


spring by VMware Tanzu
Why Spring
Learn
Projects
Academy
Solutions
Community
Spring Boot
Spring Framework
Spring Data
Spring Cloud
Spring Cloud Data Flow
Spring Security
Spring Authorization Server
Spring for GraphQL
Spring Session
Spring Integration
Spring HATEOAS
Spring Modulith
Spring REST Docs
Spring AI
Spring Batch
Spring CLI
Spring AMQP
Spring CredHub
Spring Modulith 1.2.1
OVERVIEW
LEARN
SAMPLES
Spring Modulith allows developers to build well-structured Spring Boot applications and guides developers in finding and working with application modules driven by the domain. It supports the verification of such modular arrangements, integration testing individual modules, observing the application's behavior on the module level and creating documentation snippets based on the arrangement created.
Quickstart
1. Create a Spring Boot application on https://start.spring.io
2. Create a Java package arrangement that puts business modules as direct sub-packages of the application's main package.
COPY
□ Example
└ □ src/main/java
├ □ example <1>
│ └ Application.java
├ □ example.inventory <2>
│ └ …
└ □ example.order <2>

last week

odrotbohm

1.2.4

3d28ea6

Compare

1.2.4 Latest

💡 Improvements

- Improve/fix Kotlin code examples #814
- Wrong assert message of `HourHasPassed` #793
- Invalid package reference in `JacksonEventSerializer` #790

Aug 21, 2023

odrotbohm

1.0.0

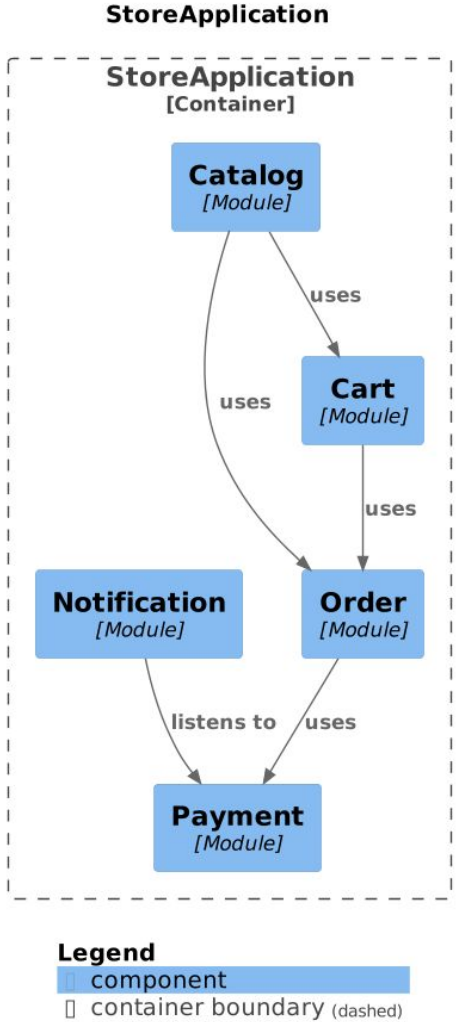
6197d35

Compare

1.0 GA

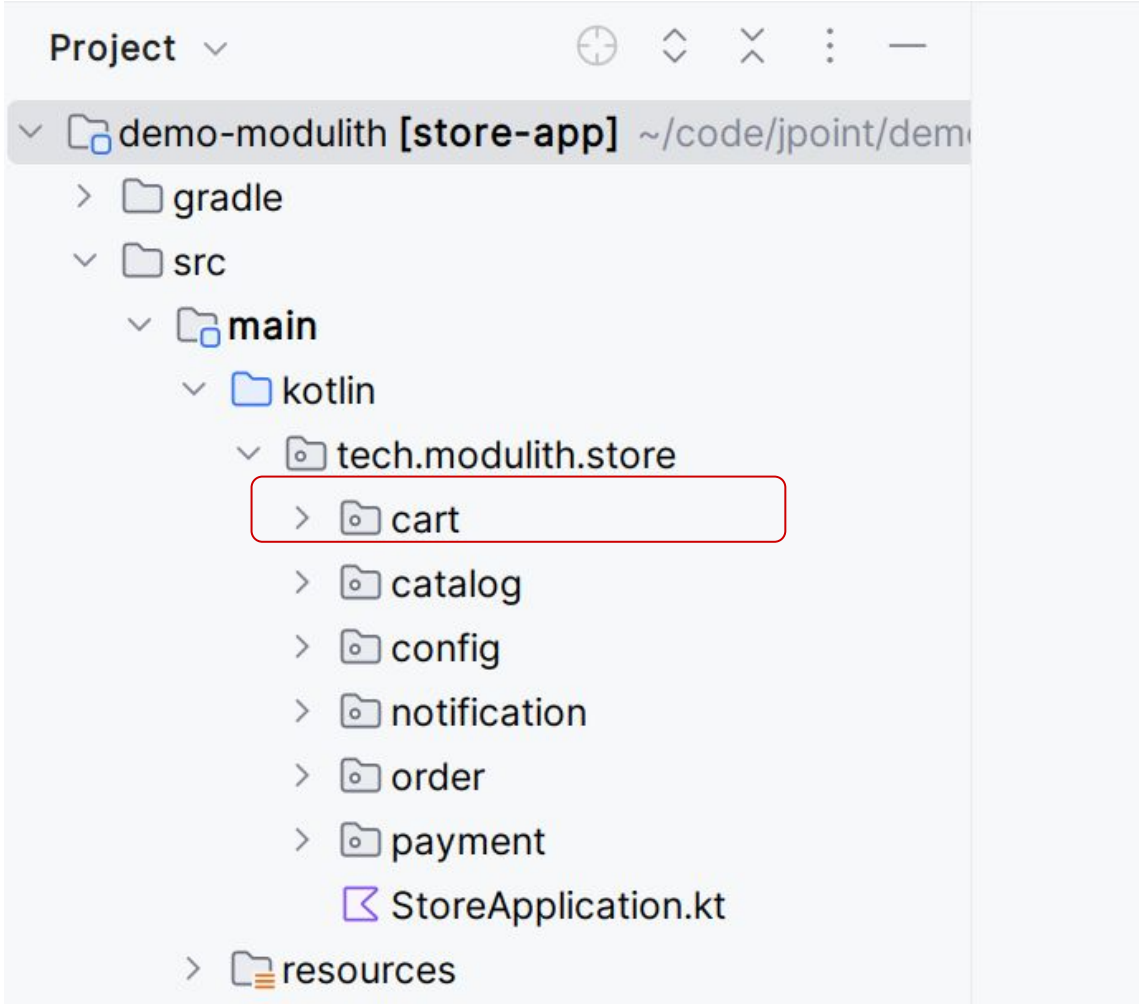
💡 Improvements

- Remove experimental declaration from `Scenario` #273
- Remove Spring Modulith Events parent POM from BOM #271


StoreApplication
StoreApplication
[Container]
Catalog
[Module]
uses
uses
Cart
[Module]
uses
Notification
[Module]
Order
[Module]
listens to
uses
Payment
[Module]
Legend
component
container boundary (dashed)

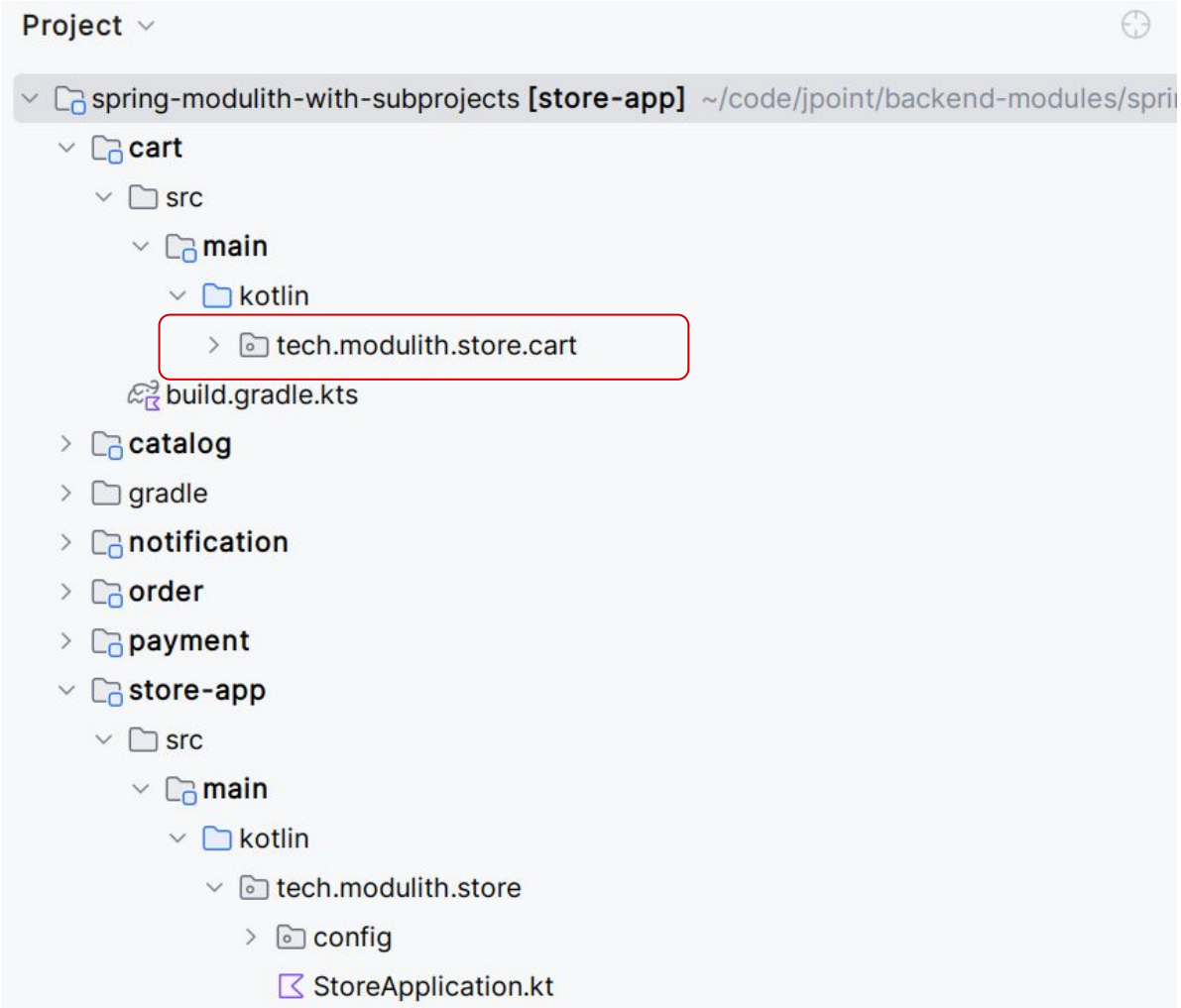
Структура проекта

gradle single level

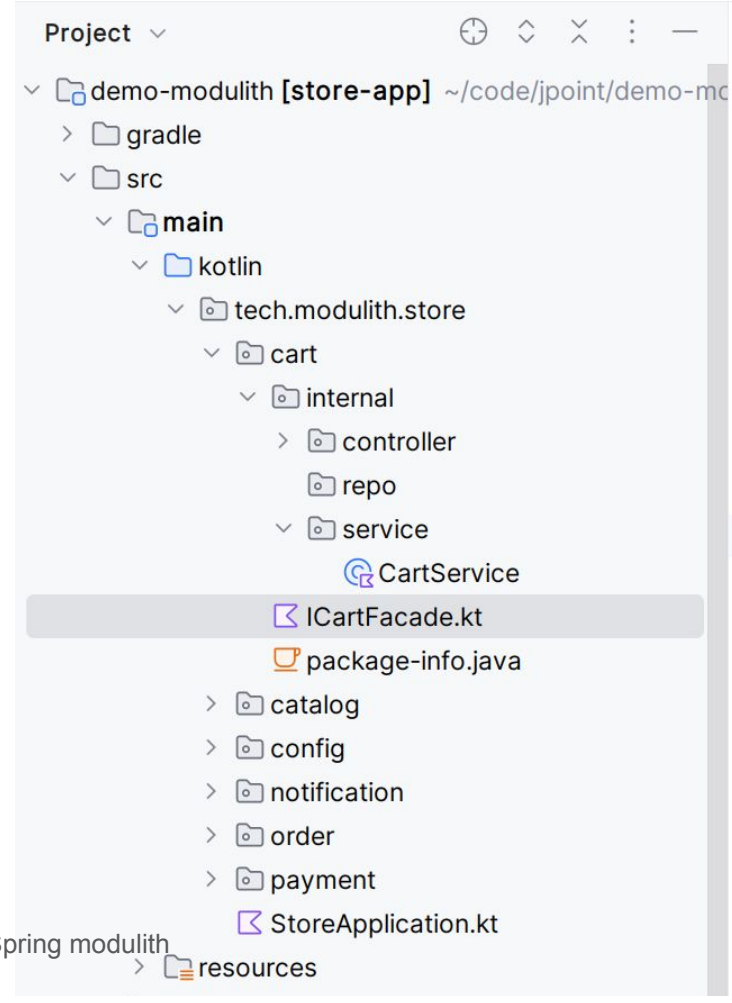


Структура проекта

gradle multi level

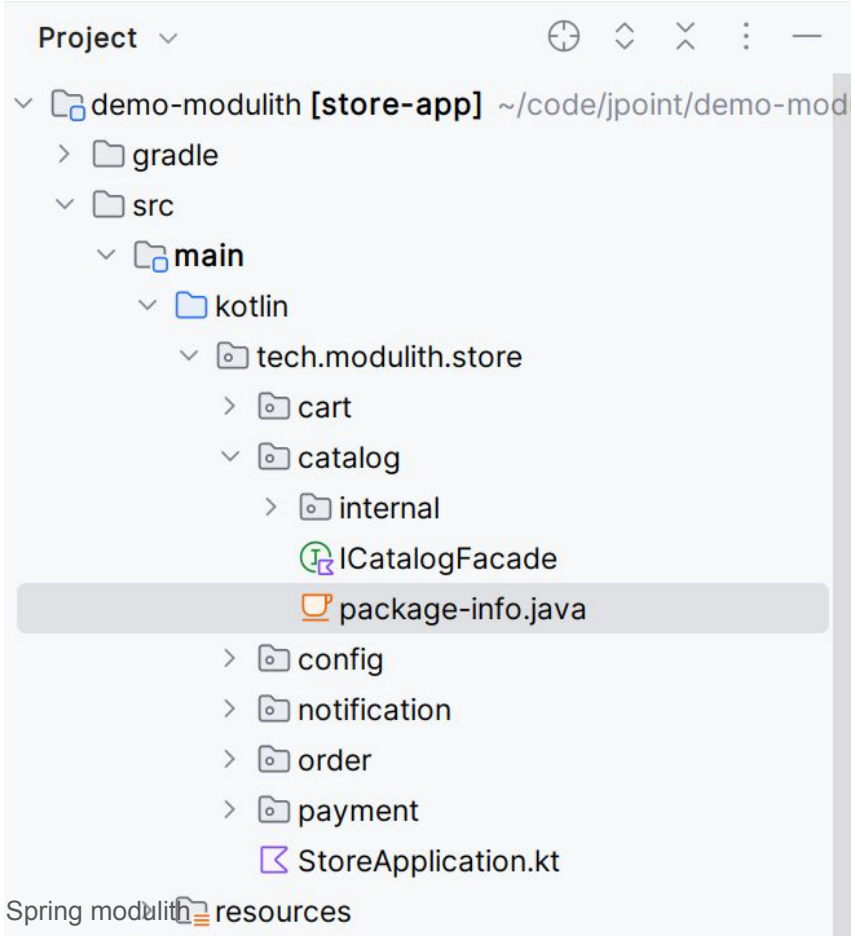


API модуля, инкапсуляция




```kotlin
package tech.modulith.store.cart

/**
 * Facade for cart module functions
 */
interface ICartFacade {

    fun createItem(rq: CreateItemRq): CreateItemRs

    fun cartToOrder(rq: CartToOrderRq): CartToOrderRs


}

data class CreateItemRq(
    val productId: String
)

data class CreateItemRs(
    val error: Exception? = null,
    val cartItemId: String? = null
)
```

Зависимость между модулями



package-info.java

```
@org.springframework.modulith.ApplicationModule
(
        allowedDependencies = { "cart","order"}
)
package tech.modulith.store.catalog ;
```

Взаимодействие модулей - синхронные вызовы

```
@Service
@Transactional
class CatalogService(
    private val cart: ICartFacade,
) : ICatalogFacade {

    fun productToCart(productId: String): String {
        val res = cart.createItem(CreateItemRq(productId))
        ...
    }

}
```

Взаимодействие модулей - события

```
@Service
class NotificationService() {

    @ApplicationModuleListener
    fun notifyAboutPayment(event: PaymentProcessed) {
        send("text")
    }
}
```

```
@TransactionalEventListener
@Transactional(
    propagation = Propagation.REQUIRES_NEW
)
@Async
@Target({ElementType.METHOD, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface ApplicationModuleListener
```

Взаимодействие модулей - события

```
@Service
class NotificationService () {

    @ApplicationModuleListener
    fun notifyAboutPayment (event: PaymentProcessed) {
        send("text")
    }
}
```

```
@TransactionalEventListener
@Transactional (
    propagation = Propagation.REQUIRES_NEW
)
@Async
@Target({ElementType.METHOD, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface ApplicationModuleListener
```

Экстернализация событий

Broker	Artifact	Description
Kafka	`spring-modulith-events-kafka`	Uses Spring Kafka for the interaction with the broker. The logical routing key will be used as
AMQP	`spring-modulith-events-amqp`	Uses Spring AMQP for the interaction with any compatible broker. Requires an explicit dependency declaration for Spring Rabbit for example. The logical routing key will be used as AMQP routing key.
JMS	`spring-modulith-events-jms`	Uses Spring’s core JMS support. Does not support routing keys.
SQS	`spring-modulith-events-aws-sqs`	Uses Spring Cloud AWS SQS support. The logical routing key will be used as SQS message group id. When routing key is set, requires SQS queue to be configured as a FIFO queue.
SNS	`spring-modulith-events-aws-sns`	Uses Spring Cloud AWS SNS support. The logical routing key will be used as SNS message group id. When routing key is set, requires SNS to be configured as a FIFO topic with content based deduplication enabled.

Тестирование архитектуры, генерация документации

```
class ModulithSnippetsTest {
    private val modules: ApplicationModules =
            ApplicationModules.of(StoreApplication::class.java)

    @Test
    fun verifyModules() {
        modules.verify()
    }

    @Test
    fun createDoc() {
        Documenter(modules).writeDocumentation()
    }

}
```

Интеграционные тесты

```
@ApplicationModuleTest
class CartIntegrationTest  {

    @Autowired
    lateinit var cartFacade: ICartFacade

    @MockBean
    lateinit var orderFacadeMock: IOrderFacade

    @Test
    fun `place product to cart test` () {
        val rq = CreateItemRq(productId =
"grill")

        val res = cartFacade.createItem(rq)

        assertThat(res.error).isNull()
        assertThat(res.cartItemId)
            .isNotNull()
            .contains("grill")
    }
}
```

```
@ApplicationModuleTest
class NotificationIntegrationTest  {

    @Test
    fun `payment event test` (scenario: Scenario)
{
        val paymentEvent = PaymentProcessed()

      scenario.publish(paymentEvent)
        .andWaitForStateChange(
            {send("text")}
        )

    }

}
```

Интеграционные тесты

```
@ApplicationModuleTest
class CartIntegrationTest  {

    @Autowired
    lateinit var cartFacade: ICartFacade

    @MockBean
    lateinit var orderFacadeMock: IOrderFacade

    @Test
    fun `place product to cart test` () {
        val rq = CreateItemRq(productId =
"grill")

        val res = cartFacade.createItem(rq)

        assertThat(res.error).isNull()
        assertThat(res.cartItemId)
            .isNotNull()
            .contains("grill")
    }
}
```

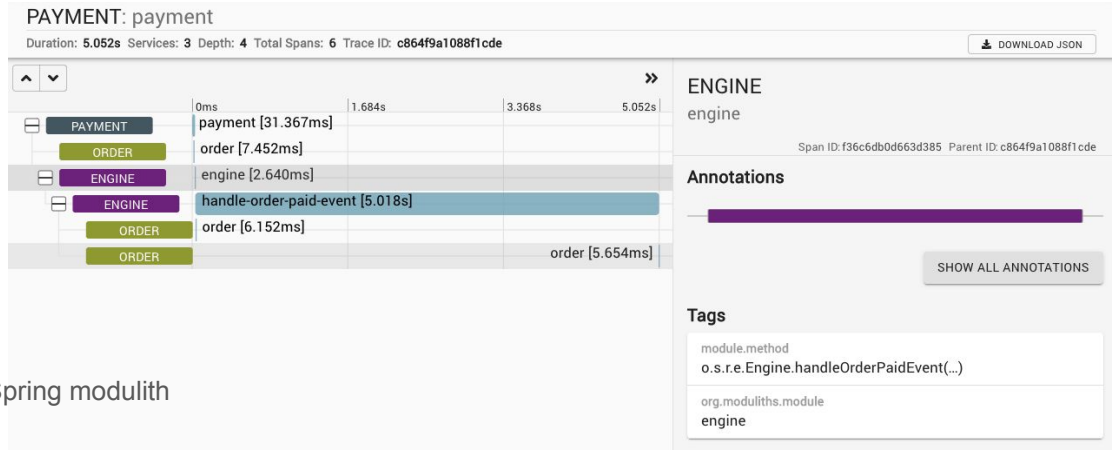
```
@ApplicationModuleTest
class NotificationIntegrationTest  {

    @Test
    fun `payment event test` (scenario: Scenario)
{
        val paymentEvent = PaymentProcessed()

      scenario.publish(paymentEvent)
        .andWaitForStateChange(
            {send("text")}
        )

    }

}
```

Runtime features

/actuator/modulith


```
{
  "payment": { 3 properties, 97 bytes "displayName": "Payment", "basePackage": "tech.modulith.s
  "order": { 3 properties, 151 bytes "displayName": "Order", "basePackage": "tech.modulith.stor
  "cart": { 3 properties, 147 bytes "displayName": "Cart", "basePackage": "tech.modulith.store.
  "catalog": { 3 properties, 208 bytes "displayName": "Catalog", "basePackage": "tech.modulith.
  "notification": { 3 properties, 165 bytes "displayName": "Notification", "basePackage": "tech
}
```


observability (zipkin, brave, micrometer)



Spring Modulith

2023 год

Что мы обсудили

1. Применение монолитной архитектуры для определённого класса приложений вполне оправдано
2. Есть проблемы о которых нужно знать заранее, учитывать их
3. Часть проблем решается с помощью модульности
4. Рассмотрели 2 варианта организации модульного приложения