Implement the monolith

A modularized one

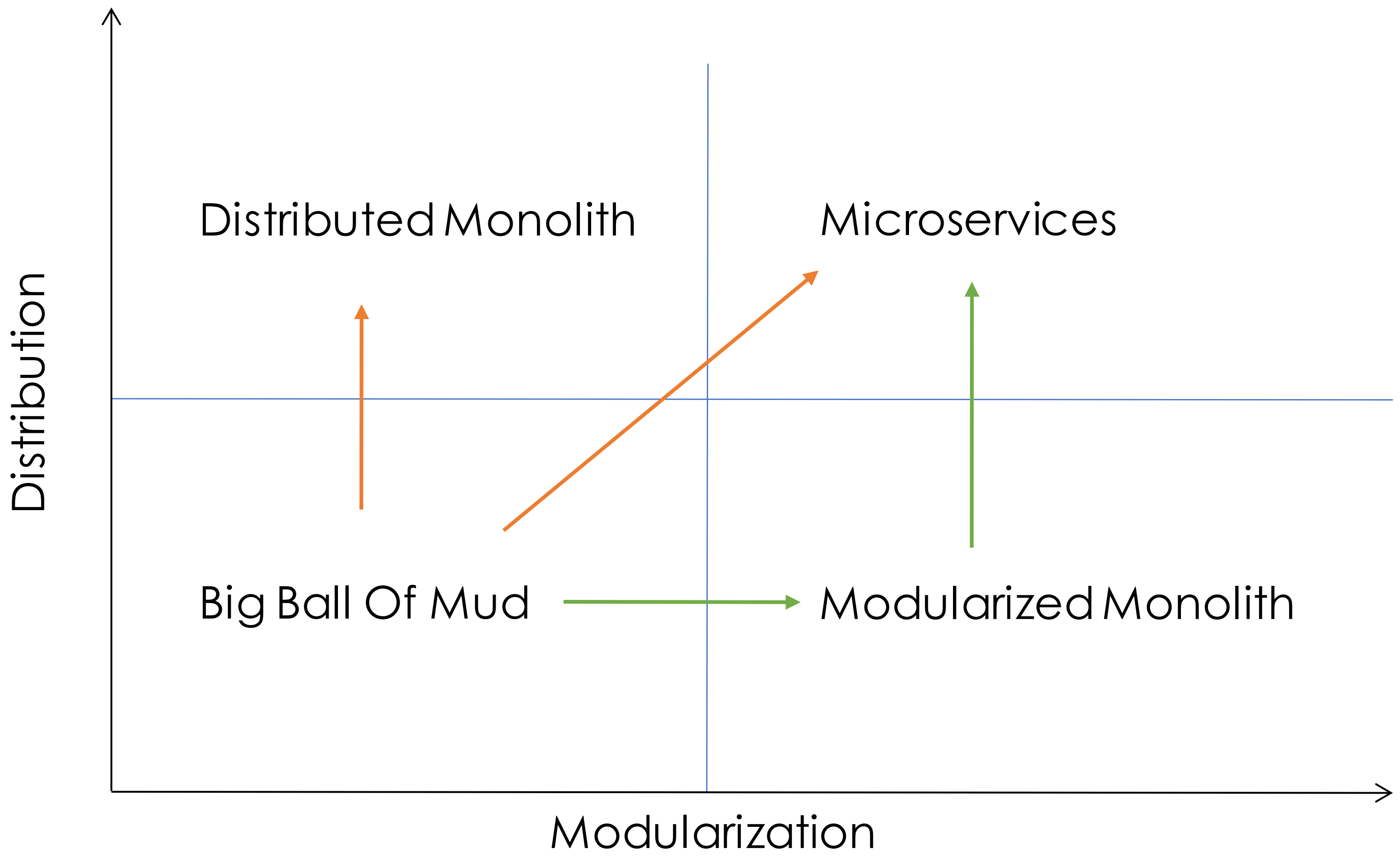
Jacek Milewski

CIRCLE K BUSINESS CENTRE

jacek.milewski.k@gmail.com

@jacek_mil

Start with monolith first



A kind of agenda

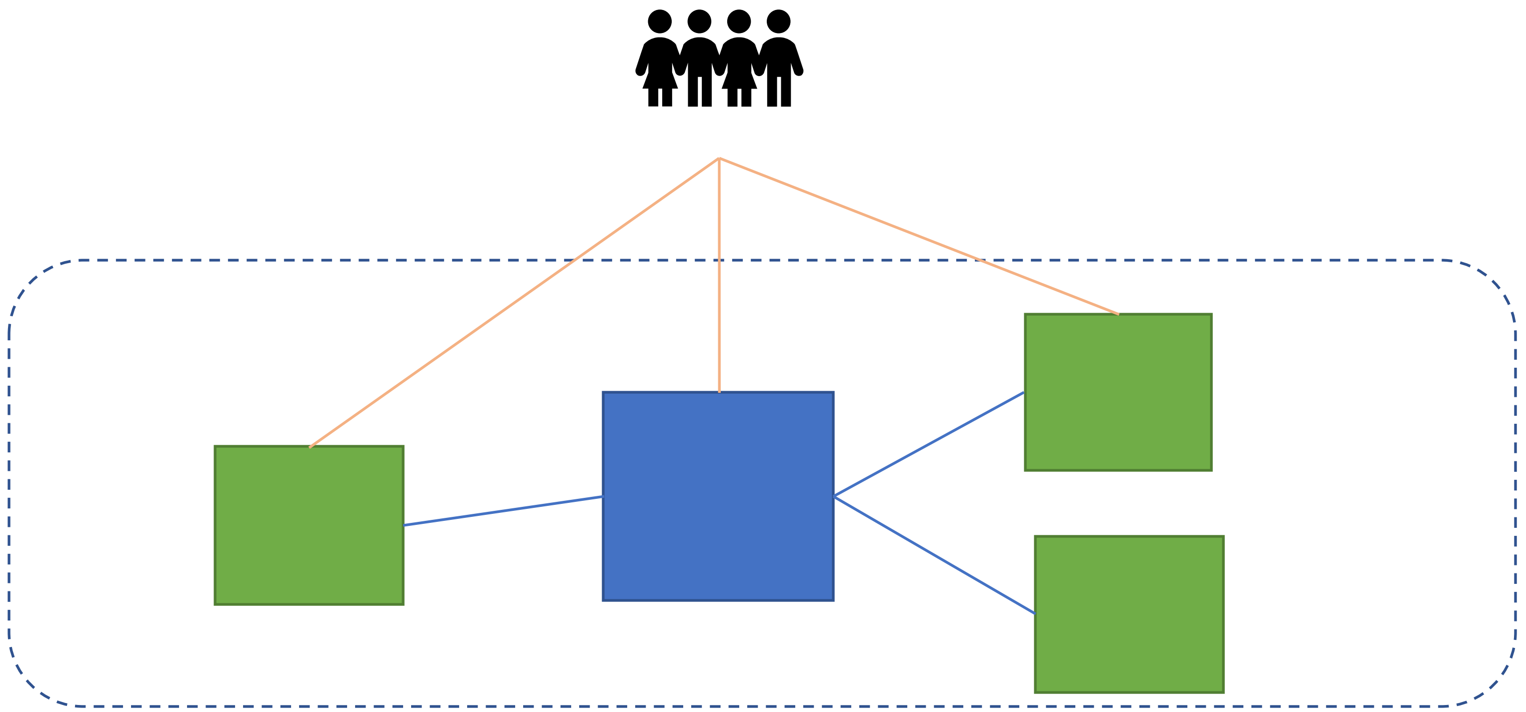
1. Strategy: My path and decisions
2. Tactics: How things are implemented

* Focus on interactions between modules (not defining the modules)

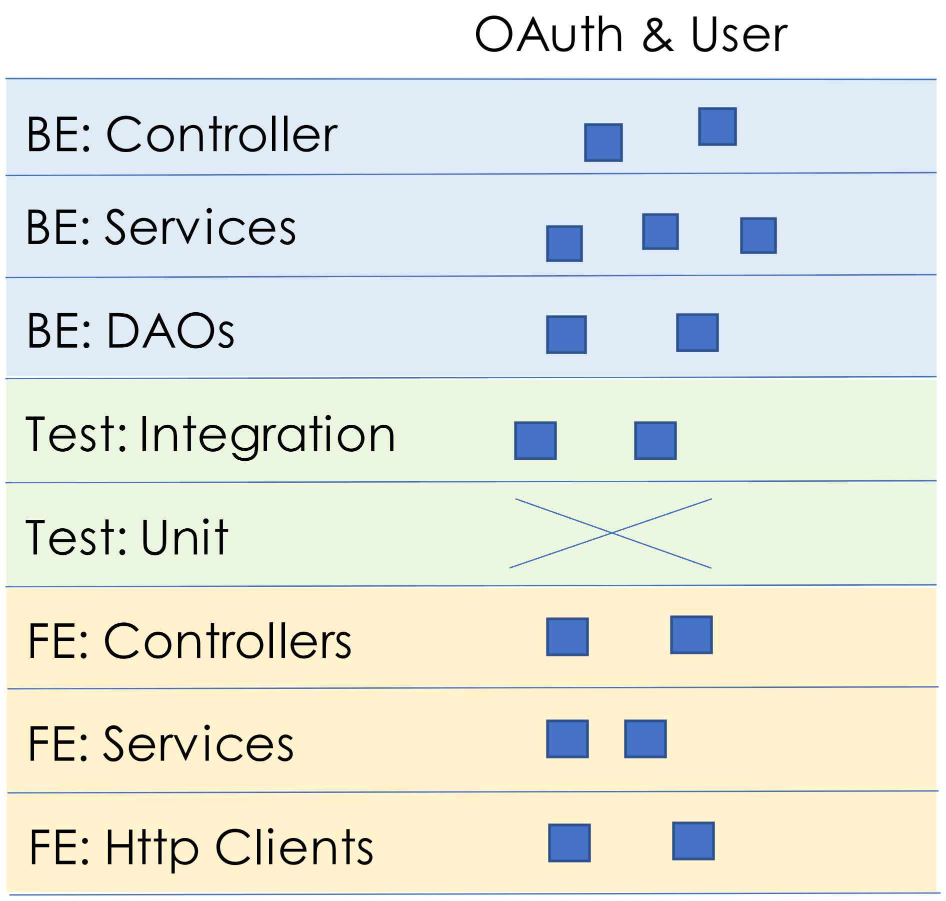
Long story short

The monolith or a microservice?

- Identity Management system for Customers
- Internal microservice used by other services



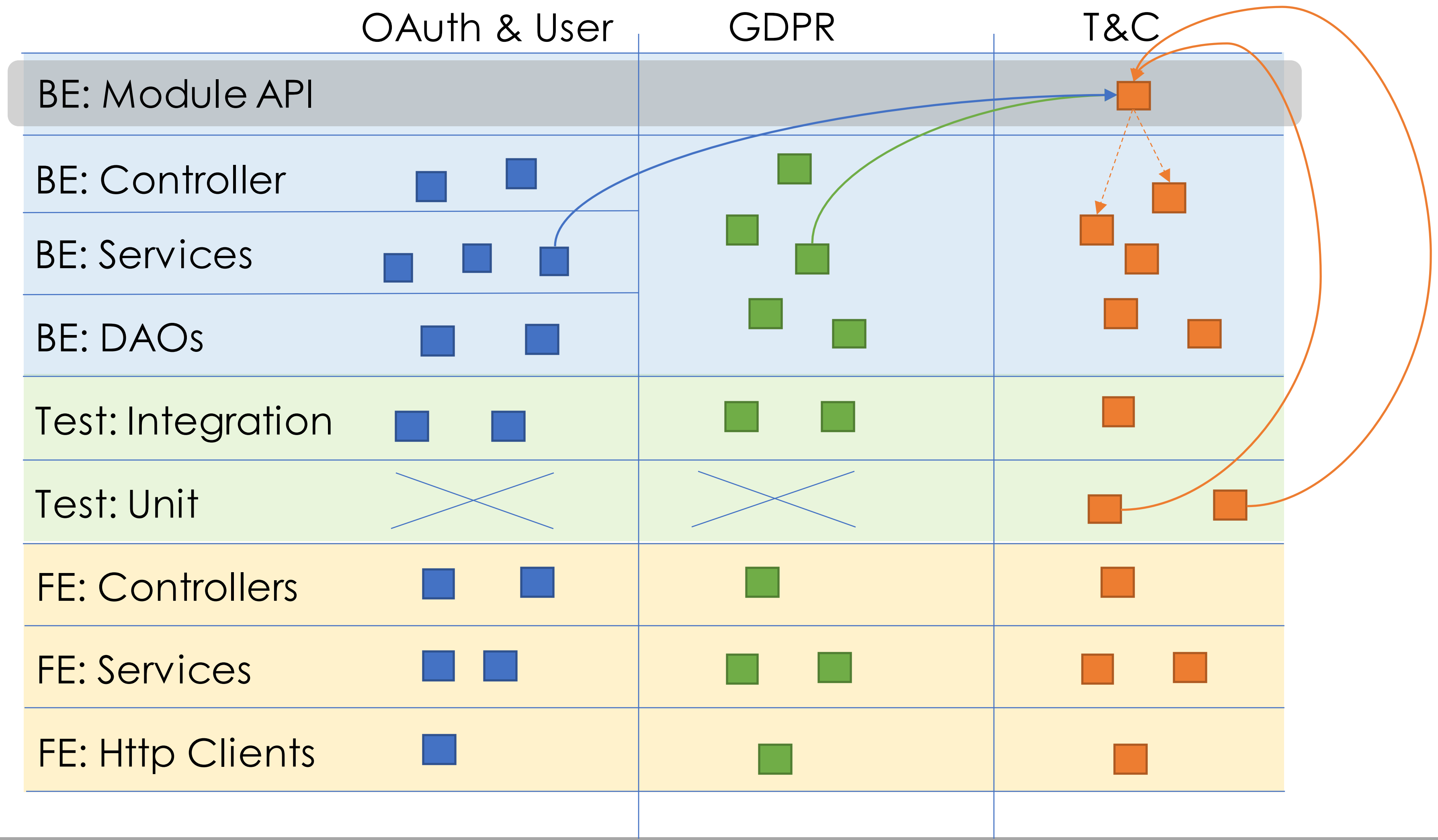
The perfect microservice



There’s this small feature… GDPR

	OAuth & User	GDPR
BE: Controller	🟦 🟦	🟩
BE: Services	🟦 🟦 🟦	🟩 🟩
BE: DAOs	🟦 🟦	🟩 🟩
Test: Integration	🟦 🟦	🟩 🟩
Test: Unit	✕	✕
FE: Controllers	🟦 🟦	🟩
FE: Services	🟦 🟦	🟩 🟩
FE: Http Clients	🟦	🟩

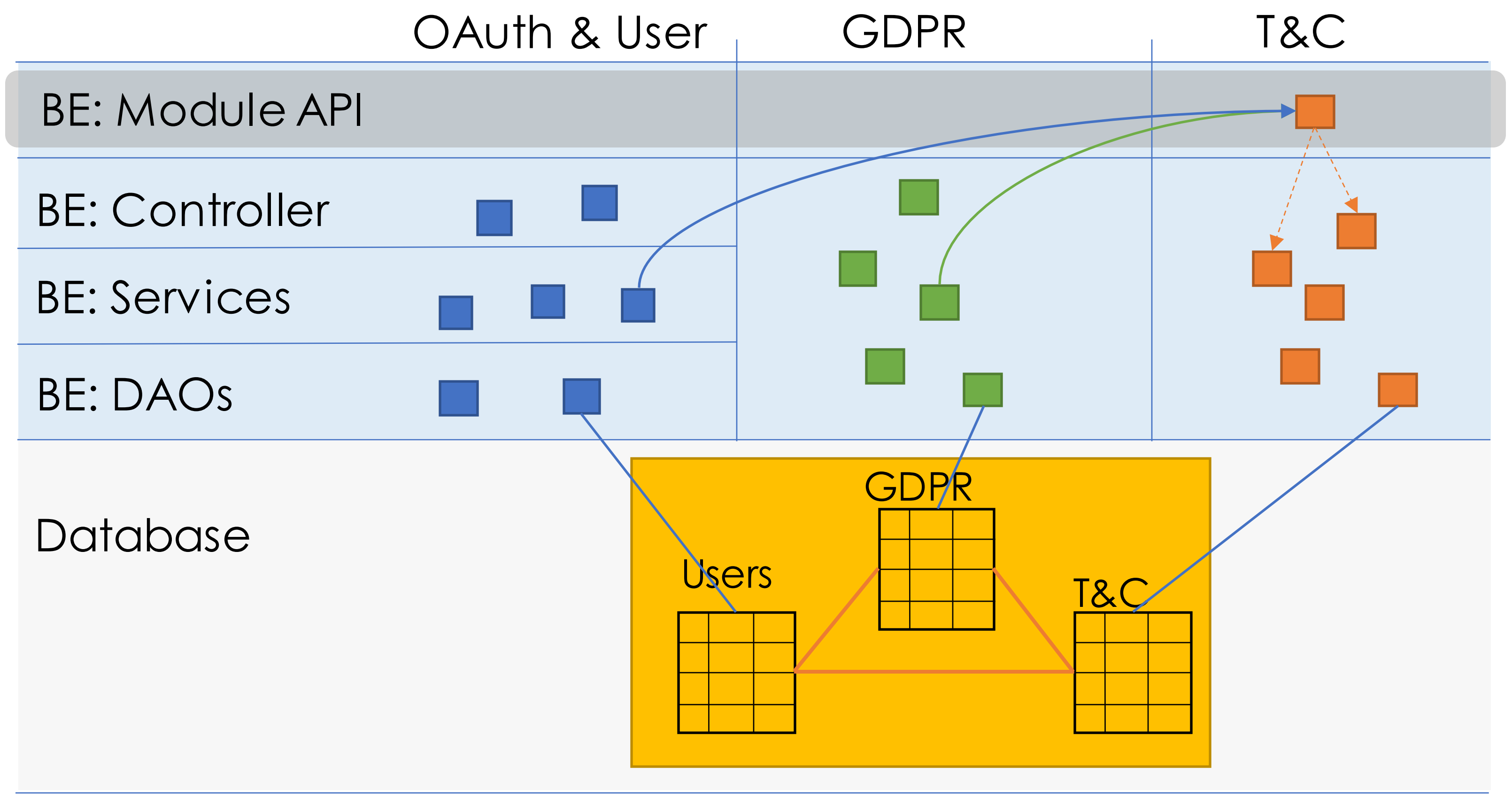
And one more: T&C



Two modules and... the rest

- @ModuleApi created for `user` and `oauth2` modules
  - Designing new modules, deciding on boundaries

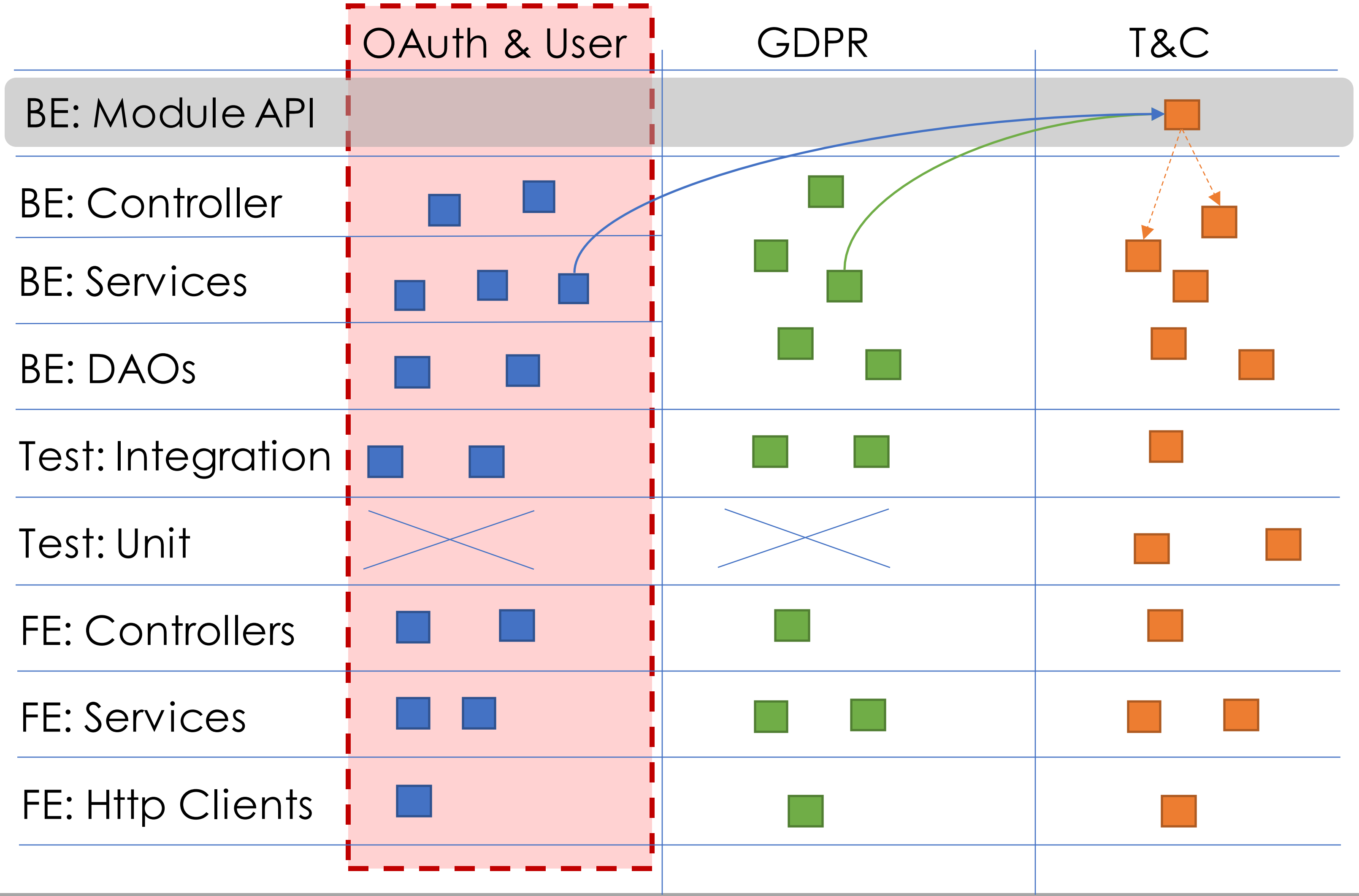
DB Relations



The future: replace Identity module

- the main product functionality might be replaced with Off-the shelf solution
  - Why investing time in modularization and refactoring, then?
    - let's do it because it's the latest opportunity to do it in this codebase - hurry up and play around! :)
  - Remaining modules might stay
    - Seems that we might have the case when one module is cut off to a separate microservice
    - only part of the product will be replaced.
    - a part - means one module. then we need to have it separated

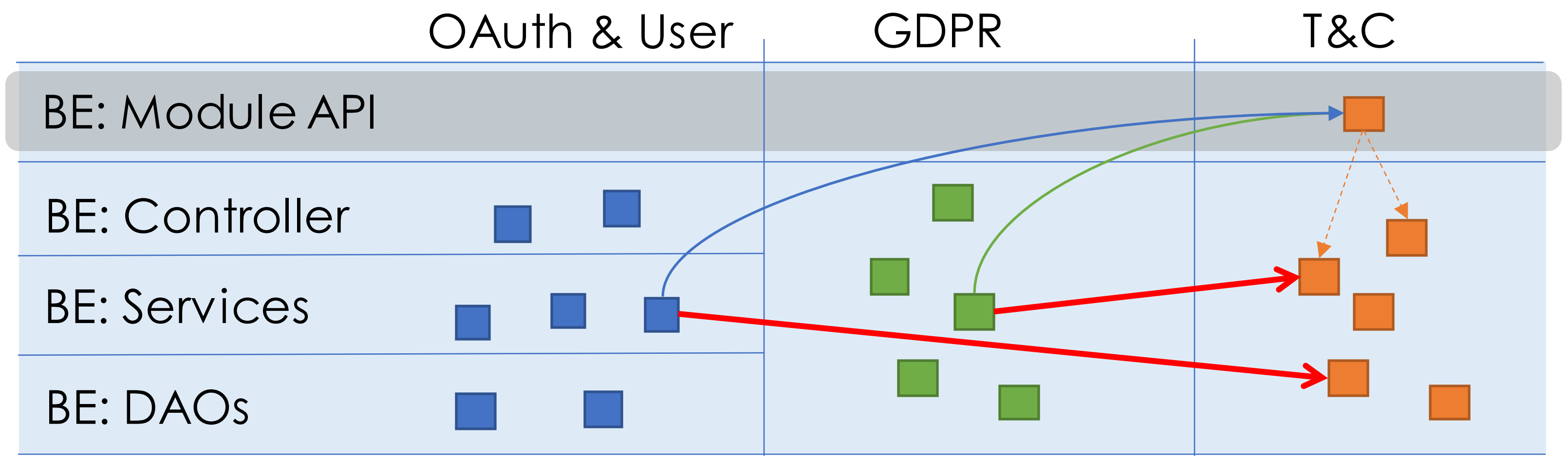
Future – replace CIAM



We know where we're going

- Modules' definitions are mature and still growing
- Current setup works well for a long time
- time to strengthen the boundaries
  - Enforce on new development
  - Find gaps where we’re not modularized

Broken boundaries



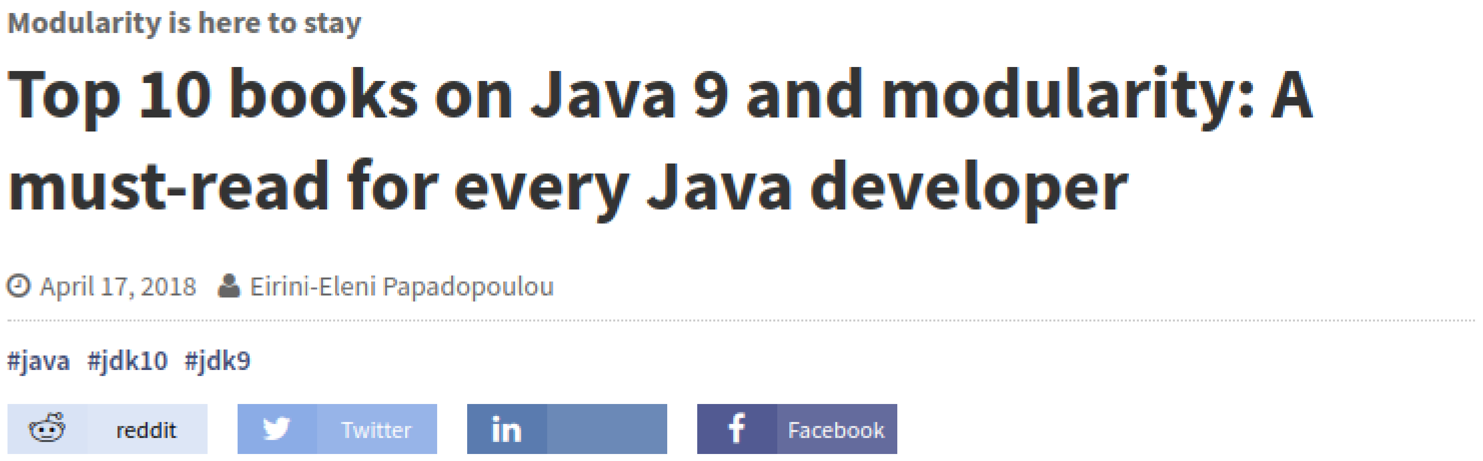
Tooling

Is there a tool to help us?

1. Automate rule checking on new development

2. Detect where we're not yet modularized

Java 9

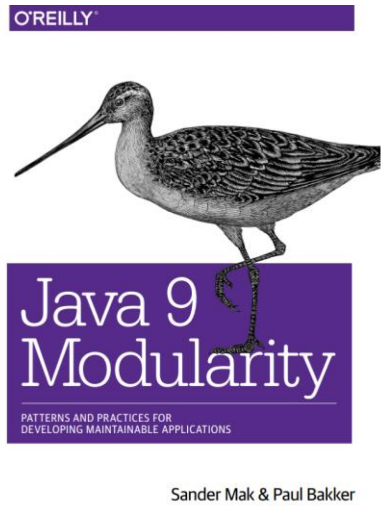


Modularity is here to stay

Top 10 books on Java 9 and modularity: A must-read for every Java developer

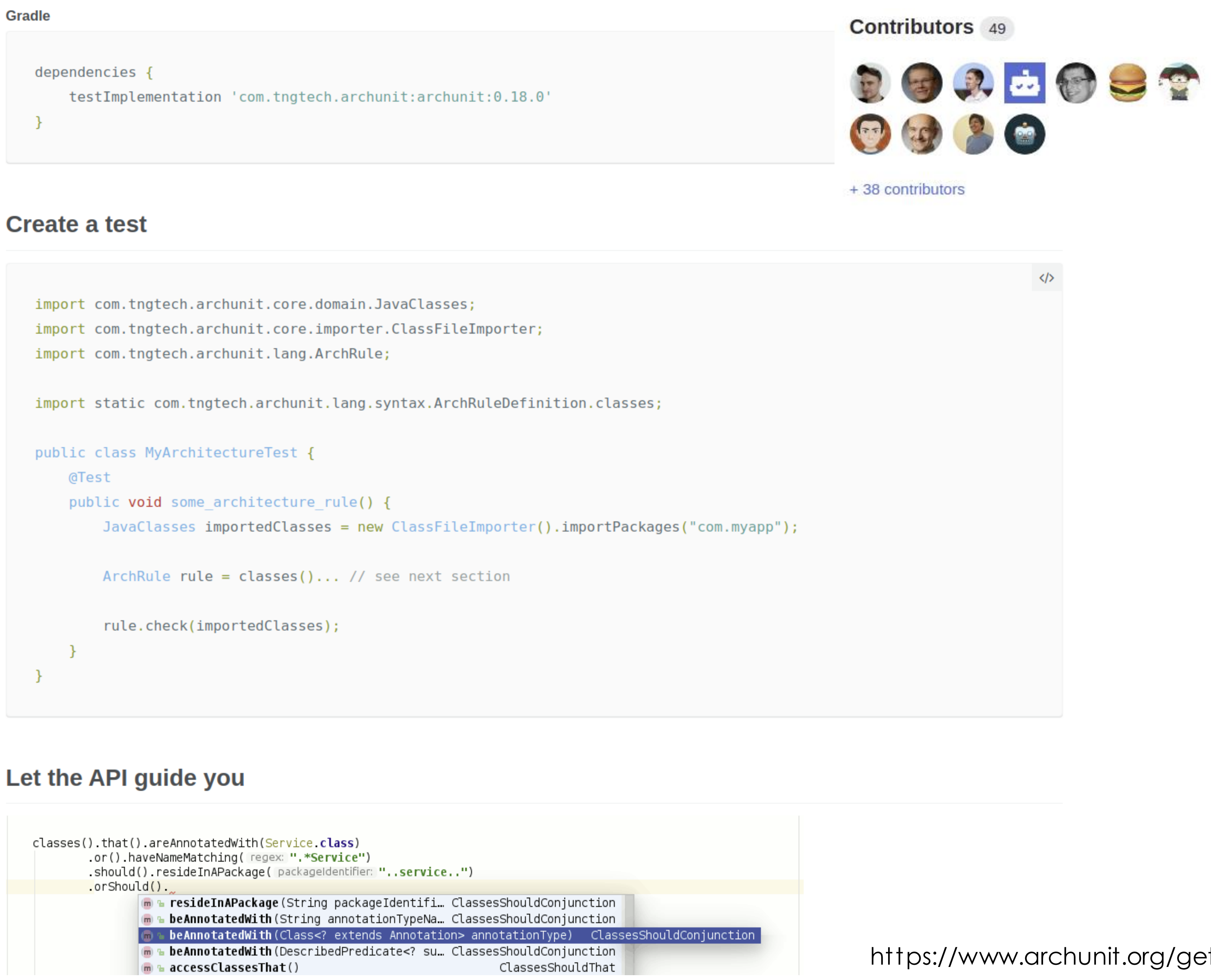
April 17, 2018 Eirini-Eleni Papadopoulou

#java #jdk10 #jdk9


O'REILLY
Java 9 Modularity
PATTERNS AND PRACTICES FOR DEVELOPING MAINTAINABLE APPLICATIONS
Sander Mak & Paul Bakker

https://jaxenter.com/top-10-books-on-java-9-modularity-143518.html

ArchUnit



Gradle

```
dependencies {
    testImplementation 'com.tngtech.archunit:archunit:0.18.0'
}
```

Contributors 49

+ 38 contributors

Create a test

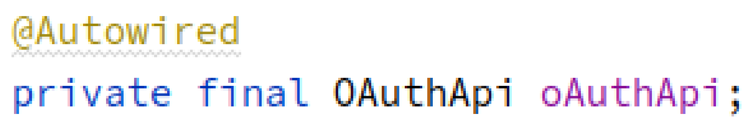
```
import com.tngtech.archunit.core.domain.JavaClasses;
import com.tngtech.archunit.core.importer.ClassFileImporter;
import com.tngtech.archunit.lang.ArchRule;

import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.classes;

public class MyArchitectureTest {
    @Test
    public void some_architecture_rule() {
        JavaClasses importedClasses = new ClassFileImporter().importPackages("com.myapp");

        ArchRule rule = classes()... // see next section

        rule.check(importedClasses);
    }
}
```

Let the API guide you

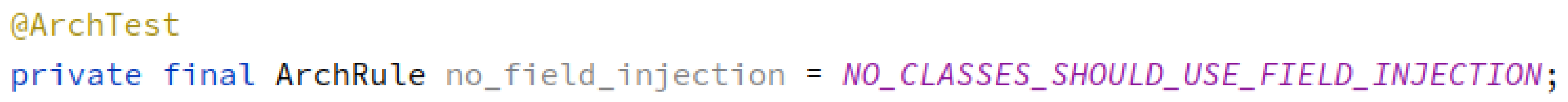
```
classes().that().areAnnotatedWith(Service.class)
        .or().haveNameMatching( regex: ".*Service")
        .should().resideInAPackage( packageIdentifier: "..service..")
        .orShould().
            resideInAPackage(String packageIdentifi…  ClassesShouldConjunction
            beAnnotatedWith(String annotationTypeNa…  ClassesShouldConjunction
            beAnnotatedWith(Class<? extends Annotation> annotationType)  ClassesShouldConjunction
            beAnnotatedWith(DescribedPredicate<? su…  ClassesShouldConjunction
            accessClassesThat()                       ClassesShouldThat
```

https://www.archunit.org/getting-started

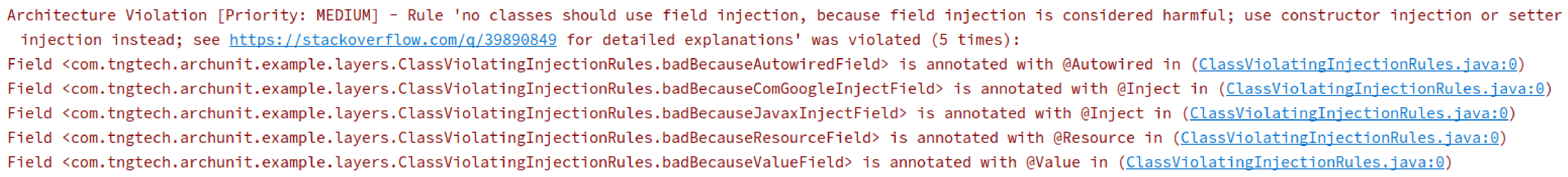
Example: Field injection



```
@Autowired
private final OAuthApi oAuthApi;
```

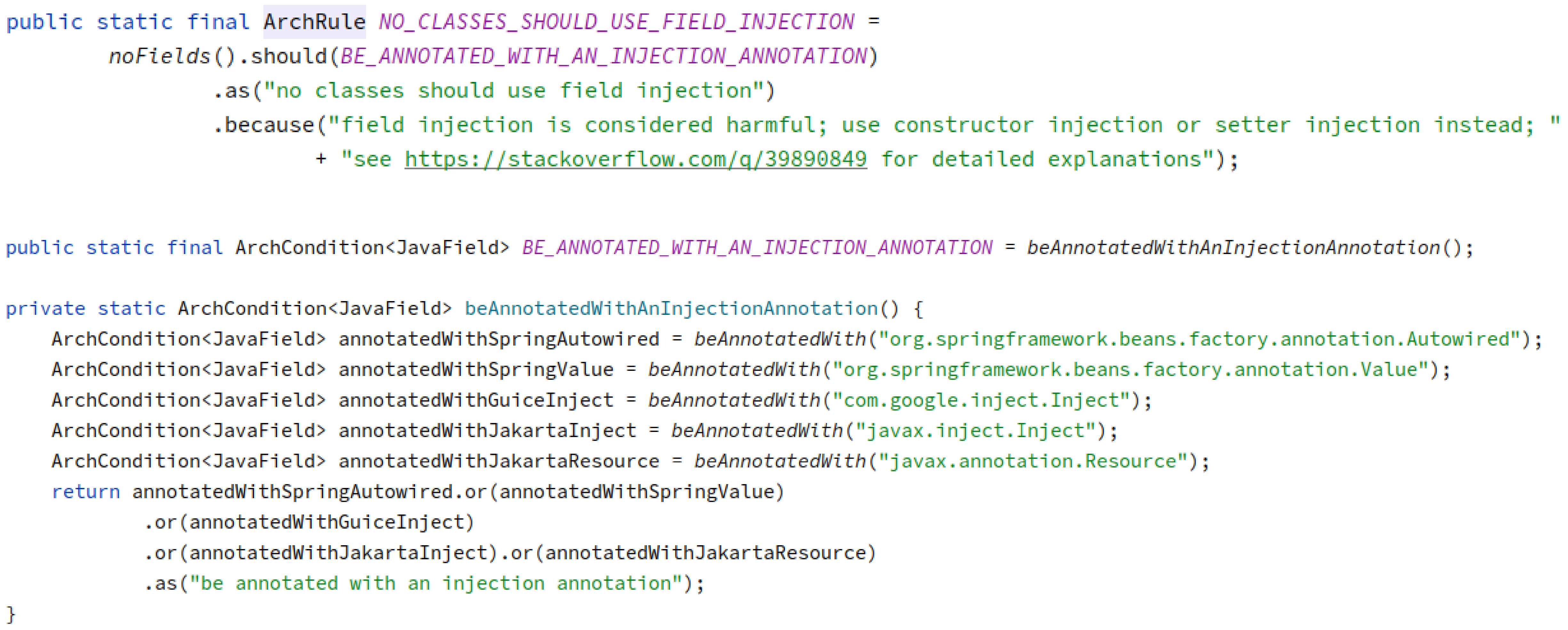


```
@ArchTest
private final ArchRule no_field_injection = NO_CLASSES_SHOULD_USE_FIELD_INJECTION;
```



```
Architecture Violation [Priority: MEDIUM] - Rule 'no classes should use field injection, because field injection is considered harmful; use constructor injection or setter
 injection instead; see https://stackoverflow.com/q/39890849 for detailed explanations' was violated (5 times):
Field <com.tngtech.archunit.example.layers.ClassViolatingInjectionRules.badBecauseAutowiredField> is annotated with @Autowired in (ClassViolatingInjectionRules.java:0)
Field <com.tngtech.archunit.example.layers.ClassViolatingInjectionRules.badBecauseComGoogleInjectField> is annotated with @Inject in (ClassViolatingInjectionRules.java:0)
Field <com.tngtech.archunit.example.layers.ClassViolatingInjectionRules.badBecauseJavaxInjectField> is annotated with @Inject in (ClassViolatingInjectionRules.java:0)
Field <com.tngtech.archunit.example.layers.ClassViolatingInjectionRules.badBecauseResourceField> is annotated with @Resource in (ClassViolatingInjectionRules.java:0)
Field <com.tngtech.archunit.example.layers.ClassViolatingInjectionRules.badBecauseValueField> is annotated with @Value in (ClassViolatingInjectionRules.java:0)
```

Example: Field injection



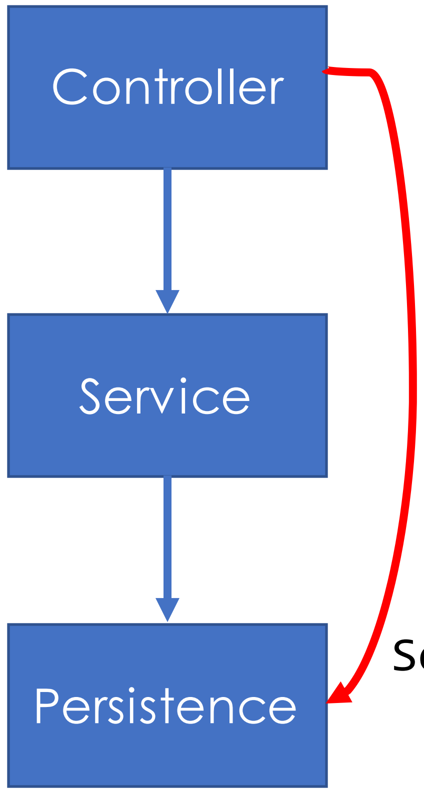
```java
public static final ArchRule NO_CLASSES_SHOULD_USE_FIELD_INJECTION =
        noFields().should(BE_ANNOTATED_WITH_AN_INJECTION_ANNOTATION)
                .as("no classes should use field injection")
                .because("field injection is considered harmful; use constructor injection or setter injection instead; "
                        + "see https://stackoverflow.com/q/39890849 for detailed explanations");


public static final ArchCondition<JavaField> BE_ANNOTATED_WITH_AN_INJECTION_ANNOTATION = beAnnotatedWithAnInjectionAnnotation();

private static ArchCondition<JavaField> beAnnotatedWithAnInjectionAnnotation() {
    ArchCondition<JavaField> annotatedWithSpringAutowired = beAnnotatedWith("org.springframework.beans.factory.annotation.Autowired");
    ArchCondition<JavaField> annotatedWithSpringValue = beAnnotatedWith("org.springframework.beans.factory.annotation.Value");
    ArchCondition<JavaField> annotatedWithGuiceInject = beAnnotatedWith("com.google.inject.Inject");
    ArchCondition<JavaField> annotatedWithJakartaInject = beAnnotatedWith("javax.inject.Inject");
    ArchCondition<JavaField> annotatedWithJakartaResource = beAnnotatedWith("javax.annotation.Resource");
    return annotatedWithSpringAutowired.or(annotatedWithSpringValue)
            .or(annotatedWithGuiceInject)
            .or(annotatedWithJakartaInject).or(annotatedWithJakartaResource)
            .as("be annotated with an injection annotation");
}
```

Example: Coding rules

```
System.out.println("Not allowed!");
```

```
@ArchTest
private final ArchRule no_access_to_standard_streams = NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;

@ArchTest
private void no_access_to_standard_streams_as_method(JavaClasses classes) {
    noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
}
```

Example: layered architecture

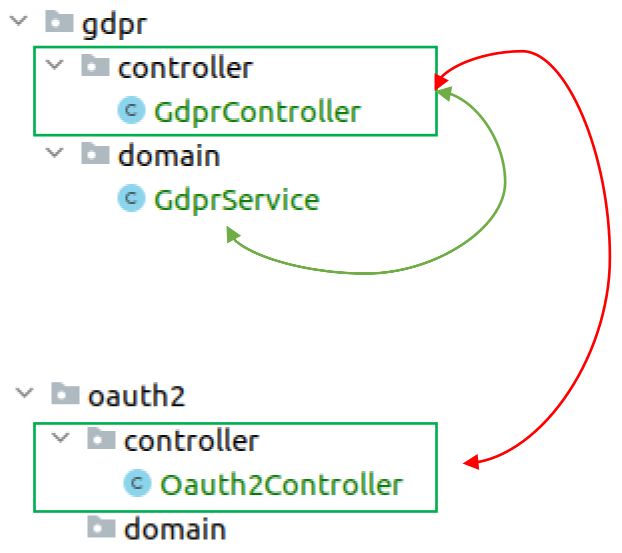


```
@ArchTest
public static final ArchRule layer_dependencies_are_respected_with_exception = layeredArchitecture()
```

```
}
```

```
ServiceViolatingLayerRules.class
```

Example: Slices



```
@ArchTest
public static final ArchRule controllers_should_only_use_their_own_slice =
        slices().matching("..controller.(*)..").namingSlices("Controller $1")
                .as("Controllers").should().notDependOnEachOther();
```

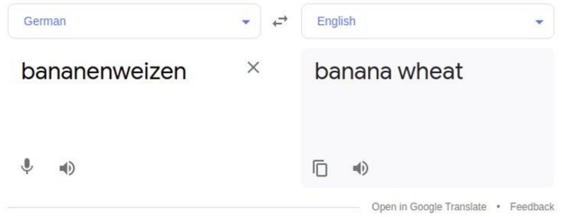
More examples

On GitHub: https://github.com/TNG/ArchUnit-Examples

Contributors 2

codecholeric Peter Gafert

Bananeweizen Michael Keppler

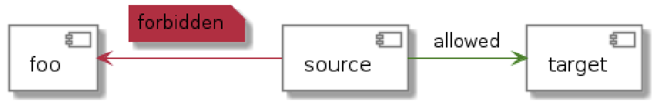

German ⇄ English

bananenweizen

banana wheat

In docs: https://www.archunit.org/userguide/html/000_Index.html#_what_to_check

4.1. Package Dependency Checks



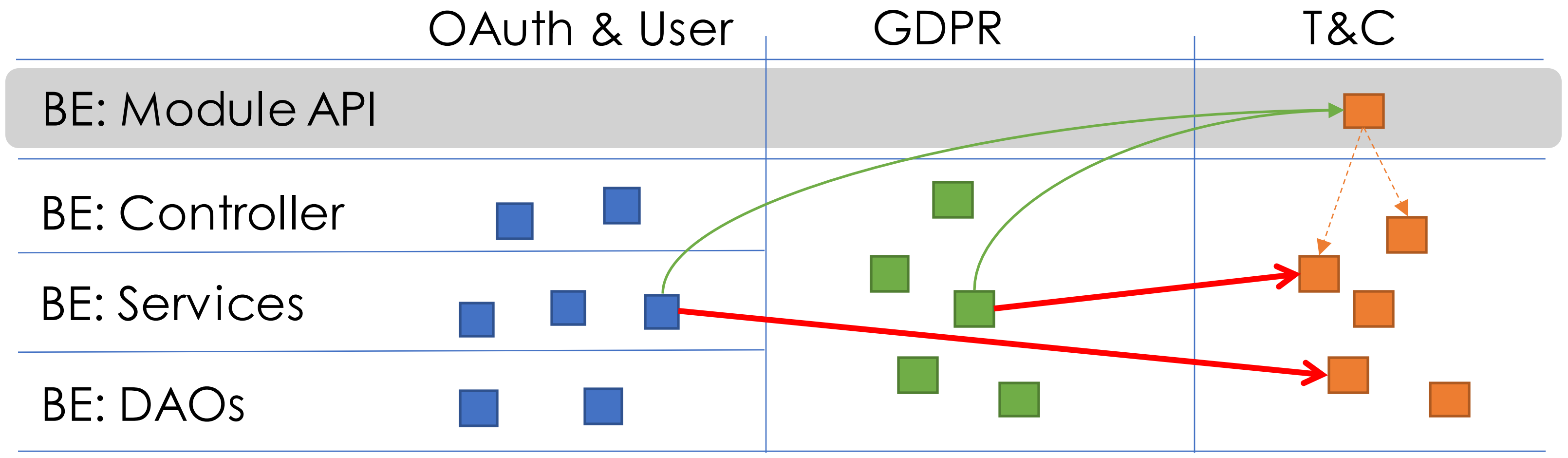
```
noClasses().that().resideInAPackage("..source..")
    .should().dependOnClassesThat().resideInAPackage("..foo..")
```

The Rules

Rule #1: Access module using its API

Module can only be accessed via API

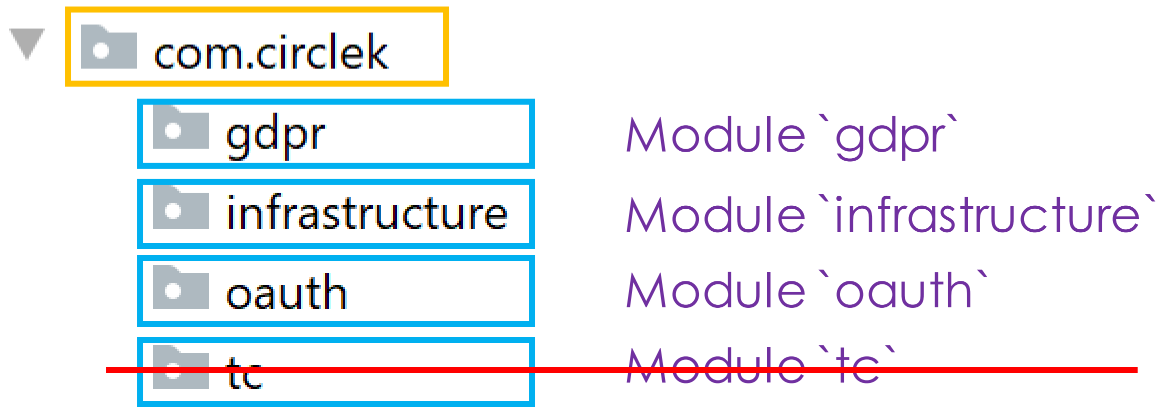
Rule #1: Access Module via API



```
Rule 'Modules should depend on other modules via API only' was violated (2767 times)
```

Rule #1: Implementation

```
@ArchTest
ArchRule allowCrossModuleDependenciesOnlyViaModuleApis =
        slices()
            .matching("com.circlek.(*)..").namingSlices("Module `$1`").as("Modules")
            .that(shouldBeIncludedInScan)
            .should(dependOnOtherModulesViaApiOnly);
```



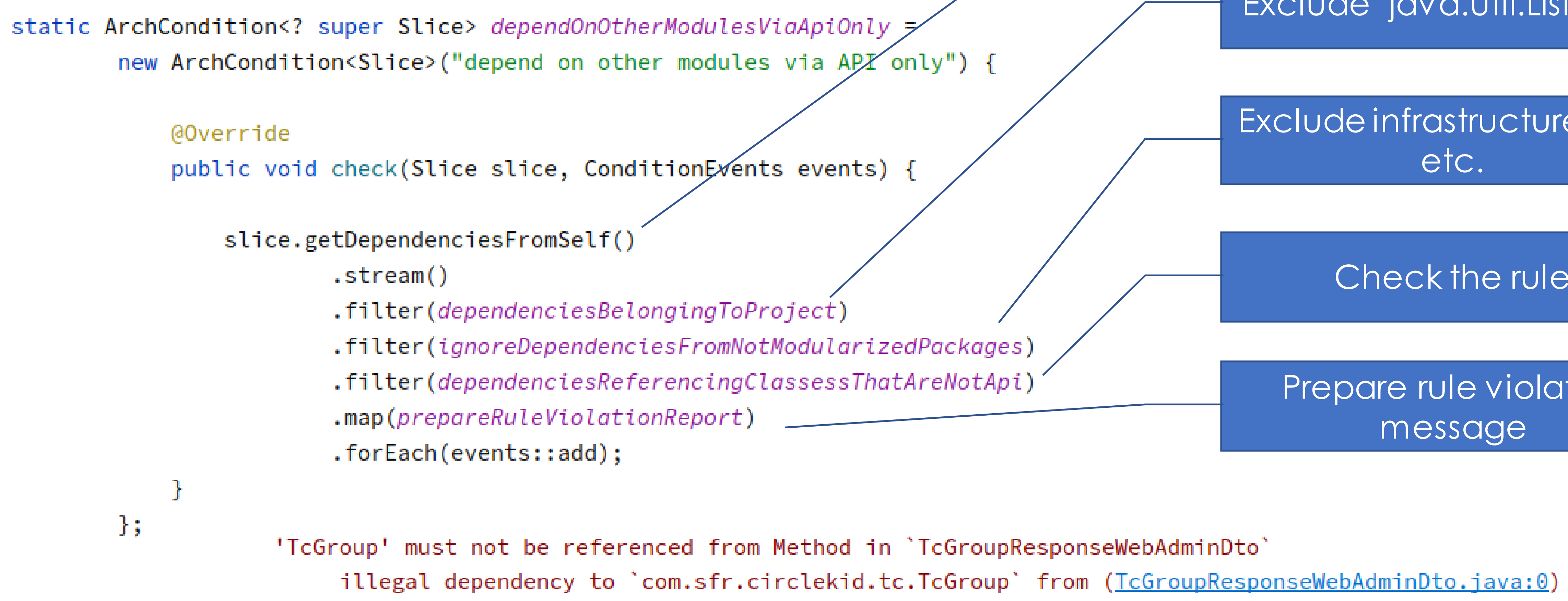
com.circlek	
gdpr	Module `gdpr`
infrastructure	Module `infrastructure`
oauth	Module `oauth`
~~tc~~	~~Module `tc`~~

Rule #1: Should be included in scan

```
private static final Set<String> SCANNED_MODULES = Set.of(
        "gdpr",
        "oauth"
        //"tc"  //TODO: enable later
);
```

```
static DescribedPredicate<? super Slice> shouldBeIncludedInScan =
        new DescribedPredicate<>("are included scan", SCANNED_MODULES) {
            @Override
            public boolean apply(Slice slice) {
                return SCANNED_MODULES.stream().anyMatch(module -> slice.getDescription().contains(module));
            }
        };
```

Rule #1: Rule check



```
static ArchCondition<? super Slice> dependOnOtherModulesViaApiOnly =
        new ArchCondition<Slice>("depend on other modules via API only") {

            @Override
            public void check(Slice slice, ConditionEvents events) {

                slice.getDependenciesFromSelf()
                        .stream()
                        .filter(dependenciesBelongingToProject)
                        .filter(ignoreDependenciesFromNotModularizedPackages)
                        .filter(dependenciesReferencingClassessThatAreNotApi)
                        .map(prepareRuleViolationReport)
                        .forEach(events::add);
            }
        };
```

```
'TcGroup' must not be referenced from Method in `TcGroupResponseWebAdminDto`
     illegal dependency to `com.sfr.circlekid.tc.TcGroup` from (TcGroupResponseWebAdminDto.java:0)
```

Check who the class is calling

Exclude `java.util.List`, etc.

Exclude infrastructure, utils, etc.

Check the rule

Prepare rule violation message

Rule #1: Actual dependency check

```
static Predicate<Dependency> dependenciesReferencingClassessThatAreNotApi =
        dependency ->
                !(dependency.getTargetClass().isAnnotatedWith(ModuleApi.class)
                        || dependency.getTargetClass().getPackageName().contains("application.api"));
```

Rule #2: Module has API

Each business module must have an API

Rule #2: Implementation

```
@ArchTest
static final ArchRule everyBusinessModuleShouldHaveApi =
        slices().matching("com.circlek.(*)..").namingSlices("Module `$1`").as("Modules")
                .that(areBusinessModules)
                .should(haveApi);


static ArchCondition<? super Slice> haveApi =
        new ArchCondition<>("have API") {
            @Override
            public void check(Slice item, ConditionEvents events) {

                boolean hasApiClass = item.stream().anyMatch(javaClass -> javaClass.isAnnotatedWith(ModuleApi.class));
                boolean hasApiPackage = item.stream().anyMatch(javaClass -> javaClass.getPackageName().contains("application.api"));

                events.add(new SimpleConditionEvent(item, hasApiPackage || hasApiClass,
                        String.format("%s is required to have an API (@ModuleApi class or `application.api` package)",
                                item.getDescription())
                ));
            }
        };
```

Rule #3: Module API is unit tested

Every Module API method must have tests

Rule #4: Module internal architecture

Layered?

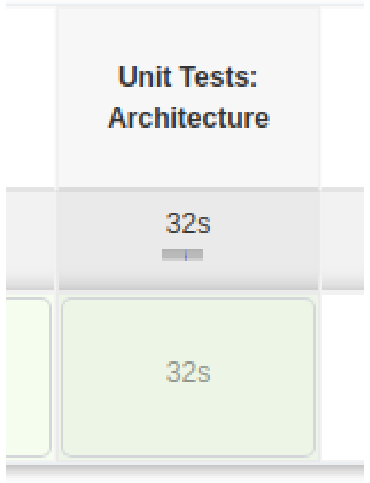
Onion?

CRUD?

Implementig it

New build step

CI Build got a new step to verify Architecture rules



Start with a simple one

Identified utility classes

```
class UuidParser
```

Elevated DTO to Module API

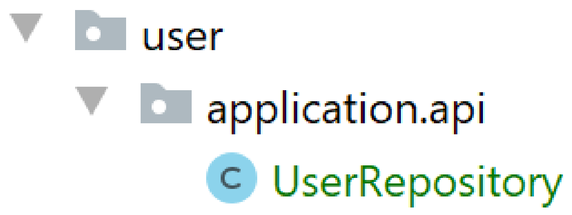
```
public enum ApplicationType {
    PUBLIC, CONFIDENTIAL
}
```

Approach

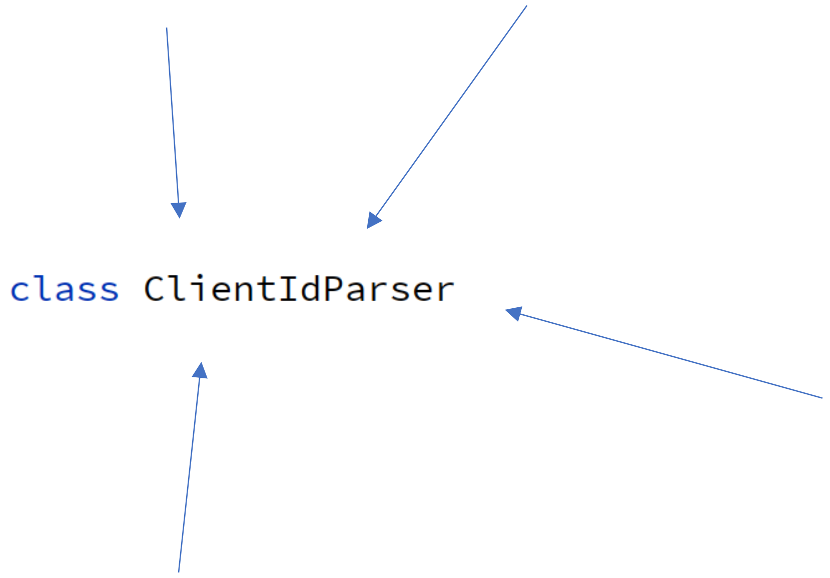
1. Enable the rule to see tests fail
2. Make tests pass with a minimal effort
3. Fix the design in separate PRs

The next one

UserRepository in module API



ClientIdParser as a code smell



```
class ClientIdParser
```

Mob programming

When you more need to discuss than to code

Other simple modules

LoggingContext moved to cross-cutting logging package

DeviceName moved to shared models

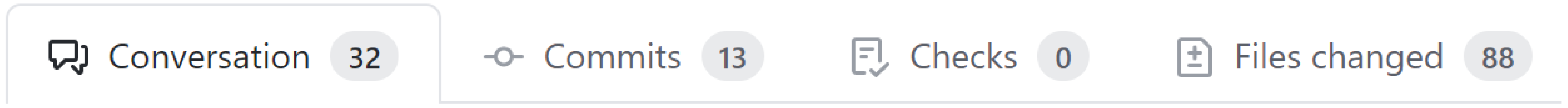
Then it started to get boring

Following refactorings were similar

Create a new module

From classes scattered across whole code base

…and used from anywhere

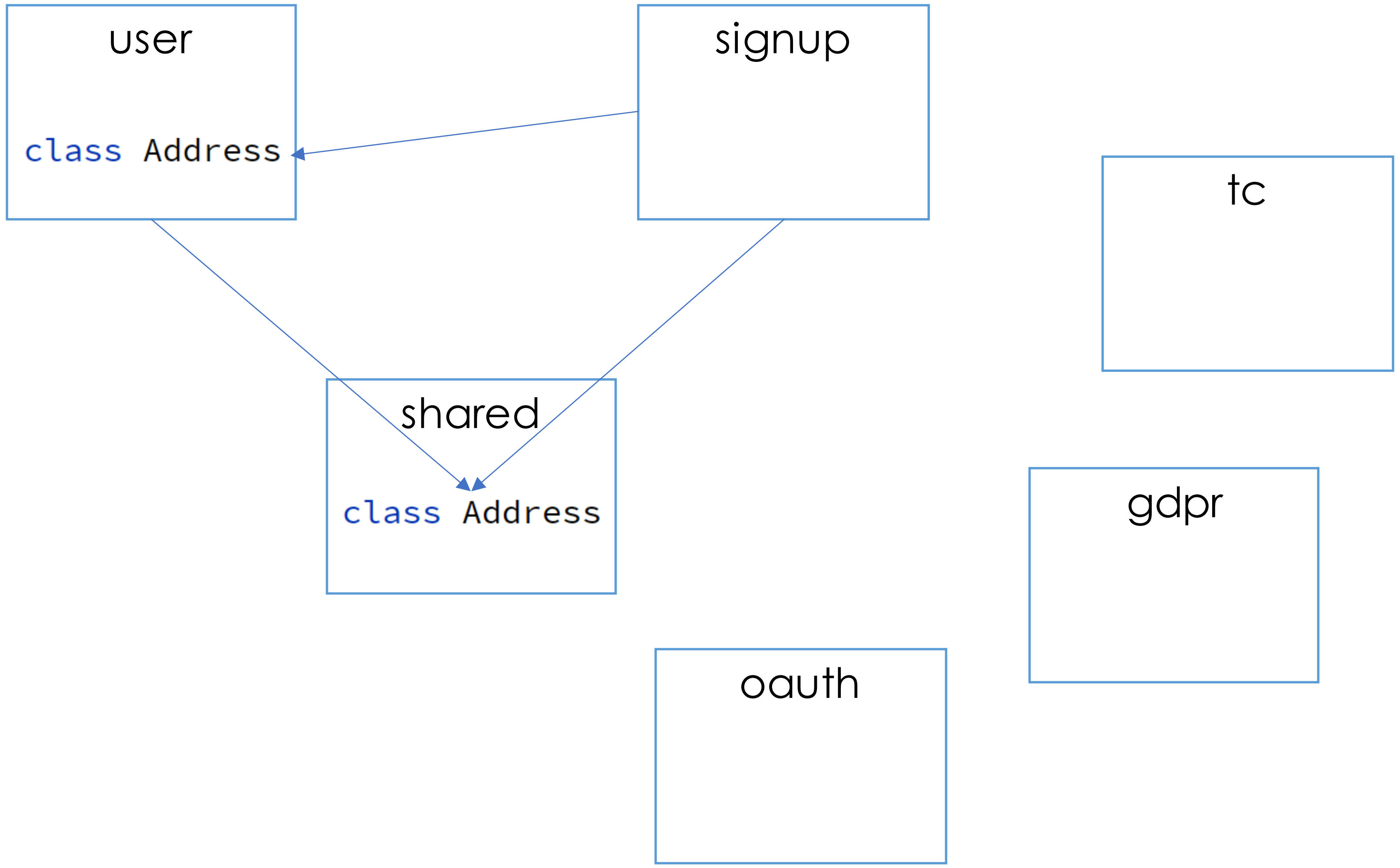


Largest modules comes last

Starting with small was good

A trap: Cohesion

Cohesion


user
class Address
signup
tc
shared
class Address
gdpr
oauth

Are we done?

Next steps

Submodules?

Rules on module internals?

Thank you

Jacek Milewski

CIRCLE K
BUSINESS CENTRE

jacek.milewski.k@gmail.com

@jacek_mil