March 6th 2020

MICRONAUT TESTING TIPS AND TRICKS

ocitraining.com

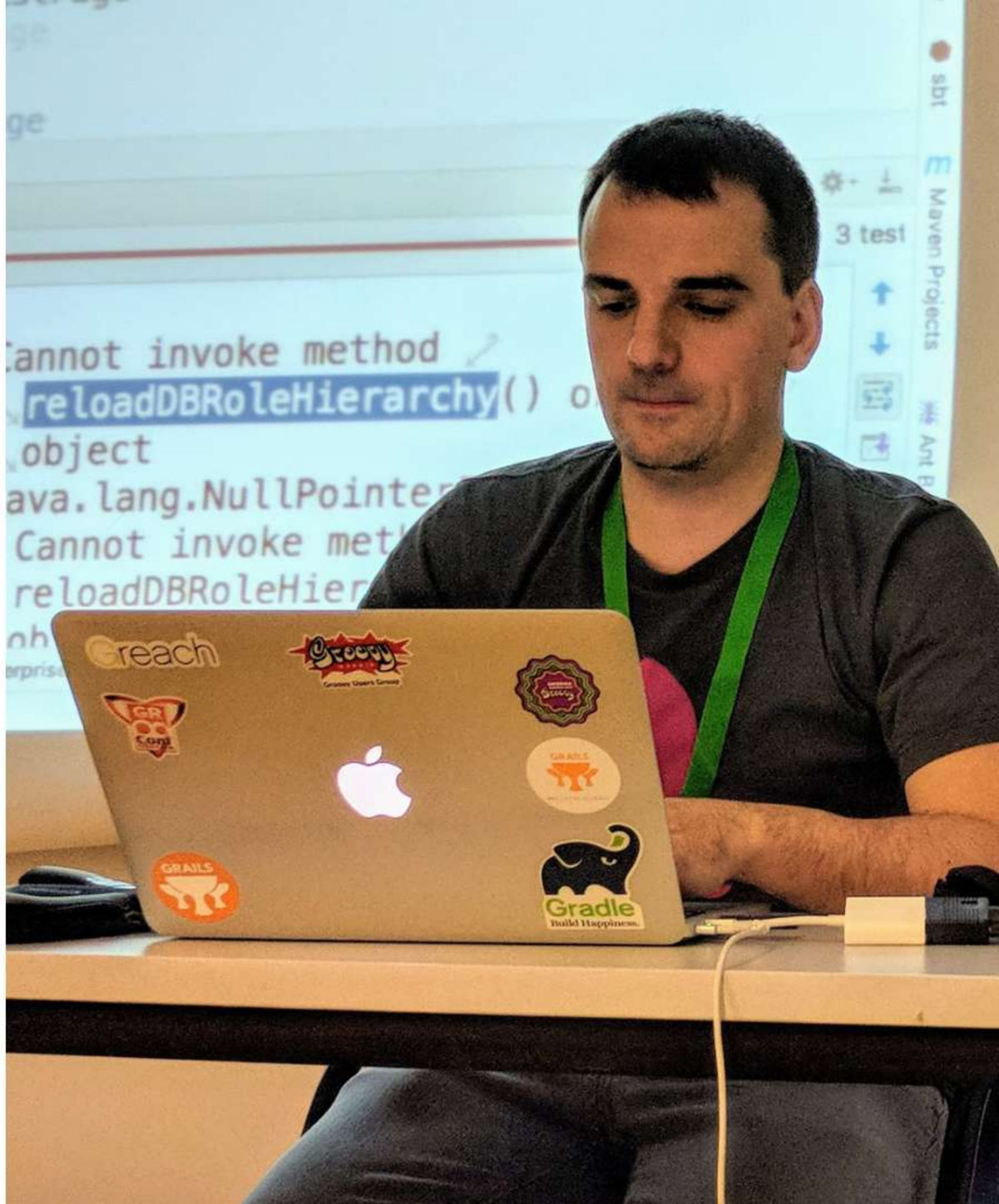
OCI | TRAINING

OCI
12140 Woodcrest Exec. Dr., Ste. 300
Saint Louis, MO 63141 USA

Software Engineering Training

- 25+ years experience
- Over 50,000 trained
- 150 current courses
- More than 40 instructors on staff
- All training delivered by practitioners and SME’s in their respective fields
- Customized to fit your specific needs

- Micronaut / Grails OCI Team
- Guadalajara, Spain
- Curator of Groovycalamari.com
- @sdelamo
- http://sergiodelamo.es
- greachconf.com organizer



OCI | WE ARE SOFTWARE ENGINEERS.

Micronaut Testing Goals

Eliminate the artificial separation imposed by traditional frameworks between functional and unit tests due to slow startup times and memory consumption.

Micronaut is Test Framework agonostic

Spock


```
plugins {
    ...
    id "groovy"
}
dependencies {
    ...
    testAnnotationProcessor platform("io.micronaut:micronaut-bom:1.3.2")
    testAnnotationProcessor "io.micronaut:micronaut-inject-java"

    testImplementation platform("io.micronaut:micronaut-bom:1.3.2")
    testImplementation("org.spockframework:spock-core") {
        exclude group: "org.codehaus.groovy", module: "groovy-all"
    }
    testImplementation "io.micronaut:micronaut-inject-groovy"
    testImplementation "io.micronaut.test:micronaut-test-spock"
    ...
}
```

JUnit 5


```
dependencies {
    testAnnotationProcessor platform("io.micronaut:micronaut-bom:1.3.2")
    testAnnotationProcessor "io.micronaut:micronaut-inject-java"

    testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine"
    testImplementation platform("io.micronaut:micronaut-bom:1.3.2")
    testImplementation "org.junit.jupiter:junit-jupiter-api"
    testImplementation "io.micronaut.test:micronaut-test-junit5"
}

test {
    useJUnitPlatform()
}
```

```
dependencies {
    ...
    kaptTest platform("io.micronaut:micronaut-bom:1.3.2")
    kaptTest "io.micronaut:micronaut-inject-java"

    testImplementation platform("io.micronaut:micronaut-bom:1.3.2")
    testImplementation "io.micronaut.test:micronaut-test-kotlintest"
    testImplementation "io.mockk:mockk:1.9.3"
    testImplementation "io.kotlintest:kotlintest-runner-junit5:3.3.2"
}
```


But, we like Spock.

Java Code, Spock tests


```
mn create-app example.micronaut.app --features=spock
```

micronaut-cli.yml

```
profile: service
defaultPackage: example.micronaut
---
testFramework: spock
sourceLanguage: java
```

```
$ mn

| Starting interactive mode...
| Enter a command name to run. Use TAB for completion:


mn> create-controller Generated

| Rendered template Controller.java to destination
src/main/java/example/micronaut/GeneratedController.java
| Rendered template ControllerSpec.groovy to destination
src/test/groovy/example/micronaut/GeneratedControllerSpec.groovy
```

Dependency Injection

Micronaut uses compile time data to implement dependency injection.

MN IoC container goals

- Use reflection as a last resort
- Avoid proxies
- Optimize start-up time
- Reduce memory footprint
- Provide clear, understandable error handling

BeanContext

The core BeanContext abstraction which allows for dependency injection of classes annotated with `@Inject`.

ApplicationContext

An application context extends a BeanContext and adds the concepts of configuration, environments and runtimes.

The entry point for IoC is the `ApplicationContext` interface, which includes a run method.

`test` environment is detected by default.

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import spock.lang.Specification

class EnvironmentDetectionSpec extends Specification {

    def "test environment is detected automatically"() {
        given:
        ApplicationContext ctx = ApplicationContext.run()

        expect:
        ctx.environment.getActiveNames().contains(Environment.TEST)

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```java
import io.micronaut.context.ApplicationContext;
import io.micronaut.context.env.Environment;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class EnvironmentDetectionTest {

    @Test
    public void testEnvironmentIsDetectedAutomatically() {
        try(ApplicationContext ctx = ApplicationContext.run()) {
            assertTrue(ctx.getEnvironment().getActiveNames().contains(Environment.TEST));
        }
    }
}
```


```
import io.kotlintest.specs.StringSpec
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment

class EnvironmentDetectionTest : StringSpec({
    "test environment is detected automatically" {
        val ctx = ApplicationContext.run()

        assert(ctx.environment.activeNames.contains(Environment.TEST))

        ctx.close()
    }
})
```

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import spock.lang.Specification
class EnvironmentDetectionSpec extends Specification {
    @AutoCleanup
    @Shared
    ApplicationContext ctx = ApplicationContext.run()

    def "test environment is detected automatically"() {
        expect:
        ctx.environment.getActiveNames().contains(Environment.TEST)
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT


```
import io.kotlintest.specs.StringSpec
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import io.micronaut.test.annotation.MicronautTest

@MicronautTest
class EnvironmentDetectionTest(ctx: ApplicationContext) : StringSpec({
    "test environment is detected automatically" {
        assert(ctx.environment.activeNames.contains(Environment.TEST))
    }
})
```

```
//src/test/groovy/example/micronaut/EnvironmentDetectionSpec
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import spock.lang.Specification

class EnvironmentDetectionSpec extends Specification {

    def "you can start the context with custom environments"() {
        given:
        String customEnv = "customevn"
        ApplicationContext ctx = ApplicationContext.run(customEnv)

        expect:
        ctx.environment.getActiveNames().contains(Environment.TEST)

        and:
        ctx.environment.getActiveNames().contains(customEnv)

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Verify bean exists


```
// src/main/java/example/micronaut/HomeController
import io.micronaut.http.HttpStatus;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.http.annotation.Status;

@Controller("/")
public class HomeController {

    @Status(HttpStatus.OK)
    @Get
    public void index() {
    }
}
```

```groovy
import io.micronaut.context.ApplicationContext
import spock.lang.Specification

class VerifyBeanExistsSpec extends Specification {

    def "verify HomeController bean exists"() {
        given:
        ApplicationContext ctx = ApplicationContext.run()

        expect:
        ctx.containsBean(HomeController)

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Verify property value

```
# src/main/resources/application.yml
micronaut:
    application:
        name: app
```

```groovy
import io.micronaut.context.ApplicationContext
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification

class PropertyValueSpec extends Specification {

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = ApplicationContext.run()

    def "application name is app"() {
        expect:
        applicationContext.getProperty('micronaut.application.name', String).get() == 'app'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT


```java
import io.micronaut.context.ApplicationContext;
import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Test;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class PropertyValueTest {

    private static ApplicationContext applicationContext;

    @BeforeAll
    public static void setupServer() {
        applicationContext = ApplicationContext.run();
    }

    @AfterAll
    public static void stopServer() {
        if (applicationContext != null) { applicationContext.stop(); }
    }

    @Test
    public void applicationNameIsApp() {
        Optional opt = applicationContext.getProperty("micronaut.application.name", String.class);
        assertTrue(opt.isPresent());
        assertEquals(opt.get(), "app");
    }
}
```

```
import io.kotlintest.specs.StringSpec
import io.micronaut.context.ApplicationContext
import java.util.Optional

class PropertyValueTest : StringSpec({

    "property value can be retrieved from application context" {
        val ctx = ApplicationContext.run()

        val opt : Optional<String> = ctx.getProperty("micronaut.application.name", String::class.java)
        assert(opt.isPresent())
        assert(opt.get() == "app")

        ctx.close()
    }
})
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Verify bean does not exist

```
package example.micronaut;

// Forgot to add @Controller annotation
public class BogusController {
}
```

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.exceptions.NoSuchBeanException
import spock.lang.Specification

class VerifyBeanExistsSpec extends Specification {

    def "verify BogusController bean does not exists"() {
        given:
        ApplicationContext ctx = ApplicationContext.run()

        expect:
        !ctx.containsBean(BogusController)

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.exceptions.NoSuchBeanException
import spock.lang.Specification

class VerifyBeanExistsSpec extends Specification {

    def "verify BogusController bean does not exists"() {
        given:
        ApplicationContext ctx = ApplicationContext.run()

        when:
        ctx.getBean(BogusController)

        then:
        thrown(NoSuchBeanException)

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```java
import io.micronaut.context.ApplicationContext;
import io.micronaut.context.exceptions.NoSuchBeanException;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class VerifyBeanExistsTest {
    @Test
    public void verifyBogusControllerBeanDoesNotExists() {
        assertThrows(NoSuchBeanException.class, () -> {
            try (ApplicationContext ctx = ApplicationContext.run()) {
                ctx.getBean(BogusController.class);
            }
        });
    }
}
```

Extra bean for just one test


```
dependencies {
    ...
    // Groovy Beans
    testCompile "io.micronaut:micronaut-inject-groovy"

    // Java / Kotlin Beans
    testAnnotationProcessor "io.micronaut:micronaut-inject-java"
    // In Kotlin project:
    // kaptTest "io.micronaut:micronaut-inject-java"

}
```

The **@Requires** annotation provides the ability to define one or many conditions on a bean.

```groovy
// src/test/groovy/example/micronaut/ExtraBean.groovy
import io.micronaut.context.annotation.Requires
import javax.inject.Singleton

@Requires(property = 'spec.name', value = 'extra')
@Singleton
class ExtraBean {
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.exceptions.NoSuchBeanException
import spock.lang.Specification

class ExtraBeanSpec extends Specification {

    def "setting spec.name to extra when you run ApplicationContext enables extra bean"() {
        given:
        ApplicationContext ctx = ApplicationContext.run(["spec.name": "extra"])

        when:
        ctx.getBean(ExtraBean)

        then:
        noExceptionThrown()

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
import io.micronaut.context.ApplicationContext
import io.micronaut.context.exceptions.NoSuchBeanException
import spock.lang.Specification

class ExtraBeanSpec extends Specification {

    def "if you don't set spec.name to extra, when you run ApplicationContext, extra bean is not loaded"() {
        given:
        ApplicationContext ctx = ApplicationContext.run()

        when:
        ctx.getBean(ExtraBean)

        then:
        thrown(NoSuchBeanException)

        cleanup:
        ctx.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Run the app from a test

To run the application from a unit test you can use the `EmbeddedServer` interface.

```groovy
package example.micronaut

import io.micronaut.context.ApplicationContext
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.Specification

class EmbeddedServerSpec extends Specification {

    def "possible to run the app from a unit test"() {
        given:
        EmbeddedServer server = ApplicationContext.run(EmbeddedServer)

        when:
        server.getApplicationContext().getBean(HomeController)

        then:
        noExceptionThrown()

        cleanup:
        server.close()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Starting a server in test is fast.

Really fast.

Testing endpoints with Micronaut HTTP Client


```
// src/main/java/example/micronaut/HomeController
import io.micronaut.http.HttpStatus;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.http.annotation.Status;

@Controller("/")
public class HomeController {

    @Status(HttpStatus.OK)
    @Get
    void index() {
    }
}
```

```
import io.micronaut.context.ApplicationContext
import io.micronaut.http.*
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class HomeControllerSpec extends Specification {

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    @AutoCleanup
    @Shared
    HttpClient client = HttpClient.create(embeddedServer.URL)

    def "test home controller exposes a GET endpoint at /"() {
        given:
        HttpRequest req = HttpRequest.GET("/")

        expect:
        client.toBlocking().exchange(req).status() == HttpStatus.OK
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

HttpClientResponseException

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.http.*
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class HomeControllerSpec extends Specification {

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    @AutoCleanup
    @Shared
    HttpClient client = HttpClient.create(embeddedServer.URL)

    def "test home controller does not expose a POST endpoint at /"() {
        given:
        HttpRequest req = HttpRequest.POST("/", "")

        when:
        HttpResponse rsp = client.toBlocking().exchange(req)

        then:
        HttpClientResponseException e = thrown()
        e.response.status() == HttpStatus.METHOD_NOT_ALLOWED
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Micronaut declarative HTTP Client in tests


```groovy
// src/test/groovy/example/micronaut/HomeClient.groovy
import io.micronaut.http.HttpStatus
import io.micronaut.http.annotation.Get
import io.micronaut.http.client.annotation.Client

@Client("/")
interface HomeClient {

    @Get
    HttpStatus index()
}
```

```
def "verify home controller with declarative HTTP Client"() {

        when:
        HomeClient homeClient = embeddedServer.applicationContext.getBean(HomeClient)

        then:
        noExceptionThrown()

        when:
        HttpStatus status = homeClient.index()

        then:
        status == HttpStatus.OK
    }
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Validation

Data validation


```
dependencies {
...
    annotationProcessor "io.micronaut:micronaut-validation"
    implementation "io.micronaut:micronaut-validation"
...
}
```


```java
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;
import io.micronaut.core.annotation.Introspected;

@Introspected
public class RegistrationCommand {
    @NotBlank
    private String username;

    @Size(min = 8)
    private String password;

    public RegistrationCommand() {}

    public RegistrationCommand(String username, String password) {
        this.username = username;
        this.password = password;
    }
    // Getter and Setter
}
```


```
import io.micronaut.context.ApplicationContext
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification
import javax.validation.Validator
class RegistrationCommandSpec extends Specification {

    @AutoCleanup @Shared ApplicationContext ctx = ApplicationContext.run()

    @Shared Validator validator = ctx.getBean(Validator)

    void "POJO validation"() {
        when:
        RegistrationCommand valid = new RegistrationCommand("sherlock", "elementary")
        RegistrationCommand notValid = new RegistrationCommand("sherlock", "12")

        then:
        validator.validate(valid).isEmpty()
        !validator.validate(notValid).isEmpty()
    }
```


```
import javax.inject.Singleton;
import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import io.micronaut.validation.Validated;

@Validated
@Singleton
public class RegistrationService {

    boolean save(@Valid @NotNull RegistrationCommand command) {
        return true;
    }
}
```


```
import io.micronaut.context.ApplicationContext
import spock.lang.*
import javax.validation.ConstraintViolationException

class RegistrationServiceSpec extends Specification {

    @Shared
    @AutoCleanup
    ApplicationContext applicationContext = ApplicationContext.run()

    def "invalid registration data triggers exception"() {
        given:
        RegistrationService registrationService = applicationContext.getBean(RegistrationService)

        when:
        registrationService.save(new RegistrationCommand('john', 'secret'))

        then:
        thrown(ConstraintViolationException)
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Test static resources

src/main/resources/static/images/logo.png


```
micronaut:
    application:
        name: agenda
    router:
        static-resources:
            default:
                enabled: true
                mapping: /**
                paths:
                    - classpath:static
```

```
import io.micronaut.context.ApplicationContext
import io.micronaut.http.*
import io.micronaut.http.client.HttpClient
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class LogoSpec extends Specification {

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    @AutoCleanup
    @Shared
    HttpClient client = HttpClient.create(embeddedServer.URL)

    def "logo.png is exposed"() {
        when:
        HttpResponse response = client.toBlocking().exchange(HttpRequest.GET("/images/logo.png"))

        then:
        response.status() == HttpStatus.OK
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

UI Tests with Geb

build.gradle

```
...
dependencies {
...
    implementation "io.micronaut:micronaut-views-velocity"
    runtime "org.apache.velocity:velocity-engine-core:2.0"

...

}
...
```


```java
// src/main/java/example/micronaut/HomeController.java
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.views.View;

import java.util.HashMap;
import java.util.Map;

@Controller
public class HomeController {

    @View("home")
    @Get
    public Map<String, Object> index() {
        return new HashMap<>();
    }
}
```

src/main/resources/views/home.vm

```
<!DOCTYPE html>
<html>
<head>
    <title>Micronaut</title>
</head>
<body>
</body>
</html>
```

Add Geb

build.gradle


```
dependencies {
...

    testImplementation "org.gebish:geb-spock:3.3"
    testImplementation "org.seleniumhq.selenium:htmlunit-driver:2.35.1"
    testRuntime 'net.sourceforge.htmlunit:htmlunit:2.35.0'

...

}
...
```

```groovy
// src/tests/resources/GebConfig.groovy
import org.openqa.selenium.htmlunit.HtmlUnitDriver

// default is to use htmlunit
driver = {
    HtmlUnitDriver htmlUnitDriver = new HtmlUnitDriver()
    htmlUnitDriver.javascriptEnabled = true
    htmlUnitDriver
}

environments {

    htmlUnit {
        driver = {
            HtmlUnitDriver htmlUnitDriver = new HtmlUnitDriver()
            htmlUnitDriver.javascriptEnabled = true
            htmlUnitDriver
        }
    }
}
```

```groovy
// src/test/groovy/example/micronaut
package example.micronaut

import geb.Page

class HomePage extends Page {

    static url = "/"

    static at = { title.contains 'Micronaut' }
}
```


```
import geb.spock.GebSpec
import io.micronaut.context.ApplicationContext
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class HomeSpec extends GebSpec {

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    def "if you visit the home page an HTML page is rendered whose title contains Micronaut"() {
        given:
        browser.baseUrl = embeddedServer.URL.toString()

        when:
        to HomePage

        then:
        at HomePage
    }
}
```

Configuration override for tests

src/main/resources/application.yml

```
company:
    address: 'St. Louis'
```

src/test/resources/application-test.yml

```
company:
    address: 'Guadalajara'
```

```groovy
import io.micronaut.context.ApplicationContext
import spock.lang.*

class PropertyOverrideTestEnvSpec extends Specification {

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = ApplicationContext.run()


    def """same property defined in src/test/resources/application-test.yml
           overrides src/main/resources/application.yml"""() {

        expect:
        applicationContext.getProperty('company.address', String).get() == 'Guadalajara'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Micronaut testing framework

Micronaut testing framework

- Automatically start and stop the server for the scope of a test suite
- Use mocks to replace existing beans for the scope of a test suite
- Allow dependency injection into a test instance

```groovy
import io.micronaut.http.*
import io.micronaut.http.client.HttpClient
import io.micronaut.http.client.annotation.Client
import io.micronaut.runtime.server.EmbeddedServer
import io.micronaut.test.annotation.MicronautTest
import spock.lang.Specification
import javax.inject.Inject

@MicronautTest
class HomeControllerSpec extends Specification {

    @Shared
    @Inject
    @Client("/")
    HttpClient client

    def "test home controller exposes a GET endpoint at /"() {
        given:
        HttpRequest req = HttpRequest.GET("/")

        expect:
        client.toBlocking().exchange(req).status() == HttpStatus.OK
    }
}
```

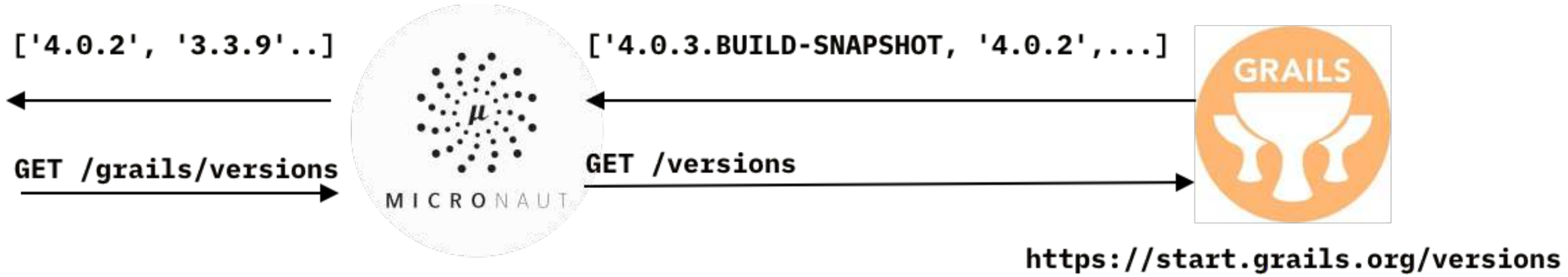
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

When you run a test within `@MicronautTest`, it is running your real application.

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.exceptions.NoSuchBeanException
import io.micronaut.test.annotation.MicronautTest
import spock.lang.Specification
import javax.inject.Inject

@MicronautTest
class VerifyBeanExistsSpec extends Specification {

    @Inject
    ApplicationContext ctx

    def "verify HomeController bean exists"() {
        expect:
        ctx.containsBean(HomeController)
    }

    def "verify BogusController bean does not exists"() {
        expect:
        !ctx.containsBean(BogusController)
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Test integration with a third party service


['4.0.2', '3.3.9'..]
GET /grails/versions
MICRONAUT
['4.0.3.BUILD-SNAPSHOT, '4.0.2',...]
GET /versions
GRAILS
https://start.grails.org/versions

```
curl start.grails.org/versions

["3.1.13","3.1.14","3.1.15","3.1.16","3.1.17.BUILD-
SNAPSHOT","3.2.2","3.2.3","3.2.4","3.2.5","3.2.6","3.2.7","
3.2.8","3.2.9","3.2.10","3.2.11","3.2.12","3.2.13","3.2.14.
BUILD-
SNAPSHOT","3.3.0","3.3.1","3.3.2","3.3.3","3.3.4","3.3.5","
3.3.6","3.3.7","3.3.8","3.3.9","3.3.10.BUILD-SNAPSHOT"]
```

```
// src/main/java/example/micronaut/GrailsClient.java
import java.util.List;

public interface GrailsClient {
    List<String> versions();
}
```


```
// src/main/java/example/micronaut/GrailsApplicationForge.java
import io.micronaut.http.annotation.Get;
import io.micronaut.http.client.annotation.Client;
import java.util.List;

@Client("grailsappforge")
public interface GrailsApplicationForge extends GrailsClient {

    @Override
    @Get("/versions")
    List<String> versions();
}
```

src/main/resources/application.yml


```yaml
micronaut:
    http:
        services:
            grailsappforge:
                url: 'https://start.grails.org'
```


```java
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import java.util.List;
import java.util.stream.Collectors;

@Controller("/grails")
public class VersionsController {
    private final GrailsClient grailsClient;

    public VersionsController(GrailsClient grailsClient) {
        this.grailsClient = grailsClient;
    }

    @Produces(MediaType.TEXT_PLAIN)
    @Get("/versions")
    public List<String> versions() {
            return grailsClient.versions().stream()
                    .filter(s -> !s.endsWith("BUILD-SNAPSHOT"))
                    .collect(Collectors.toList());
    }
}
```


```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.core.type.Argument
import io.micronaut.http.HttpRequest
import io.micronaut.http.client.HttpClient
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class VersionsControllerSpec extends Specification {

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    @AutoCleanup
    @Shared
    HttpClient client = HttpClient.create(embeddedServer.URL)

    def "versions do not include BUILD-SNAPSHOT"() {
        given:
        HttpRequest req = HttpRequest.GET("/grails/versions")

        when:
        List<String> versions = client.toBlocking().retrieve(req, Argument.listOf(String))

        then:
        versions
        !versions.any { it.endsWith('BUILD-SNAPSHOT')}
    }
}
```

```
List<String> versions = client
    .toBlocking()
    .retrieve(req, Argument.listOf(String))
```

Use `Argument` to bind response's body to `List<?>`.

Don't use `GrailsApplicationForge`. Use a different implementation of `@GrailsClient`

OPTION # 1 - Micronaut Test @MockBean

```groovy
@MicronautTest
class VersionsControllerSpec extends Specification {
    @Inject EmbeddedServer embeddedServer
    @Inject GrailsClient grailsClient

    def "versions do not include BUILD-SNAPSHOT"() {
        given:
        HttpClient client = HttpClient.create(embeddedServer.URL)
        HttpRequest req = HttpRequest.GET("/grails/versions")
        when:
        List<String> versions = client.toBlocking().retrieve(req, Argument.listOf(String))
        then:
        1 * grailsClient.versions() >> ['3.3.9', '3.3.10.BUILD-SNAPSHOT']
        versions
        versions.size() == 1
        !versions.any { it.endsWith('BUILD-SNAPSHOT')}
        cleanup:
        client.close()
    }
    @MockBean(GrailsApplicationForge)
    GrailsClient grailsClient() {
        Mock(GrailsClient)
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT


```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
@MicronautTest
public class VersionsControllerTest {
    @Inject EmbeddedServer embeddedServer;
    @Inject GrailsClient grailsClient;

    @Test
    public void versionsDoNotIncludeBuildSnapshot() {
        when(grailsClient.versions())
                .then(invocation -> Arrays.asList("3.3.9", "3.3.10.BUILD-SNAPSHOT"));
        try(HttpClient client = HttpClient.create(embeddedServer.getURL())) {
            HttpRequest req = HttpRequest.GET("/grails/versions");
            List<String> versions = client.toBlocking().retrieve(req, Argument.listOf(String.class));
            assertNotNull(versions);
            assertEquals(versions.size(), 1);
            assertFalse(versions.stream().anyMatch(i -> i.endsWith("BUILD-SNAPSHOT")));
        }
        verify(grailsClient).versions();
    }
    @MockBean(GrailsApplicationForge.class)
    GrailsClient grailsClient() {
        return mock(GrailsClient.class);
    }
}
```

Option 2 - @Replaces


```
import io.micronaut.context.annotation.Replaces
import io.micronaut.context.annotation.Requires
import io.micronaut.context.env.Environment
import javax.inject.Singleton

@Replaces(GrailsApplicationForge)
@Requires(env = Environment.TEST)
@Requires(property= 'spec.name', value = 'atreplaces')
@Singleton
class GrailsClientReplacement implements GrailsClient {

    @Override
    List<String> versions() {
        ['3.3.9', '3.3.10.BUILD-SNAPSHOT']
    }
}
```

```groovy
class VersionsControllerAtReplacesSpec extends Specification {
    @Shared Map<String, Object> conf = ['spec.name': 'atreplaces']

    @AutoCleanup @Shared EmbeddedServer server = ApplicationContext.run(EmbeddedServer, conf)

    @AutoCleanup @Shared HttpClient client = HttpClient.create(server.URL)

    def "versions do not include BUILD-SNAPSHOT"() {
        given:
        HttpRequest req = HttpRequest.GET("/grails/versions")

        when:
        List<String> versions = client.toBlocking().retrieve(req, Argument.listOf(String))

        then:
        versions
        versions.size() == 1
        !versions.any { it.endsWith('BUILD-SNAPSHOT')}
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Option 3 - @Primary

```groovy
import io.micronaut.context.annotation.Primary
import io.micronaut.context.annotation.Requires

import javax.inject.Singleton

@Primary
@Requires(env = Environment.TEST)
@Requires(property= 'spec.name', value = 'primary')
@Singleton
class GrailsClientPrimary implements GrailsClient {

    @Override
    List<String> versions() {
        ['3.3.9', '3.3.10.BUILD-SNAPSHOT']
    }
}
```

OPTION # 4 - Fallback


```groovy
package example.micronaut

import io.micronaut.context.annotation.Requires
import io.micronaut.context.env.Environment
import io.micronaut.retry.annotation.Fallback

@Fallback
@Requires(env = Environment.TEST)
@Requires(property= 'spec.name', value = 'atfallback')
class GrailsClientFallback implements GrailsClient {

    @Override
    List<String> versions() {
        ['3.3.9', '3.3.10.BUILD-SNAPSHOT']
    }
}
```

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.core.type.Argument
import io.micronaut.http.HttpRequest
import io.micronaut.http.client.*
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class VersionsControllerAtFallbackSpec extends Specification {

    @Shared
    Map<String, Object> conf = ['spec.name': 'atfallback',
                                'micronaut.http.services.grailsappforge.url': 'https://nonreachable.grails.org'
    ]

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, conf)

    @AutoCleanup
    @Shared
    HttpClient httpClient = HttpClient.create(embeddedServer.URL)

    BlockingHttpClient getClient() {
        httpClient.toBlocking()
    }

    def "versions do not include BUILD-SNAPSHOT"() {
        given:
        HttpRequest req = HttpRequest.GET("/grails/versions")

        when:
        List<String> versions = client.retrieve(req, Argument.listOf(String))

        then:
        versions
        versions.size() == 1
        !versions.any { it.endsWith('BUILD-SNAPSHOT')}
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Option 5 - Mock Http Server - Ersatz

Ersatz Server

Mock HTTP server for testing client code.

http://stehno.com/ersatz/

```
dependencies {
    ...
    testImplementation "com.stehno.ersatz:ersatz:1.9.0"
    ...
}
```


```groovy
class VersionsControllerErsatzSpec extends Specification {

  def "versions do not include BUILD-SNAPSHOT"() {
        setup:
        String path = '/versions'
        ErsatzServer ersatz = new ErsatzServer()
        ersatz.expectations {
            get(path) {
                called 1
                responder {
                    body("['3.3.9', '3.3.10.BUILD-SNAPSHOT']", ContentType.TEXT_PLAIN)
                }
            }
        }

        and:
        EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, [
                'micronaut.http.services.grailsappforge.url': ersatz.httpUrl
        ], Environment.TEST)
        HttpClient httpClient = HttpClient.create(embeddedServer.URL)
       ....
```

```
        ...
        when:
        HttpRequest req = HttpRequest.GET("/grails/versions")
        List<String> versions = httpClient.toBlocking().retrieve(req, Argument.of(List, String))

        then:
        versions
        versions.size() == 1
        !versions.any { it.endsWith('BUILD-SNAPSHOT') }

        and:
        ersatz.verify()

        cleanup:
        ersatz.stop()

        and:
        httpClient.close()

        and:
        embeddedServer.close()
    }
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Option 5 - Mock Http Server - Micronaut


```
package example.micronaut

import io.micronaut.context.annotation.Requires
import io.micronaut.http.MediaType
import io.micronaut.http.annotation.Controller
import io.micronaut.http.annotation.Get
import io.micronaut.http.annotation.Produces

@Requires(property = 'spec.name', value='mockHttpServer')
@Controller("/")
class MockVersionsController  {

    @Produces(MediaType.TEXT_PLAIN)
    @Get("/versions")
    List<String> index() {
        ['3.3.9', '3.3.10.BUILD-SNAPSHOT']
    }
}
```


```
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import io.micronaut.core.io.socket.SocketUtils
import io.micronaut.core.type.Argument
import io.micronaut.http.HttpRequest
import io.micronaut.http.client.HttpClient
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.Specification

class VersionsControllerMicronautMockHttpServerSpec extends Specification {

    def "versions do not include BUILD-SNAPSHOT"() {
        setup:
        int mockHttpServerPort = SocketUtils.findAvailableTcpPort()
        EmbeddedServer mockHttpServer = ApplicationContext.run(EmbeddedServer, [
                        'micronaut.security.enabled': false,
                        'spec.name': 'mockHttpServer',
                        'micronaut.server.port': mockHttpServerPort
                ])

        expect:
        mockHttpServer.applicationContext.containsBean(MockVersionsController)

        when:
        String mockHttpServerUrl = "http://localhost:$mockHttpServerPort"
        EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, [
                'micronaut.http.services.grailsappforge.url': mockHttpServerUrl
        ], Environment.TEST)
        ...
```


```
        ...
        HttpClient httpClient = HttpClient.create(embeddedServer.URL)
        HttpRequest req = HttpRequest.GET("/grails/versions")
        List<String> versions = httpClient.toBlocking().retrieve(req, Argument.of(List, String))

        then:
        versions
        versions.size() == 1
        !versions.any { it.endsWith('BUILD-SNAPSHOT') }

        and:
        httpClient.close()

        and:
        embeddedServer.close()

        and:
        mockHttpServer.close()
    }
}
```

Or if TEST env, don't load

```
@Requires(notEnv = Environment.TEST)
```


```
import io.micronaut.context.annotation.Requires;
import io.micronaut.context.env.Environment;
import io.micronaut.http.annotation.Get;
import io.micronaut.http.client.annotation.Client;
import java.util.List;

@Client("grailsappforge")
@Requires(notEnv = Environment.TEST)
public interface GrailsApplicationForge extends GrailsClient {

    @Override
    @Get("/versions")
    List<String> versions();
}
```

Test-specific configuration

```
// src/main/java/example/micronaut/Address.java
public interface Address {
    String getCity();
    String getState();
}
```


```java
// src/main/java/example/micronaut/AddressConfigurationProperties.java
import io.micronaut.context.annotation.ConfigurationProperties;

@ConfigurationProperties("address")
public class AddressConfigurationProperties implements Address {
    private String city;
    private String state;

    public void setCity(String city) { this.city = city; }

    public void setState(String state) { this.state = state; }

    @Override
    public String getCity() { return this.city; }

    @Override
    public String getState() { return this.state; }
}
```

```groovy
import io.micronaut.context.annotation.Property
import io.micronaut.test.annotation.MicronautTest
import spock.lang.Specification
import javax.inject.Inject

@MicronautTest
@Property(name = "address.city", value = "St. Louis")
@Property(name = "address.state", value = "MO")
class TestSpecificPropertySpec extends Specification {

    @Inject
    Address address

    def "load properties"() {
        expect:
        address.state == 'MO'
        address.city == 'St. Louis'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```groovy
package example.micronaut

import io.micronaut.context.ApplicationContext
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.Specification

class TestSpecificPropertySpec extends Specification {

    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, [
            "address.city": "St. Louis",
            "address.state": "MO",
    ])

    def "load properties"() {
        given:
        Address address = embeddedServer.applicationContext.getBean(Address)

        expect:
        address.state == 'MO'
        address.city == 'St. Louis'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Specify additional `propertySources` in any supported format (YAML, JSON, Java properties file etc.) using the `@MicronautTest` annotation

src/test/properties/contact.properties

```
address.city=St. Louis
address.state=MO
```

```
import io.micronaut.test.annotation.MicronautTest
import spock.lang.Specification
import javax.inject.Inject

@MicronautTest(propertySources = "classpath:address.properties")
class AddressSpec extends Specification {

    @Inject
    Address address

    def "load properties"() {
        expect:
        address.state == 'MO'
        address.city == 'St. Louis'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Security

src/main/resources/application.yml

```
micronaut:
    security:
        enabled: true
```

build.gradle

```
...
dependencies {
...
    annotationProcessor "io.micronaut:micronaut-security"
    implementation "io.micronaut:micronaut-security"
}
```


```java
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.security.annotation.Secured;
import io.micronaut.security.rules.SecurityRule;

@Secured(SecurityRule.IS_AUTHENTICATED)
@Controller("/address")
public class AddressController {

    private final Address address;

    public AddressController(Address address) {
        this.address = address;
    }

    @Get
    public Address index() {
        return address;
    }
}
```

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import io.micronaut.http.*
import io.micronaut.http.client.HttpClient
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.*

class AddressControllerSpec extends Specification {

    @Shared @AutoCleanup EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, [
            "spec.name": 'AddressControllerSpec',
            "micronaut.security.enabled": false,
            "address.city": "Guadalajara",
            "address.state": "Castilla-La Mancha",
    ])

    @Shared @AutoCleanup HttpClient client = HttpClient.create(embeddedServer.URL)

    def "verify address endpoint"() {
        when:
        HttpResponse<Map> resp = client.toBlocking().exchange('/address', Map)

        then:
        resp.status() == HttpStatus.OK
        resp.body().city == 'Guadalajara'
        resp.body().state == 'Castilla-La Mancha'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT


```
import io.micronaut.context.ApplicationContext
import io.micronaut.context.exceptions.NoSuchBeanException
import io.micronaut.http.*
import io.micronaut.http.client.HttpClient
import io.micronaut.http.client.exceptions.HttpClientResponseException
import io.micronaut.runtime.server.EmbeddedServer
import io.micronaut.security.authentication.AuthenticationProvider
import io.micronaut.security.filters.SecurityFilter
import io.micronaut.security.token.reader.TokenReader
import spock.lang.*

class SecuritySpec extends Specification {

    @AutoCleanup @Shared EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, [
            'spec.name': 'SecuritySpec'
    ])

    @AutoCleanup @Shared HttpClient client = HttpClient.create(embeddedServer.URL)


    def "/address is secured"() {
        when:
        client.toBlocking().exchange(HttpRequest.GET("/address"), Map)

        then:
        HttpClientResponseException e = thrown()
        e.response.status() == HttpStatus.UNAUTHORIZED
    }
}
```

or get fancy with HTTP Client filters


```
import io.micronaut.context.annotation.Requires
import io.micronaut.context.env.Environment
import io.micronaut.security.authentication.*
import io.micronaut.security.authentication.UserDetails
import io.reactivex.Flowable
import org.reactivestreams.Publisher
import javax.inject.Singleton

@Requires(env = Environment.TEST)
@Requires(property = 'spec.name', notEquals = 'SecuritySpec')
@Singleton
class MockAuthenticationProvider implements AuthenticationProvider {

    @Override
    Publisher<AuthenticationResponse> authenticate(AuthenticationRequest authenticationRequest) {
        if ( authenticationRequest.identity == 'user' && authenticationRequest.secret == 'password' ) {
            return Flowable.just(new UserDetails('user', []))
        }
        return Flowable.just(new AuthenticationFailed())
    }
}
```


```groovy
import io.micronaut.context.annotation.Requires
import io.micronaut.context.env.Environment
import org.reactivestreams.Publisher
import io.micronaut.http.annotation.Filter
import io.micronaut.http.*
import io.micronaut.http.filter.*

@Filter('/**')
@Requires(env = Environment.TEST)
@Requires(property = 'spec.name', notEquals = 'SecuritySpec')
class SecurityBypassFilter implements HttpClientFilter {

    @Override
    Publisher<? extends HttpResponse<?>> doFilter(MutableHttpRequest<?> request, ClientFilterChain chain) {
        request.basicAuth('user', 'password')
        return chain.proceed(request)
    }
}
```

```groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.context.env.Environment
import io.micronaut.runtime.server.EmbeddedServer
import io.micronaut.http.*
import io.micronaut.http.client.*
import spock.lang.*

class AddressControllerSpec extends Specification {

    @Shared @AutoCleanup EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, [
            "spec.name": 'AddressControllerSpec',
            "address.city": "Guadalajara",
            "address.state": "Castilla-La Mancha",
    ])

    @Shared
    @AutoCleanup
    HttpClient httpClient = embeddedServer.applicationContext.createBean(HttpClient, embeddedServer.URL)

    def "verify address endpoint"() {
        when:
        HttpResponse<Map> resp = client.toBlocking().exchange('/address', Map)
        then:
        resp.status() == HttpStatus.OK
        resp.body().city == 'Guadalajara'
        resp.body().state == 'Castilla-La Mancha'
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

If you create the client with `HttpClient.create`, you need to shutdown the client and, of course, no dependency injection will occur for the created client.

```
HttpClient client = HttpClient.create(embeddedServer.URL)
```

If you want features such as `HttpClientFilter`, register the client as a bean:

```
HttpClient httpClient = embeddedServer.applicationContext
                                      .createBean(HttpClient, embeddedServer.URL)
```

Package scanning Testing

Although Micronaut itself doesn’t scan the classpath, some integrations do (such as JPA and GORM).

For these cases, you may wish to specify either the application class or the packages.


```java
package example.micronaut.entities;

import javax.annotation.Nonnull;
import javax.persistence.*;
import javax.validation.constraints.*;

@Entity
@Table(name = "book")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Nonnull
    private Long id;

    @Nonnull
    @NotBlank
    @Column(name = "name", nullable = false, unique = true)
    private String name;

    public Book() {}

    public Book(@Nonnull @NotNull String name) {
        this.name = name;
    }

    // Getters and Setters
}
```


```
datasources:
    default:
        username: ${JDBC_USER:sa}
        password: ${JDBC_PASSWORD:""}
        driverClassName: ${JDBC_DRIVER:org.h2.Driver}
        url: ${JDBC_URL:`jdbc:h2:mem:default;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE`}
jpa:
    default:
        packages-to-scan:
            - 'example.micronaut.entities'
        properties:
            hibernate:
                hbm2ddl:
                    auto: update
                show_sql: true
```

```
package example.micronaut.repository;

import example.micronaut.entities.Book;

import javax.validation.constraints.NotEmpty;

public interface BookRepository {
    Book save(@NotEmpty String name);
}
```


```java
package example.micronaut.repository;

import example.micronaut.entities.Book;
import io.micronaut.configuration.hibernate.jpa.scope.CurrentSession;
import io.micronaut.runtime.ApplicationConfiguration;
import io.micronaut.spring.tx.annotation.Transactional;

import javax.annotation.Nonnull;
import javax.inject.Singleton;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.validation.constraints.NotEmpty;

@Singleton
public class BookRepositoryImpl implements BookRepository {

    @PersistenceContext
    private EntityManager entityManager;

    public BookRepositoryImpl(@CurrentSession EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    @Override
    @Transactional
    public Book save(@Nonnull @NotEmpty String name) {
        Book book = new Book(name);
        entityManager.persist(book);
        return book;
    }
}
```

```groovy
package example.micronaut.repository

import io.micronaut.test.annotation.MicronautTest
import spock.lang.Specification
import javax.inject.Inject

@MicronautTest(packages="example.micronaut.entities")
class BookRepositorySpec extends Specification {

    @Inject
    BookRepository bookRepository

    def "save book works"() {
        when:
        bookRepository.save('Building microservices')

        then:
        noExceptionThrown()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```groovy
package example.micronaut.repository

import io.micronaut.context.ApplicationContext
import spock.lang.*

class BookRepositorySpec extends Specification {

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = ApplicationContext.build()
                                                              .packages("example.micronaut.entities")
                                                              .start()


    def "save book works"() {
        when:
        BookRepository bookRepository = applicationContext.getBean(BookRepository)

        then:
        noExceptionThrown()

        when:
        bookRepository.save('Building microservices')

        then:
        noExceptionThrown()
    }
}
```

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Test Containers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Add tests container dependencies


```
dependencies {
    ...
    testImplementation "org.testcontainers:spock:1.12.5"
    testImplementation "org.testcontainers:mysql:1.12.5"
}
```


```groovy
import org.testcontainers.containers.MySQLContainer

class MySQL {
    static MySQLContainer mysqlContainer

    static init() {
        if (mysqlContainer == null) {
            mysqlContainer = new MySQLContainer()
                    .withDatabaseName('db')
                    .withUsername('sherlock')
                    .withPassword('elementary')
            mysqlContainer.start()
        }
    }
    static void destroy() {
        if (mysqlContainer != null) {
            mysqlContainer.stop()
        }
    }
}
```


```
trait MySQLContainerFixture {

    Map<String, Object> getMySQLConfiguration() {
        if (MySQL.mysqlContainer == null || !MySQL.mysqlContainer.isRunning()) {
            MySQL.init()
        }
        [
                'datasources.default.url'     : MySQL.mysqlContainer.getJdbcUrl(),
                'datasources.default.password': MySQL.mysqlContainer.getPassword(),
                'datasources.default.username': MySQL.mysqlContainer.getUsername(),
        ]
    }
}
```

Cleanup


```
import org.spockframework.runtime.extension.AbstractGlobalExtension

class MySQLCleanup extends AbstractGlobalExtension {

    @Override
    void stop() {
        MySQL.destroy()
    }

}
```

ApplicationContextSpecification


```
trait ConfigurationFixture implements MySQLContainerFixture {
    Map<String, Object> getConfiguration() {

        Map<String, Object> m = [:]

        if (specName) {
            m['spec.name'] = specName
        }
        m += mySQLConfiguration
        m
    }

    String getSpecName() {
        null
    }
}
```


```groovy
//src/test/groovy/example/micronaut/ApplicationContextSpecification.groovy
import io.micronaut.context.ApplicationContext
import io.micronaut.core.io.socket.SocketUtils
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification

abstract class ApplicationContextSpecification extends Specification
        implements ConfigurationFixture, LeakageDetector {

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = ApplicationContext.run(configuration)

    def cleanup() {
        assert !hasLeakage()
    }
}
```

```
trait LeakageDetector extends RepositoriesFixture {

    boolean hasLeakage() {
        if (userRepository.count() > 0) {
            println "there are still users"
        }

        tallyRepository.count() > 0
    }
}
```

```
import example.micronaut.UserRepository
import io.micronaut.context.ApplicationContext

trait RepositoriesFixture {
    abstract ApplicationContext getApplicationContext()

    UserRepository getUserRepository() {
        applicationContext.getBean(UserRepository)
    }

}
```

```
class UserRepositorySpec
    extends ApplicationContextSpecification {
    void "it is possible to save a user with name"() {
        given:
        String name = 'Sergio'

        when:
        UserEntity user = userRepository.save('Sergio')

        then:
        user.count() == old(user.count()) + 1

        cleanup:
        userRepository.delete(user)
    }
}
```

EmbeddedServerSpecification


```
import io.micronaut.context.ApplicationContext
import io.micronaut.http.client.BlockingHttpClient
import io.micronaut.http.client.HttpClient
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification

abstract class EmbeddedServerSpecification extends Specification implements ConfigurationFixture, LeakageDetector {

    @AutoCleanup @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, configuration)

    @AutoCleanup @Shared
    ApplicationContext applicationContext = embeddedServer.applicationContext

    @AutoCleanup @Shared
    HttpClient httpClient = embeddedServer.applicationContext.createBean(HttpClient, embeddedServer.URL)

    BlockingHttpClient getClient() {
        httpClient.toBlocking()
    }
    def cleanup() {
        assert !hasLeakage()
    }
}
```


```
class EndpointsSpec extends EmbeddedServerSpecification {

    @Unroll
    def "#endpoint is available"(String endpoint, String jsonKey) {
        when:
        HttpResponse<String> response = client.exchange(HttpRequest.GET(endpoint), String)

        then:
        noExceptionThrown()
        response.status() == HttpStatus.OK
        response.body().contains("\"${jsonKey}\"")

        where:
        endpoint   | jsonKey
        '/health'  | "status"
        '/info'    | "version"
}
```

Testing Functions

```
interface PirateTranslator {
    fun translate(message: String): String
}
```


```
package example.micronaut

import javax.inject.Singleton

@Singleton
class DefaultPirateTranslator : PirateTranslator {

    val replacements = mapOf("Hello" to "Ahoy!", "Yes" to "Aye!", "Yes, Captain!" to "Aye Aye!")

    override fun translate(message: String): String {
        var result = message
        replacements.forEach { k, v -> result = result.replace(k, v) }
        return result
    }
}
```

Input POJO

```
package example.micronaut

data class HandlerInput(val message: String)
```

Output POJO

```
package example.micronaut

data class HandlerOutput(val pirateMessage: String)
```


```
package example.micronaut

import io.micronaut.function.FunctionBean
import java.util.function.Function

@FunctionBean("pirate-translator")
class PirateTranslatorFunction(val translator: PirateTranslator) : Function<HandlerInput, HandlerOutput> {

    override fun apply(t: HandlerInput): HandlerOutput {
        return HandlerOutput(translator.translate(t.message))
    }
}
```

```
dependencies {
    ...
    // favourite testing framework dependencies
    testRuntime "io.micronaut:micronaut-http-server-netty"
    testRuntime "io.micronaut:micronaut-function-web"
}
```


```
package example.micronaut

import io.micronaut.function.client.FunctionClient
import io.micronaut.http.annotation.Body
import io.reactivex.Single
import javax.inject.Named

@FunctionClient
interface PirateTranslatorClient {

    @Named("pirate-translator")
    fun index(@Body input: HandlerInput): Single<HandlerOutput>

}
```


```
package example.micronaut

import io.micronaut.context.ApplicationContext
import io.micronaut.runtime.server.EmbeddedServer
import org.jetbrains.spek.api.Spek
import org.jetbrains.spek.api.dsl.describe
import org.jetbrains.spek.api.dsl.it
import org.junit.jupiter.api.Assertions.assertEquals

class PirateTranslatorFunctionTest: Spek({

    describe("pirate-translator function") {
        val server = ApplicationContext.run(EmbeddedServer::class.java)
        val client = server.applicationContext.getBean(PirateTranslatorClient::class.java)

        it("Hello is translated to Ahoy") {
            val input = HandlerInput("Hello, I am Captain Jack Sparrow")
            assertEquals("Ahoy!, I am Captain Jack Sparrow", client.index(input).blockingGet().pirateMessage)
        }

        afterGroup {
            server.stop()
        }
    }
})
```

Q & A