

Testing

Spock

Default test runner since Grails 2.3

Behavior-style test framework in Groovy with support for easy data-driven testing

Behavior-Style Testing

Test cases separated into three main sections

given (setup)

when (execute method under test)

then (verify results)

Testing Strategies

Three different strategies for testing

- Unit
- Integration
- Functional

Unit Tests

- Pared-down application context
- No rich features, like metaClass decorations, database, or object marshallers
- Must explicitly Bootstrap and mimic aspects that you need
 - No "Grails Environment" is explicitly bootstrapped

Unit Tests

- Super fast
- Can be run directly as JUnit tests
- Excellent for testing a distinct feature
 - Like, ensuring an interaction occurs
- Interest is in interaction facts -- "a method *did* get called, as I expected"

Unit Tests in Grails

Uses `@TestFor` annotation to tell Grails what you are testing.

```
@TestFor(User)
```

```
@TestFor(QuestionController)
```

Unit Tests in Grails

Uses `@Mock` annotation to tell Grails what classes to substitute behaviour.

```
@TestFor (QuestionController)
```

```
@Mock (Question)
```

- The QuestionController is ready for you to test.
- Question domain's behaviour is replaced.

Writing a Spock Test

Test class name ends in Spec or Specification
and class extends `spock.lang.Specification`

```
class UserSpec extends Specification {  
    ...  
}
```

Test Case Name

Test case method names can be descriptive sentences

```
void "test custom validator for no users with the name Justin Bieber"() {  
    ...  
}
```

Test Case Body

```
void "Question should validate"() {  
    given:  
    def u = Mock(User)  
  
    when:  
    def q = new Question(title: 'What is def?',  
        text: 'Please explain this thing called def.',  
        user: u)  
    q.validate()  
  
    then:  
    q.hasErrors() == false  
}
```

Data Driven tests

Run same test body with multiple sets of test inputs and expected outputs

Data Driven Testing

where:

input1		input2		output
4		6		5
12		18		15
20		14		17

@Unroll

```
@Unroll

void "User custom validation"() {
    mockForConstraintsTests(User)

    when:

    User u = new User(...)

    u.validate()

    then:

    u.hasErrors() == !valid

    where:

    username | firstName | lastName | email | valid
    "theBiebs" | "Justin"| "Bieber" | "justin@example.com" | false
```

Adding more info to Test output

```
void "User #username, #firstName, #lastName passes custom validation #valid"() {  
    ...  
}
```

Groovy Power Assert

```
def "x plus y equals z"() {  
    when:  
        int x = 4  
        int y = 5  
        int z = 10  
  
    then:  
        assert x + y == z  
}
```


Detailed output

Condition not satisfied:

$x + y == z$

| | | | |

4 9 5 | 10

false

Testing Controllers

```
@TestFor (QuestionController)
@Mock (Question)
class QuestionControllerSpec extends Specification {
    ...
}
```

Testing Controllers

```
void "Question Controller returns json"() {  
    given:  
        controller.response.format = 'json'  
  
    when:  
        controller.index()  
  
    then:  
        controller.response.status == 200  
        response.contentType ==  
            'application/json;charset=UTF-8'  
}
```

Workshop

Create a test in `VoteControllerSpec` that validates the redirect for `voteUpQuestion`

Integration Testing

- Ability to test more extensive feature sets
 - Domain class validation
 - Constraints
 - Custom data retrieval (HQL/JPQL)
 - Data made it through an entire workflow
- Less granular than unit testing

Integration Testing

Integration tests spin up a full Grails environment, including wiring all beans and connecting to a database

- All Grails dynamic methods are available
- Beans can be injected into tests
- Data can be added in `Bootstrap.groovy`
- Full lifecycle tests can be performed

Create Integration test

```
> grails create-integration-test com.opi.  
QuestionController
```

Integration Tests

Controllers are not injected, so you have to create them.

Services can be manually injected

```
QuestionController controller
```

```
MyService myService
```

```
def setup() {
```

```
    controller = new QuestionController()
```

```
}
```


Simple Example

```
void "Question Controller returns json"() {  
    given:  
        controller.response.format = 'json'  
        controller.params.id = '1'  
  
    when:  
        controller.show()  
  
    then:  
        controller.response.status == 200  
        controller.response.contentType ==  
            'application/json;charset=UTF-8'  
}
```

Functional Testing

Grails application is now listening and responding to actual HTTP requests

useful for end-to-end testing scenarios, such as making REST calls against a JSON API.

Enabling the Functional Test Phase

If your project doesn't already have a plugin that enables functional testing (e.g. Geb, Webdriver, etc.), then you'll need to add code similar to this to your `scripts/_Events.groovy` file to enable the functional test phase.

If you do have a plugin that enables the functional test phase, don't add this to your `_Events.groovy` file. Otherwise your functional tests may run twice.

Enabling Functional Testing Phase

```
eventAllTestsStart = {  
    if (getBinding().variables.containsKey("functionalTests"))  
    {  
        functionalTests << "functional"  
    }  
}
```

You can now run

```
grails test-app functional:
```

For More info:

<http://www.objectpartners.com/2014/07/15/grails-api-functional-testing/>

Functional API Testing

With Rest Client Builder

```
def 'should fetch person with Grails REST client
builder'() {
    given:
        User user = new User(username: "sabersd", firstName:
            "Doug", lastName: "sabers", email: "me@opi.
            com")

    RestBuilder rest = new RestBuilder()
```

Functional API Testing

when:

```
RestResponse response = rest.get(
    "http://localhost:8080/question/user/${user.id}.json")
{
    // Need to set the accept content-type to JSON,
    //otherwise it defaults to String
    // and the API will throw a 415 'unsupported media
type'
    accept JSON
}
```

then:

```
assert response.status == 200
assert response.json.firstName == user.firstName
assert response.json.lastName == user.lastName
```

Functional Spock

Allows you to write and run Spock specs under the functional test scope

<http://grails.org/plugin/functional-spock>

Geb

Geb is a groovy wrapper around WebDriver

“brings together the power of WebDriver, the elegance of jQuery content selection, the robustness of Page Object modelling and the expressiveness of the Groovy language”

<http://www.gebish.org>

Has good documentation

Brief History

Selenium 1 (JavaScript)

WebDriver (Native)

Selenium 1 + WebDriver = Selenium 2 (Native)

Selenium 2 + Groovy = Geb

Geb Example

```
import geb.Browser

Browser.drive {
  go "http://myapp.com/login"

  assert $("h1").text() == "Please Login"

  $("form.login").with {
    username = "admin"
    password = "password"
    login().click()
  }

  assert $("h1").text() == "Admin Section"
}
```

Dynamic Content

If you have a lot of ajax on the screen, waitFor is your friend

```
waitFor {  
    $("div.alert").displayed  
}
```

```
waitFor {  
    $("div.message").text() == "Update successful"  
}
```

Page Object Pattern

- Abstract page-specific details into helper classes (Page Objects)
- Re-used across tests
- Single point of maintenance
- Enhance test readability

Page example

```
class QuestionCreatePage extends geb.Page {  
  static url = "question/create"  
  static at = { $("div#create-question").displayed }  
  
  static content = {  
    titleField(wait: true) { $("input#title") }  
    descriptionField(wait: true) { $("textarea#text") }  
  
    createButton(wait: true, to: QuestionShowPage) {  
      $("input#create") }  
    }  
  }  
}
```

Spock Test

```
def "should create questions"() {  
    given: "the question creation page"  
    QuestionCreatePage questionCreatePage = to(QuestionCreatePage)  
  
    when: "the user creates a new question"  
    String title = "I have a great Question"  
    String text = "Is this going to work?"  
  
    QuestionShowPage questionShowPage =  
        questionCreatePage.createQuestion(title, text)  
    then: "should show the new question"  
    assert questionShowPage.titleText == title  
    assert questionShowPage.textText == text  
}
```

Cross Browser Testing

- Manually downloading driver libraries on each machine is a pain
- Instead, let's automatically download the library Download based on current OS
- See example GebConfig.groovy

Headless Testing

X Virtual Frame Buffer (xvfb)

Simulated browsers (HtmlUnit, PhantomJS)

HTML Unit

Uses Rhino JavaScript engine

Can get into dependency conflict hell

PhantomJS

Used by several libraries, including JavaScript unit testing runners

Uses WebKit JavaScript engine

GhostDriver - WebDriver implementation for PhantomJS

I have seen PhantomJS run almost 50% faster than running an actual browser

Change Browsers

```
grails -Dgeb.env=chrome test-app functional:
```

The environments are setup in the `GebConfig.groovy` file

```
// Default browser
driver = { new FirefoxDriver() }

environments { chrome {

    System.setProperty('webdriver.chrome.driver',
                        '/path/to/chromedriver')

    driver = { new ChromeDriver() }
}
}
```

Testing your JS with Spock

Why?

Tests integrate with your existing Spock / JUnit-powered test suite, reporting, and IDE.

If your JavaScript uses Java APIs, you have them available for use.

You get [all of the things you love about Spock](#): power assertions, data-driven testing, mocks, and most importantly: readable tests.

Testing JS with Spock

```
class TransformSpec extends Specification {
    ScriptEngine engine = new ScriptEngineManager().getEngineByName
('nashorn');

    def setup() {
        def source = this.class.getResource('/js/transforms.js').text
        engine.eval(source);
    }

    def "transform"() {
        when:
            Map result = engine.invokeFunction('transform', [name: [first:
'James', last: 'Bond']])

        then:
            result.firstName == 'James'
            result.lastName == 'Bond'
    }
}
```

The JS

```
function transform(person) {  
    return {firstName: person.name.first, lastName: person.  
name.last}  
}
```

Want to see more?

<http://www.objectpartners.com/2014/05/29/unit-test-your-server-side-javascript-with-spock/>