



IstQ 2024

Dare. Explore. Change

Effective Unit and Integration Testing

Petyo Dimitrov | Senior Software Architect

Agenda



What is a
good test?



Anti-patterns



How to improve
your tests?

What is a good test?

Provides Confidence



ISTC 2024

```
05-25T20:40:00.100Z - info] <...>
core > infosystem > collection > parseFilter - undefined should return an empty array
core > infosystem > collection > parseFilter - undefined should return an empty array
core > infosystem > collection > parseFilter - Keep array format if array
core > infosystem > collection > parseFilter - multiple filter types
core > infosystem > collection > Invalid limit should throw an error (114ms)
core > infosystem > collection > offset out of range should throw an error
core > infosystem > groupParameters > transform parameters into their complex groups, when they have subIndex (223ms)
core > infosystem > models > insert unique of a single event with new EventClass (191ms)
core > infosystem > models > insert unique of an existing single event (185ms)
core > infosystem > models > Get GUIDD by filename that does not exist will throw an error (178ms)
core > infosystem > models > Default limit should be 10 (378ms)
core > infosystem > collection > Get a list of items with limit (400ms)
core > infosystem > collection > Filter with filter option (378ms)
core > infosystem > collection > Filter with filter option (411ms)
core > infosystem > collection > Select only a few properties (400ms)
core > infosystem > models > Execute two info system methods in one transaction (242ms)
core > infosystem > collection > group result by a property (400ms)
core > infosystem > collection > group result by two property (485ms)
core > infosystem > collection > group Devices by their Location (504ms)
core > infosystem > collection > Setting limit as infinity or -1 should get all entries (598ms)
core > infosystem > collection > Filter with equals operator should be the same with omitted (693ms)
core > infosystem > collection > Execute two transactions simultaneously (739ms)
core > infosystem > models > Count adapter for a set of properties (770ms)
core > infosystem > models > Update adapter by QRCode (811ms)
core > infosystem > models > Count devices by setting ConnectionState (856ms)
core > infosystem > models > Connect device by setting ConnectionState (856ms)
core > infosystem > models > Disconnect device by setting ConnectionState (894ms)
core > infosystem > models > Insert new DeviceControl (965ms)
core > infosystem > models > Insert new DeviceFile (965ms)
core > infosystem > models > Insert new DeviceFile (1s)
core > infosystem > models > Update existing DeviceFile (1.1s)
core > infosystem > models > Insert new Event to EPC (1.1s)
core > infosystem > models > Insert new EPCCConfig (1.1s)
core > infosystem > models > Insert an Error in ErrorMapper (1.2s)
core > infosystem > models > Disappeared Event is no longer active (1.3s)
core > infosystem > models > update a single event (1.3s)
core > infosystem > models > Update Value for ISDU Diagnosis entry identified by DevID, Index, Subindex (1.3s)
core > infosystem > models > Insert into ISDU Diagnosis (1.3s)
core > infosystem > models > Insert new ProcessData (1.3s)
core > infosystem > models > insert of a process data mode (1.4s)
core > infosystem > models > Update vendor by ID (1.4s)
core > infosystem > models > Get adapter by id (1.7s)
core > infosystem > models > Get selected properties of an adapter by filter (1.7s)
core > infosystem > models > Get selected properties of an adapter by filter (1.7s)
core > infosystem > models > Get selected properties datatype (1.8s)
core > infosystem > models > Insert unique existing datatype (1.8s)
core > infosystem > models > Insert selected properties of a device by id (1.8s)
```

Quick Feedback



Robust



Maintainable



Anti-patterns and fixes

Test structure

```
def "test method name" () {  
  given: "prepare"  
  ...  
  when: "stimulus"  
  ...  
  then: "assertion"  
  ...  
}
```

Specific syntax

```
def user = Mock(User)
```

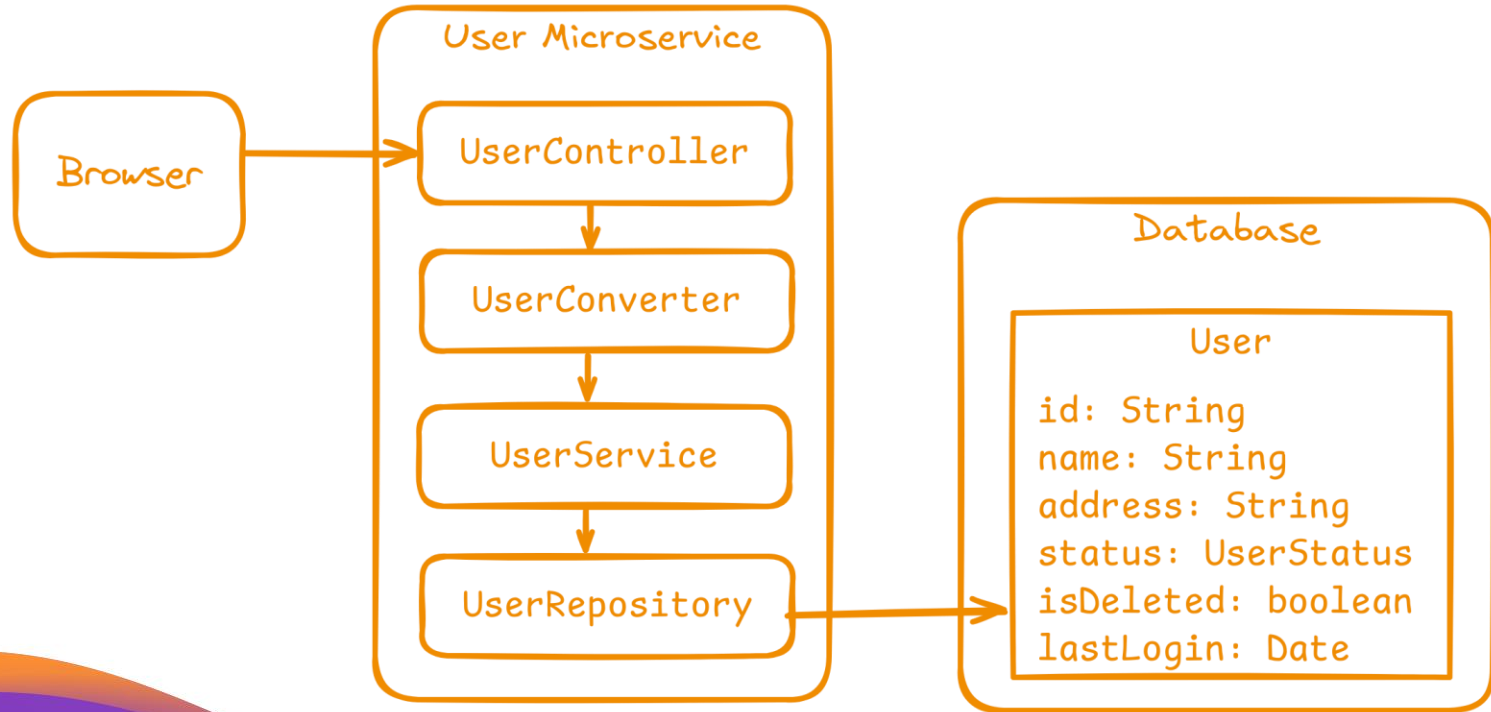
```
user.name >> "Johnny Bravo"
```

```
user.isDeleted == false
```

```
1 * converter.convert(user)
```

```
0 * converter._
```

Context



Anti-pattern #1: Mocking overuse

```
def user = Mock(User)

user.name >> "Johnny Bravo"

user.address >> "Sofia"

user.status >> UserStatus.Active

user.isDeleted >> false

user.lastLogin >> new Date()
```



Use real objects

```
def johnny = new User("Johnny Bravo",  
                    "Sofia", UserStatus.Active)  
  
johnny.isDeleted = false  
johnny.lastLogin = new Date()
```

Invest in test data

```
def johnny = TestData.JOHNNY
```

```
def littleSuzy = TestData.littleSuzy()
```

Object Mother pattern

```
UserMother.johnny()  
AddressMother.sofia()
```

```
UserMother.johnny()  
  .withAddress(  
    AddressMother.sofia().build()  
  ).build()
```


Anti-pattern #2: Repetition

```
def "user conversion works" () {  
  given:  
    def input = new User("Johnny Bravo",  
                        "Sofia", UserStatus.Active)  
  when:  
    def result = converter.convert(input)  
  
  then:  
    result.name == "Johnny Bravo"  
    result.address == "Sofia"  
    result.status == Active  
}
```



```
def "user conversion works" () {  
  given:  
    def input = new User("Johnny Bravo",  
                          "Sofia", UserStatus.Active)  
  when:  
    def result = underTest.convert(input)  
  
  then:  
    result.name == input.name  
    result.address == input.address  
    result.status == input.status  
}
```



```
def "user conversion works" () {  
  given:  
    def input = new User("Johnny Bravo",  
                        "Sofia", UserStatus.Active)  
  when:  
    def result = underTest.convert(input)  
  
  then:  
    Utils.matchProperties(result, input,  
                        "name", "address", "status", ...) }  
}
```



Anti-pattern #3: Testing the implementation

```
def "user service can create user" () {  
  given:  
    def user = UserMother.johnny().build()  
    def userForm = new UserForm(user.name, ...)  
  
  when:  
    def result = underTest.createUser(userForm)  
  
  then:  
    1 * otherService.isNameOK(_ as UserForm) >> true  
    1 * userRepository.persist(_ as User) >> user  
    0 * userRepository._  
    result == user  
}
```

```
def "user service can create user" () {  
  given:  
    def user = UserMother.johnny().build()  
    def userForm = new UserForm(user.name, ...)   
    // name matches rules  
    otherService.isNameOK(_) >> true  
    // user is persisted  
    userRepository.persist(_) >> user  
  
  when:  
    def result = underTest.createUser(userForm)  
  
  then:  
    0 * userRepository._  
    result == user  
}
```

Anti-pattern #4: Ineffective tests

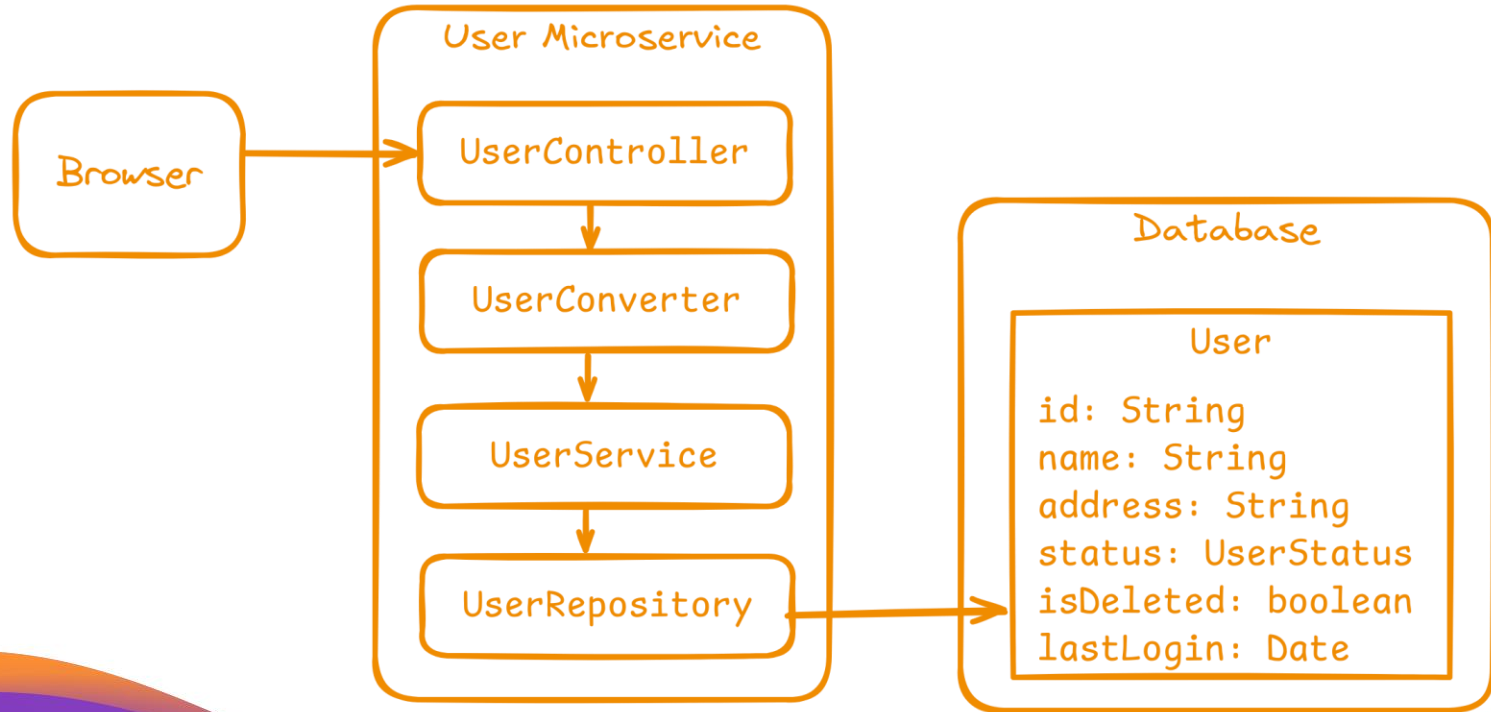

```
@PostMapping  
@ResponseBody  
public UserDto create(@RequestBody @Valid UserForm form) {  
    User user = userService.create(form);  
    return userConverter.convert(user);  
}
```

```
def "controller calls the service and converter" () {  
  given:  
    def user = UserMother.johnny().build()  
    def userForm = new UserForm(user.name, ...)   
    userService.create(_) >> user  
    userConverter.convert(_) >> ...  
  
  when:  
    def result = underTest.create(form)  
  
  then:  
    result.name == user.name  
    result.address == user.address  
}
```

Don't write unit tests!
Do write **integration tests!**



A refresher



```
def "user is created, updated and deactivated" () {  
    given:  
        def user = UserMother.johnny().build()  
  
    when:  
        def createdUserDto = /* trigger create REST API */  
  
    then:  
        createdUserDto.name == user.name  
        userRepository.findByName(user.name) != null  
}
```

when:

```
user.lastLogin = now().minusDays(30)
```

```
def updatedUserDto = /* trigger update REST API */
```

then:

```
updatedUserDto.name == user.name
```

```
updatedUserDto.lastLogin == user.lastLogin
```



when:

```
jobsUtil.triggerUserPeriodicJob()  
def userDto = /* trigger read REST API */
```

then:

```
userDto.name == user.name  
userDto.status == UserStatus.Inactivate
```

**Don't write
integration tests!
Do write unit tests!**




```
@Unroll
```

```
def "update changes status from #currentStatus to #newStatus"() {  
  given:  
    def user = new User(status: currentStatus)  
  when:  
    user.updateStatus(newStatus)  
  then:  
    user.status == newStatus  
  where:  
    currentStatus      | newStatus  
    UserStatus.Inactive | UserStatus.Active  
    UserStatus.Inactive | UserStatus.Deleted
```



Summary

Avoid mock overuse

Don't Repeat Yourself

Test contracts, not implementations

Don't test for the sake of testing

Distribute tests across the test pyramid

*To me, legacy code
is simply code...
without tests*

Michael Feathers
Working Effectively With Legacy Code

No code coverage, no problems!



Refactoring code
without unit tests

Looks like it's working!

O RLY?

Sir Crashalot

Thanks!

petyo.dimitrov@gmail.com
petyo.dimitrov@musala.com

www.linkedin.com/in/petyo-dimitrov