# **Testing APIs using Pact-js**

## Overview

The Pact family of frameworks provide support for Consumer Driven Contracts testing.

A Contract is a collection of agreements between a client (Consumer) and an API (Provider) that describes the interactions that can take place between them.

Consumer Driven Contracts is a pattern that drives the development of the Provider from its Consumers point of view.

Pact is a testing tool that guarantees those Contracts are satisfied.

## Prerequisites

For this example we will use Pact-js, whith Mocha (test framework) and Chai (assertion library).

You can use the Pact flavour that is most suited with you, for more informations please check their document page.

We will need:

- · Node.js environment
- Docker

To start using the Pact please follow the Get Started documentation.

## **Consumer Validations**

Consumer driven contract testing with Pact will allow you to validate the contracts between the consumer and the provider sooner in the pipeline. This approach is driven by the consumer, so the provider development will be defined by the consumer point of view.

In order to demonstrate this approach we have defined an API that consists in a Comic store with operations to get all existent comics available or to fetch one particular comic. We have added also an authentication mechanism based on the authorization header.

From the consumer point of view we are going to define the interactions that the consumer is expecting from the provider using Pact-js. We will then run those interactions against a mocked provider. To achieve these results we have defined the following classes that will represent the consumer API:

#### ./consumer.js

```
const express = require("express")
const request = require("superagent")
const server = express()
const getApiEndpoint = () => process.env.API_HOST || "http://localhost:
8081"
const authHeader = {
 Authorization: "Bearer 1234",
}
// Fetch all comics
// Comics Service
const availableComics = () => {
 return request
    .get(`${getApiEndpoint()}/comics/available`)
    .set(authHeader)
    .then(res => res.body)
}
// Find comics by their ID
const getComicsById = id => {
 return request
   .get(`${getApiEndpoint()}/comics/${id}`)
    .set(authHeader)
    .then(
     res => res.body,
     () => null
    )
}
module.exports = {
 server,
 availableComics,
 getComicsById,
}
```

#### consumerService.js

```
const { server } = require("./consumer.js")
server.listen(8080, () => {
   console.log("Comics Service listening on http://localhots:8080")
})
```

And using Pact we have defined the expected iterations:

#### test/consumer.spec.js

```
const path = require("path")
const chai = require("chai")
const chaiAsPromised = require("chai-as-promised")
const expect = chai.expect
const { Pact, Matchers } = require("@pact-foundation/pact")
const { log } = require("console")
const LOG_LEVEL = process.env.LOG_LEVEL || "WARN"
chai.use(chaiAsPromised)
describe("Pact", () => {
  const provider = new Pact({
    consumer: "e2e Consumer Example",
```

```
provider: "e2e Provider Example",
    log: path.resolve(process.cwd(), "logs", "mockserver-integration.log"),
    dir: path.resolve(process.cwd(), "pacts"),
    logLevel: LOG_LEVEL,
    spec: 2,
  })
  // Alias flexible matchers for simplicity
  const { eachLike, like, term, iso8601DateTimeWithMillis } = Matchers
  // comic to match
  const comic_to_match = {
   id: 2.
    title: "Batman: no return",
    pages: 22
  }
  const MIN_COMICS = 2
 const comicBodyExpectation = {
    id: like(1),
   title: like("X-MEN"),
   pages: like(50)
  }
  // Define comics list payload, reusing existing object matcher
  const comicListExpectation = eachLike(comicBodyExpectation, {
   min: MIN_COMICS,
  })
  // Setup a Mock Server before unit tests run.
  \ensuremath{{\prime}}\xspace // This server acts as a Test Double for the real Provider API.
  // We then call addInteraction() for each test to configure the Mock
Service
  // to act like the Provider
 // It also sets up expectations for what requests are to come, and will
fail
  \ensuremath{{\prime}}\xspace // if the calls are not seen.
 before(() =>
   provider.setup().then(opts => {
      // Get a dynamic port from the runtime
      process.env.API_HOST = `http://localhost:${opts.port}`
   })
  )
  // After each individual test (one or more interactions)
  // we validate that the correct request came through.
  // This ensures what we <code>_expect_</code> from the provider, is actually
  \ensuremath{{\prime}}\xspace // what we've asked for (and is what gets captured in the contract)
  afterEach(() => provider.verify())
  // Configure and import consumer API
  // Note that we update the API endpoint to point at the Mock Service
  const {
   availableComics,
   getComicsById,
  } = require("../consumer")
 // Verify service client works as expected.
  11
  // Note that we don't call the consumer API endpoints directly, but
  // use unit-style tests that test the collaborating function behaviour -
  // we want to test the function that is calling the external service.
  describe("when a call to list all comics from the Comic Service is
made", () => {
   describe("and the user is not authenticated", () => {
      before(() =>
        provider.addInteraction({
          state: "is not authenticated",
          uponReceiving: "a request for all comics",
          withRequest: {
```

```
method: "GET",
           path: "/comics/available",
          },
          willRespondWith: {
           status: 401,
          },
       })
      )
      it("returns a 401 unauthorized", () => {
       return expect(availableComics(comic_to_match)).to.eventually.be.
rejectedWith(
         "Unauthorized"
       )
     })
    })
    describe("and the user is authenticated", () => {
      describe("and there are comics in the database", () => {
       before(() =>
         provider.addInteraction({
            state: "Has some comics",
            uponReceiving: "a request for all comics",
            withRequest: {
             method: "GET",
             path: "/comics/available",
             headers: { Authorization: "Bearer 1234" },
            },
            willRespondWith: {
             status: 200,
             headers: {
                "Content-Type": "application/json; charset=utf-8",
             },
             body: comicListExpectation,
            },
         })
        )
        it("returns a list of comics", done => {
         const comicsReturned = availableComics()
          expect(comicsReturned)
            .notify(done)
        })
      })
   })
  })
  describe("when a call to the Comic Service is made to retreive a single
comic by ID", () => {
    describe("and there is an comic in the DB with ID 1", () => {
     before(() =>
       provider.addInteraction({
         state: "Has an comic with ID 1",
          uponReceiving: "a request for an comic with ID 1",
          withRequest: {
           method: "GET",
           path: term({ generate: "/comics/1", matcher: "/comics/[0-9]+"
}),
           headers: { Authorization: "Bearer 1234" },
          },
          willRespondWith: {
            status: 200,
           headers: {
            "Content-Type": "application/json; charset=utf-8",
            },
           body: comicBodyExpectation,
          },
       })
      )
      it("returns the animal", done => {
```

```
const comicsRetuned = getComicsById(11)
        expect(comicsRetuned)
          .to.eventually.have.deep.property("id", 1)
          .notify(done)
     })
   })
   describe("and there no comics in the database", () => \{
     before(() =>
       provider.addInteraction({
         state: "Has no comics",
         uponReceiving: "a request for an comic with ID 100",
         withRequest: {
           method: "GET",
           path: "/comics/100",
           headers: { Authorization: "Bearer 1234" },
         },
         willRespondWith: {
           status: 404,
          },
       })
      )
     it("returns a 404", done => {
       const comicReturned = getComicsById(100)
        expect(comicReturned)
          .to.eventually.be.a("null")
          .notify(done)
     })
   })
 })
 // Write pact files
 after(() => {
   return provider.finalize()
 })
})
```

In the above class we have defined a new Pact between a consumer, that we have named: "e2e Consumer Example" and a provider named: "e2e Provider Example" (notice that we have also defined other parameters such as: the log path and a log level).

```
...
describe("Pact", () => {
  const provider = new Pact({
    consumer: "e2e Consumer Example",
    provider: "e2e Provider Example",
    log: path.resolve(process.cwd(), "logs", "mockserver-integration.log"),
    dir: path.resolve(process.cwd(), "pacts"),
    logLevel: LOG_LEVEL,
    spec: 2,
  })
...
```

Before starting these validations we want to start a mock server (that will represent the provider) and point our client to it.

```
before(() =>
    provider.setup().then(opts => {
        // Get a dynamic port from the runtime
        process.env.API_HOST = `http://localhost:${opts.port}`
    })
...
```

Finally we need to define the various interactions this consumer expects from the provider. In our case we have defined 4 interactions:

- when a call to list all comics from the Comic Service is made
  - $^{\circ}\;$  and the user is not authenticated
  - and the user is authenticated
  - and there are comics in the database
- · when a call to the Comic Service is made to retrieve a single comic by ID
  - $^{\circ}~$  and there is a comic in the DB with ID 1  $^{\circ}$
  - $^{\circ}\;$  and there no comics in the database

In each of these we have defined a state that will define a desired state of the provider before executing the test (this state name will be used afterwards in the provider to setup the expected state), we also have defined the request details (such as method, path and possible headers) and the response expected.

```
. . .
describe("and there no comics in the database", () => {
      before(() =>
        provider.addInteraction({
          state: "Has no comics",
          uponReceiving: "a request for an comic with ID 100",
          withRequest: {
            method: "GET",
            path: "/comics/100",
            headers: { Authorization: "Bearer 1234" },
          },
          willRespondWith: {
            status: 404,
          },
       })
      )
. . .
```

Finally we will check if the expectation matches the answer obtained.

```
it("returns a 404", done => {
    const comicReturned = getComicsById(100)
    expect(comicReturned)
        .to.eventually.be.a("null")
        .notify(done)
    })
...
```

#### Once the code is implemented we can execute it with the following command:

npm run test:consumer

This will generate an immediate result in the console showing the status of the tests:



A Junit report and the pact file are generated from the execution:

#### junit\_consumer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites name="Mocha Tests" time="1.1280" tests="4" failures="0">
  <testsuite name="Root Suite" timestamp="2022-02-16T14:50:35" tests="0"
time="0.0000" failures="0">
  </testsuite>
  <testsuite name="Pact" timestamp="2022-02-16T14:50:35" tests="0" file="
/Users/cristianocunha/Documents/Projects/xray-pact/test/consumer.spec.js"
time="0.0000" failures="0">
  </testsuite>
  <testsuite name="when a call to list all comics from the Comic Service
is made" timestamp="2022-02-16T14:50:36" tests="0" file="/Users
/cristianocunha/Documents/Projects/xray-pact/test/consumer.spec.js" time="
0.0000" failures="0">
  </testsuite>
  <testsuite name="and the user is not authenticated" timestamp="2022-02-
16T14:50:36" tests="1" file="/Users/cristianocunha/Documents/Projects/xray-
pact/test/consumer.spec.js" time="0.0310" failures="0">
    <testcase name="Pact when a call to list all comics from the Comic
Service is made and the user is not authenticated returns a 401
unauthorized" time="0.0120" classname="returns a 401 unauthorized">
    </testcase>
  </testsuite>
  <testsuite name="and the user is authenticated" timestamp="2022-02-16T14:</pre>
50:36" tests="0" file="/Users/cristianocunha/Documents/Projects/xray-pact
/test/consumer.spec.js" time="0.0000" failures="0">
  </testsuite>
  <testsuite name="and there are comics in the database" timestamp="2022-
02-16T14:50:36" tests="1" file="/Users/cristianocunha/Documents/Projects
/xray-pact/test/consumer.spec.js" time="0.0170" failures="0">
    stestcase name="Pact when a call to list all comics from the Comic
Service is made and the user is authenticated and there are comics in the
database returns a list of comics" time="0.0050" classname="returns a list
of comics">
   </testcase>
  </testsuite>
  <testsuite name="when a call to the Comic Service is made to retrieve a
single comic by ID" timestamp="2022-02-16T14:50:36" tests="0" file="/Users
/cristianocunha/Documents/Projects/xray-pact/test/consumer.spec.js" time="
0.0000" failures="0">
  </testsuite>
  <testsuite name="and there is a comic in the DB with ID 1" timestamp="
2022-02-16T14:50:36" tests="1" file="/Users/cristianocunha/Documents
/Projects/xray-pact/test/consumer.spec.js" time="0.0150" failures="0">
    <testcase name="Pact when a call to the Comic Service is made to
retrieve a single comic by ID and there is a comic in the DB with ID 1
returns the animal" time="0.0050" classname="returns the animal">
    </testcase>
  </testsuite>
  <testsuite name="and there no comics in the database" timestamp="2022-02-
16T14:50:36" tests="1" file="/Users/cristianocunha/Documents/Projects/xray-
pact/test/consumer.spec.js" time="0.0210" failures="0">
    <testcase name="Pact when a call to the Comic Service is made to
retrieve a single comic by ID and there no comics in the database returns
a 404" time="0.0040" classname="returns a 404">
    </testcase>
  </testsuite>
</testsuites>
```

e2e\_consumer\_example-e2e\_provider\_example.json

```
{
  "consumer": {
    "name": "e2e Consumer Example"
  },
  "provider": {
```

```
"name": "e2e Provider Example"
},
"interactions": [
 {
   "description": "a request for all comics",
   "providerState": "is not authenticated",
    "request": {
     "method": "GET",
     "path": "/comics/available"
    },
    "response": {
     "status": 401,
      "headers": {
     }
   }
 },
  {
    "description": "a request for all comics",
    "providerState": "Has some comics",
    "request": {
     "method": "GET",
      "path": "/comics/available",
      "headers": {
       "Authorization": "Bearer 1234"
     }
    },
    "response": {
     "status": 200,
      "headers": {
       "Content-Type": "application/json; charset=utf-8"
     },
      "body": [
       {
         "id": 1,
         "title": "X-MEN",
         "pages": 50
       },
        {
         "id": 1,
         "title": "X-MEN",
          "pages": 50
       }
      ],
      "matchingRules": {
        "$.body": {
         "min": 2
        },
        "$.body[*].*": {
         "match": "type"
        },
        "$.body[*].id": {
         "match": "type"
        },
        "$.body[*].title": {
         "match": "type"
        },
        "$.body[*].pages": {
         "match": "type"
        }
     }
   }
 },
  {
    "description": "a request for an comic with ID 1",
    "providerState": "Has an comic with ID 1",
    "request": {
     "method": "GET",
      "path": "/comics/1",
      "headers": {
       "Authorization": "Bearer 1234"
      },
```

```
"matchingRules": {
          "$.path": {
            "match": "regex",
            "regex": "\\/comics\\/[0-9]+"
          }
        }
      },
      "response": {
        "status": 200,
        "headers": {
          "Content-Type": "application/json; charset=utf-8"
        },
        "body": {
          "id": 1,
          "title": "X-MEN",
          "pages": 50
        },
        "matchingRules": {
          "$.body.id": {
            "match": "type"
          },
          "$.body.title": {
            "match": "type"
          },
          "$.body.pages": {
            "match": "type"
          }
        }
     }
   },
    {
      "description": "a request for an comic with ID 100",
      "providerState": "Has no comics",
      "request": {
        "method": "GET",
        "path": "/comics/100",
        "headers": {
          "Authorization": "Bearer 1234"
        }
      },
      "response": {
        "status": 404,
        "headers": {
        }
     }
   }
 ],
 "metadata": {
    "pactSpecification": {
      "version": "2.0.0"
   }
 }
}
```

This concludes the consumer validations, next we need to share the pact file with the provider. There are two ways to do this, either we send the file to the provider (in a shared directory or email) or we can use the Pact Broker for it.

## Pact Broker

The Pact broker is an open source tool that enables you to share your pacts and verification results between projects. It is the recommended way forward for serious Pact development.

For our example we will use the Pact broker available in a Docker image. We have included a *docker-compose.yaml* file in the solution, so to start it you just have to use the following command:

docker-compose up

This option will require you to deploy, administer and host the broker yourself. If you would prefer a plugand-play option, you can use Pactflow.

Once the broker is up you can use the following command to push the pact to the broker:



We can see in the output if the command was successful or not and an URL of the Broker.



To double check you can access the url of your broker and check that a new Pact was uploaded:

					API B	owser
Pacts						
Search	Submit Reset					
Consumer 14	Provider 14		Latest pact published	Webhook status	Last verified	
e2e Consumer Example	e2e Provider Example	₽ #	less than a minute ago	Create		
		« 1 »				
		1 of 1 pacts				

Notice that the column "Last Verified" is still empty because the provider has not yet validated this Pact on his side.

## **Provider Validations**

On the provider side we have defined the provider API with the following two classes:

provider.js

```
const express = require("express")
const cors = require("cors")
const bodyParser = require("body-parser")
const Repository = require("./repository")
const server = express()
server.use(cors())
server.use(bodyParser.json())
server.use(
 bodyParser.urlencoded({
   extended: true,
 })
)
server.use((req, res, next) => {
 res.header("Content-Type", "application/json; charset=utf-8")
 next()
})
server.use((req, res, next) => {
 const token = req.headers["authorization"] || ""
 if (token !== "Bearer 1234") {
   res.sendStatus(401).send()
  } else {
   next()
  }
})
const comicRepository = new Repository()
// Load default data into a repository
const importData = () => {
 const data = require("./data/comicsData.json")
 data.reduce((a, v) => {
   v.id = a + 1
   comicRepository.insert(v)
   return a + 1
 }, 0)
}
// Get all comics
server.get("/comics", (req, res) => {
 res.json(comicRepository.fetchAll())
})
// Get all available comics
server.get("/comics/available", (req, res) => {
 res.json(comicRepository.fetchAll())
})
// Find an comic by ID
server.get("/comics/:id", (req, res) => {
  const response = comicRepository.getById(req.params.id)
 if (response) {
  res.end(JSON.stringify(response))
  } else {
   res.writeHead(404)
   res.end()
  }
})
module.exports = {
 server,
 importData,
 comicRepository,
}
```

#### providerService.js

```
const { server, importData } = require("./provider.js")
importData()
server.listen(8084, () => {
   console.log("Comics Profile Service listening on http://localhost:8084")
})
```

To perform the pact validations we have defined the following class:

#### provider.spec.js

```
const { Verifier } = require("@pact-foundation/pact")
const chai = require("chai")
const chaiAsPromised = require("chai-as-promised")
chai.use(chaiAsPromised)
const { server, importData, comicRepository } = require("../provider.js")
const path = require("path")
server.listen(8084, () => {
 importData()
 console.log("Comics Service listening on http://localhost:8084")
})
// Verify that the provider meets all consumer expectations
describe("Pact Verification", () => {
  it("validates the expectations of Comics Service", () => \{
    let token = "INVALID TOKEN"
    let opts = {
     provider: "e2e Provider Example",
      logLevel: "DEBUG",
      providerBaseUrl: "http://localhost:8084",
     requestFilter: (req, res, next) => {
        console.log(
         "Middleware invoked before provider API - injecting
Authorization token"
        req.headers["MY_SPECIAL_HEADER"] = "my special value"
       // e.g. ADD Bearer token
       req.headers["authorization"] = 'Bearer ' + token
       next()
      },
      stateHandlers: {
        "Has no comics": () => {
         comicRepository.clear()
         token = "1234"
         return Promise.resolve('Comics removed to the db')
        },
        "Has some comics": () => {
         token = "1234"
         importData()
         return Promise.resolve('Comics added to the db')
        },
        "Has an comic with ID 1": () => {
         token = "1234"
         importData()
         return Promise.resolve('Comic added to the db')
        },
        "is not authenticated": () => {
```

```
token = ""
         Promise.resolve('Invalid bearer token generated')
       },
     },
     // Fetch pacts from broker
     pactBrokerUrl: "http://localhost:8000",
      // Fetch from broker with given tags
     consumerVersionTags: ["master", "test", "prod"],
     // Enables "pending pacts" feature
     enablePending: true,
     pactBrokerUsername: "pact_workshop",
     pactBrokerPassword: "pact_workshop",
     publishVerificationResult: true,
     providerVersion: "1.0.0",
   }
   return new Verifier(opts).verifyProvider().then(output => {
     console.log("Pact Verification Complete!")
     console.log(output)
   })
 })
})
```

Notice that we have defined that we need to fetch the pact from the broker and that we have defined "*stat e handlers*" that are defining the provider state before the validations (remember that the names used here must match the *providerState* defined in the consumer validations above).

In order to execute the provider validations we run the following command:

npm run test:provider

The results appear in the console output as we can see, and are also recorded in a Junit file.



#### junit\_provider.xml

Now when you access the broker you can see that the pact now is validated by the consumer and provider for that combination of versions.

					AJ	Pl Browse
Pacts						
earch	Submit Reset					
Consumer 14	Provider 14		Latest pact published	Webhook status	Last verified	/
e2e Consumer Example	e2e Provider Example	₽ ■	12 minutes ago	Create	1 minute ago	
		« 1 »				
		1 of 1 pacts				

The broker will provide more info that only this view, if you want to know more please check the Pact docu mentation.

Pact also has available a tool that will use the information in the broker to help decide if we can proceed with the deployment or not called: *can-i-deploy*, more details of this functionality here.

### Integrating with Xray

As we saw in the above example, where we are producing Junit reports with the result of the tests, it is now a matter of importing those results to your Jira instance, this can be done by simply submitting automation results to Xray through the REST API, by using one of the available CI/CD plugins (e.g. for Jenkins) or using the Jira interface to do so.

In this particular case, as we have two Junit files we need to repeat the process for each result. The importation of results is usually done in each API pipeline/process.

As the importation of the Junit results is the same, either if it is from the consumer or the provider side, we will only exemplify the one from the consumer side.

### API

#### API

Once you have the report file available you can upload it to Xray through a request to the REST API endpoint, and for that the first step is to follow the instructions in v1 or v2 (depending on your usage) to include authentication parameters in the following requests.

#### JUnit XML results

We will use the API request with the addition of some parameters that will set the Project to where the results will be uploaded and the Test Plan that will hold the Execution results.

In the first version of the API the authentication uses a login and password (not the token that is used in Cloud).

```
curl -H "Content-Type: multipart/form-data" -u admin:admin -F
"file=@junit_consumer.xml" http://yourserver/rest/raven/1.0/import
/execution/junit?projectKey=XT&testPlanKey=XT-344
```

With this command we are creating a new Test Execution in the referred Test Plan with a generic summary and four tests with a summary based on the test name.

	utorial	-pact-js							
🖋 Edit	Q Con	nment Assig	n More 🛩	To Do	In Progress	Done Admir	n •		
Details									
Type:		🔽 Test P	lan			Status:	TO DO (V)	ew Workflow)	
Priority:		O Trivial				Resolution:	Unresolved	1	
Labels:		None							
Descrip	tion								
Click to	add desc	ription							
Tests									
Add T	ests 🛩	Create Test Ex	ecution 🛩						Test Plan Board
Overall E	Execution S	itatus							
4 PAS	SS sts: 4								
4 PAS Total Tes	SS sts:4 ∓Filter(s)								
4 PAS Total Tes	SS sts:4 ≕ Filter(s)						Show 10 v entri	es All Environments	✓ Columns ▼
4 pas Total Tee	SS sts: 4 〒 Filter(s) Key	Summary			Requireme	nts #Test Execu	Show 10 v entri	es All Environments Latest Status	✓ Columns ▼
4 pas Total Tee	SS sts: 4 T Filter(s) Key	Summary Pact when a ca	II to list all comi	ics from the	Requireme	nts #Test Execu	Show 10 v entri tions issue Assignee Xnand IT	es All Environments Latest Status	✓ Columns ✓
4 pA: Total Tes	SS sts: 4 T Filter(s) Key XT- 346	Summary Pact when a ca comic Service i authenticated r	II to list all com is made and the etums a 401 ur	ics from the user is not authorized	Requireme	nts @Test Execu 1	Show 10 v entri tions issue Assignee Xpand IT Admin	es All Environments Latest Status PASS	✓ Columns ▼
4 pAS	SS sts: 4 T Filter(s) Key XT- 346	Summary Pact when a ca Comic Service i authenticated r Pact when a ca	If to fist all comi s made and the sturns a 401 ur If to fist all comi	cs from the suser is not authorized ics from the	Requireme	nts øTest Execu 1	Show 10 v) entri tions Issue Assignee Xpand IT Admin	es All Environments Latest Status PASS	• Columns •
	SS sts: 4 = Filter(s) Key XT- 346 XT- 347	Summary Pact when a ca Comic Service authenticated a Pact when a ca Comic Service authenticated a database return	I to list all com is made and the eturns a 401 ur is fait all com is made and the is a list of com	ics from the suser is not authorized ics from the suser is mics in the cs	Requireme t	nts #Test Execu 1 1	Show 10 v entri tions issue Assignee Xpand IT Admin Xpand IT Admin	es All Environments Latest Status PASS	• Columns • 
4 PAS	SS sts: 4 = Filter(s) Key XT- 346 XT- 347	Summary Pact when a ca Comic Service authenticated r Pact when a ca Comic Service authenticated auth	I to list all com is made and the eturns a 401 ur It to list all com is made and the nd there are oc is a list of com I to the Comic	ics from the user is not authorized ics from the user is mics in the cs Service is	Requireme	nts #Test Execu 1 1	Show [10 ••) entri Issue Assignee Xpand IT Admin Xpand IT Admin	es All Environments Latest Status PASS	<ul> <li>Columns -</li> <li></li> </ul>

### Jira UI

### Jira UI

Kets Q Comment Assign More Y To Do In Progress Done Admin Y      Paralle     Type:      Test Status:      Labels     None      Paccoption      Yop:      Gravity      Construct Admin Y      Paccoption      Yop:      Gravity      Construct Admin Y      Paccoption      Paccoption	10 Do In Progress Done Admin v Status: COSC (View Workflow) Resolution: Unresolved
Vortalis     Vortalis	Status: ECC (View Workflow) Resolution: Unresolved
Type:     If Test     Status:     IDDE (View Workforw)       Priority:     O Trivial     Resolution:     Unresolved       Labels:     Type:     Generic     Unresolved       Type:     Generic     Definition:     view a single comic by ID and there no comics in database returns a 404       Pre-Conditions     Pre-Conditions     Pre-Conditions	Status: EDD (view Workflow) Repolution: Unresolved
Promity:     O Trivial     Resolution:     Unreadved       Labels:     None	Resolution: Unresolved
Libers: None Description Type: Generic Definition: returns a 404-Parct when a call to the Comic Service is made to retrieve a single comic by ID and there no comics in database returns a 404 Pre-Conditions Pre-Conditions	all to the Comic Service is made to retrieve a single comic by ID and there no comics in the
Description     Test Details     Type:     Generic     Definition:     returns: 404.Ploct when a call to the Comic Service is made to retrieve a single comic by ID and there no comics in     database returns: a 404	all to the Comic Service is made to retrieve a single comic by ID and there no comics in the
<ul> <li>▼ Test Details         Type: Generic         Definition: returns a 404.Pact when a call to the Comic Service is made to retrieve a single comic by ID and there no comics in database returns a 404     </li> <li>&gt; Pre-Conditions</li> </ul>	all to the Comic Service is made to retrieve a single comic by ID and there no comics in the
Type: Generic Definition: returns a 404 Pact when a call to the Comic Service is made to retrieve a single comic by ID and there no comics in database returns a 404 > Pre-Conditions	all to the Comic Service is made to retrieve a single comic by ID and there no comics in the
Definition: returns a 404-Petr when a call to the Comic Service is made to retrieve a single comic by ID and there no comics in detabase returns a 404  Pre-Conditions	all to the Comic Service is made to retrieve a single comic by ID and there no comics in the
> Pre-Conditions	
> Test Sets	
> Test Plans	
✓ Test Runs	
Execute In V	
New Test Execution	
Existing Test Execution New Test Execution	
dependent) otatu pro-	-

2

Fill in the necessary fields and press "Create"

Create new te	st execution to run XT-349		
Project*	Xray Tutorials		
Summary*	Ad-hoc execution for Pact when a call to the Comic Service is	made to r	ettic
Assignee	O Xpand IT Admin		~
	Choose a user to assign the Test Execution		
Priority	Slocker		~
	Start typing to get a list of possible matches or press down to select.		
Fix Version/s			~
	Start typing to get a list of possible matches or press down to select.		
Sprint			~
	Start typing to get a list of possible matches or press down to select.		
est Environments			~
	Start typing to get a list of possible matches or press down to select. Each environment where the Test is to be executed		
Revision			
	The system revision for the test execution		
	<ul> <li>Execute Immediately</li> </ul>		
		Create	Cance

3

#### Open the Test Execution and import the JUnit report



4

#### Choose the results file and press "Import"

Import Execution Results		
Choose file No file chosen The file with the execution results for the Test Execution.		
	Import	Cancel

5

The Test Execution is now updated with the test results imported

Edit	Q Commer	nt Assi	ign More 🛩	To Do	In Progres	s Done	Admin	~			
Details											
Type:		🔼 Test I	Execution			St	atus:		TO DO (Vie	w Workflow)	
Priority:		O Trivia	4			Re	solution:		Unresolved		
Labels: Teet Dier		None XT 244									
Test Env	vironments:	None									
Descrip	tion										
Tests											
Add Ie	osts v										
Overall E	versition Status										
<b>4</b> PAS	SS										
4 PAS	SS its: 4										
4 PAS Total Tes	SS its: 4 〒 Filter(s)										
4 PAS Total Tes	SS rts: 4 〒 Filter(s) Apply Rani	c								Show 100 ♥ entries	Columns <del>-</del>
4 <sub>PAS</sub> Total Tes	SS #ts:4 F Filter(s) Apply Rank ∳ Rank	Key	¢ Summary	¢ 1	fest Type	₽Raq	#Def	Assignee	Dataset	Show[100▼] entries ¢ Status	Columns +
4 pas Total Tes	SS F Filter(s) Apply Rank \$	к * Кеу XT-346	Summary Pact when a call list all comics fro the Comic Servic made and the us	¢1 to tam te is teris Ge	fest Type	#Req 0	#Def	Assignee Xpand IT Admin	Dataset	Show 100 V) entries \$ Status	Columns -
4 pas Total Tes	SS tts: 4 F Filter(s) ↓ Rank ↓ Rank	к * Кеу XT-346	Summary Pact when a call list all comics fire the Comic Strett made and the us not authenticate returns a 40 unauthorized	∲1 m ceis eeris Ge d	fest Type	#Req 0	#Def	Assignee Xpand IT Admin	Dataset	Show 100 v) entries \$ Status PASS	Columns -
4 PAS	SS Its: 4 Filter(s) ¢ Rank 1	к * Кау XT-346	Summary Pact when a call list all comics for the Comic Service made and the us not authenticat returns a 401 unauthorized Pact when a call list all comics for the Comic Service	¢ 1 to ternis ternis Ge d to to to to to	fest Type	#Req D	#Def	Assignee Xpand IT Admin	Dataset	Show 100 V entries 0 \$ Status PASS	Columns •

Tests implemented will have a corresponding Test entity in Xray. Once results are uploaded, Test issues corresponding to the tests are auto-provisioned, unless they already exist.

In Xray, results are stored in a Test Execution, usually a new one. The Test Execution contains a Test Run per each test that was executed.



Detailed results, including logs and exceptions reported during execution of the test, can be seen on the execution screen details of each Test Run, accessible through the *Execution details*:

×	Test Runs											/lew on Board		
	Decute in +													
	· FLTERS													
	Project	Version ( depende	project st)	Statu		51m	4	End						
	All Projects	• Selecta	project to ena	ble .		¥ 00	MM-YYYY HOLMA	00-MM-YYYY H	NV. Clear					
								Show	10 v entries	Columns •				
	Rey Fix 0 0 Varalae/a	Revision )	Executed By	Ranted (	Finished (	Defects ()	Test † Environments	Summary 0	TestEstimation 0	Original () Estimate	E Original Estimate	Stetus 0		
	хт- 345		Xpand IT Admin	12 minutes ago	12 minutes ago		None	Execution results - junit_consumerami				MIS	<b>P</b>	
	Showing 1 to 1 of 1 entrie	9							First Previous	Next Las			Execute I	ende cator.
*	Attachments			C	Drop files	to attach, o	r browse.						EXECUTE MUNE TODO EXECUTINO	
*	Activity Al <u>Comments</u> W There are no comment	fork Log H s yet on this i	istory Activision.	vity									FAL ABORTED BLOCKED PENDING	

As we can see here:

And any and any	t when a call to list all comics fro there are comics in the databas	extrate (	Export Test as Test.      A Return to Test Execute Execute with Exploratory App     Free/ous     Next.
A Sammer      Constant constant     Constan	eoutien Status PASS* Statued On: 10/Feb/32 5392 PM (**	Filiphol On: 10/Feli22 5/02 PM	Assignee Xgand IT Admin Versions Executed by Xgand IT Admin Revisions Texts - environments
Con a divisional la facta de la división de la divi	∧ Comment	^ Execution Defects (8) ⊕	^ Execution Evidence (0) ⊕
A reperture	Cloir to add comment	No defects yet	No evidence yet
A calance Arriver and a second	^ Test Details ercera		
Software         manual bit of careful Aperi whom and is bit of careful have in each or the care is a sub-reference and there are nones. In the database means, bit of careful Aperi Ape	Gustern Fields There are no list than Custern Pields defined.     Treat Description Text Totar: Denortic		
A heads Dated Dapat Darion States	Definition: returns a list of comics Pa	tt when a call to list all comics from the Comic Service is made and the user is authenticated and there	are comics in the database returns a list of comics
Contest Dutjust Duration States			
	<ul> <li>Results</li> </ul>		

### Tips

- after results are imported, in Jira Tests can be linked to existing requirements/user stories, so you can track the impacts on their coverage. results from multiple builds can be linked to an existing Test Plan, to facilitate the analysis of
- ٠ test result trends across builds.
- results can be associated with a Test Environment, in case you want to analyze coverage and test results by that environment later on. A Test Environment can be a testing stage (e.g. dev, staging, prepod, prod) or a identifier of the device/application used to interact with the system (e. g. browser, mobile OS).

### References

- https://github.com/pact-foundation/pact-js
- https://docs.pact.io/
- https://docs.pact.io/implementation\_guides/javascript
- https://pactflow.io/
- https://github.com/pact-foundation/pact\_broker