EN
English

# Software Architecture

## Acceptance tests

2020-21

Jose Emilio Labra Gayo
Pablo González
Irene Cid
Paulino Álvarez

## Acceptance tests and BDD

- Tests that can be run in front of the client
  - If the tests pass, the product is accepted
- Behaviour-Driven Development (BDD)
  - Variant of TDD
    - Acceptance test driven development
  - Behaviour = User Stories
  - Also known as: *Specification by example*
  - Goal: Executable specifications
- Some tools:
  - cucumber, jBehave, concordion

## BDD - User Stories

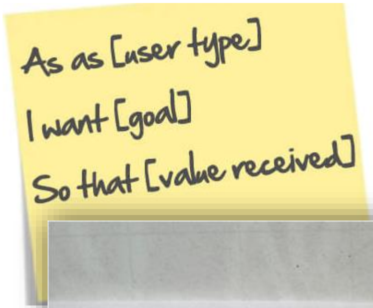
- Simple
- Readable by domain experts (business people)
- Approved by domain experts
- Other advisable characteristics:
  - Independent (with no strong relationships)
  - Negotiable (with no specific details)
  - Valuable for the customer
  - Estimable (to add them to Sprints)
  - Small (or consider division)
  - Testable (automatic tests)

## User story structure

Feature: *Title (one line describing the story)*

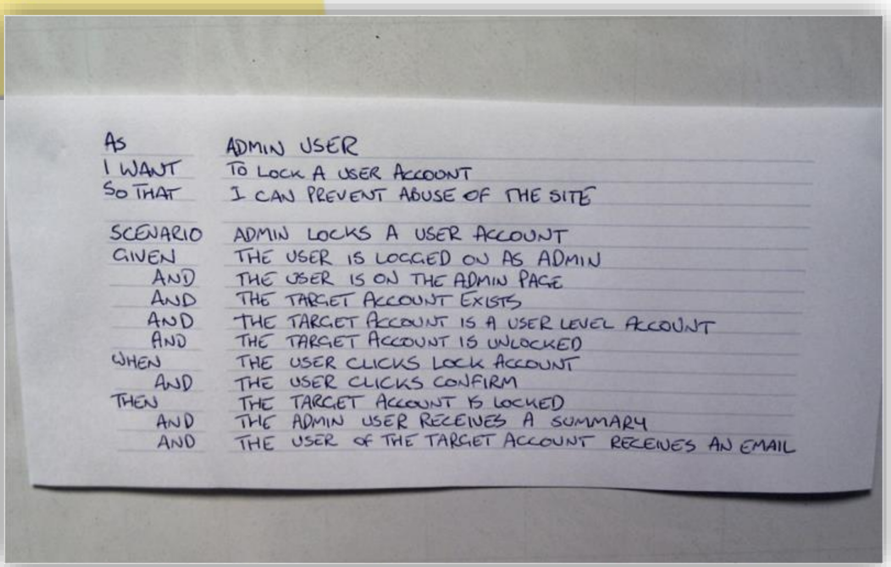
The following structure is recommended:

*As a* [role]
*I want* [feature]
*So that* [benefit]


As as [user type]
I want [goal]
So that [value received]

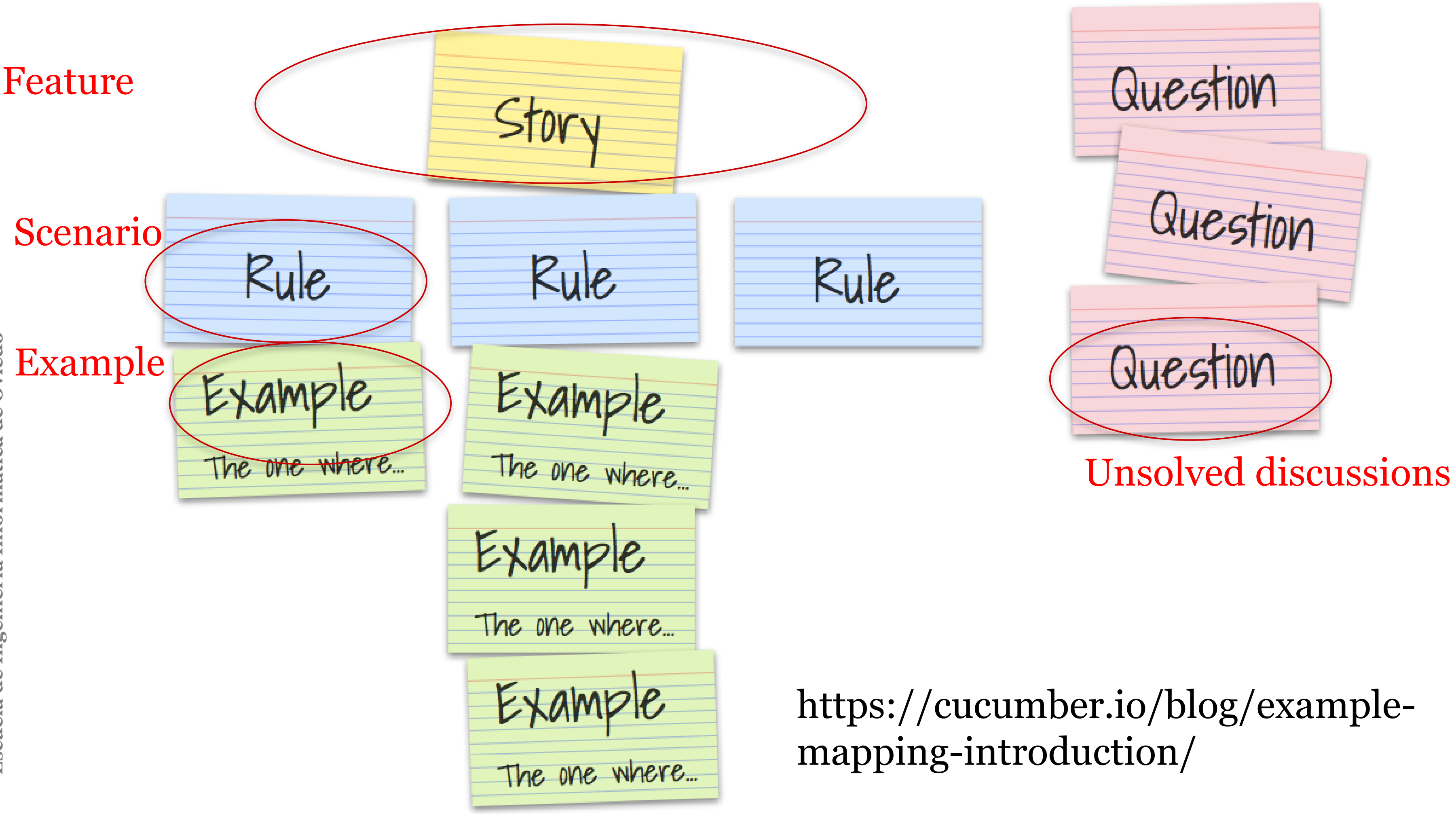
Scenarios

*Given* [Context]
*And* [Some more context]
*when* [Event]
*then* [Outcome]
*And* [Another outcome]



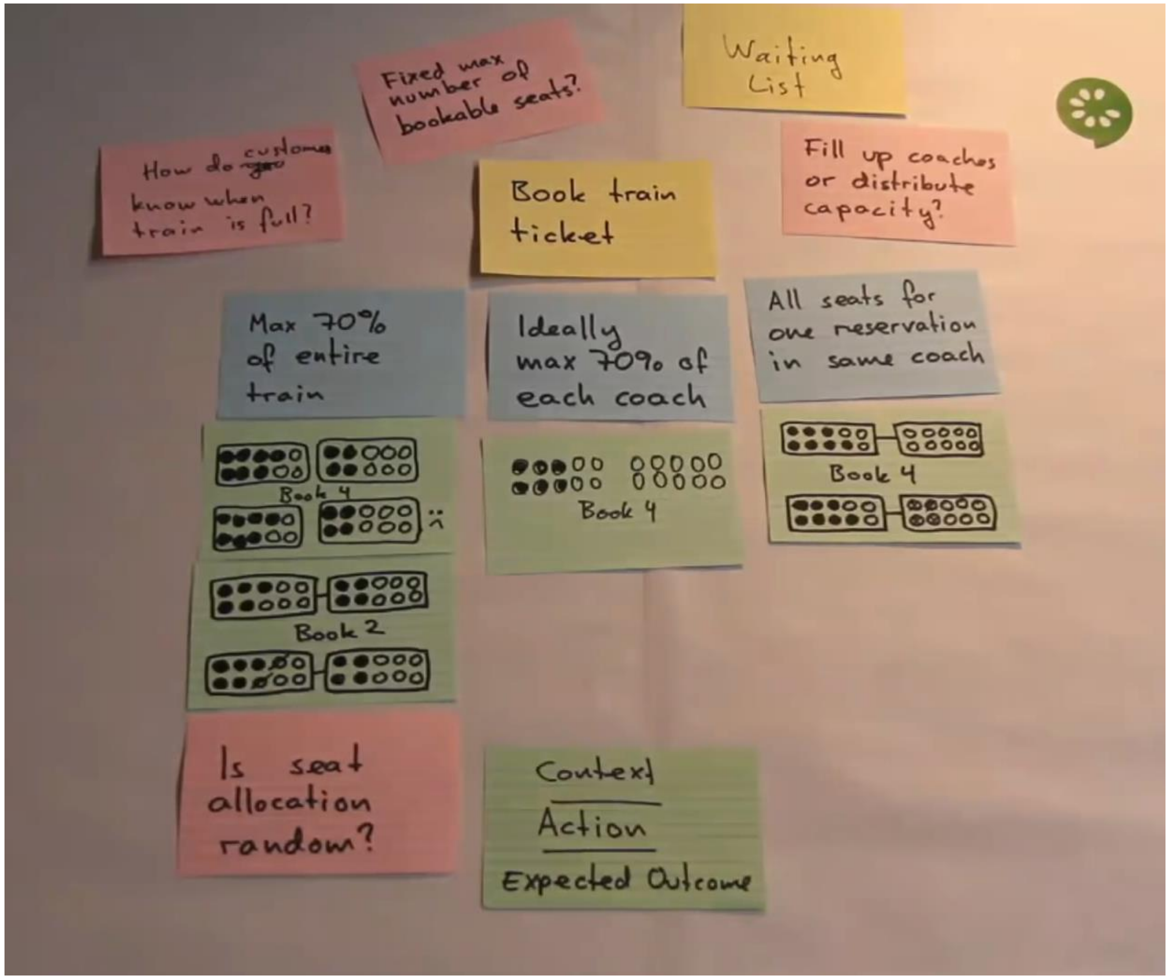
| | |
|---|---|
| AS | ADMIN USER |
| I WANT | TO LOCK A USER ACCOUNT |
| SO THAT | I CAN PREVENT ABUSE OF THE SITE |
| SCENARIO | ADMIN LOCKS A USER ACCOUNT |
| GIVEN | THE USER IS LOGGED ON AS ADMIN |
| AND | THE USER IS ON THE ADMIN PAGE |
| AND | THE TARGET ACCOUNT EXISTS |
| AND | THE TARGET ACCOUNT IS A USER LEVEL ACCOUNT |
| AND | THE TARGET ACCOUNT IS UNLOCKED |
| WHEN | THE USER CLICKS LOCK ACCOUNT |
| AND | THE USER CLICKS CONFIRM |
| THEN | THE TARGET ACCOUNT IS LOCKED |
| AND | THE ADMIN USER RECEIVES A SUMMARY |
| AND | THE USER OF THE TARGET ACCOUNT RECEIVES AN EMAIL |

## BDD – Example Mapping



https://cucumber.io/blog/example-mapping-introduction/

# BDD – Example Mapping



Fixed max number of bookable seats?
Waiting List
How do customers know when train is full?
Book train ticket
Fill up coaches or distribute capacity?
Max 70% of entire train
Ideally max 70% of each coach
All seats for one reservation in same coach
Book 4
Book 4
Book 4
Book 2
Is seat allocation random?
Context
Action
Expected Outcome

[Introducing example mapping [video]]

# BDD using Cucumber

- Cucumber = developed in Ruby (2008)
  - RSpec (Ruby), jbehave (Java)
  - Based on Gherkin
    - internal language to define user stories
  - Web: `http://cukes.info`
  - Support for multiple languages
- Java: cucumber-jvm
  - `https://github.com/cucumber/cucumber-jvm`

## BDD using cucumber

- Features define some functionality
  - Gherkin language
    https://cucumber.io/docs/gherkin/
    Can be used in several languages
- User stories are linked to step definitions
  - Step definitions can be run to validate user stories

## BDD using cucumber

Feature: Describes a system feature

- A feature can have several scenarios
- Scenario:
  - How must the system behave in some context
    - *Given*: Prepares scenario
    - *When*: Interact with the system
    - *Then*: Checks the state
    - *Examples*: Specific data

## BDD

- Step by step guide to a user story
  - Install Cucumber
  - Write a first scenario in Gherkin
  - Write steps definitions in a chosen programming language
  - Run cucumber

## BDD with cucumber

- Depends on programming language/environment
  - Java/Javascript/Python/…
  - Installation: https://cucumber.io/
- React: https://github.com/Arquisoft/radarin_0
  - jest-cucumber: Module to define user stories in Gherkin
    - And convert them to executable tests by Jest

```
$ npm install --save-dev puppeteer jest-cucumber
```

  - jest-puppeteer. Module to run the tests in a browser
    - It could be configured to use Selenium.

```
$ npm install --save-dev puppeteer jest-puppeteer
```

  - expect-puppeteer: Module with high level selectors for e2etests

```
$ npm install --save-dev expect-puppeteer
```

## BDD

- User Story example using Node.js

```
Feature: Registering a new user

Scenario: The user is not registered in the site
  Given An unregistered user
  When I fill the data in the form and press submit
  Then A welcome message should be shown in the screen

Scenario: The user is already registered in the site
  Given An already registered user
  When I fill the data in the form and press submit
  Then An error message should be shown in the screen
```

Feature

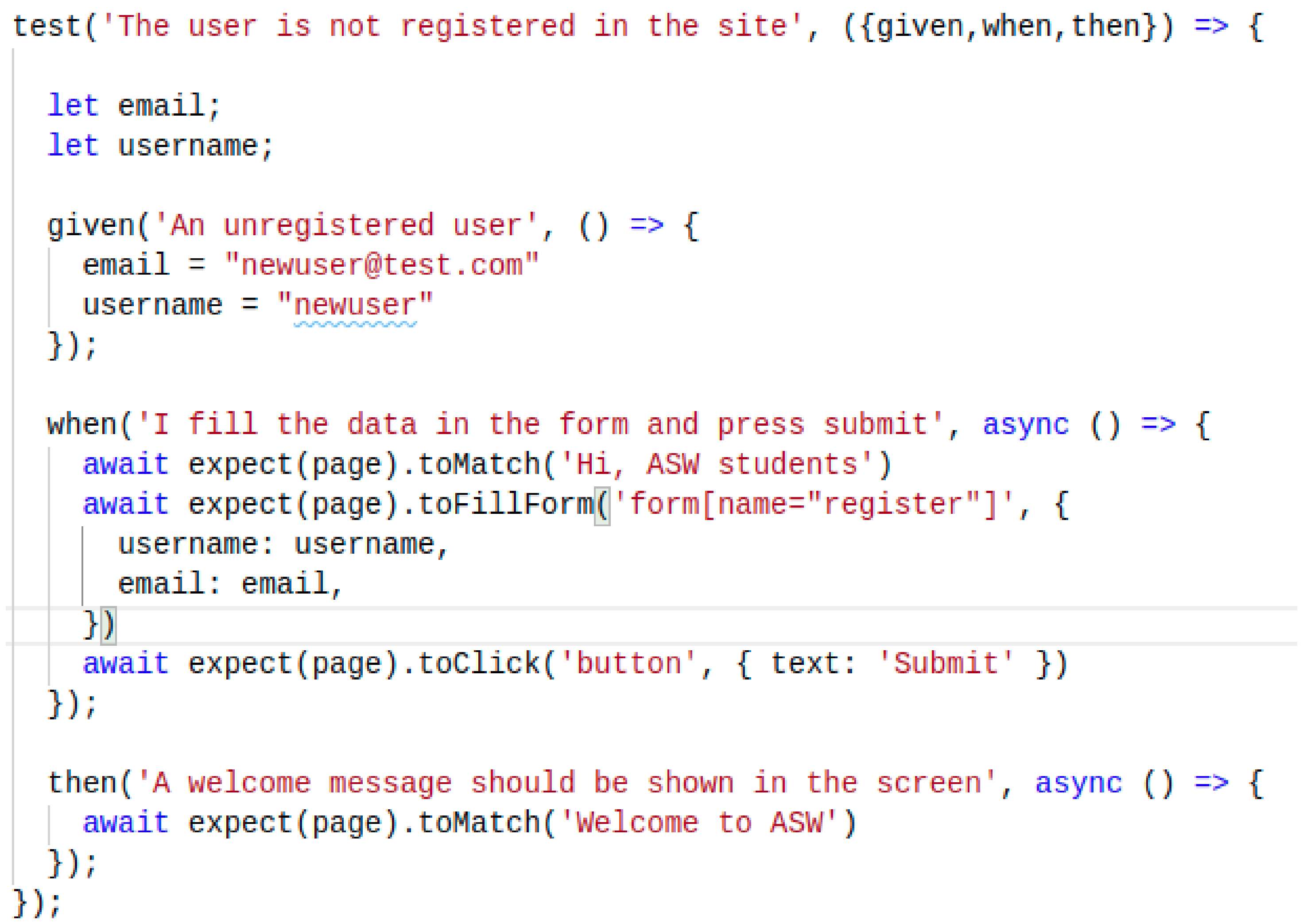
Scenario

Scenario

e2e/features/register-form.feature

## BDD

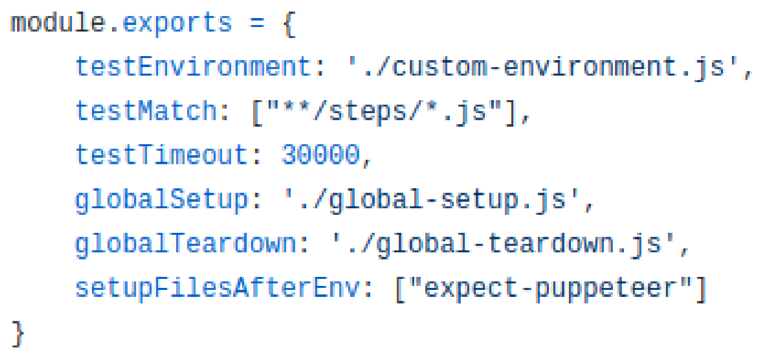
e2e/features/step-definition/register-form-steps.js



```
test('The user is not registered in the site', ({given,when,then}) => {

  let email;
  let username;

  given('An unregistered user', () => {
    email = "newuser@test.com"
    username = "newuser"
  });

  when('I fill the data in the form and press submit', async () => {
    await expect(page).toMatch('Hi, ASW students')
    await expect(page).toFillForm('form[name="register"]', {
      username: username,
      email: email,
    })
    await expect(page).toClick('button', { text: 'Submit' })
  });

  then('A welcome message should be shown in the screen', async () => {
    await expect(page).toMatch('Welcome to ASW')
  });
});
```

## BDD [Configuration]

- jest-config.js
  - This file links everything together
  - Tell jest where are the step test files
  - Configuration to start and teardown the tests



```
module.exports = {
    testEnvironment: './custom-environment.js',
    testMatch: ["**/steps/*.js"],
    testTimeout: 30000,
    globalSetup: './global-setup.js',
    globalTeardown: './global-teardown.js',
    setupFilesAfterEnv: ["expect-puppeteer"]
}
```

## BDD [Configuration]

- jest-puppeteer.config.js
  - Configures how to launch the browser to perform the tests
  - We use **puppeteer** for this task
  - Can be also configured with other browsers.
  - We use **headless=true** to run the tests in the CI system but we can change it to false to run them locally.
  - The **slowMo** parameter is useful to slowdown the tests and see what is happening

```
var NodeEnvironemnt = require('jest-environment-node')
var puppeteer = require('puppeteer')
class CustomEnvironment extends NodeEnvironemnt {
    constructor(config, context){
        super(config, context)
    }
    async setup(){
        await super.setup()
        this.global.browser = await puppeteer.launch({
            headless: true,
            //slowMo: 20
        })
        this.global.page = await this.global.browser.newPage()
    }
    async teardown(){
        await this.global.browser.close()
        await super.teardown()
    }
}
module.exports = CustomEnvironment
```

# BDD [Configuration]

- global-setup.js
  - Configures how to lunch the system
  - For testing this app we need, the databse, the restapi and the webapp
  - The database and the restapi are launched using two extra scripts.
  - BROWSER=none indicates that we do not want to launch the browser with npm start (we already configured how to launch the browser.

```
const { setup: setupDevServer } = require("jest-dev-server")
module.exports = async () => {
    await setupDevServer([
    {
        command: 'node start-db.js',
        launchTimeout: 100000,
        debug:true,
        port: 27017,
    },
    {
        command: 'node start-restapi.js',
        launchTimeout: 60000,
        debug:true,
        port: 5000,
    },
    {
        command: 'BROWSER=none npm start',
        launchTimeout: 60000,
        debug: true,
        port: 3000
    }])
}
```

## BDD [Configuration]

- start-db.js
  - Creates an in-memory instance of mongodb
  - Reuses the *startdb()* function. Used also for launching an in-memory database for the restapi unitary tests

```
const server = require('../../restapi/tests/server-for-tests')
server.startdb()
```

- start-restapi.js
  - Creates the restapi server with express and connects to the mongo in-memory database
  - Reuses the function *startserver()* from the restapi (also used in the restapi unitary tests)

```
const server = require('../../restapi/tests/server-for-tests')
server.startserver()
```

## BDD

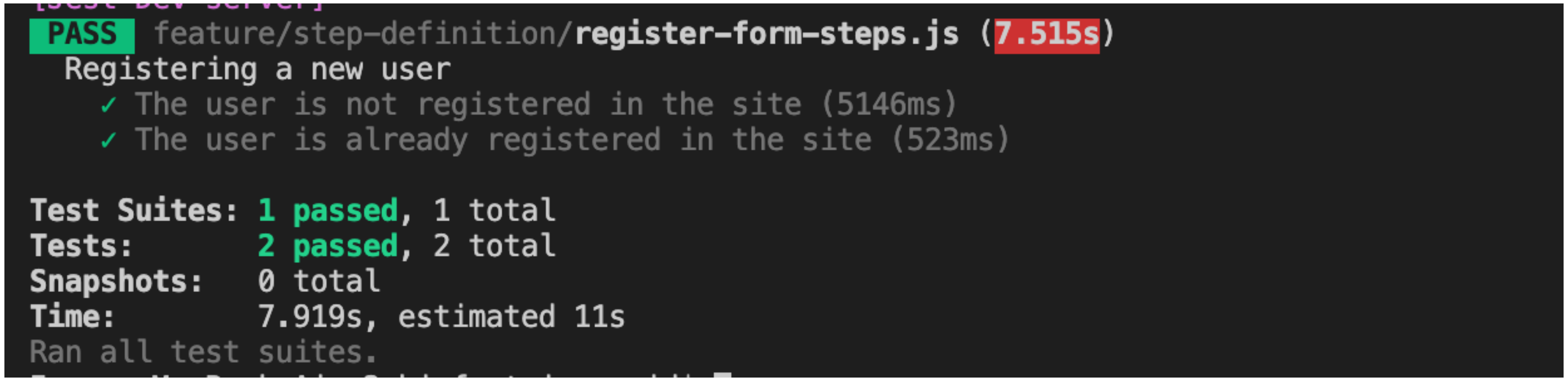
- Configuration package.json (scripts section):

```
"test:e2e": "cd e2e && jest",
```

- Running the tests:
  - npm run test:e2e

## BDD

- Result



```
 PASS  feature/step-definition/register-form-steps.js (7.515s)
  Registering a new user
    ✓ The user is not registered in the site (5146ms)
    ✓ The user is already registered in the site (523ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        7.919s, estimated 11s
Ran all test suites.
```

## Other example cucumber + selenium + java (spring boot) from previous years:

https://github.com/arquisoft/votingSystem0

## Browser-based tests

- Browser automation
  - https://cucumber.io/docs/reference/browser-automation
- Several systems
  - Selenium WebDriver - http://docs.seleniumhq.org/
  - Capybara - http://teamcapybara.github.io/capybara/
  - Watir - https://watir.com/
  - Serenity - http://serenity-bdd.info

## Selenium

- Selenium IDE: Allows to record actions
  - Firefox and Chrome plugins
- Generates code to execute those actions
- Travis configuration
  - https://lkrnac.net/blog/2016/01/run-selenium-tests-on-travisci/

# Bibliography and links

- User Story Mapping by Jeff Patton
  - **User Story Mapping: Discover the Whole Story, Build the Right Product, 1st Edition**
    https://www.amazon.com/User-Story-Mapping-Discover-Product/dp/1491904909
- User stories
  - Scrum. Historias de Usuario (Fernando Llopis, Universidad de Alicante)
    https://fernandollopis.dlsi.ua.es/?p=39
  - User stories with Gherkin and Cucumber (Michael Williams)
    https://medium.com/@mvwi/story-writing-with-gherkin-and-cucumber-1878124c284c
  - Cucumber 10 minutes tutorial (JS)
    https://docs.cucumber.io/guides/10-minute-tutorial/
- Browser based tests
  - Automated UI Testing with Selenium and JavaScript
    https://itnext.io/automated-ui-testing-with-selenium-and-javascript-90bbe7ca13a3