Testing 1, 2, 3, ...

.......

Wouter Lagerweij
@wouterla | wouter@lagerweij.com
We are infinitely creative in finding ways to break software
https://gitlab.com/wouterla/lowfunchightest/
**BDD/Functional tests**

- Create environment from scratch
- Automate deployment
- Protocol / component tests
- Integration / end-to-end tests

**Well-defined deployment artifacts**

- Smoke-tests / monitoring
- Automate update/data-migrations
- Controlled production deploys
- In-production test / monitoring
### BDD / ATDD

<table>
<thead>
<tr>
<th>Goal</th>
<th>Show the software does what was requested by the customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Commit Stage</td>
</tr>
</tbody>
</table>
**Scenario:** Update recipe

Given the following recipes in the system

<table>
<thead>
<tr>
<th>Recipe Name</th>
<th>Net Carbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fat Bombs</td>
<td>5</td>
</tr>
</tbody>
</table>

When we change the Net Carbs for "Fat Bombs" to "7"

Then the recipe "Fat Bombs" has "7" of Net Carbs
Then('the recipe {string} has {string} of Net Carbs', function (recipeName, net_carbs) {
  this.getRecipe(recipeName).then(function(recipe) {
    expect(recipe.net_carbs).to.equal(net_carbs);
  });
});
## Unit Test

<table>
<thead>
<tr>
<th>Goal</th>
<th>Prove to the developer the code does what he thought it would</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Commit Stage</td>
</tr>
</tbody>
</table>
it("should calculate the sum of all ingredient carbs", function() {
    let recipe = new Recipe();
    recipe.addIngredient(new Ingredient( name: 'test1', net_carbs: 4));
    recipe.addIngredient(new Ingredient( name: 'test2', net_carbs: 4));
    expect(recipe.calculateTotalCarbs().toEqual(8));
});

calculateTotalCarbs() {
    let totalCarbs = this.getIngredients().reduce(function(total, ingredient) {
        return total + ingredient.net_carbs;
    }, 0);
    return totalCarbs;
}
### Data Storage

<table>
<thead>
<tr>
<th>Goal</th>
<th>Test the code that stores data in a storage engine</th>
</tr>
</thead>
</table>
| Pipeline | Commit Stage (mocked)  
Test (to deployed/configured storage engine) |
beforeEach(function() {
    recipe = new Recipe();
    recipe.name = "Test";
    recipe.net_carbs = 10;

    if (global.gConfig.recipe_service === "memory") {
        recipeStorage = new RecipeRepositoryMemory();
        // console.log("using in-memory storage");
    } else {
        recipeStorage = new RecipeRepositoryFirestore();
        // console.log("using firebase");
    }
})

it("should be able to store a recipe", function() {
    recipeStorage.add(recipe);

    recipeStorage.findByName("Test").then(function(recipeFound) {
        expect(recipeFound.net_carbs).toEqual(10);
    });
});
## Integration

<table>
<thead>
<tr>
<th>Goal</th>
<th>Can the component reach, and use, another component</th>
</tr>
</thead>
</table>
| Pipeline | Test (between deployed component)  
Any environment using status endpoints |
```json
{
    "storage_engine": "firestore",
    "data_mode": "persist",
    "db_reached": true,
    "ingredient_service_endpoint": "http://production.lowfunchigtest.lagerweij.com",
    "ingredient_service_status": 200
}
```
## Accessing Dependencies - Gateway

<table>
<thead>
<tr>
<th>Goal</th>
<th>Can the consumer access another service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Commit (mocked)</td>
</tr>
</tbody>
</table>
it("should create a new ingredient with a name", async function() {
  let ingredients = new IngredientGateway();
  mock.put(apiEndpoint + '/api/ingredients', function(req) {
    return {
      status: 201,
      body: {
        id: '1',
        name: req.body.name,
        net_carbs: req.body.net_carbs
      }
    }
  });
  let ingredient = new Ingredient(name: 'test', net_carbs: 6);
  let result = await ingredients.add(ingredient);
  expect(result.name).toEqual(ingredient.name);
});
## Component Test

<table>
<thead>
<tr>
<th>Goal</th>
<th>Test the whole service as a black box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Commit (mocked, internal)</td>
</tr>
<tr>
<td></td>
<td>Test (external)</td>
</tr>
</tbody>
</table>
describe('GET /api/recipes', function() {
  it('responds with a list of recipes', function(done) {
    request(app)
      .get('/api/recipes')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect('a: 200', done);
  });
});
## Accessing Dependencies - Contract

<table>
<thead>
<tr>
<th>Goal</th>
<th>Test the contract in use between consumer and provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>Commit (consumer)  &lt;br&gt; Contract (provider, with specific versions deployed)</td>
</tr>
</tbody>
</table>
```javascript
const item = global.providers.addInteraction({
  state: 'jv1bzPVHTE',
  params: ingredientOne,
  uponReceiving: 'a request for a specific ingredient',
  withRequest: {
    method: 'GET',
    path: '/api/ingredients/' + ingredientOne.id,
  },
  willRespondWith: {
    status: 200,
    body: {
      id: 'jv1bzPVHTE',
      name: like('test'),
      net_carbs: like(3)
    }
  }
})
```
it('can get an ingredient by id', async function() {
    let id = ingredientOne.id;
    let response = await request.get(apiEndpoint + '/api/ingredients/' + id);

    let obj = JSON.parse(response.text);

    expect(obj).to.have.property("name", 'test');
    expect(obj).to.have.property('net_carbs', 3);
});
"consumer": {
  "name": "recipe_service"
},
"provider": {
  "name": "ingredient_service"
},
"interactions": [
  {
    "description": "a request for a specific ingredient",
    "providerState": "jv1bzPVHTE",
    "request": {
      "method": "GET",
      "path": "/api/ingredients/jv1bzPVHTE"
    },
    "response": {
      "status": 200,
      "headers": {
      },
      "body": {
        "id": "jv1bzPVHTE",
        "name": "test",
        "net_carbs": 3
      },
      "matchingRules": {
        "$.body.name": {
          "match": "type"
        },
        "$.body.net_carbs": {
          "match": "type"
        }
      }
    }
  },
  
}
A pact between recipe_service and ingredient_service

Requests from recipe_service to ingredient_service

- A new ingredient
- A request for a specific ingredient given jv1bzPVHTE

Interactions

Upon receiving a new ingredient from recipe_service, with

```json
{
  "method": "PUT",
  "path": "/api/ingredients",
  "headers": {
    "Content-Type": "application/json"
  },
  "body": {
    "name": "test2",
    "net_carbs": 3
  }
}
```

ingredient_service will respond with:

```json
{
  "status": 201,
  "body": {
    "id": "hW1OHaxRw",
    "name": "test2",
    ...
  }
}
```
return new Verifier(opts).verifyProvider().then(output => {
    console.log('Pact Verification Complete!')
    console.log(output)
})
Client

- Client code...
- Contract Tests
- UI Code
- UI (appraise, quixote, ...
End-to-end

- Necessary
- But limited in number
- Flow through application
- Don’t try to test combinations...
Plus...

- Assembly
- Monitoring
- Infrastructure
- Performance
- Security
- Be creative....
Thanks!

Wouter Lagerweij
@wouterla | wouter@lagerweij.com