



# APEX UI Testing: Best Practices and Pitfalls

16-November-2023

Philipp Hartenfeller, Senior Consultant

UKOUG 2023, Reading, UK

# We are one of the top 20 IT service providers in Germany!



> 800 employees  
> 14 offices globally  
> 150 customers  
> 10 industries



\$ whoami



## Philipp Hartenfeller

- Düsseldorf, Germany
- Master IT-Management
- Since 2016 @ MT GmbH
- Senior Consultant – Oracle APEX
- Mostly doing WebDev, DBs and APEX Testing (<https://lct.software>)

Blog: [hartenfeller.dev/blog/](https://hartenfeller.dev/blog/)



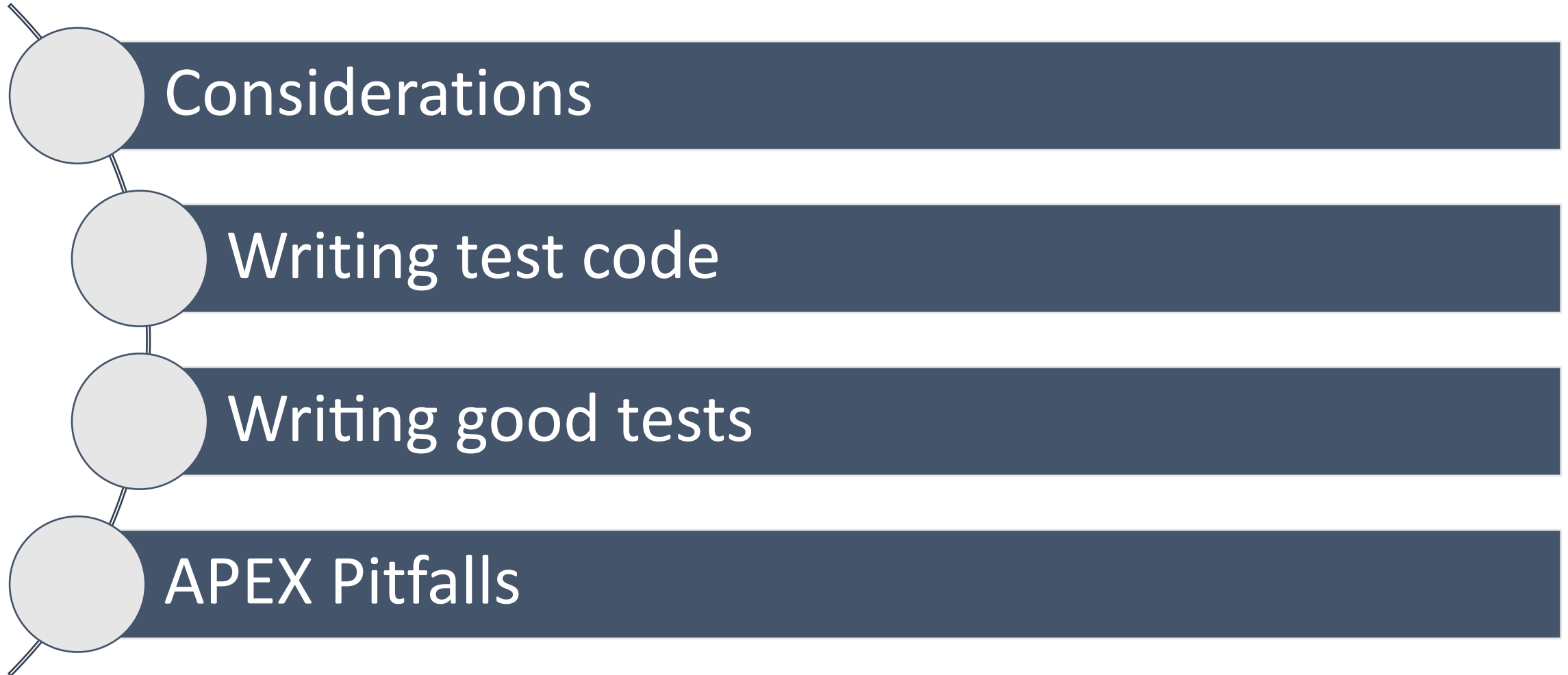
[@phartenfeller](https://twitter.com/phartenfeller)

[hartenfeller.dev/links](https://hartenfeller.dev/links)

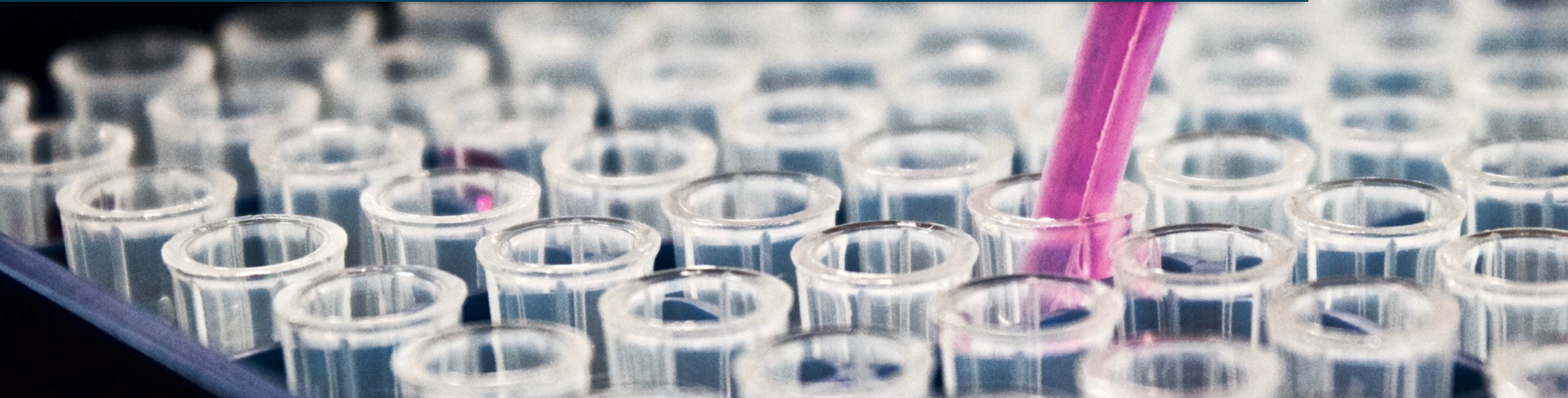


Oracle ACE  
Associate

# What this talk is about? / Agenda



# Considerations

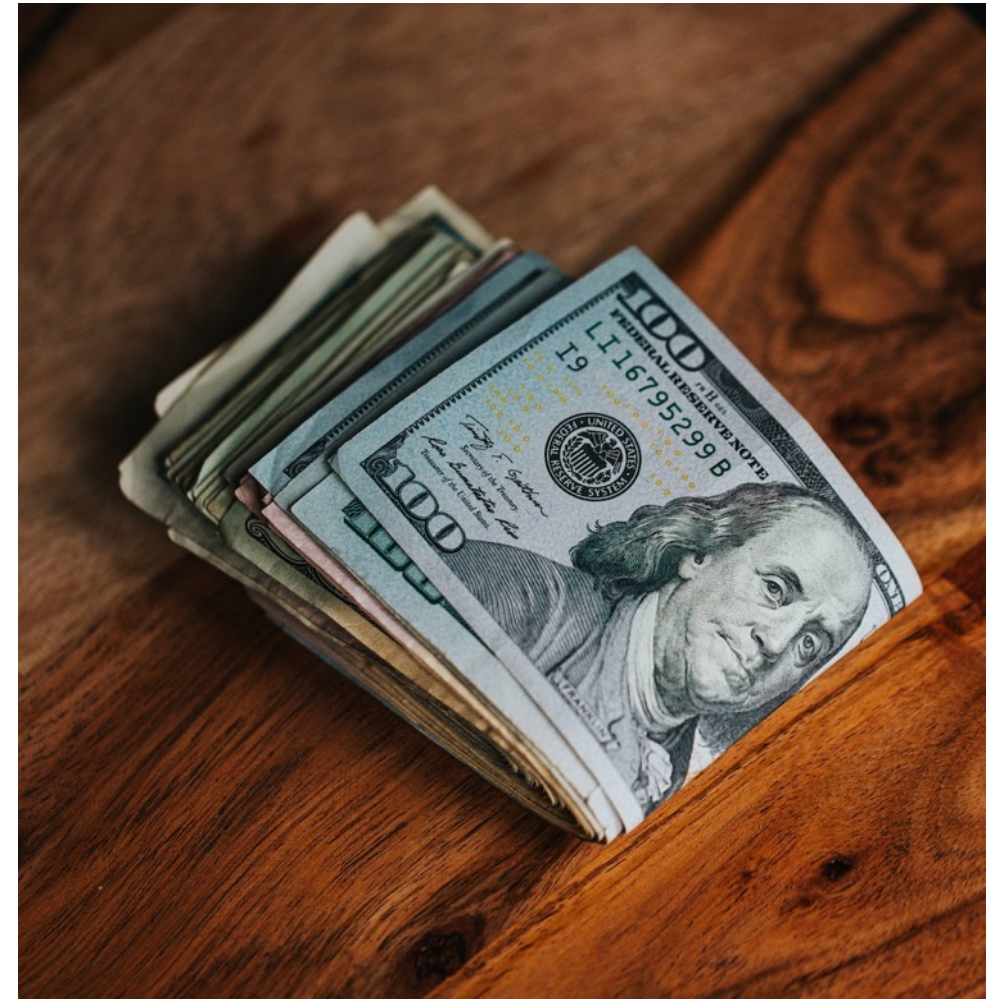


# What are we aiming for?

- **Maintainability**  
(Your app will change; it should be easy to adjust your tests)
- **Robust test routines**  
(tests should not break on APEX upgrades / positional changes)
- **Continuous testing**  
(testing while development; maybe Test-driven development)
- **Valuable testing**  
(Don't just tick the box of having any tests)

# There are no shortcuts

- Effective testing will require:
  - **Lots of time**
  - **Expertise**
  - **Constant maintenance**
- You want to make sure to use it as efficiently as possible!
- Consider **how costly bugs are** in your case and **how much impact** bugs have
- In some cases, manual testing is cheaper



Source: [Nathan Dumlao](#)

# Which framework to use?

## Things to consider

- Browser support
- Performance
- Ease of writing test code
- Ease of use / getting started
- How well is it maintained (browsers get updates very frequently)
- How easy it is to debug / find errors
- Documentation
- Maturity / Feature completeness
- Measures against flakiness



# Which framework to use?

## Things you should not consider

- Programming language support  
(its mostly scripting and modern frameworks > language familiarity)
- Record and Playback features
  - Does not understand APEX --> bad selectors like stylistic instead of descriptive classes
  - Easy to start from 0 but how to maintain?
  - Good for navigating around but you also want assertions

# My recommendations

## Cypress (2017)



Language: JS, TS

License: MIT (Open Source)

Company: Cypress.io, Inc.

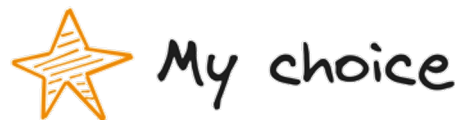
## Playwright (2020)



Languages: JS, TS, Python, C#, Java

License: Apache 2.0 (Open Source)

Company: Microsoft



Key differences:

- PW: more mature browser support (but both Chromium, WebKit and Firefox)
- PW: more comprehensive in things like web-APIs, browser events, multi-context, etc.
- CY financed by commercial cloud PW not commercial (sponsored by Microsoft + OSS community)
- **Both heavily used**

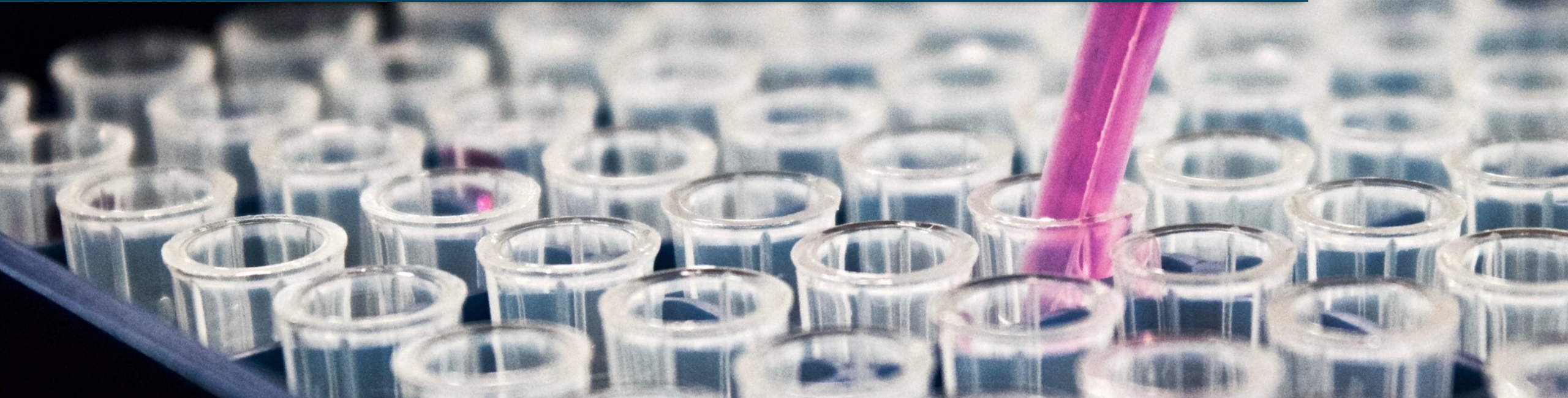
## Playwright: Actionability Checks

Clicking on something will do following checks:

- Attached to DOM
- Visible
- Enabled
- Stable (wait until animation completed)
- Element not obscured by other elements

→ Auto waits and less flaky tests

# Writing Test Code (Playwright)



## Requires Node.js



```
npm init playwright@latest
```

# playwright.config.js

```
const { defineConfig, devices } = require('@playwright/test');

module.exports = defineConfig({
  testDir: './tests',
  /* Run tests in files in parallel */
  fullyParallel: true,
  /* Retry on CI only */
  retries: process.env.CI ? 2 : 0,
  /* Opt out of parallel tests on CI. */
  workers: process.env.CI ? 1 : undefined,
  /* Reporter to use. See https://playwright.dev/docs/test-reporters */
  reporter: 'html',
  /* Shared settings for all the projects below.
  See https://playwright.dev/docs/api/class-testoptions. */
  use: {
    trace: 'on-first-retry',
  },
});
```

For now: keep as it is

Make sure to browse [documentation](#)

# tests/my\_test.spec.js

```
test.describe("Test Group", () => {  
    test("Isolated Test", async ({ page }) => {  
        await page.goto("https://apex.x.com/ords/myapp");  
        await page.locator("#P9999_USERNAME").fill("testuser");  
        await page.locator("#P9999_PASSWORD").fill("test");  
        await page.locator("#LOGIN_BTN").click();  
    });  
    test("Isolated Test 2", async ({ page }) => { ... });  
});
```

# Locators -> how to point to elements



```
await page.getByLabel('King Charles')
await page.getByRole('button', { name: 'Sign in' })
await page.getByText('T N Biscuits <3')

// CSS
await page.locator('#static_id') //id
await page.locator('.t-Button') //class
await page.locator('#filter_group .t-Form-labelContainer input[type="checkbox"]')

// XPath
await page.locator('//*[@id="tsf"]/div[2]/div[1]')
```



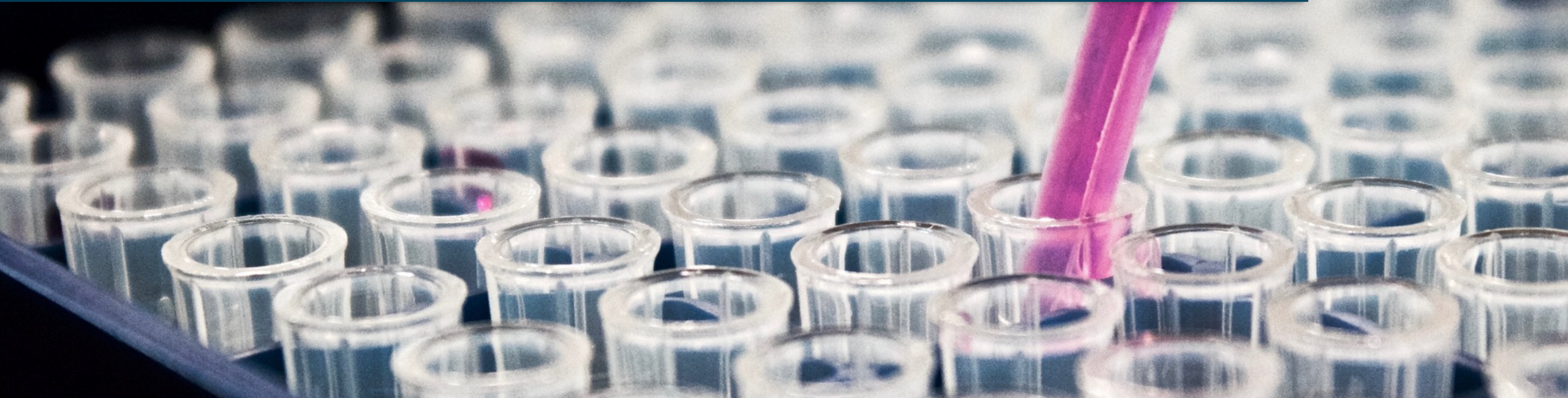
# Demo – First Test



# Demo – Debugging



# Writing Good Tests







# What to test

- Test the **UI** not your backend
  - for PL/SQL use [utPL/SQL](#)
- Test workflows that users perform regularly in your app
- Watch / talk to your end users and replicate **their behavior** in tests
- Test with different users / authorizations
- Test your app not the APEX framework
  - Trust that the APEX devs do their work
  - E.g. fill a date input and don't try to interact with the date picker

## Test environment

- Run tests against dev and/or test environments regularly
- Continuous feedback instead of 40 % fails after 3 months of development
- Disable Single-Sign-On on test environment
  - Test as multiple users with different authorizations
  - Getting Kerberos etc. working in the test runners for multiple users is hard
  - Filling login form during tests is way easier

# Test scope

- Write small tests for specific functionalities
- You want 37  | 3   
not 0  | 1   
to see what exactly is broken
- More information and tests abort after an error occurred
- Developer experience better in smaller tests

Example structure:

## Customer Details Page

- Create
- Edit
- Delete
- Delete button disabled when user has active contracts
- Error: e-mail unique
- Error: last name required

## Contract Page

...

# Test isolation

- When the previous test fails the next one should not be affected
- This means that every test should start at zero
  - New session, new login
  - Data preparation (more on that later)

```
test.beforeEach(async ({ page }) => {  
  await page.goto("https://apex.x.com/ords/myapp");  
  await page.locator("#P9999_USERNAME").fill("testuser");  
  await page.locator("#P9999_PASSWORD").fill("test");  
  await page.locator("#LOGIN_BTN").click();  
});
```

# Assertions

- Add a lot of assertions to your tests
- Fail fast (navigation leads to wrong page, directly check header text)
- Test small things at the side next to main test case (e.g. create customer)
- Your tests are more robust and less flaky



```
await expect(page.getByTestId('todo-item').first()).toBeVisible();
await expect(locator).toHaveText(/Welcome, Test User/);
await expect(locator).toContainText('substring');
await expect(locator).toBeDisabled();
await expect(locator).toHaveClass(/selected/);
```



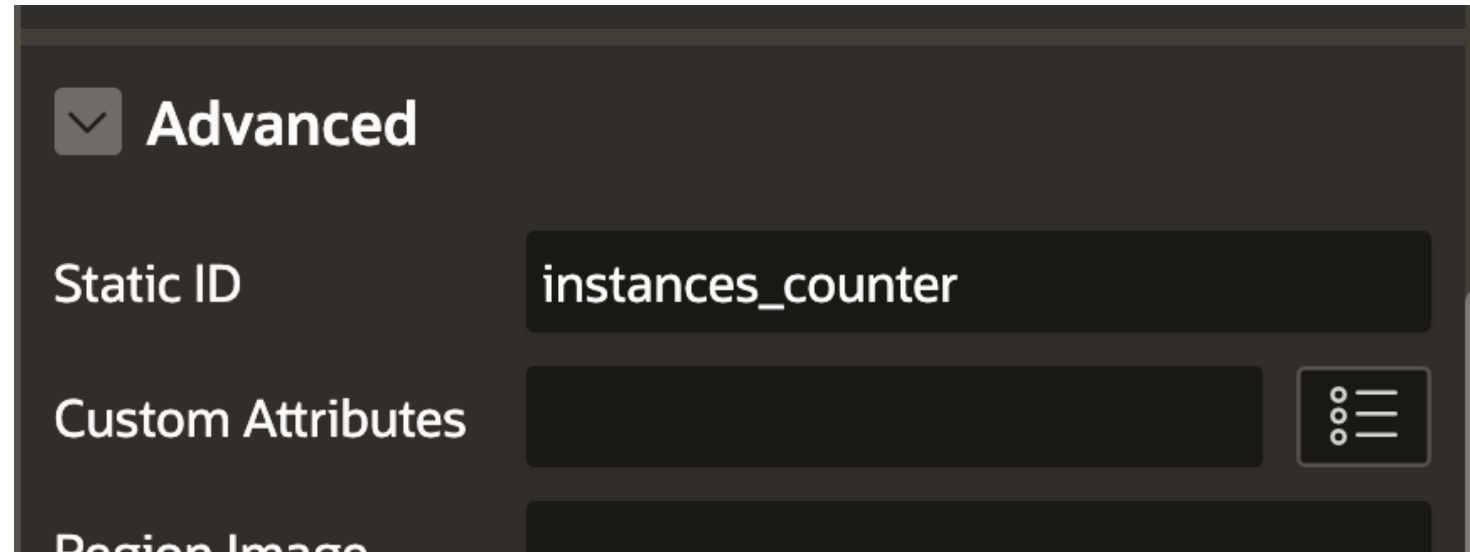
# Test Data Handling

- You will run into data related issues
  - Unique constraints
  - Value missing in LOV
- To make sure our tests work **every time** you need data preparation routines
- PL/SQL procedure that does inserts/deletes etc.
- Create ORDS endpoint for that procedure
- Call endpoint at the start of the test

```
const response = await request.post(
  'apex.site.com/ords/prep_emp_data'
, {}
);
expect(response.status()).toBe(200);
```

# Give static IDs to regions / elements in APEX

- Autogenerated IDs change if you not explicitly check „Export with Original IDs”
- Static IDs give context and are easy to read
- Static IDs likely never change where classes, labels, etc. may do



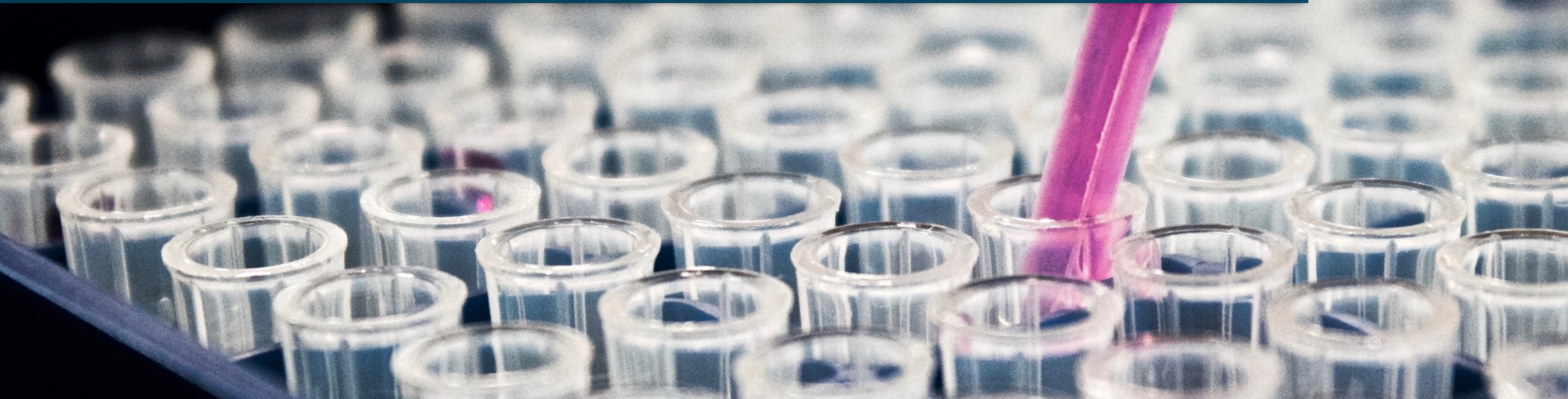
# Extract commonly used functionalities into reusable functions

- Test code is just JavaScript
- Just extract test code you need again to functions

```
async function sampleDbAppLogin({ page, username, password }) {
  await page.goto("https://apex.x.com/ords/myapp");
  await page.locator("#P9999_USERNAME").fill(username);
  await page.locator("#P9999_PASSWORD").fill(password);
  await page.locator("#LOGIN_BTN").click();
}

test("Add Customer", async ({ page }) => {
  await sampleDbAppLogin({
    page,
    username: "testuser",
    password: "test"
  });
  // ...
});
```

# APEX Pitfalls



# Session in URL

- APEX stores the session ID in the URL
- For explicit navigations (not link click) the URL need to include the current session ID

```
await page.waitForFunction(() => {
  if (!window.apex) {
    throw new Error(`No APEX Context available. Page still loading?`);
  }

  try {
    const sid = window.apex.item('pInstance').getValue();
    return sid;
  } catch (e) {
    throw new Error(
      `Error obtaining session: ${e}`
    );
  }
}
```

# Modals

- APEX uses „iframes“ to display modal pages
- iframe is basically HTML document inside HTML document
- Needs special handling

```
▼ <div id="apex_dialog_1" class="ui-dialog-content ui-wid
get-content js-dialogReady" style="width: auto; min-hei
ght: 0px; max-height: none; height: 651.143px;">
...
▼ <iframe src="f?p=126:45:21066082632676:::RP,45::&cs=3
vQMm4HUS5i5-5X2m_xXHBQLhK1JP...Px2U5bFM2f0w1cyfq2yRTwI7
bXBwg6LkAvV_fZ-oVNpwJXNooQs8_wenSutR5pUeyh31T7zKfg"
title="Worksheet Details" width="100%" height="100%"
style="min-width: 95%;height:100%;" scrolling="aut
o"> == $0
▼ #document
  <!DOCTYPE html>
  ▼ <html class="page-45 app-LCT" lang="en">
    ▶ <head> ... </head>
    ▼ <body class="t-Dialog-page t-Dialog-page--standa
rd position:fixed apex-side-nav apex-icons-fonta
```

# Modals



```
const modal = await page.frameLocator("iframe");

// steps from modal
await expect(modal.locator(".t-Dialog-footer")).toBeVisible();
await modal.locator("#P7_CUST_FIRST_NAME").fill("Paddington");
// ...
await modal.locator("#submit").click();

// uses parent context after modal closed
await expect(page.locator("#success_msg")).toBeVisible();
```

# Complex Components: Popup LOV

- Lots of variations:
  - Single / Multi cols
  - Search while typing / button
  - Single / multi values
  - Inline / popup dialog
  - Allow manual values
  - Required / quickpicks
  - In normal / modal page
  - Page item / Interactive Grid

Find interaction code that works for all variants.

## Examples

The examples are arranged in two columns. The first column contains 10 examples, and the second column contains 4 examples. Each example is a light gray box with a label and a small icon or dropdown arrow. The labels include: 'Popup LOV (default)', 'Popup LOV (search while typing)', 'Popup LOV (with modal values)', 'Popup LOV (modal & search as u type)', 'Popup LOV (multiple values)', 'Popup LOV (allow manual entries)', 'Popup LOV (multi value / allow manual entries)', 'Popup LOV (value required)', 'Popup LOV (manual allowed / max. 10 characters)', 'Popup LOV (with quick picks)', 'Popup LOV (with multiple quick picks)', and 'Popup LOV (with multiple quick picks) - M'. Below the 'Popup LOV (with quick picks)' and 'Popup LOV (with multiple quick picks)' examples, there are links for 'Quick Pick 1, Quick Pick 2, Quick Pick 3'.

Validate and submit page



## Popup LOV strategy

- Click on popup trigger button
- Grab popup dialog element (**always on parent page**)
- Clear search input and type in term
- Check if a search button is present and click
- Wait for network request that includes „/wwv\_flow.ajax“
- Check if there is a „no data found“ element present
- Iterate results and click on the one that exactly matches the search term (order matters -> “Kevin“ can appear above “Kev“ while searching for “Kev”)

## Editing the Interactive Grid

- Delete row -> use delete button from 🍔 menu
- New row -> click add row button
- Edit row -> filter grid so that only the row you want to edit is displayed

**Edit columns:** better not by position but by column ID

## Editing the Interactive Grid

Edit a column:

- Click into cell that you want to edit (**next slide**)
- Find input by column ID (generated or static ID)
- Fill Input

```
<input type="text" id="C725575308089259406" name="7  
25575308089259406" class="text_field apex-item-text  
js-ignoreChange js-tabbable" value maxlength="60"  
data-text-case="UPPER" tabindex="-1"> == $0
```

## Editing the Interactive Grid

Problem: column ID not in table layout (unlike inputs)

Solution: find column index with column ID:

- Get all elements from selector: „#ig\_id .a-GV-header > span:first-child“ (table header labels)
- Loop over each and find **index** of the one with the right ID
- We can then click on „locator('#ig\_id .a-GV-table tr.a-GV-row:first-child .a-GV-cell').nth(**index**)“

```
▼ <th role="columnheader" class="a-GV-header u-tS" data-idx="3" aria-haspopup="true" tabindex="-1">  
  <span class="a-GV-headerLabel" id="C725575843278259407_HDR">Job</span> == $0
```

# How it works with LCT

### Add Value(s)

Pick a Step ✓ Add Values ●

Fill a field with the given content

---

Step Name ? Execution Sequence ?  
Next sequence: 300

Item	Step Parameters and Values	Options
Selector Type <input type="radio"/> Custom <input checked="" type="radio"/> <b>Interactive Grid Column</b> <input type="radio"/> Page Element	Value <span>?</span>	Step Timeout (ms) <span>?</span> Default: 0
Application <span>?</span> 102 - Sample Interactive Grids <span>▼</span>		Force Interaction <span>?</span> <input type="radio"/> Yes <input checked="" type="radio"/> <b>No</b>
Page <span>?</span> - Select - <span>▼</span>		
Interactive Grid <span>?</span> <span>▼</span>		
Interactive Grid Column Element <span>?</span> <span>▼</span>		

< Cancel Create

# Testing APEX Apps is now as easy as creating them.

- Tailored to APEX
- Save a lot of time on regression tests
- Use our intuitive LCT-App and don't write any test code
- Testing on multiple platforms simultaneously

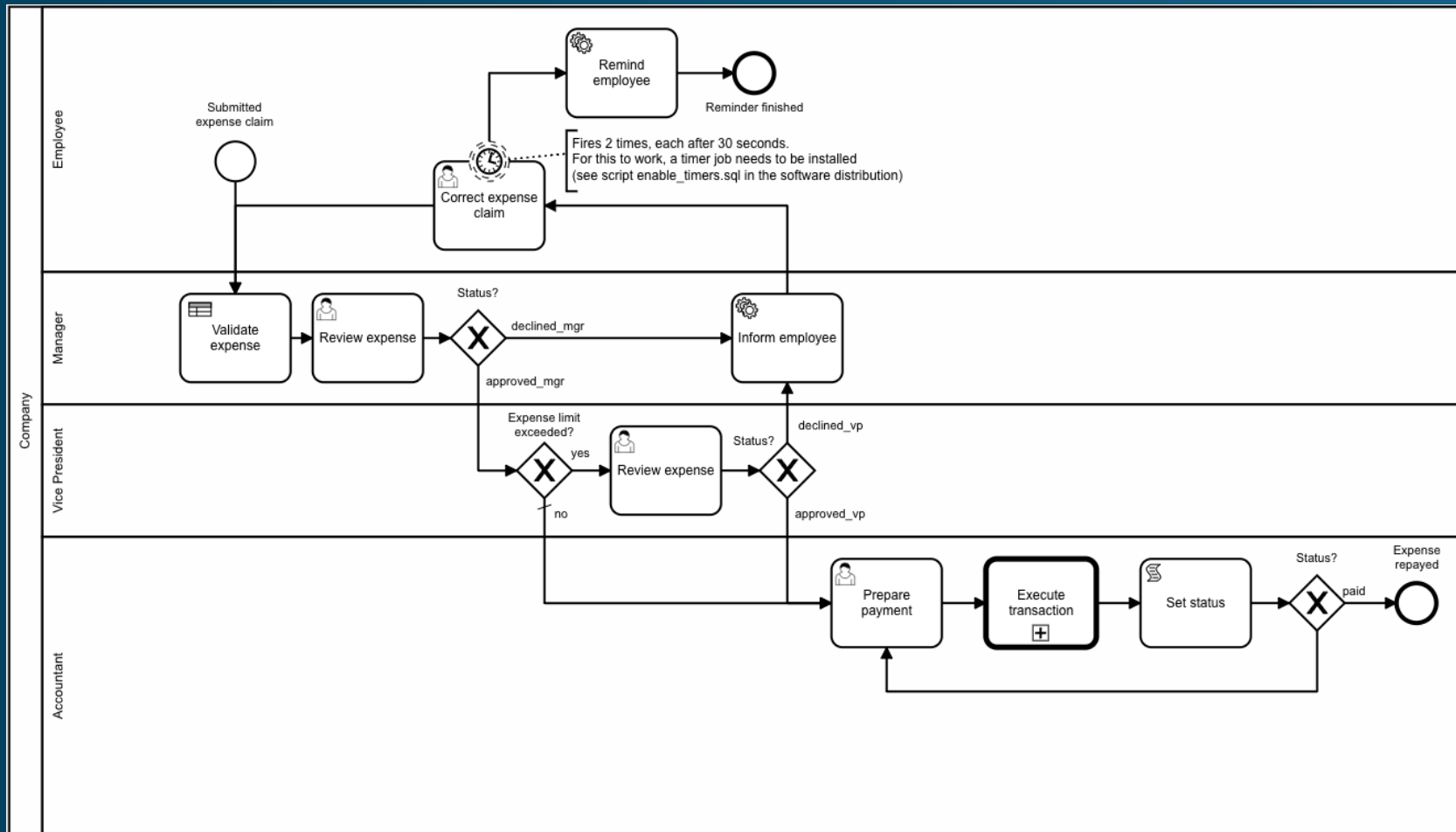


# LCT



# Flows for APEX

## BPMN 2.0 Workflows for APEX



- Open Source
- Community Driven
- Support available





**UKOUG : 23**  
**CONFERENCE**

— INDEPENDENT —  
**UKOUG**  
UK ORACLE USER GROUP

**ORACLE**



---

**CONFERENCE SPONSORS**