Level up your backend with Cucumber feature tests

by Koen Bollen @ Golab 2023

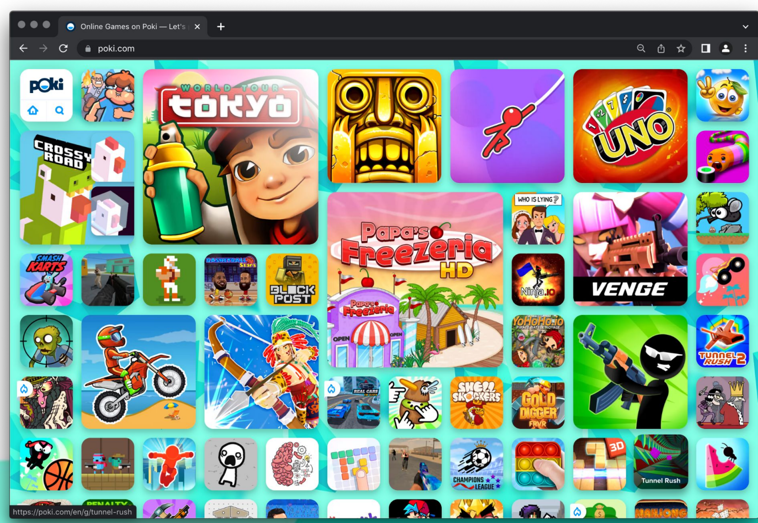
Quick intro!

History of Koen Bollen:

- Taught at University of Applied Sciences in Amsterdam
- Gamedev → PC & Mobile
- Using Go since 2014
- Started at Poki in 2019

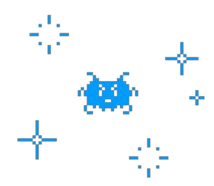
About Poki

- Web game portal
- ±50 million unique users per month
- Work with over 300 developers on a revenue share basis
- We use a lot of Cucumber!



Level up your backend with feature tests

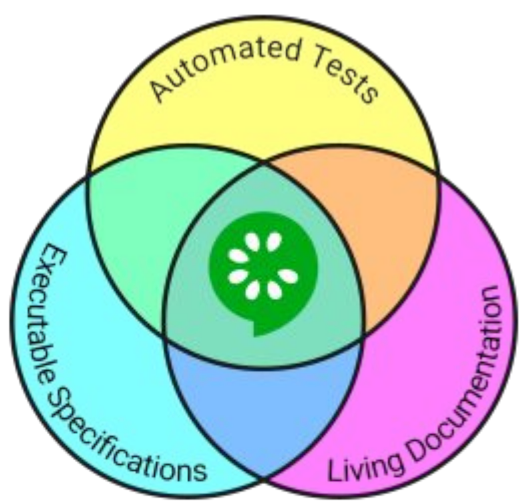
1. 👨🏻‍💻 Introductions
2. 🥒 What is Cucumber
3. 🌐 Testing your APIs
4. 🏎️ Why?
5. 🎛️ Demo time
6. 🚀 Protips
7. 🗒️ Closing notes



🥒 What is Cucumber

100+

What is Cucumber?




```
# Comment
Feature: Eating too many cucumbers may not be good for you
  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```

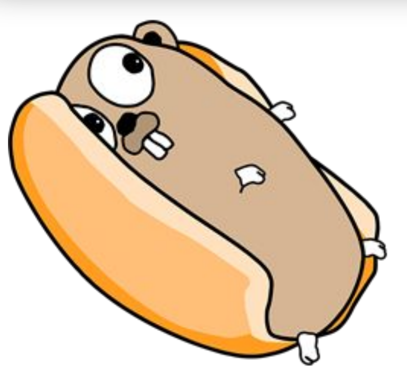
Running a cucumber scenario

```
# Comment
Feature: Eating too many cucumbers may not be good for
  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```

```
people := make(map[string]int)
var activePerson string
step(`/(.*) is hungry/`, func(name string) {
  people[name] = 0
  activePerson = name
})
```

```
step(`/she eats (\d+) cucumbers/`, func(amount int) {
  people[activePerson] += amount
})
step(`/then she will be full/`, func() error {
  if people[activePerson] < FullnessThreshold {
    return fmt. Errorf("%s is not full", activePerson)
  }
})
```



github.com/cucumber/godog

Examples:

```
Feature:
  Scenario: Terms and Conditions effective from 17 May 2022
    Given the contract is created
    When the contract is ready to sign
    Then the Seller agrees to Term and Conditions version 17 May 2022
```

```
Feature: API Service
  Scenario: GET the heatlh endpoint
    When the client does a GET request to "/health"
    Then the response code equals 200 (OK)
    And the response header "Content-Type" should be "application/json"
    And the response body should be:
      """json
      {
        "healthy": true
      }
      """
```

```
Feature: Players can create and connect a network of players

  Background:
    Given the "signaling" backend is running

  Scenario: A player can create a network to join a game
    When "green" creates a network for game "164aae2e-c6e5-4073-80bf-b2a03ad4c9b7"
    Then "green" receives the network event "ready"
    And "green" has recieved the peer ID "h5yzwyizlwao"
```

Keywords

- Feature
- Background
- Scenario
- Steps:
  - Given
  - When
  - Then

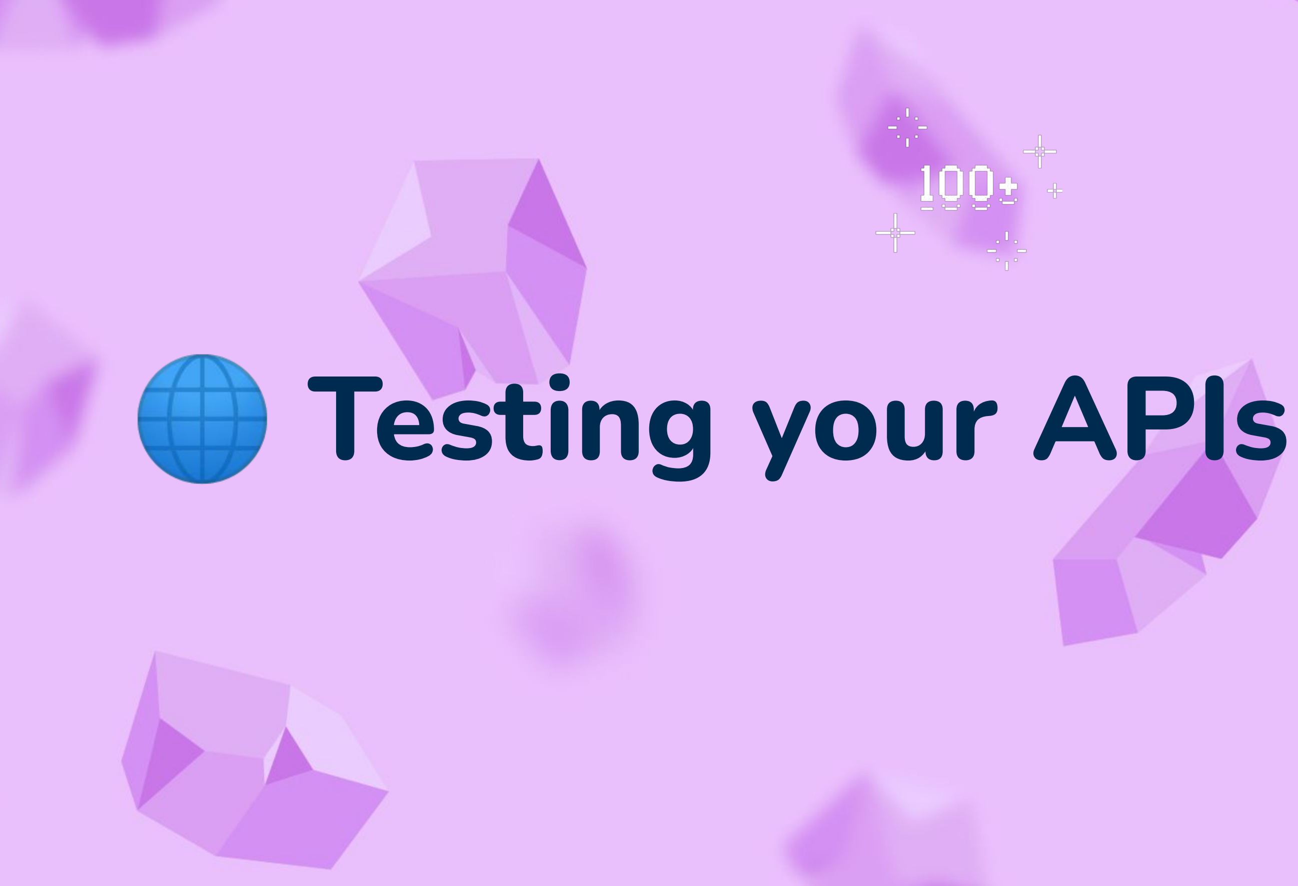
```
Feature: Custom CORS Headers

  In order to prevent large scale abuse of this service we only allow cl
  requests from Poki domains. Some games can be configured to be allowed
  custom other domains.

  Scenario: CORS is always allowed on poki.com
    Given the game "c79e39b4-3b34-42ac-b538-c2fe0d49944f" exists

    When the client does a OPTIONS request to "/v0/c79e39b4-3b34-42ac-b5
      | Origin                        | https://poki.com |
      | Access-Control-Request-Method | POST             |

    Then the response code should be 204 (No Content)
    And the response header "Access-Control-Allow-Origin" should be "htt
    And the response header "Access-Control-Allow-Methods" should be "DE
    And the response header "Access-Control-Allow-Headers" should be "*"
```



🌐 Testing your APIs

Using Go’s standard testing package

```go
func TestHandler_ListUsers(t *testing.T) {
    req := httptest.NewRequest("GET", "/users", nil)
    resp := httptest.NewRecorder()

    mockStore := &stores.Memory{}
    fakeClient := &cloudflare.MockClient{}

    mockStore.AddMockUser("1", "john")
    mockStore.AddMockUser("2", "kate")

    handler := Handler(context.Background(), mockStore, fakeClient)

    handler.ServeHTTP(resp, req)

    if resp.Code != http.StatusOK {
        t.Errorf("expected status code %d, got %d", http.StatusOK, resp.Code)
    }

    // test body...
}
```

Using a library

```go
func TestGetMessage(t *testing.T) {
    handler := func(w http.ResponseWriter, r *http.Request) {
        msg := `{"message": "hello"}`
        _, _ = w.Write([]byte(msg))
        w.WriteHeader(http.StatusOK)
    }

    apitest.New().
        HandlerFunc(handler).
        Get("/message").
        Expect(t).
        Body(`{"message": "hello"}`).
        Status(http.StatusOK).
        End()
}
```

Maybe we use cucumber?

```gherkin
Feature: List users
  Scenario: List all users from a team
    Given these "teams" records:
      | id | name   |
      | 1  | A Team |
      | 2  | B Team |
    And these "users" records:
      | id | team_id | name |
      | 1  | 1       | Kate |
      | 2  | 2       | John |
      | 3  | 1       | Jane |
    When the client does a GET request to "/teams/1/users"
    Then the response code equals 200 (OK)
    And the response header "Content-Type" should be "application/json"
    And the response body should be:
      """json
      [
        {"id": 1, "team_id": 1, "name": "Kate"),
        {"id": 3, "team_id": 1, "name": "Jane")
      ]
      """
```

Full example

```
Feature: Vote Counting

  A common feature of userdata is for other users to up- and downvote entries.
  These 'votes' need to be tracked so users can't keep continue to vote.

  Background:
    Given these "userdata" records exist:
      | id                   | values                                    | data     | game
      | cevc2tkllhclvn23d8r0 | {"title": "Best Map", "up-vote": 42}      | "binary" | be2546ff-dfcd-4fc9-8276-62476
      | cevc2usllhcm1snp7gh0 | {"title": "Good Map 1", "up-vote": 32}    | "binary" | be2546ff-dfcd-4fc9-8276-62476
      | cevc2vkllhcm21kvugjg | {"title": "Good Map 2", "up-vote": 32}    | "binary" | be2546ff-dfcd-4fc9-8276-62476
      | cevc3l4llhcmctncgbug | {"title": "Bad", "up-vote": 4}            | "binary" | be2546ff-dfcd-4fc9-8276-62476

  Scenario: Vote on a level
    Given the time is "2006-01-02T18:04:05Z"
    And the client's remote address is "235.209.157.191"

    When the client does a POST request to "/v0/be2546ff-dfcd-4fc9-8276-62476f625870/userdata/levels/cevc2vkllhc
/_vote?key=up-vote"

    Then the response code should be 200 (OK)
    And the response body should be the following "application/json":
```

```
    Then the response code should be 200 (OK)
    And the response body should be the following "application/json":
      """json
      {
        "id": "cevc2vkllhcm21kvugjg",
        "meta": {
          "revision": 1,
          "created_at": "2006-01-02T15:04:05Z",
          "updated_at": "2006-01-02T18:04:05Z",
          "expires_at": "2007-01-02T18:04:05Z",
          "expires_in": 31536000
        },
        "values": {
          "title": "Good Map 2",
          "up-vote": 33
        }
      }
      """
    And this "userdata" record exists:
      | id       | cevc2vkllhcm21kvugjg                   |
      | values   | {"title": "Good Map 2", "up-vote": 33} |
      | revision | 1                                      |
    And this "votes" record exists:
      | userdata_id | cevc2vkllhcm21kvugjg |
      | value_key   | up-vote              |
      | user_ip     | 235.209.157.191      |
      | day         | 2006-01-02T00:00:00Z |
      | created_at  | 2006-01-02T18:04:05Z |
```

Step definitions

```
scenario.Step(`^the response code should be (\d+) \([^\)]+\)$`,
    s.ThenStatusShouldBe)

func (s *HTTPSteps) ThenStatusShouldBe(ctx context.Context, status int) error {
    if s.RecordedResponse == nil {
        return fmt.Errorf("no request was made")
    }
    if s.RecordedResponse.Code != status {
        body := strings.TrimSpace(s.RecordedResponse.Body.String())
        return fmt.Errorf("expected status %d, got %d (body: %v)", status, s.Rec
    }
    return nil
}
```

```go
scenario.Step(`/^these "([^"]* )" records:$/`, givenTheseRecords)

func givenTheseRecords(tableName string, data *godog.Table) error {
    tx, err := database.Begin()
    if err != nil {
        return err
    }
    fields := ExtractTableHeaders(data)
    marks := strings.Repeat("?, ", len(fields)-1) + "?"

    stmt, err := tx.Prepare("REPLACE INTO " + tableName + " (`" + strings.Join(fields, "`, `") +
        "`) VALUES (" + marks + ")")
    if err != nil {
        tx.Rollback()
        return err
    }
    for i := 1; i < len(data.Rows); i++ {
        var vals []any
        for _, cell := range data.Rows[i].Cells {
            vals = append(vals, cell.Value)
        }
        if _, err := stmt.Exec(vals...); err != nil {
            tx.Rollback()
            return err
        }
    }
    return tx.Commit()
}
```

Why?

Pros

- Everyone can read these tests
- Super easy to write tests upfront before worrying about implementation
- Cucumber can be used in any programming language
- 100% test coverage of documented features!
- We don’t run our codebase locally anymore

Pro: Everyone can read the tests

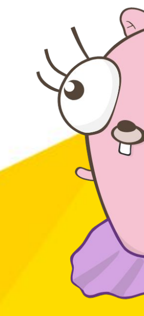
- It serves as documentation
- Early discussion before implementation
  - Ask our frontender if this is an endpoint they can work with.

Pro: Easy to write test upfront

- Allows you to think about a feature before implementing it
- Makes it clear to colleagues what an endpoint is going to be

  (we sometimes make a pull-request with just a test)

Pro: Cucumber is reusable



- Skill learned in Ruby you can use in Go
- Porting a (sub)service to another technology? Copy over the test suite!
- One testing syntax for multiple projects within an organization

Pro: 100% test coverage*

*of documented features.

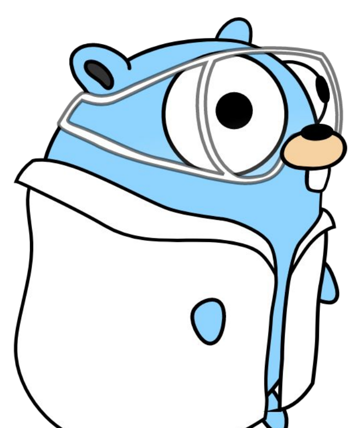
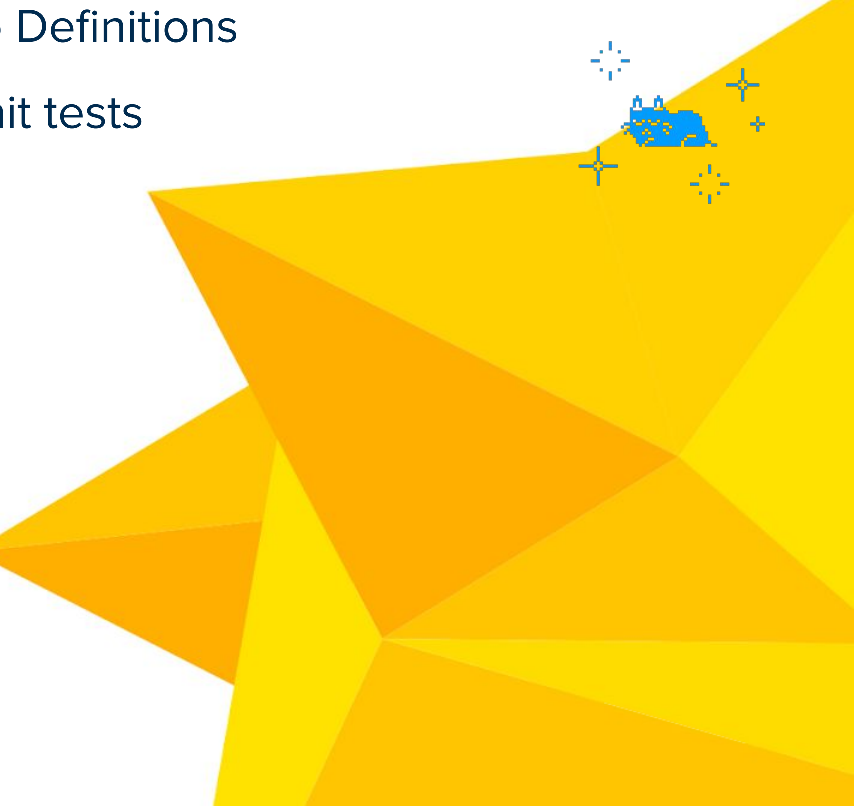
- If the feature tests serve as documentation, then the documentation is tested
- Super easy to add a fringe edge-case as a feature test to fix a bug

Pro: Easy development

- We don’t run our codebase locally
- Write a feature test, then implement the feature while running the test to try out your code.
- Easy to run just one test or a section of test

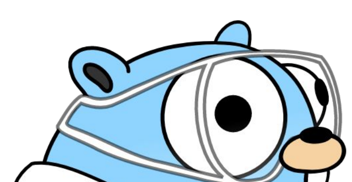
Cons

- More work upfront implementing Step Definitions
- Might run a little slower then native unit tests
- Need to learn cucumber+gherkin
- Debugging is a bit more involved

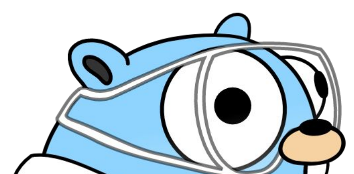
Con: More work upfront

- Making step-definitions takes time, especially when making them generalized
- You always need a step-definitions, in normal tests you can add code inline



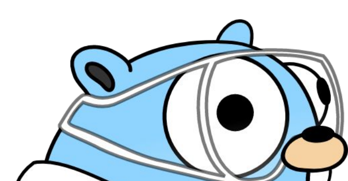
Con: Performance overhead

- Might run a little slower than native unit tests
- Had to compile the entire app in one binaries
- Has to parse the tests and execute them instead of directly running a TestXxx() function



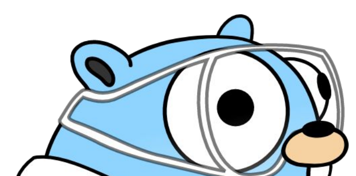
Con: New skill

- You need to learn a new syntax
- It’s a different way of testing/working with feature tests



Con: Debugging is more involved

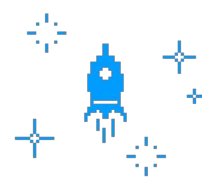
- No VScode integration (yet)
- You can’t add breakpoints in the feature test files
- You can run tests using delve (also in vscode)



🎛️ Demo time

🚀 Tips

1. Isolate your tests
2. Generalize your step definitions
3. But also allow for specific steps for readability
4. Write test for the reader (also true for coding in general)
5. Use actual dependencies when they contain logic (e.g. database)



🗒️ Closing notes

- We are super happy with our cucumber feature tests
- Super useful for regression tests and red/green tests
- Easier in communication with frontenders and other stakeholders
- We still write a bunch of unit tests to cover small edge cases
- It saves us a lot of time!

Thanks! Any questions?

Links

https://cucumber.io/

https://github.com/cucumber/godog

https://cucumber-rs.github.io/cucumber/current/

https://github.com/egonelbre/gophers

Code snippets created with: https://carbon.now.sh/

fin