# Mocking Opportunities in Pacemaker

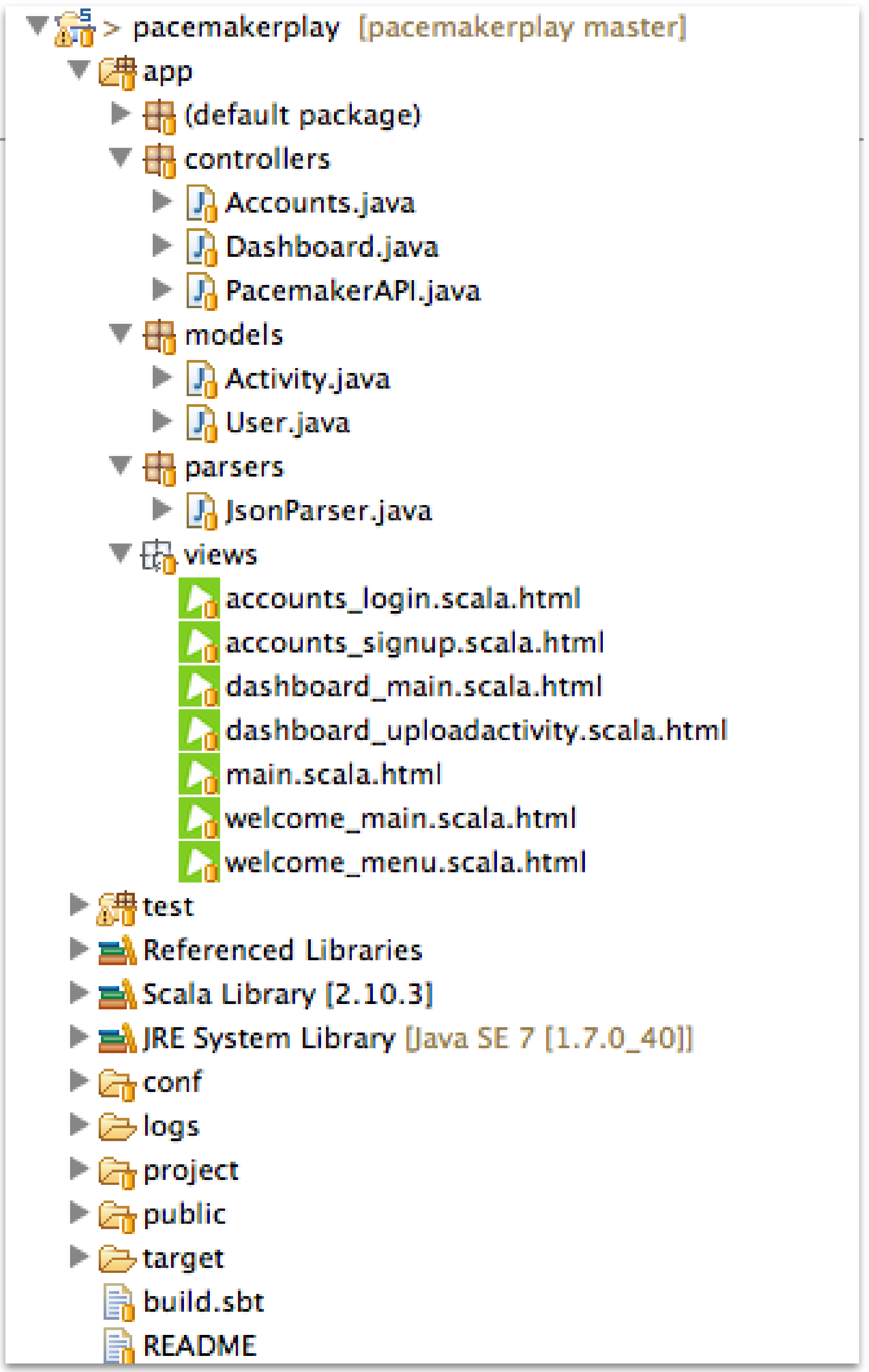
Produced by: Eamonn de Leastar (edeleastar@wit.ie)

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

# Pacemaker 2.0

- REST Web Service
- Standard Web UI



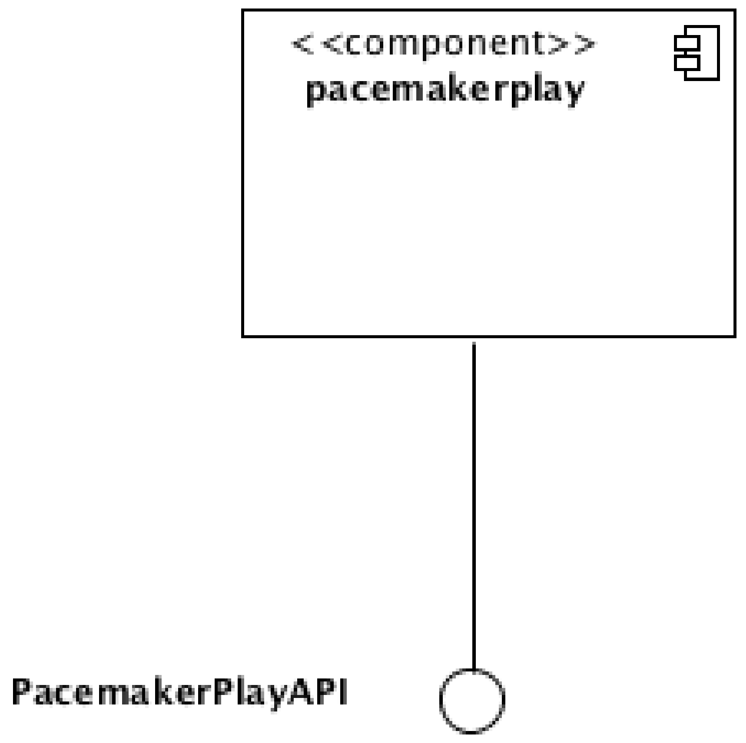
```
▼ pacemakerplay [pacemakerplay master]
  ▼ app
    ▶ (default package)
    ▼ controllers
      ▶ Accounts.java
      ▶ Dashboard.java
      ▶ PacemakerAPI.java
    ▼ models
      ▶ Activity.java
      ▶ User.java
    ▼ parsers
      ▶ JsonParser.java
    ▼ views
        accounts_login.scala.html
        accounts_signup.scala.html
        dashboard_main.scala.html
        dashboard_uploadactivity.scala.html
        main.scala.html
        welcome_main.scala.html
        welcome_menu.scala.html
  ▶ test
  ▶ Referenced Libraries
  ▶ Scala Library [2.10.3]
  ▶ JRE System Library [Java SE 7 [1.7.0_40]]
  ▶ conf
  ▶ logs
  ▶ project
  ▶ public
  ▶ target
    build.sbt
    README
```

## Assignment Rubric for Assignment 2 (top marks deployment + any 2 others)

| Standard | Deployment | Features | UX | DX |
|---|---|---|---|---|
| Baseline | REST (Local) | activities *(see runkeeper)* | Simple Web (up to 15 “UI” routes). | TDD – REST Tests |
| Good | REST (Deployed) | reports *(see runkeeper)* | Simple Web (> 15 “UI” routes and some Semantic UI or similar). | Modelling and Analysis |
| Excellent | REST (Secured e.g. OAuth) | friends *(see runkeeper)* | Simple Web (> 15 “UI” routes, widespread integration of Semantic UI or similar, very good UX) | TDD – Mocks / Doubles (factor out two services) |
| Outstanding | REST (2 x cloud) | dashboard *(see runkeeper)* | Rich Client / Mobile App | API Documentation |

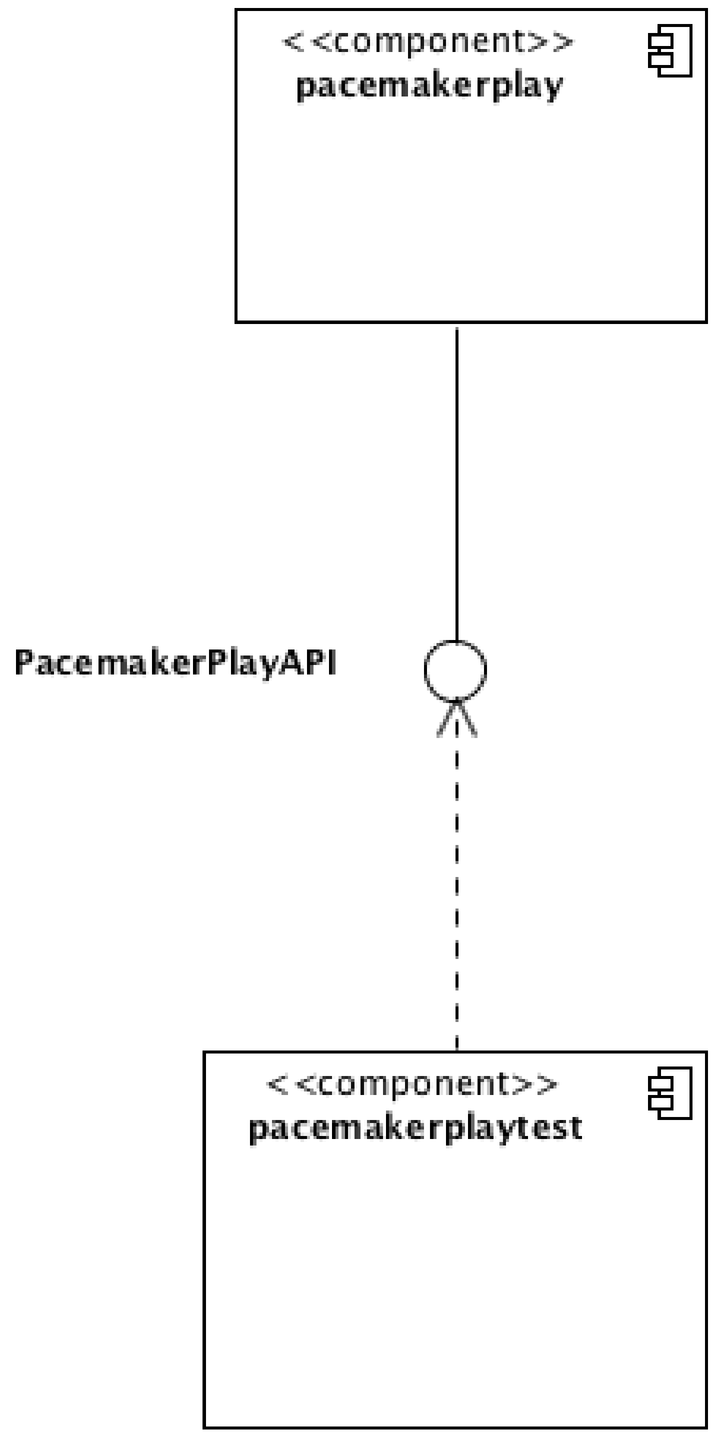
# pacemakerplay service

- Provides an API for managing:
  - users
  - activities
  - routes (within activities)



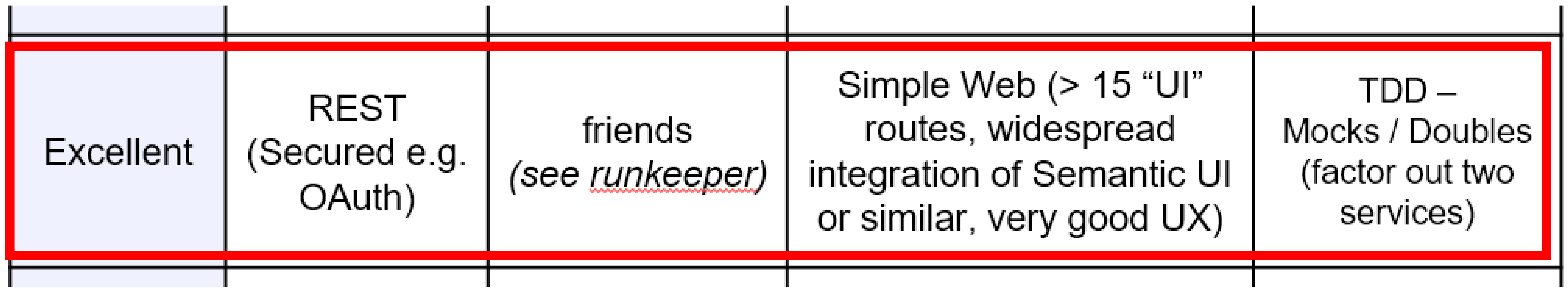
```
# API

GET       /api/users                                 controllers.PacemakerAPI.users()
DELETE    /api/users                                 controllers.PacemakerAPI.deleteAllUsers()
POST      /api/users                                 controllers.PacemakerAPI.createUser()

GET       /api/users/:id                             controllers.PacemakerAPI.user(id: Long)
DELETE    /api/users/:id                             controllers.PacemakerAPI.deleteUser(id: Long)
PUT       /api/users/:id                             controllers.PacemakerAPI.updateUser(id: Long)

GET       /api/users/:userId/activities              controllers.PacemakerAPI.activities(userId: Long)
POST      /api/users/:userId/activities              controllers.PacemakerAPI.createActivity(userId: Long)

GET       /api/users/:userId/activities/:activityId  controllers.PacemakerAPI.activity(userId: Long, activityId:Long)
DELETE    /api/users/:userId/activities/:activityId  controllers.PacemakerAPI.deleteActivity(userId: Long, activityId:Long)
PUT       /api/users/:userId/activities/:activityId  controllers.PacemakerAPI.updateActivity(userId: Long, activityId:Long)
```

## pacemakerplaytest



- Exercise the API over HTTP
- Full set of tests to verify key features

## New Feature - ‘Social’



| | | | | |
|---|---|---|---|---|
| Excellent | REST (Secured e.g. OAuth) | friends *(see runkeeper)* | Simple Web (> 15 “UI” routes, widespread integration of Semantic UI or similar, very good UX) | TDD – Mocks / Doubles (factor out two services) |

- Follow friends
- View Friends activities
- ‘Feeds’, etc…

- Consider modelling this as a separate service
- With its own API for managing
  - social graph
  - updates
  - follow/unfollow etc…

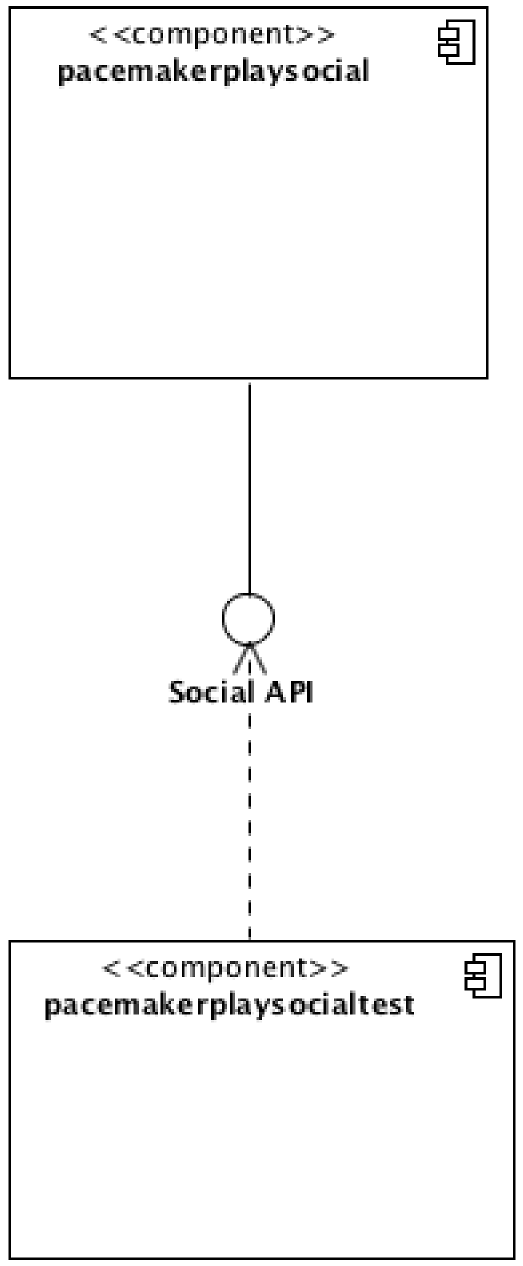
## pacemakerplaysocial

- Uncouple the social aspects from the core activity service.

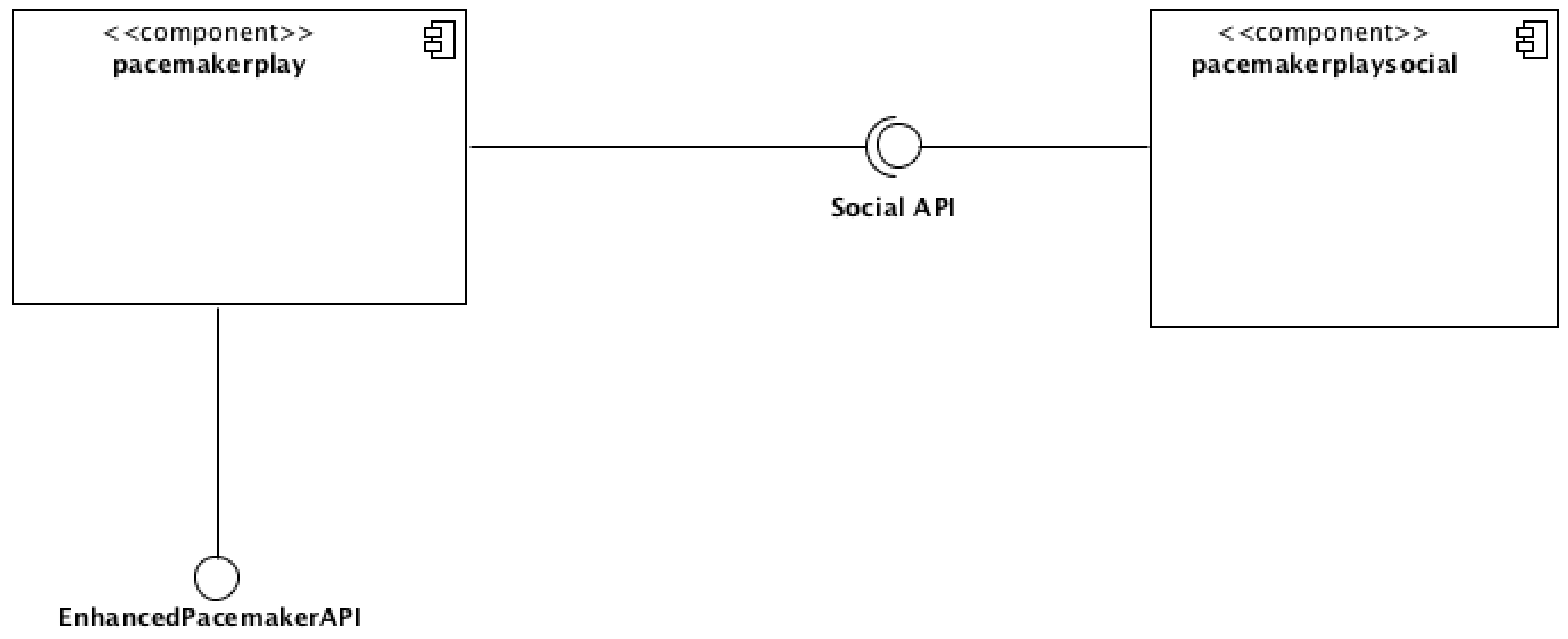
- Allows the social service to be constructed and optimised independently:

  - NoSQL database more appropriate.

  - Interfaces to Twitter, FaceBook, etc…



# pacemakerplay → pacemakerplaysocial

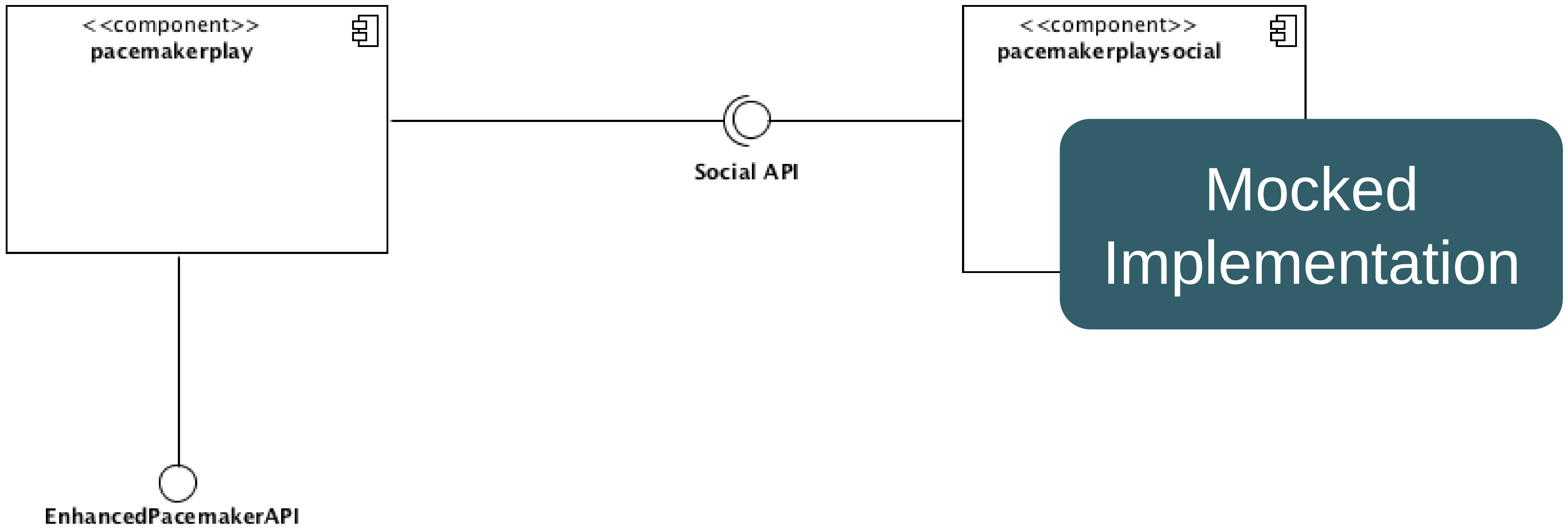


- ‘Enhanced’ API uses Social features, provided by pacemakerplaysocial service

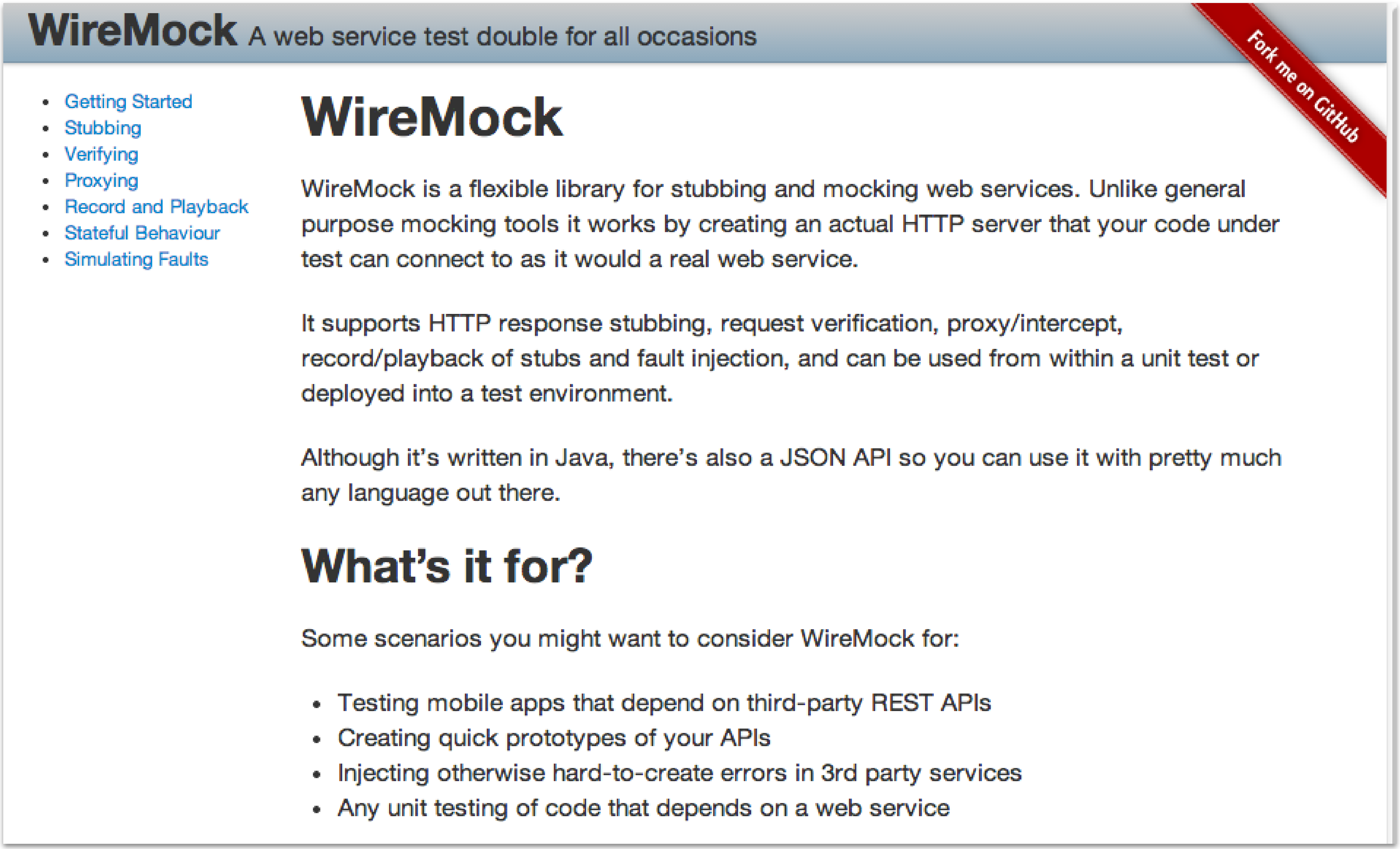
## The Role of Mock Objects

- Postpone the Construction of the pacemakerplaysocial component.
- Model the API first as a REST API.
- Mock out its implementation.
- Then:
    - Write Tests against the Mocked out API.
    - Enhance pacemaker play API to incorporate social features.

## pacemakerplay → pacemakerplaysocial



- Use REST Mocking Services to deliver realistic test data to pacemakerplay.



**WireMock** A web service test double for all occasions

- Getting Started
- Stubbing
- Verifying
- Proxying
- Record and Playback
- Stateful Behaviour
- Simulating Faults

# WireMock

WireMock is a flexible library for stubbing and mocking web services. Unlike general purpose mocking tools it works by creating an actual HTTP server that your code under test can connect to as it would a real web service.

It supports HTTP response stubbing, request verification, proxy/intercept, record/playback of stubs and fault injection, and can be used from within a unit test or deployed into a test environment.

Although it's written in Java, there's also a JSON API so you can use it with pretty much any language out there.

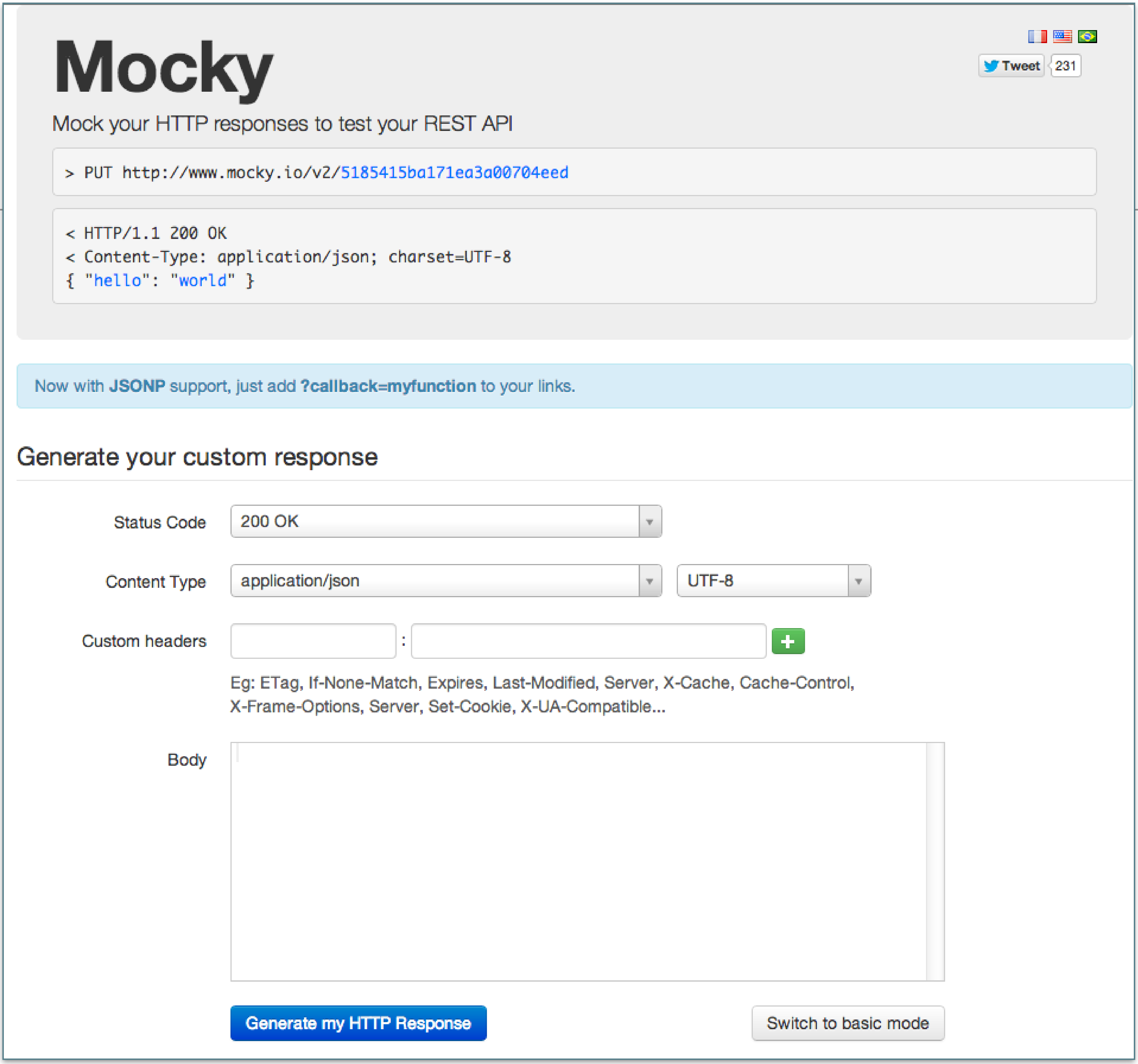
## What's it for?

Some scenarios you might want to consider WireMock for:

- Testing mobile apps that depend on third-party REST APIs
- Creating quick prototypes of your APIs
- Injecting otherwise hard-to-create errors in 3rd party services
- Any unit testing of code that depends on a web service

## WireMock Video

- Gives a very good introduction to WireMock with examples of how to use it:
  - https://skillsmatter.com/skillscasts/6853-scalable-management-of-test-data-making-tests-readable#video



Mocky

Mock your HTTP responses to test your REST API

```
> PUT http://www.mocky.io/v2/5185415ba171ea3a00704eed
```

```
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=UTF-8
{ "hello": "world" }
```

Now with **JSONP** support, just add **?callback=myfunction** to your links.

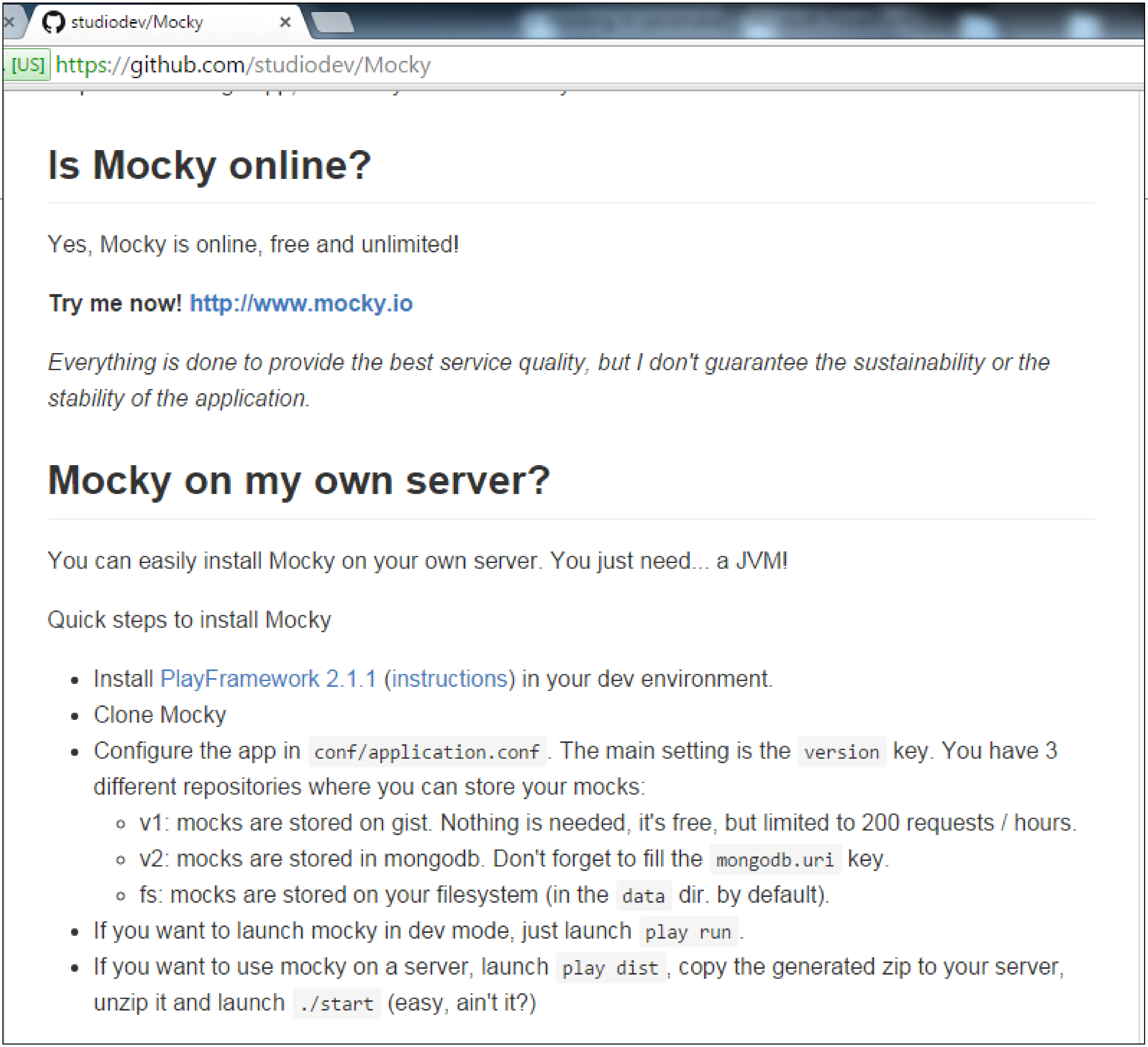
Generate your custom response

| | |
|---|---|
| Status Code | 200 OK |
| Content Type | application/json · UTF-8 |
| Custom headers | : |
| | Eg: ETag, If-None-Match, Expires, Last-Modified, Server, X-Cache, Cache-Control, X-Frame-Options, Server, Set-Cookie, X-UA-Compatible... |
| Body | |

http://www.mocky.io/

## Mocky

- Mocky is a service that allows web developers to mock their HTTP responses in order to test the functionality of a REST API.
- Available as:
  - A Web Console – www.mocky.io.
  - Downloadable software – for installation on your own server.



## Is Mocky online?

Yes, Mocky is online, free and unlimited!

**Try me now! http://www.mocky.io**

*Everything is done to provide the best service quality, but I don't guarantee the sustainability or the stability of the application.*

## Mocky on my own server?

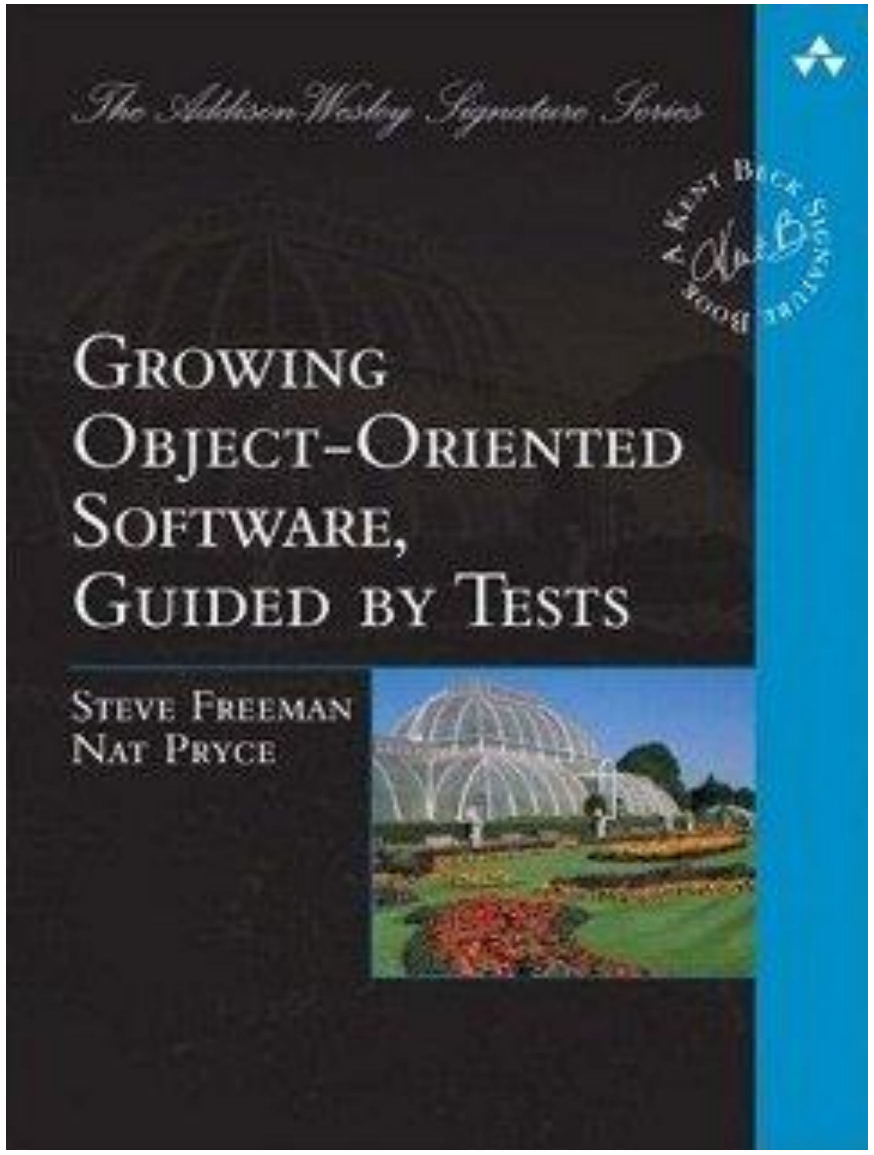
You can easily install Mocky on your own server. You just need... a JVM!

Quick steps to install Mocky

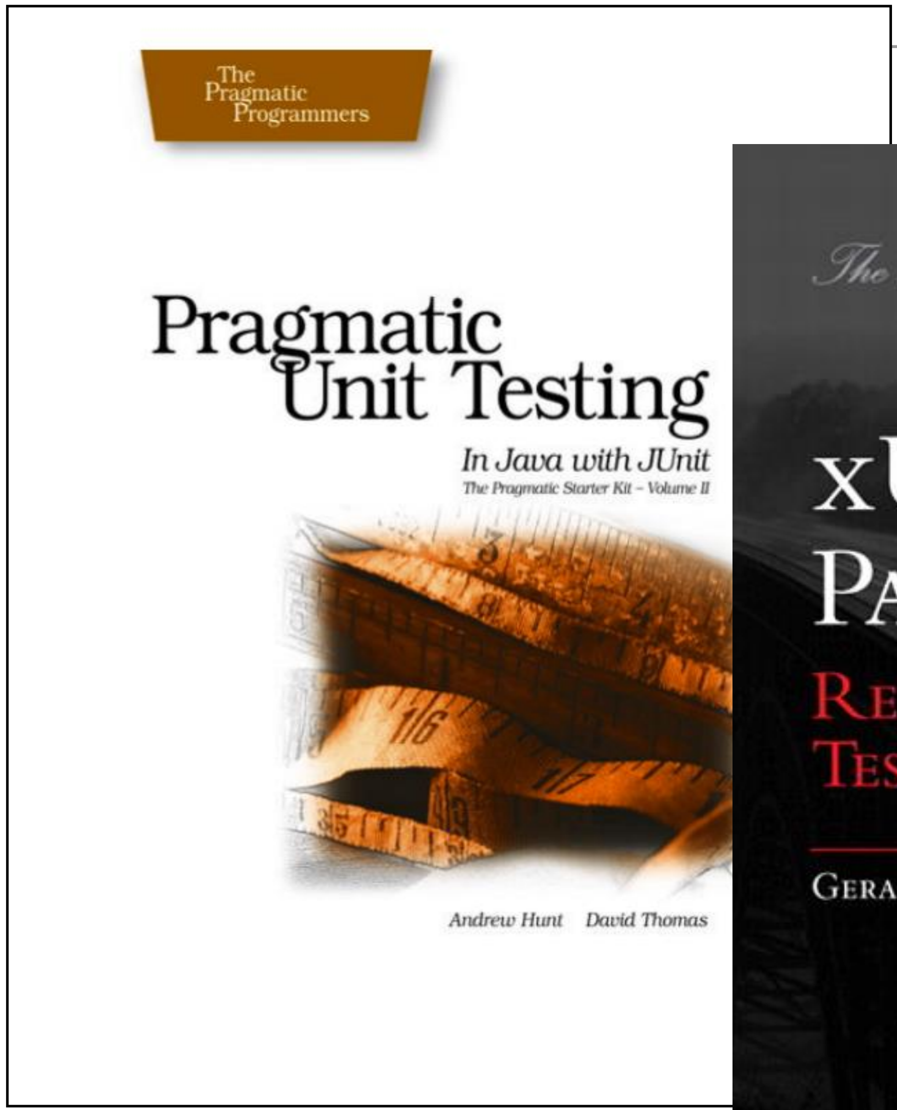
- Install PlayFramework 2.1.1 (instructions) in your dev environment.
- Clone Mocky
- Configure the app in `conf/application.conf`. The main setting is the `version` key. You have 3 different repositories where you can store your mocks:
    - v1: mocks are stored on gist. Nothing is needed, it's free, but limited to 200 requests / hours.
    - v2: mocks are stored in mongodb. Don't forget to fill the `mongodb.uri` key.
    - fs: mocks are stored on your filesystem (in the `data` dir. by default).
- If you want to launch mocky in dev mode, just launch `play run`.
- If you want to use mocky on a server, launch `play dist`, copy the generated zip to your server, unzip it and launch `./start` (easy, ain't it?)

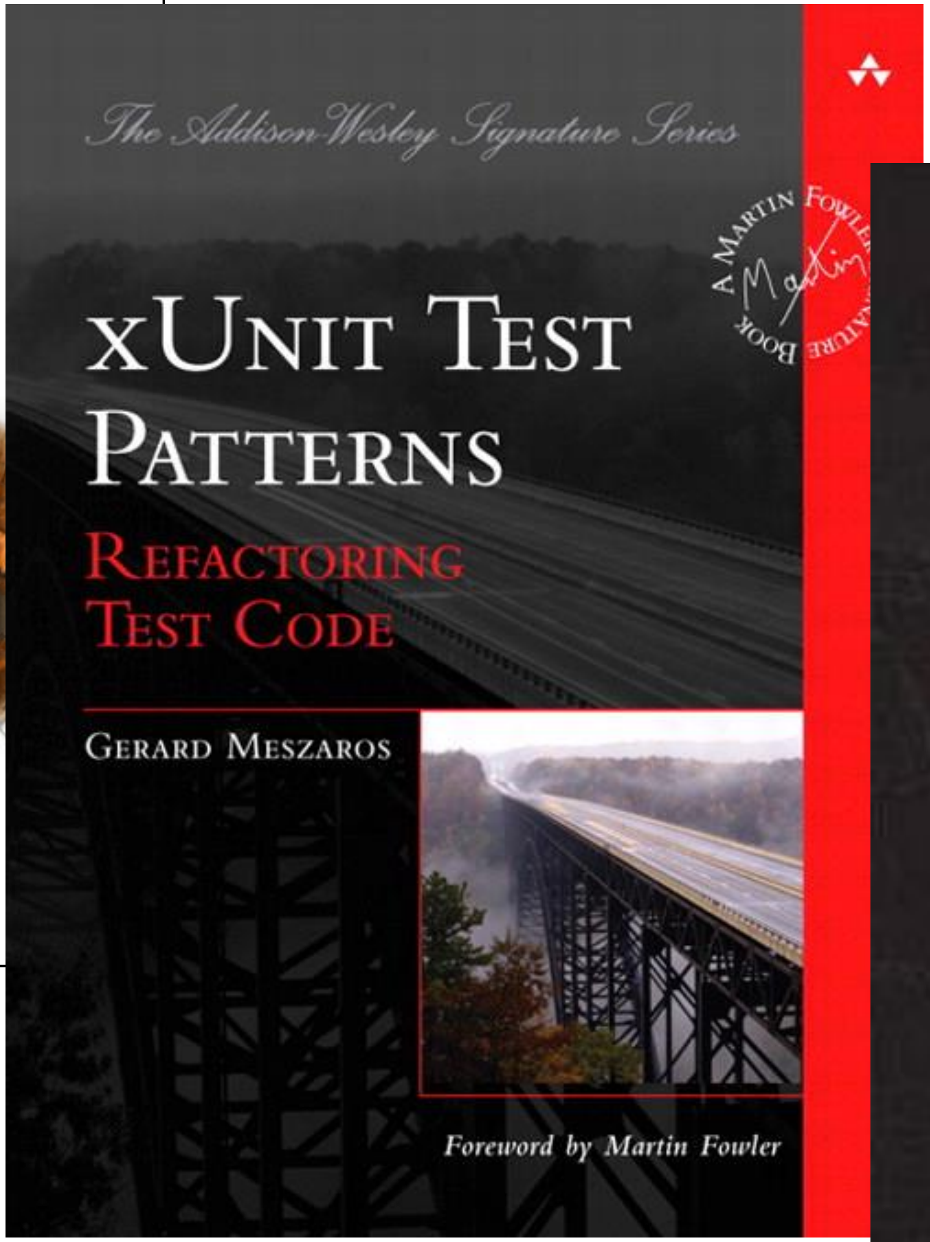
## Key Text on Employing Test Doubles Effectively

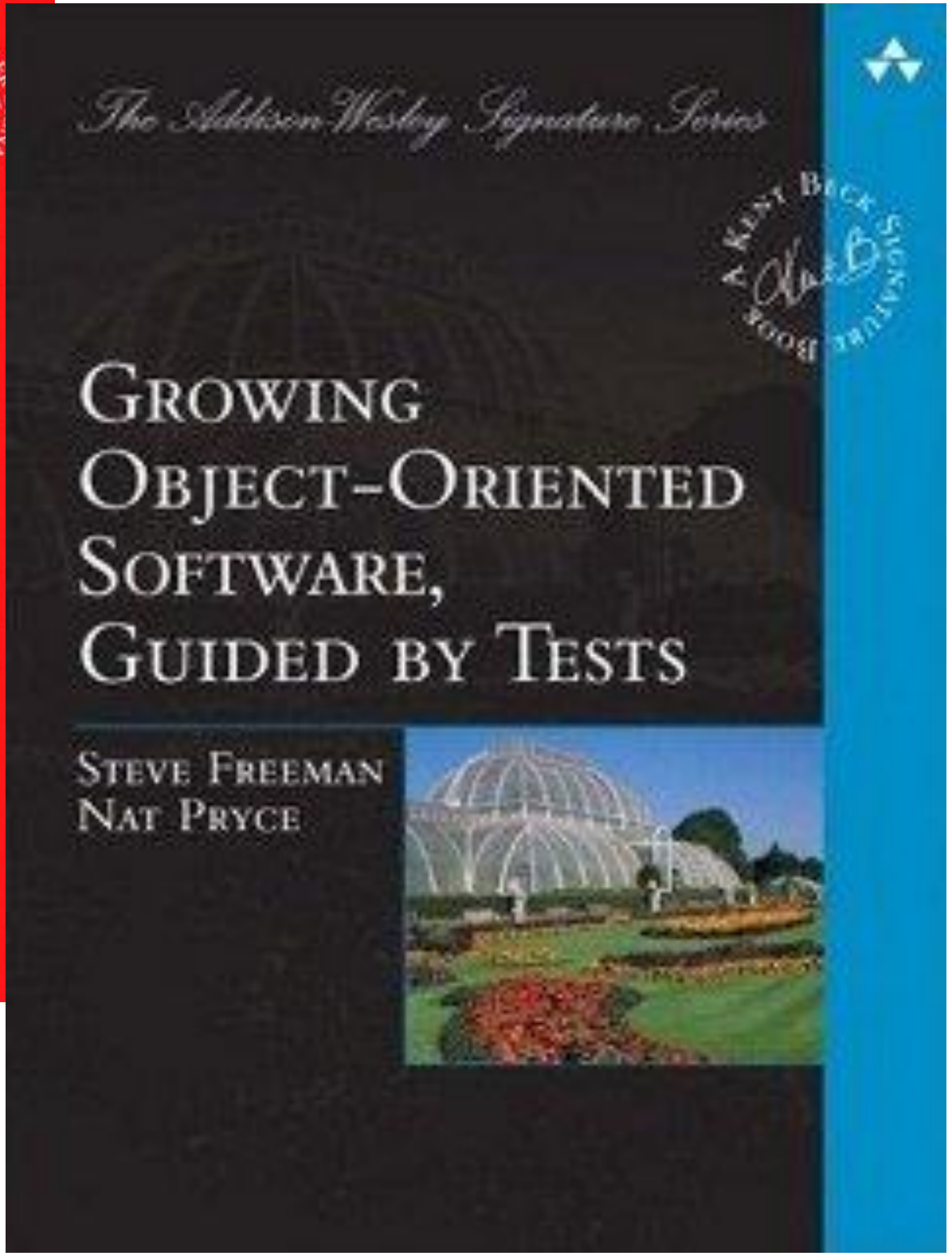
- Implementing TDD effectively: getting started, and maintaining your momentum throughout the project.
- Creating cleaner, more expressive, more sustainable code.
- Using tests to stay relentlessly focused on sustaining quality.
- Understanding how TDD, Mock Objects, and Object-Oriented Design come together in the context of a real software development project.
- Using Mock Objects to guide object-oriented designs.
- Succeeding where TDD is difficult: managing complex test data, and testing persistence and concurrency.


The Addison Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce

## TDD - Key Texts




The Addison-Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit