No API? No problem!

API mocking with WireMock

An open source workshop by …

What are we going to do?

_Stubbing, mocking and service virtualization

_WireMock

_Exercises, examples, …

Preparation

_Install JDK (Java 17 or newer)

_Install IntelliJ IDEA (or any other IDE)

_Download or clone project

_Import Maven project in IDE

Section 0:

An introduction to service virtualization

Problems in test environments

_Systems are constructed out of of many different
 components

_Not all of these components are always available
 for testing
   _Parallel development
   _No control over test data
   _Fees required for using third party components
   _…

Problems in test environments
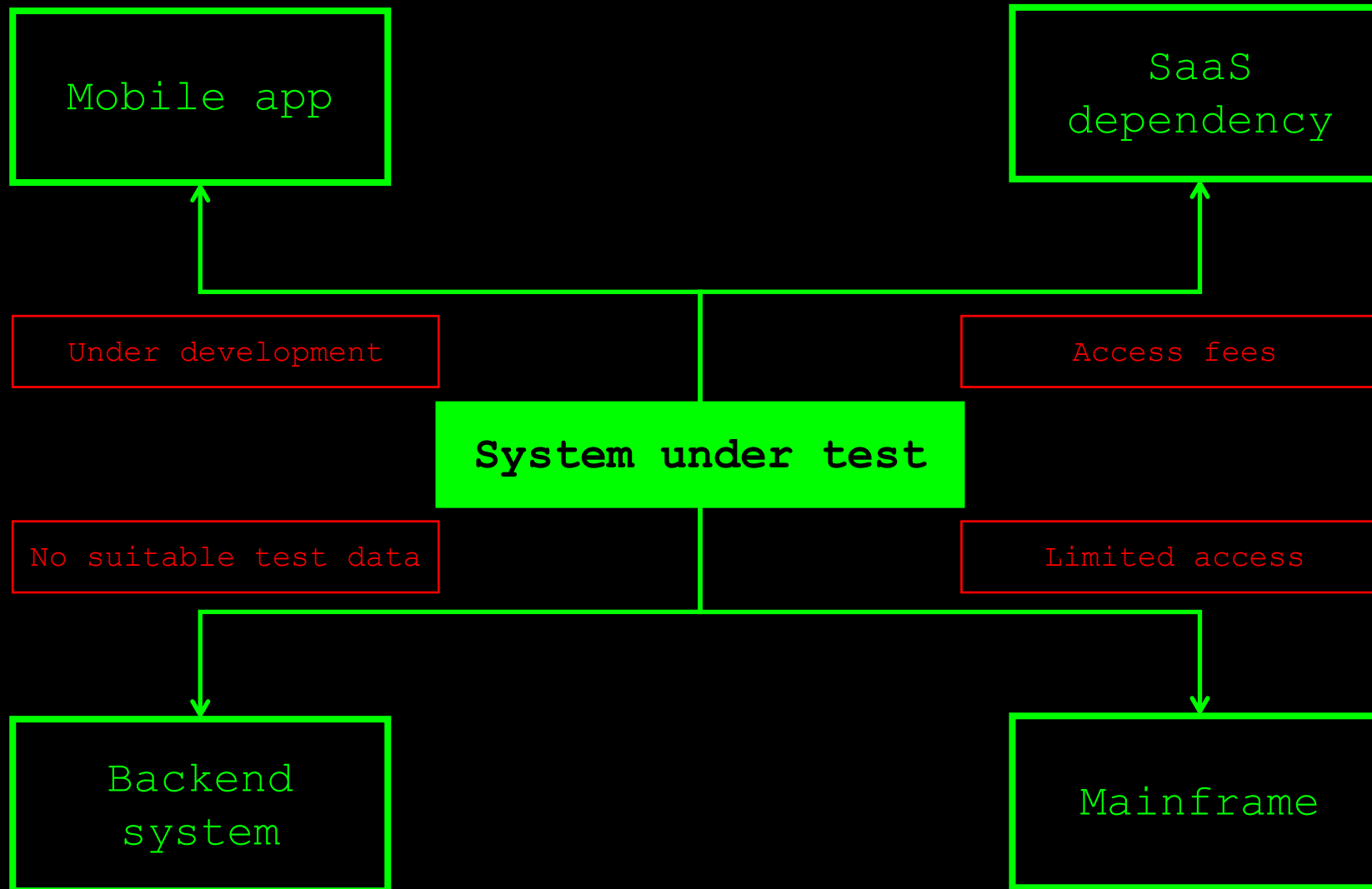
Simulation during test execution

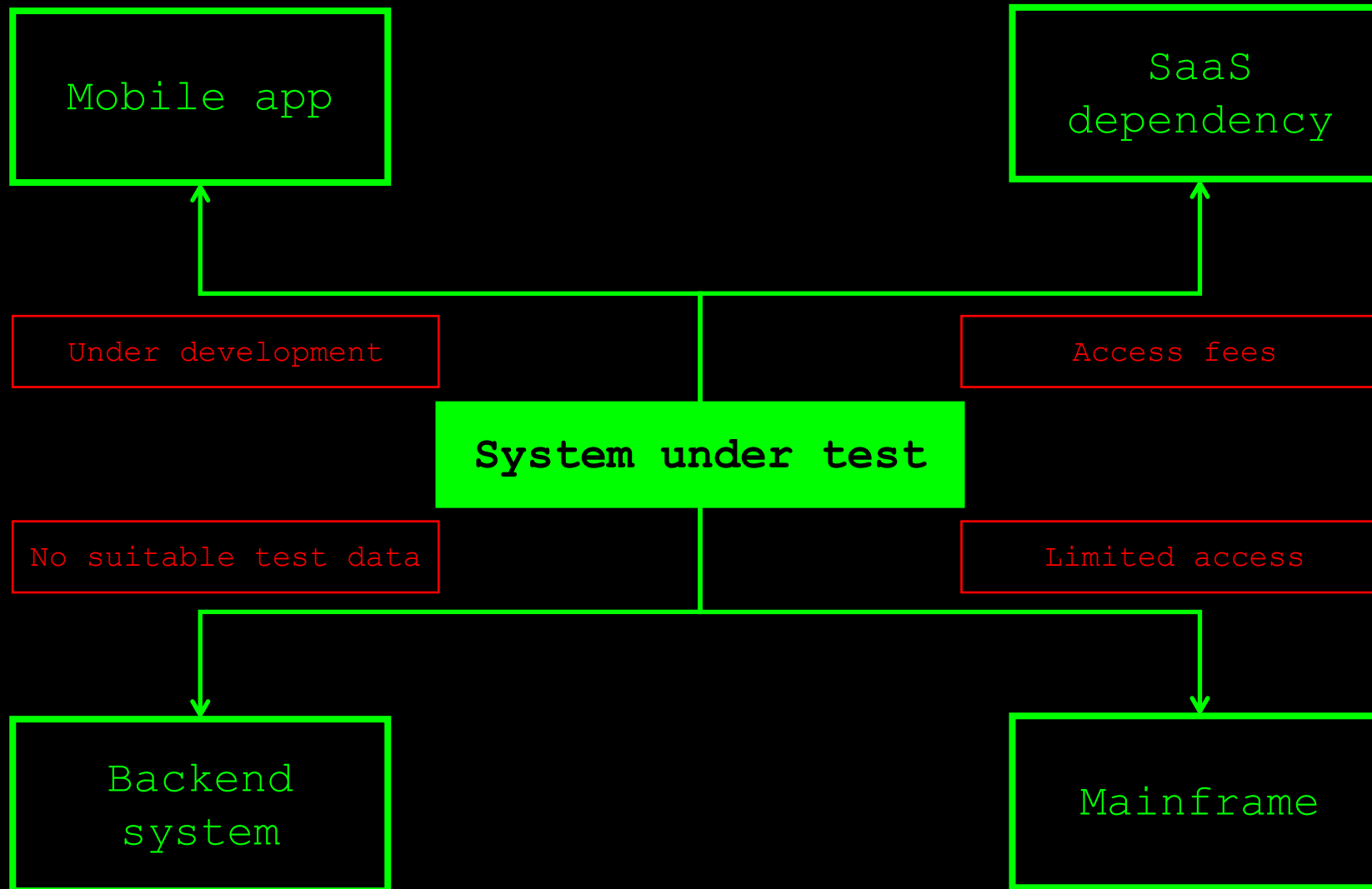_Simulate dependency **behaviour**

_Regain control over test environment
  _Dependencies available on demand
  _Control over test data (edge cases!)
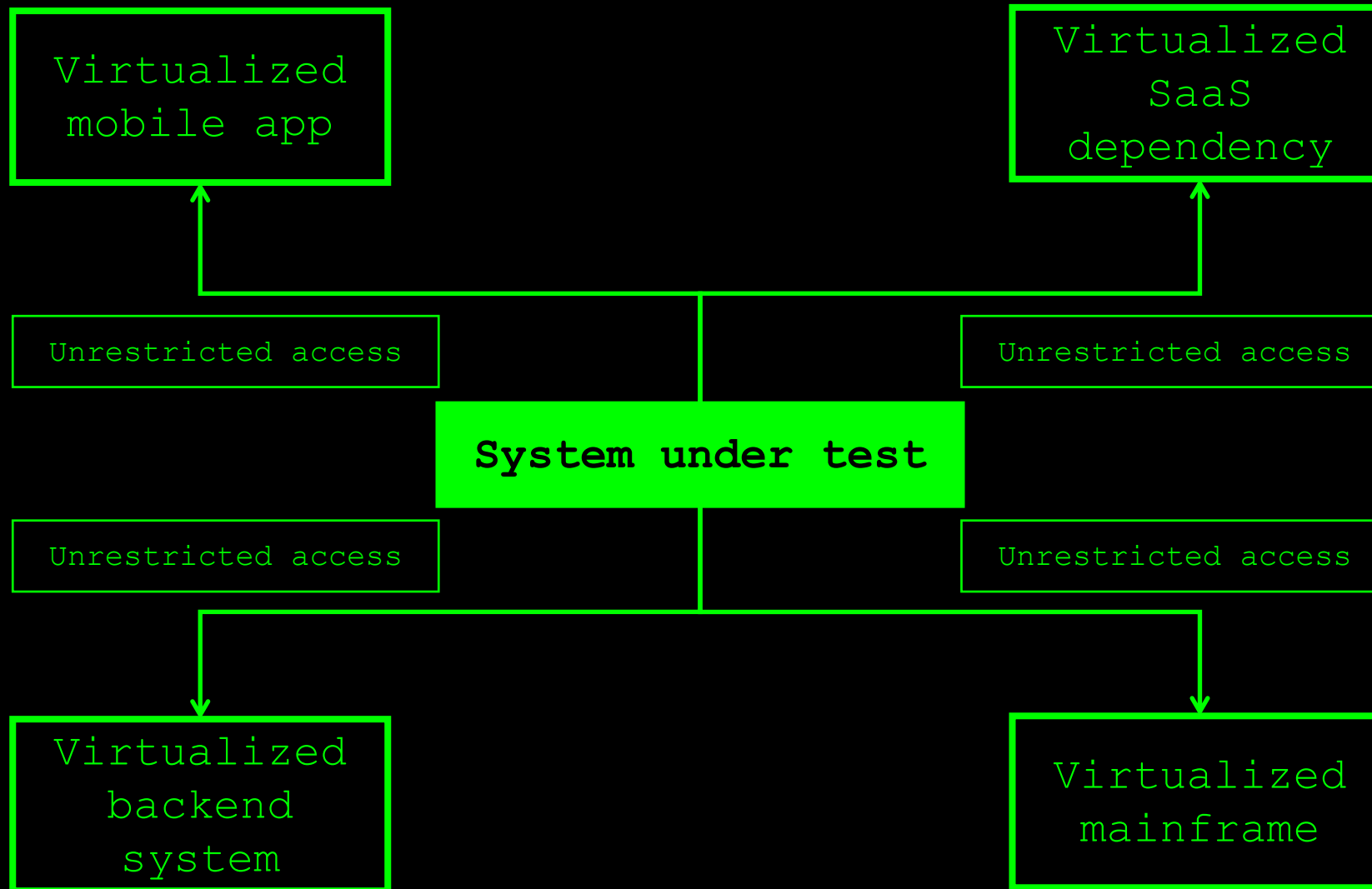 _Eliminate third party component usage fees
  _…

Problems in test environments

```
┌─────────────────┐                                    ┌─────────────────┐
│                 │                                    │      SaaS       │
│   Mobile app    │                                    │   dependency    │
│                 │                                    │                 │
└─────────────────┘                                    └─────────────────┘
        ↑                                                       ↑
        │                                                       │
        └───────────────────────────┬───────────────────────────┘
┌─────────────────────────┐         │         ┌─────────────────────────┐
│   Under development      │         │         │       Access fees       │
└─────────────────────────┘         │         └─────────────────────────┘
                          ┌─────────────────────┐
                          │  System under test  │
                          └─────────────────────┘
┌─────────────────────────┐         │         ┌─────────────────────────┐
│  No suitable test data   │         │         │     Limited access      │
└─────────────────────────┘         │         └─────────────────────────┘
        ┌───────────────────────────┴───────────────────────────┐
        │                                                       │
        ↓                                                       ↓
┌─────────────────┐                                    ┌─────────────────┐
│     Backend     │                                    │                 │
│     system      │                                    │    Mainframe    │
│                 │                                    │                 │
└─────────────────┘                                    └─────────────────┘
```

Simulation in test environments

```
Virtualized
mobile app
```

```
Virtualized
SaaS
dependency
```

| Unrestricted access | | Unrestricted access |

System under test

| Unrestricted access | | Unrestricted access |

```
Virtualized
backend
system
```
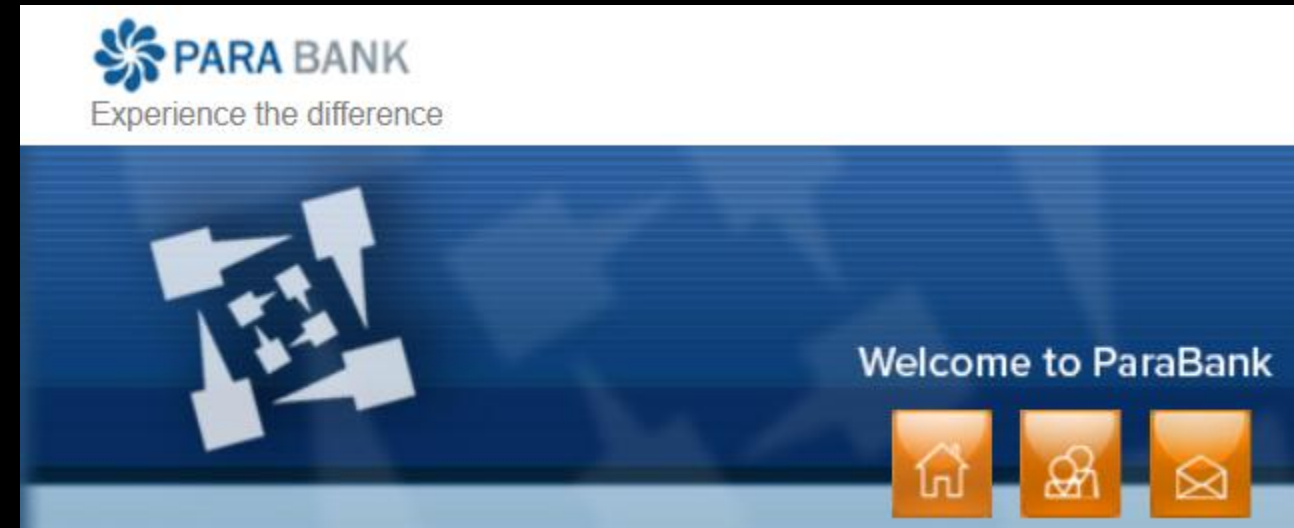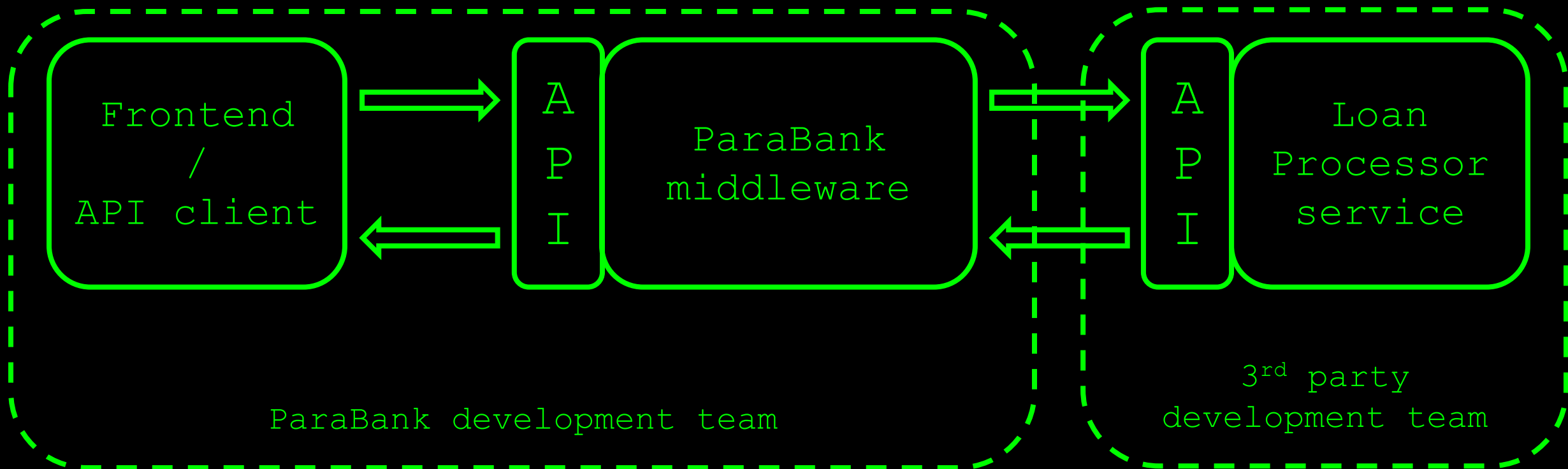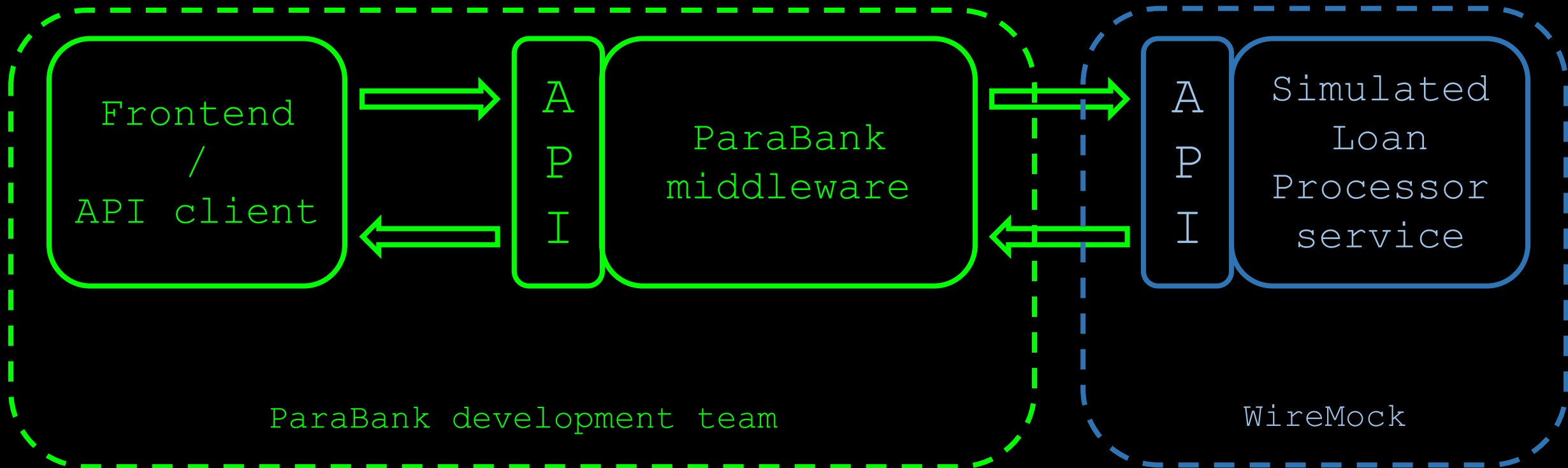
```
Virtualized
mainframe
```

Our system under test

_ParaBank

_The world's least safe online bank

_Request Loan process

_Loan application is processed by 3rd party loan provider component

Frontend / API client → API | ParaBank middleware ⇄ API | Loan Processor service

ParaBank development team

3rd party development team

Early testing against features under development

Easy setup of state for edge cases

What might we want to simulate?

Delays, fault status codes, malformatted responses, …

…

Section 1:

Getting started with WireMock

WireMock

_https://wiremock.org

_Java
 _ports and adapters are available for many other
 languages

_HTTP mock server
 _only supports HTTP(S)

_Open source

WireMock Cloud

https://www.wiremock.io

Install WireMock

_Maven

```xml
<dependency>
    <groupId>org.wiremock</groupId>
    <artifactId>wiremock</artifactId>
    <version>3.9.2</version>
    <scope>test</test>
</dependency>
```

Starting WireMock (JUnit 4)

_Via JUnit 4 @Rule

```java
@Rule
public WireMockRule wireMockRule = new WireMockRule( port: 9876);
```

_Without using JUnit 4 @Rule

```java
WireMockServer wireMockServer =
        new WireMockServer(new WireMockConfiguration().port(9876));

wireMockServer.start();
```

Starting WireMock (JUnit 5)

_Uses the JUnit 5 Jupiter extension mechanism

_Via @WireMockTest class annotation (basic configuration)

```
@WireMockTest(httpPort = 9876)
public class WireMockAnswers1Test {
```

_Using @RegisterExtension (full configuration)

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance()
        .options(wireMockConfig().port(9876).globalTemplating(true))
        .build();
```

Starting WireMock (standalone)

_Useful for exploratory testing purposes

_Allows you to share WireMock instances between
 teams

_Long-running instances

_Download the .jar first

java -jar wiremock-standalone-3.9.2.jar --port 9876

Configure responses

_In (Java) code

_Using JSON mapping files

An example mock defined in Java

```java
public void helloWorld() {

    stubFor(
        get(
            urlEqualTo(testUrl: "/helloworld")
        )

            .willReturn(
                aResponse()
                    .withHeader(key: "Content-Type", ...values: "text/plain")
                    .withStatus(200)
                    .withBody("Hello world!")));
}
```

Some useful WireMock features

_Verification
 _Verify that certain requests are sent by application under test

_Record and playback
 _Generate mocks based on request-response pairs (traffic)

_Fault simulation

_…

_Full documentation at https://wiremock.org/docs/

Now it's your turn!

_exercises > WireMockExercises1Test.java

_Create a couple of basic mocks
  _Implement the responses as described in the comments

_Verify your solution by running the tests in the same
 file

_Answers are in answers > WireMockAnswers1Test.java

_Examples are in examples > WireMockExamples1Test.java

Section 2:

Request matching strategies and fault simulation

Request matching

_Send a response only when certain properties in
the request are matched

_Options for request matching:
  _URL
  _HTTP method
  _Query parameters
  _Headers
  _Request body elements
  _…

Example: URL matching

```java
public void setupStubURLMatching() {

    stubFor(get((urlEqualTo(testUrl: "/urlmatching"))
        .willReturn(aResponse()
            .withBody("URL matching")
        ));
}
```

_Other URL options:
  _urlPathEqualTo (matches only path, no query parameters)
  _urlMatching (using regular expressions)
  _urlPathMatching (using regular expressions)

Example: header matching

```java
public void setupStubHeaderMatching() {


    stubFor(get(urlEqualTo( testUrl: "/headermatching"))
        .withHeader( s: "Content-Type", containing( value: "application/json"))
        .withHeader( s: "DoesntExist", absent())
        .willReturn(aResponse()
            .withBody("Header matching")
        ));

}
```

_absent(): check that header is **not** in request

Example: using logical AND and OR

```java
public void setupStubLogicalAndHeaderMatching() {

    stubFor(get(urlEqualTo( testUrl: "logical-or-matching"))
        .withHeader( s: "my-header",
            matching( regex: "[a-z]+").and(containing( value: "somevalue"))
        )
        .willReturn(aResponse()
            .withBody("Logical AND matching"))
    );

}
```

_ 'somevalue' is matched

_ 'bananasomevaluebanana' is matched

_ 'banana' is not matched (does not contain 'somevalue')

_ '123somevalue' is not matched (contains numeric characters)

Some more examples…

```java
public void setupStubLogicalAndHeaderMatchingMoreVerbose() {

    stubFor(get(urlEqualTo( testUrl: "logical-or-matching"))
        .withHeader( s: "my-header", and(
            matching( regex: "[a-z]+"),
            containing( value: "somevalue"))
        )
        .willReturn(aResponse()
            .withBody("Logical AND matching, a little more verbose"))
    );
}
```

Same behaviour as the previous example, using a slightly different syntax

```java
public void setupStubLogicalOrHeaderMatching() {

    stubFor(get(urlEqualTo( testUrl: "logical-or-matching"))
        .withHeader( s: "Content-Type",
            equalTo( value: "application/json").or(absent())
        )
        .willReturn(aResponse()
            .withBody("Logical OR matching"))
    );
}
```

Matching on request body elements

```java
public void setupStubRequestBodyValueMatching() {

    stubFor(post(urlEqualTo( testUrl: "/request-body-matching"))
            .withRequestBody(
                    matchingJsonPath( value: "$.fruits[?(@.banana == '2')]")
            )
            .willReturn(aResponse()
                    .withBody("Request body matched successfully"))
    );
}
```
Matching only those request bodies that have a root level element
fruits with a child element *banana* with value *2*

{"fruits": {"banana": "2", "apple": "5"} } ➔ **MATCH**

{"fruits": {"apple": "5"} } ➔ **NO MATCH**

{"fruits": {"banana": "3", "apple": "5"} } ➔ **NO MATCH**

Matching using date/time properties

```java
public void setupStubAfterSpecificDateMatching() {


    stubFor(get(urlEqualTo( testUrl: "date-is-after"))
        .withHeader( s: "my-date",
            after( dateTimeSpec: "2021-07-01T00:00:00Z")
        )
        .willReturn(aResponse()
            .withBody("Date is after midnight, July 1, 2021"))
    );

}
```

Matching all dates after
midnight of July 1, 2021

```java
public void setupStubRelativeToCurrentDateMatching() {


    stubFor(get(urlEqualTo( testUrl: "date-is-relative-to-now"))
        .withHeader( s: "my-date",
            beforeNow().expectedOffset( amount: 1, DateTimeUnit.MONTHS)
        )
        .willReturn(aResponse()
            .withBody("Date is at least 1 month before current date"))
    );

}
```

Matching all dates at least 1
month before the current date

Other matching strategies

_Authentication (Basic, OAuth(2))

_Query parameters

_Multipart/form-data

_You can write your own matching logic, too

Fault simulation

_Extend test coverage by simulating faults

_Often hard to do in real systems

_Easy to do using stubs or mocks

_Used to test the exception handling of your
 application under test

Example: HTTP status code

```java
public void setupStubReturningErrorCode() {

    stubFor(get(urlEqualTo( testUrl: "/errorcode"))
        .willReturn(aResponse()
            .withStatus(500)
            .withStatusMessage("Status message goes here")
    ));
}
```

_Some often used HTTP status codes:

Consumer error   **Provider error**

403 (Forbidden)   500 (Internal server error)

404 (Not found)   503 (Service unavailable)

Example: timeout

```java
public void setupStubFixedDelay() {

    stubFor(get(urlEqualTo( testUrl: "/fixeddelay"))
        .willReturn(aResponse()
            .withFixedDelay(2000)
    ));
}
```

_Random delay can also be used
  _Uniform, lognormal distribution

_Can be configured on a per-stub basis as well as globally

Example: bad response

```java
public void setupStubBadResponse() {

    stubFor(get(urlEqualTo( testUrl: "/badresponse"))
        .willReturn(aResponse()
            .withFault(Fault.MALFORMED_RESPONSE_CHUNK)
        ));
}
```

_HTTP status code 200, but garbage in response body

_Other options:
    _RANDOM_DATA_THEN_CLOSE (as above, without HTTP 200)
    _EMPTY_RESPONSE (does what it says on the tin)
    _CONNECTION_RESET_BY_PEER (close connection, no response)

Now it's your turn!

_exercises > WireMockExercises2Test.java

_Practice fault simulation and different request
_matching strategies
    _Implement the responses as described in the comments

_Verify your solution by running the tests in the same
_file

_Answers are in answers > WireMockAnswers2Test.java

_Examples are in examples > WireMockExamples2Test.java

Section 3:

Creating stateful mocks

Statefulness

_Sometimes, you want to simulate stateful
behaviour

_Shopping cart (empty / containing items)

_Database (data present / not present)

_Order in which requests arrive is significant

Stateful mocks in WireMock

_Supported through the concept of a Scenario

_Essentially a finite state machine (FSM)
  _States and state transitions

_Combination of current state and incoming request determines the response being sent
  _Before now, it was only the incoming request

Stateful mocks: an example

```java
public void setupStubStateful() {

    stubFor(get(urlEqualTo( testUrl: "/order")).inScenario( s: "Order processing")
        .whenScenarioStateIs(Scenario.STARTED)
        .willReturn(aResponse()
            .withBody("Your shopping cart is empty")
        )
    );

    stubFor(post(urlEqualTo( testUrl: "/order")).inScenario( s: "Order processing")
        .whenScenarioStateIs(Scenario.STARTED)
        .withRequestBody(equalTo( value: "Ordering 1 item"))
        .willReturn(aResponse()
            .withBody("Item placed in shopping cart")
        )
        .willSetStateTo("ORDER_PLACED")
    );

    stubFor(get(urlEqualTo( testUrl: "/order")).inScenario( s: "Order processing")
        .whenScenarioStateIs("ORDER_PLACED")
        .willReturn(aResponse()
            .withBody("There is 1 item in your shopping cart")
        )
    );
}
```

Responses are grouped by scenario name

Response depends on both the incoming request as well as the current state

The initial state should always be Scenario.STARTED

Incoming requests can trigger state transitions

State names other than Scenario.STARTED are yours to define

Now it's your turn!

_exercises > WireMockExercises3Test.java

_Create a stateful mock that exerts the described
_behaviour
  _Implement the responses as described in the comments

_Verify your solution by running the tests in the same
_file

_Answers are in answers > WireMockAnswers3Test.java

_Examples are in examples > WireMockExamples3Test.java

Section 4:

Response templating

Response templating

_Often, you want to reuse elements from the request in the response

  _Request ID header

  _Unique body elements (client ID, etc.)

  _Cookie values

_WireMock supports this through response templating

Setup response templating (JUnit 4)

_In code: through the JUnit @Rule

```java
@Rule
public WireMockRule wireMockRule =
    new WireMockRule(wireMockConfig().
        port(9876).
        extensions(new ResponseTemplateTransformer( global: true))
    );
```

_Global == false: response templating transformer has to be enabled for individual stubs

Setup response templating (JUnit 5)

_In code: through the JUnit @RegisterExtension

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance()
        .options(wireMockConfig().port(9876).globalTemplating(true))
        .build();
```

_Argument == false: response templating has to be
 enabled for individual stubs

Enable/apply response templating

_This template reads the HTTP request method (GET/POST/PUT/…) using *{{request.method}}* and returns it as the response body

```java
public void setupStubResponseTemplatingHttpMethod() {

    wiremock.stubFor(any(urlEqualTo( testUrl: "/template-http-method"))
        .willReturn(aResponse()
            .withBody("You used an HTTP {{request.method}}")
            .withTransformers("response-template")
    ));
}
```

This call to *withTransformers()* is only necessary when response templating isn't activated globally

One thing to keep in mind…

```java
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance()
        .options(wireMockConfig().port(9876).globalTemplating(true))
        .build();
```

Because we're explicitly initializing
a WireMock instance here…

```java
public void setupStubResponseTemplatingHttpMethod() {

    wiremock.stubFor(any(urlEqualTo( testUrl: "/template-http-method"))
        .willReturn(aResponse()
            .withBody("You used an HTTP {{request.method}}")
            .withTransformers("response-template")
    ));
}
```

… we need to explicitly assign our stub definition to that instance
here, or else the stub definition will not be picked up!

Request attributes

_Many different request attributes available for use

  _request.method   : HTTP method (example)

  _request.pathSegments.[<n>]   : n^{th} path segment

  _request.headers.<key>   : header with name *key*

  _…

_All available attributes listed at

https://wiremock.org/docs/response-templating/

Request attributes (cont'd)

_Extracting and reusing body elements

_In case of a JSON request body:

{{jsonPath request.body '$.path.to.element'}}

_In case of an XML request body:

{{xPath request.body '/path/to/element/text()'}}

JSON extraction example

_When sent this JSON request body:

```json
{
    "book": {
        "author": "Ken Follett",
        "title": "Pillars of the Earth",
        "published": 2002
    }
}
```

_This stub returns a response with body "Pillars of the Earth":

```java
public void setupStubResponseTemplatingJsonBody() {

    stubFor(post(urlEqualTo( testUrl: "/template-json-body"))
            .willReturn(aResponse().
                    withBody("{{jsonPath request.body '$.book.title'}}")
                    .withTransformers("response-template")
    ));
```

Again, this call to *withTransformers()* is only necessary when response templating isn't activated globally

Now it's your turn!

_exercises > WireMockExercises4Test.java

_Create mocks that use response templating
  _Implement the responses as described in the comments

_Verify your solution by running the tests in the same
_file

_Answers are in answers > WireMockAnswers4Test.java

_Examples are in examples > WireMockExamples4Test.java

Section 5: Verification

Verifying incoming requests

_Apart from returning responses, you might also want to verify that incoming requests have certain properties

  _Fail a test if these verifications aren't met

_You can do this with WireMock in a way very similar to mocking frameworks for unit tests (e.g., Mockito for Java)

Verifying incoming requests

```java
public void setupHelloWorldStub() {

    stubFor(
        get(
            urlEqualTo( testUrl: "/hello-world")
        )
            .willReturn(
                aResponse()
                    .withHeader( key: "Content-Type", …values: "text/plain")
                    .withStatus(200)
                    .withBody("Hello world!"))
    );
}
```

Given this simple 'hello world' stub

```java
@Test
public void helloWorldVerificationTest() {

    setupHelloWorldStub();

    given().
            spec(requestSpec).
    when().
            get( s: "/hello-world").
    then().

    verify(exactly( expected: 1), getRequestedFor(urlEqualTo( testUrl: "/hello-world")));

    and().
            body(org.hamcrest.Matchers.equalTo( operand: "Hello world!"));
```

When we have this test that should invoke that stub exactly once

Then this verification can be added to the test to ensure that indeed, an HTTP GET to '/hello-world' has been made exactly once

```
verify(exactly( expected: 1), getRequestedFor(urlEqualTo( testUrl: "/hello-world")));
```

Some more verification examples

```
verify(getRequestedFor(urlEqualTo( testUrl: "/hello-world")));
```
The same as the above, but less verbose

```
verify(lessThan( expected: 5), postRequestedFor(urlEqualTo( testUrl: "/requestLoan")));
```
Verify that less than 5 HTTP POSTs were made to /requestLoan

```
verify(
        moreThanOrExactly( expected: 10),
        postRequestedFor(urlEqualTo( testUrl: "/requestLoan"))
                .withHeader( key: "Content-Type", containing( value: "application/json"))
);
```
Verify that 10 or more HTTP POSTs with a 'Content-Type' header value containing 'application/json' were made to /requestLoan

Now it's your turn!

_exercises > WireMockExercises5Test.java

_Add WireMock verifications to the tests
   _Verify request properties as described in the comments

_Verify your solution by running the tests

_Answers are in answers > WireMockAnswers5Test.java

_Examples are in examples > WireMockExamples5Test.java

Section 6:

Stub priority

Defining stub priority

_In some cases, you might want to tell WireMock to
first evaluate an incoming request against one
stub definition, before 'passing it down' to
others

_You can enforce this with defining stub priority

_1 is highest priority, Integer.MAX_VALUE is
lowest priority

Stub priority - example

```
public void setupPizzaIngredientsStub() {  1 usage  Bas Dijkstra

    stubFor(get(urlEqualTo( testUrl: "/pizza/ingredients/pineapple"))
        .atPriority( integer: 1)
        .willReturn(
            aResponse()
                .withHeader( key: "Content-Type", ...values: "text/plain")
                .withStatus(400)
                .withBody("No. Just no."))
    );


    stubFor(get(urlMatching( urlRegex: "/pizza/ingredients/(.*)"))
        .atPriority( integer: 2)
        .willReturn(
            aResponse()
                .withHeader( key: "Content-Type", ...values: "text/plain")
                .withStatus(200)
                .withBody("That's fine."))
    );
}
```

WireMock will match the incoming request with this stub definition first…

… before moving on to matching it against this stub definition

Now it's your turn!

_exercises > WireMockExercises6Test.java

_Define two stub definitions and assign the required
 priority to both of them

_Verify your solution by running the tests

_Answers are in answers > WireMockAnswers6Test.java

_Examples are in examples > WireMockExamples6Test.java

Section 7:

Extending WireMock

Extending WireMock

_In some cases, the default WireMock feature set
might not fit your needs

_WireMock is open to extensions

_Allows you to create even more powerful stubs

_Several options available

Section 7.1:

Filtering incoming requests

Request filtering

_Modify incoming requests (or halt processing)

_This has a variety of use cases:
  _Checking authentication details
  _Request header injection
  _URL rewriting

_Created by implementing the *StubRequestFilterV2* interface

Request filtering - build

```java
public class HttpDeleteFilter implements StubRequestFilterV2 {

    @Override
    public RequestFilterAction filter(Request request, ServeEvent serveEvent) {

        if (request.getMethod().equals(RequestMethod.DELETE)) {

            return RequestFilterAction.stopWith(ResponseDefinition.notPermitted("HTTP DELETE is not allowed!"));
        }

        return RequestFilterAction.continueWith(request);
    }

    @Override
    public String getName() { return "http-delete-filter"; }
}
```

If the HTTP verb used equals DELETE…

Return an HTTP 403 and stop processing the request

Else continue processing the request

Request filtering - use

```java
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance().
    options(wireMockConfig().
        port(9876).
        extensions(new HttpDeleteFilter())
    ).build();
```

An extension can be registered using:
- its class name ("*com.example.HttpDeleteFilter*")
- the class (*HttpDeleteFilter.class*)
- an instance (*new HttpDeleteFilter()*)

Now it's your turn!

_exercises > extensions > BasicAuthFilter.java

_Implement a custom request filter that filters out all requests that do not have the proper basic authentication credentials

_Verify your solution by running the tests in exercises > WireMockExercises7dot1Test.java

_Answers are in answers > extensions > BasicAuthFilter.java

_Examples are in examples > extensions > HttpDeleteFilter.java

Section 7.2:

Building a custom request matcher

Custom request matchers

_Add custom request matching logic to WireMock

_Can be combined with existing standard matchers

_Done by extending RequestMatcherExtension class

Custom request matcher - build

```java
public class BodyLengthMatcher extends RequestMatcherExtension {

    @Override
    public String getName() {
        return "body-too-long";
    }


    @Override
    public MatchResult match(Request request, Parameters parameters) {
        int maxLength = parameters.getInt( key: "maxLength");
        return MatchResult.of(request.getBody().length > maxLength);
    }
}
```

Get the value of the *maxLength* matcher parameter

Compare the request body length to the *maxLength* parameter value and return the result as a *MatchResult*

Custom request matcher - use

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance().
        options(wireMockConfig().
                port(9876).
                extensions(new BodyLengthMatcher())
        ).build();
```

Register the extension

Use custom matcher in a
stub definition using its
name (can be combined
with existing matchers)

Specify desired parameter value

```
stubFor(get(urlEqualTo( testUrl: "/custom-matching")).
    andMatching( s "body-too-long", Parameters.one( name: "maxLength", value: 20)).
    willReturn(aResponse().withStatus(400))
);
```

Now it's your turn!

- exercises > extensions > RejectedHttpVerbsMatcher.java

- Implement a custom matcher that reads a list of rejected HTTP verbs and matches the HTTP verb used in the incoming request against it

- Verify your solution by running the tests in exercises > WireMockExercises7dot2Test.java

- Answers are in answers > extensions > RejectedHttpVerbsMatcher.java

- Examples are in examples > extensions > BodyLengthMatcher.java

Section 7.3:

Using ServeEvent
listeners

ServeEvents

_Perform specific actions before or after
 processing or serving response

_Logging, writing to database, …

_Extend ServeEventListener class

ServeEvent listener - build

```java
public class DatabaseWriter implements ServeEventListener {  2 usages    Bas Dijkstra

    @Override     Bas Dijkstra
    public String getName() {
        return "database-writer";
    }


    @Override   no usages    Bas Dijkstra
    public boolean applyGlobally() {
        return false;
    }

    @Override   no usages    Bas Dijkstra
    public void afterComplete(ServeEvent serveEvent, Parameters parameters) {


        String database = parameters.getString( key: "database");


        System.out.println("Writing to database: " + database);

    }
}
```

This implements the action to execute
after serving a response has completed

ServeEvent listener - use

```java
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance().
    options(wireMockConfig().
        port( portNumber: 9876).
        extensions(new DatabaseWriter())
    ).build();
```

Register the extension

```java
public void stubForServeEventListener() {  1 usage   ▲ Bas Dijkstra

    Map<String, Object> params = new HashMap<>();
    params.put("database", "requestsDB");

    wiremock.stubFor(get(urlEqualTo( testUrl: "/serve-event")).
        withServeEventListener( s: "database-writer", Parameters.from(params)).
        willReturn(aResponse().withStatus(200))
    );
}
```

Add the ServeEvent
listener to the stub
definition and supply the
desired parameter value

Now it's your turn!

_exercises > extensions > LogRequestWithTimestamp.java

_Implement a ServeEvent listener that prints a log message containing the current date and time in the requested format to the console

_Configure and call the listener in exercises > WireMockExercises7dot3Test.java

_Answers are in answers > extensions > LogRequestWithTimestamp.java

_Examples are in examples > extensions > DatabaseWriter.java

https://wiremock.org/docs
/extending-wiremock/

Appendix A:

JSON equivalents for the Java examples

Our Hello world! mock

```json
{
    "request": {
        "method": "GET",
        "url": "/helloworld"
    },
    "response": {
        "status": 200,
        "body": "Hello world!",
        "headers": {
            "Content-Type": "text/plain"
        }
    }
}
```

URL matching

```json
{
    "request": {
        "method": "GET",
        "url": "/urlmatching"
    },
    "response": {
        "status": 200,
        "body": "URL matching"
    }
}
```

Request header matching

```json
{
    "request": {
        "method": "GET",
        "headers": {
            "headerName": {
                "equalTo": "headerValue"
            }
        }
    },
    "response": {
        "status": 200,
        "body": "Header matching"
    }
}
```

Simulating a delay

```json
{
    "request": {
        "method": "GET",
        "url": "/fixeddelay"
    },
    "response": {
        "status": 200,
        "fixedDelayMilliseconds": 2000
    }
}
```

Returning a fault response

```json
{
    "request": {
        "method": "GET",
        "url": "/badresponse"
    },
    "response": {
        "fault": "MALFORMED_RESPONSE_CHUNK"
    }
}
```

Creating a stateful mock

```
{
  "mappings": [
    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "Started",
      "request": {
        "method": "GET",
        "url": "/order"
      },
      "response": {
        "status": 200,
        "body" : "Your shopping cart is empty"
      }
    },

    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "Started",
      "newScenarioState": "ORDER_PLACED",
      "request": {
        "method": "POST",
        "url": "/order",
        "bodyPatterns": [
          { "equalTo": "Ordering 1 item" }
        ]
      },
      "response": {
        "status": 200,
```

```
      "response": {
        "status": 200,
        "body": "Item placed in shopping cart"
      }
    },

    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "ORDER_PLACED",
      "request": {
        "method": "GET",
        "url": "/order"
      },
      "response": {
        "status": 200,
        "body" : "There is 1 item in your shopping cart"
      }
    }

  ]
}
```

Use response templating

```json
{
    "request": {
        "url": "/template-http-method"
    },
    "response": {
        "status": 200,
        "body": "You used an HTTP {{request.method}}",
        "transformers": ["response-template"]
    }
}
```

Use response templating

_When sent this JSON
request body:

```json
{
    "book": {
        "author": "Ken Follett",
        "title": "Pillars of the Earth",
        "published": 2002
    }
}
```

_This stub returns a response with body "Pillars of
the Earth":

```json
{
  "request": {
    "method": "POST",
    "urlPath": "/template-json-body"
  },
  "response": {
    "body": "{{jsonPath request.body '$.book.title'}}",
    "transformers": ["response-template"]
  }
}
```

Using WireMock extensions

```json
{
    "request" : {
        "customMatcher" : {
            "name" : "body-too-long",
            "parameters" : {
                "maxLength" : 2048
            }
        }
    },
    "response" : {
        "status" : 422
    }
}
```

Using a custom matcher

```json
{
    "request": {
        "method": "GET",
        "url": "/local-transform"
    },
    "response": {
        "status": 200,
        "body": "Original body",
        "transformers": ["my-transformer", "other-transformer"]
    }
}
```

Registering a local transformer

Specifying transformer parameters

```json
{
    "request" : {
        "url" : "/transform",
        "method" : "GET"
    },
    "response" : {
        "status" : 200,
        "transformerParameters" : {
            "paramName" : "value"
        }
    }
}
```