DevNQ

August 2019 Session


JAMES COOK
UNIVERSITY
AUSTRALIA


JESI

Please no Flash Photography

Developers are afraid of the light

These slides and example code snippets are publicly available and will be linked in the Meetup event listing.

Travel vector created by Layerace - https://freepik.com/free-photos-vectors/travel

Why Test?

Performance

Reduce Bugs

Regression

Quality

Stress

Confidence

Validation

Reliability

But...

“Testing is too hard. I’ll consider it later.”

“Testing is too hard. I’ll consider it later.”

- Improves your algorithm and implementation
- Promotes clean code
- Pushes dependencies to edges
- Developer documentation
- Confidence

“I can run the tests manually before I deploy”

“I can run the tests manually before I deploy”

- Time consuming
- Cannot possibly test every use case
- Multiple environments

“Tests are not production code.
It doesn’t affect my users.”

“Tests are not production code.
It doesn’t affect my users.”

- BURP
  - Bugs
  - Usability
  - Regression
  - Promise

- Testing is part of the feature

- Paying users = Your Salary

When should we Test?

It Depends.

Every situation and platform is different

Hobby? Prototype? Core Product? Tooling?

F.I.R.S.T Principles of Unit Testing

Fast

- Run and complete in milliseconds
- Assist in a rapid development and feedback cycle

Fast

Isolated

- Order Independent
- Arrange their own data and setup
- Invoke only what needs to be invoked
- Assert only what needs to be asserted

Fast

Isolated

Repeatable

- Yield the same result every time

Fast

Isolated

Repeatable

Self-Validating

- The tests should tell you if they have failed

Fast

Isolated

Repeatable

Self-Validating

Thorough

- Business rules
- Edge cases
- Large and Illegal values
- Security or Vulnerability issues
- Aim for 100% scenario, not necessarily 100% coverage

Anatomy of a Unit Test

```
public class DevNqEvent {

    public int add(final int left, final int right) {

        return left + right;

    }
}
```

```
public class DevNqEventTest {

    @Test
    @DisplayName("Returns the addition of two integers")
    void addTest() {









    }
}
```

Anatomy of a Unit Test

```
public class DevNqEvent {

    public int add(final int left, final int right) {

        return left + right;

    }
}
```

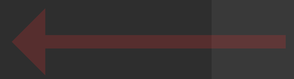
```
public class DevNqEventTest {

    @Test
    @DisplayName("Returns the addition of two integers")
    void addTest() {

        final DevNqEvent event = new DevNqEvent();

    }
}
```

Arrange

Anatomy of a Unit Test

```
public class DevNqEvent {

    public int add(final int left, final int right) {

        return left + right;

    }
}
```

```
public class DevNqEventTest {

    @Test
    @DisplayName("Returns the addition of two integers")
    void addTest() {

        final DevNqEvent event = new DevNqEvent();

        final int actual = event.add(1, 4);


    }
}
```



Arrange

Act

Anatomy of a Unit Test

```java
public class DevNqEvent {

    public int add(final int left, final int right) {

        return left + right;

    }
}
```

```java
public class DevNqEventTest {

    @Test
    @DisplayName("Returns the addition of two integers")
    void addTest() {

        final DevNqEvent event = new DevNqEvent();

        final int actual = event.add(1, 4);

        assertThat(actual).isEqualTo(5);
    }
}
```

Arrange

Act

Assert

Software Testing is like Onions

They stink?

Yea..NO!

Ohh they make you cry?

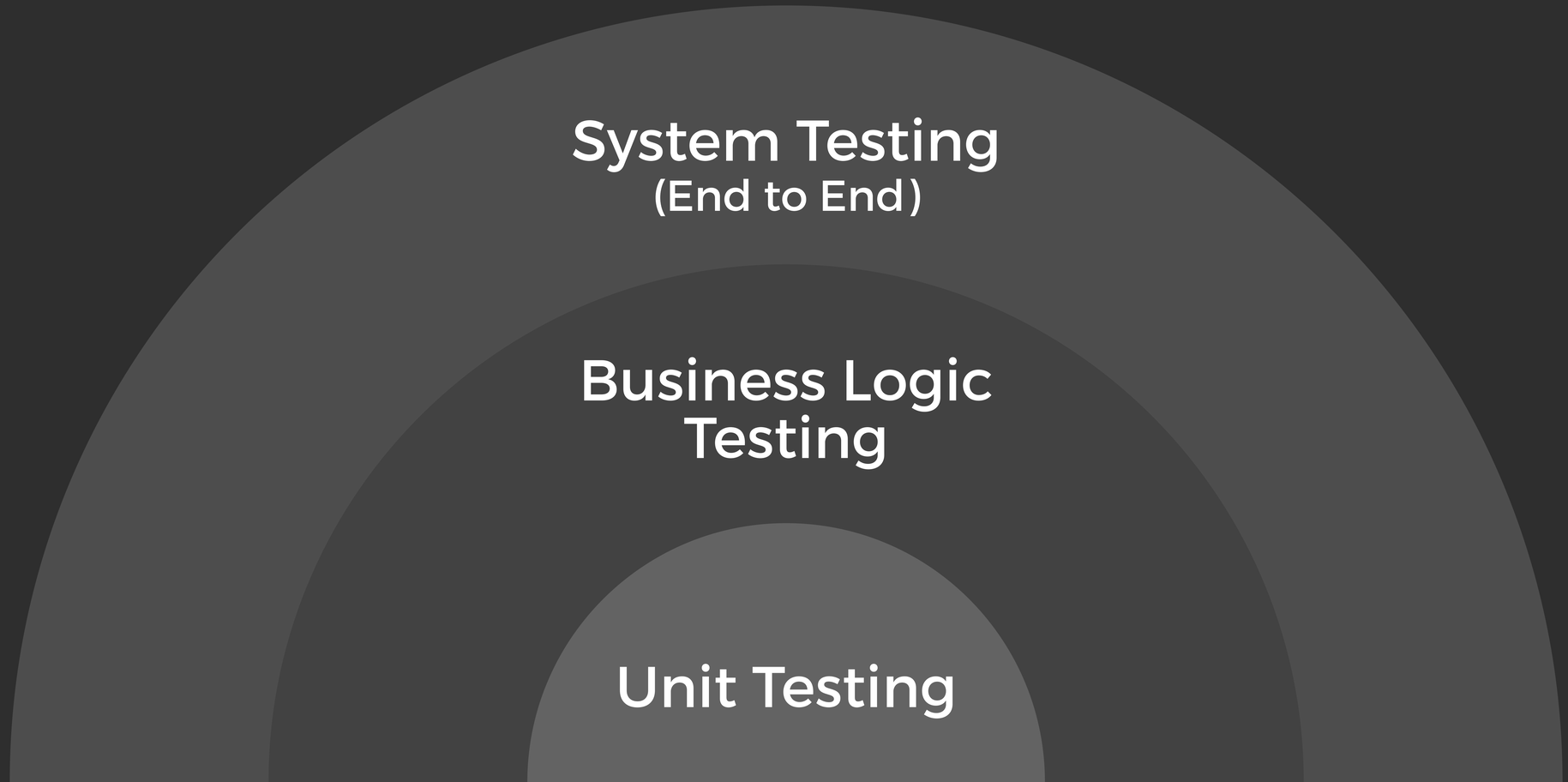
Noooo....(well sometimes)

You leave them in your code base and they start growing little bug features?

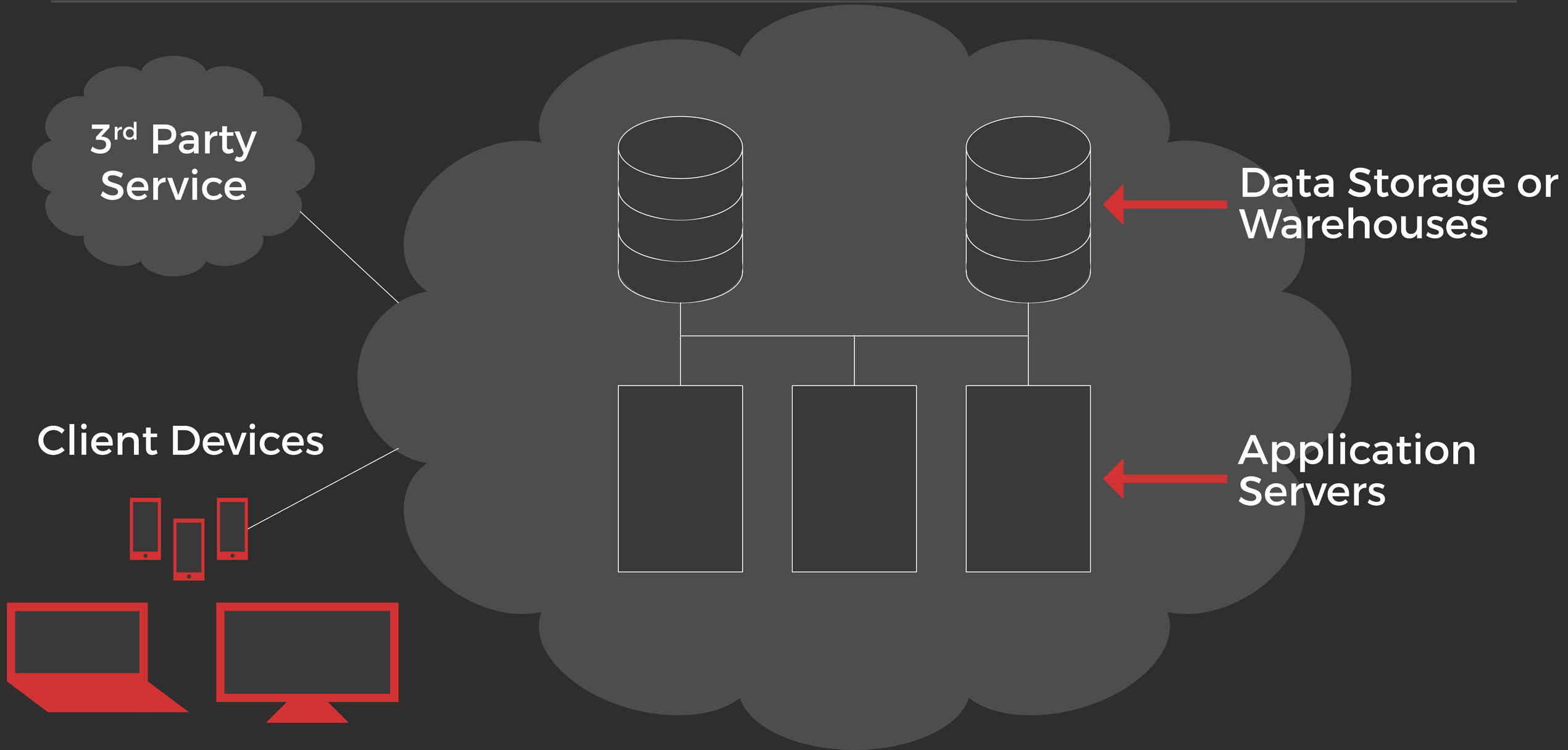
No! LAYERS! Software Testing has Layers!

Unit Testing

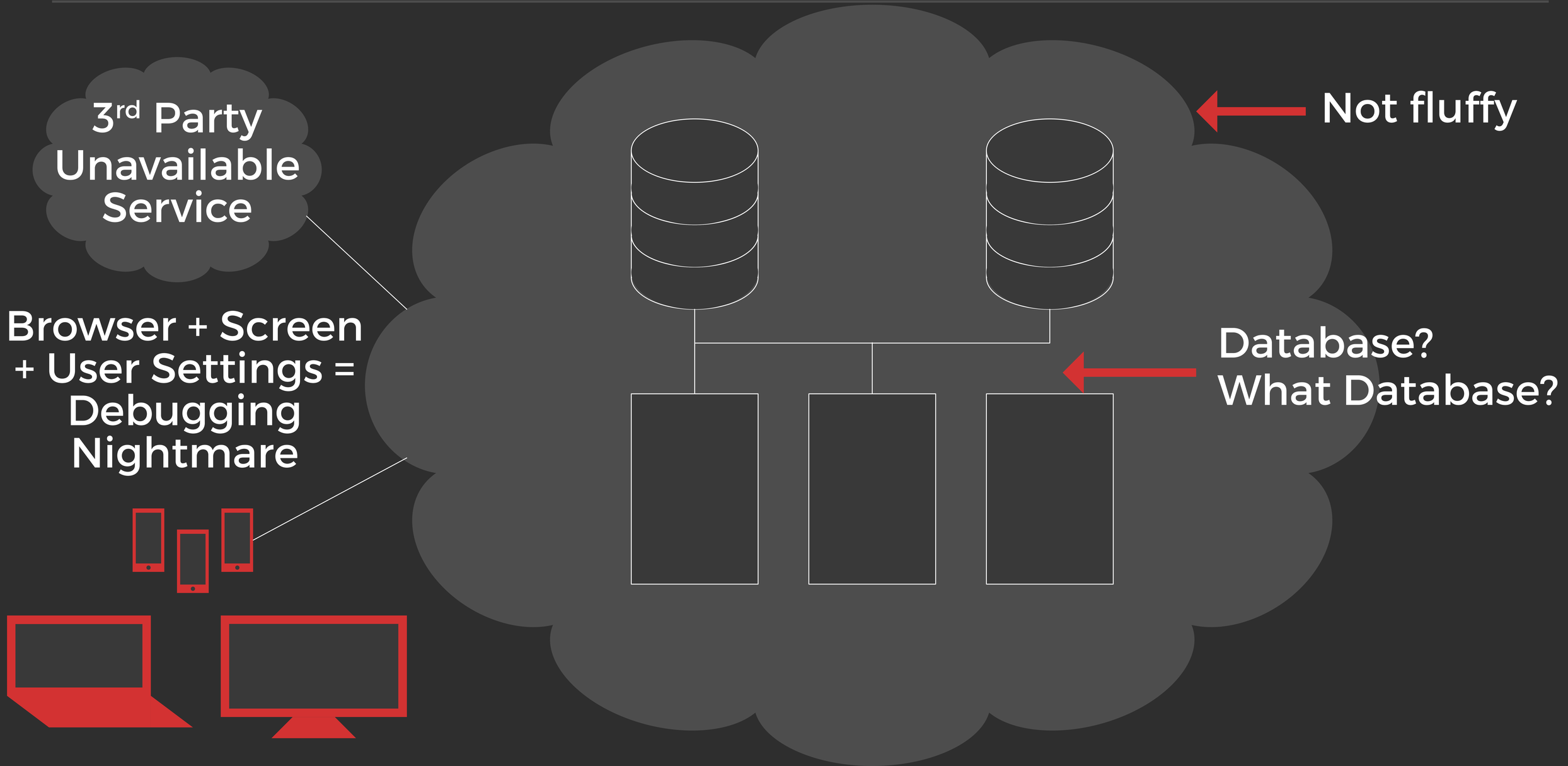

Business Logic
Testing
Unit Testing


System Testing
(End to End)
Business Logic
Testing
Unit Testing

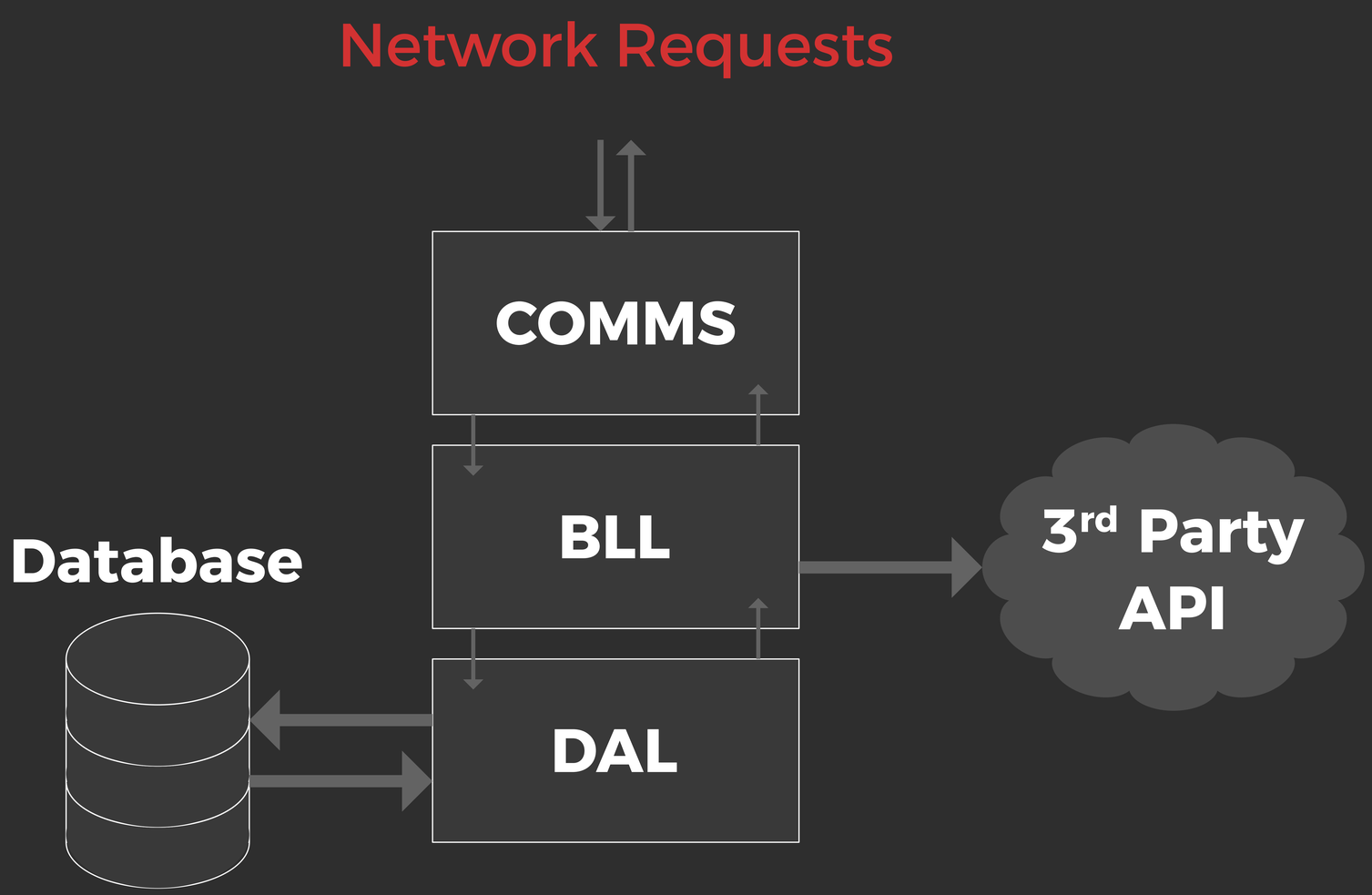
Basic SaaS Architecture



Reality of SaaS Architecture



API/Server Stack



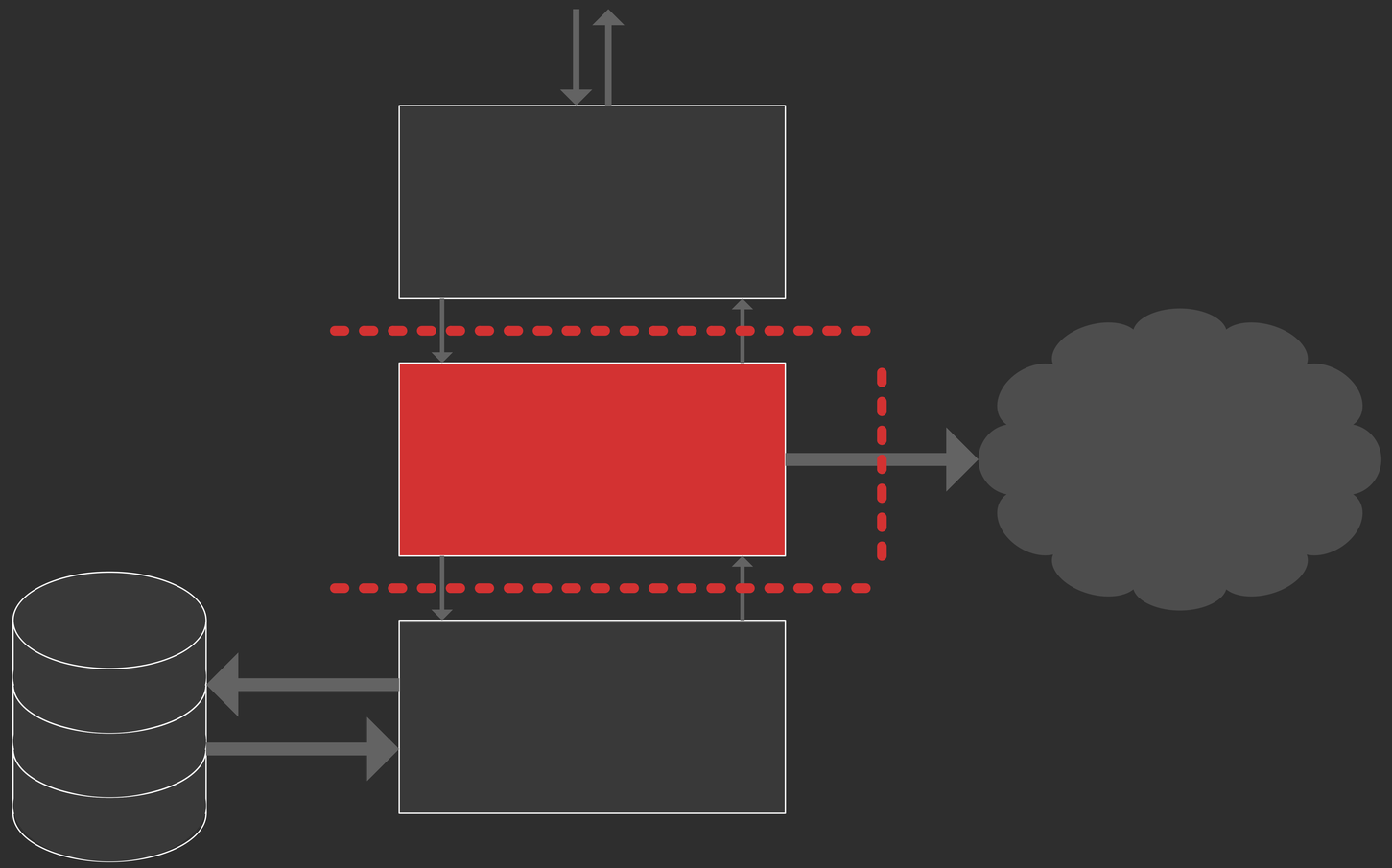
Question

We have developed a middleware function that extracts an authorised User from a request context (object that keeps track of the request-level data). The function is called from, and used, by a 3rd party library.

How do we test our code in isolation?

Mocking

Mocking



- Simulating behaviour of surrounding layers
- Spy on objects
- Verify execution

Mocking Example - Java

```java
@ExtendWith(MockitoExtension.class)
class AuthUserExtractorTest {

    @Test
    @DisplayName("Extracts the user from the context")
    void extractUserTest() {

        //Arrange
        final AuthUserExtractor extractor
            = new AuthUserExtractor();
        final User user = genNewUser().build();

        final Context context = mock(Context.class);
        when(context.getAttribute(AUTH_USER_ATTRIBUTE))
        .thenReturn(user);

        //Act
        final User actual = extractor.extract(context);

        //Assert
        assertThat(actual).isEqualTo(user);

        verify(context).getAttribute(AUTH_USER_ATTRIBUTE);
    }
}
```

Object from 3rd Party library is mocked and told what to do

Verify that the mock was called correctly

Question

In a web environment, humanised string representation of a date time is is dependent on the users' browser and system settings.

How can we test that the date values we display are natively readable by our users?

Epoch: 0

EN-AU: *Thursday, 1 January 1970*

DE-DE: *Donnerstag, 1. Januar 1970*

Mocking Example - Node.js

```
const R = require('ramda')
const userLanguage = R.path(['navigator', 'userLanguage'])
const systemLanguage = R.path(['navigator', 'language'])

const options = {...}

const browserLanguage = () => userLanguage(window)
                           || systemLanguage(window)

const humanizeEpoch = epoch => {
  const d = new Date(0)
  d.setUTCSeconds(epoch)
  const lang = browserLanguage()
  return d.toLocaleDateString(lang, options)
}

module.exports = humanizeEpoch
```

Mocking Example - Node.js

```
const R = require('ramda')
const userLanguage = R.path(['navigator', 'userLanguage'])
const systemLanguage = R.path(['navigator', 'language'])

const options = {...}

const browserLanguage = () => userLanguage(window)
                           || systemLanguage(window)

const humanizeEpoch = epoch => {
  const d = new Date(0)
  d.setUTCSeconds(epoch)
  const lang = browserLanguage()
  return d.toLocaleDateString(lang, options)
}

module.exports = humanizeEpoch
```

Mocking Example - Node.js

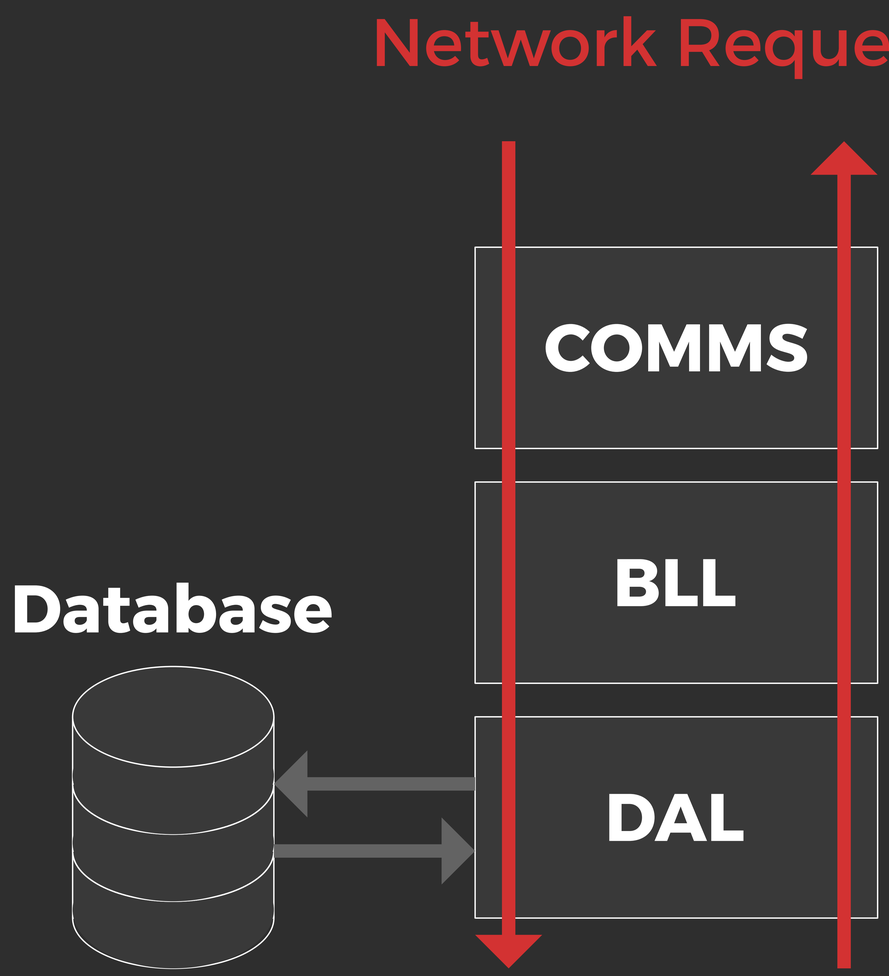
```
import test from 'ava'

const rewire = require('rewire')
const humanizeEpoch = rewire('../../src/js/humanizeEpoch')

const mockLanguage = lang => {
  humanizeEpoch.__set__('browserLanguage', () => lang)
}



test('converts an epoch time into a human readable format', t => {
    // Arrange
    const epoch = 0
    const expected = 'Thursday, 1 January 1970'

    mockLanguage('en-au')

    // Act
    const actual = humanizeEpoch(epoch)

    //Assert
    t.is(actual, expected)
  }
)
```

Rewire adds private setters and getters

Set the result we want

End-to-End or API Testing

End-to-End or API Testing



- Layers work together
- Server/Stack configuration
- Content negotiation (e.g. Read/write JSON)
- Transport protocols (HTTP, Socetc, Etc.

Question

In most SaaS systems, users are authenticated by tokens which are granted by the server after successful validation of key identifiers (typically a username / email, and password).

What would an end-to-end test involve?

What would we do in the Arrange, Act and Assert phases?

End-To-End/API Example

```
@Test
@DisplayName("valid credentials")
void validCredentialsTest() throws IOException {









}
```

End-To-End/API Example

```
@Test
@DisplayName("valid credentials")
void validCredentialsTest() throws IOException {
    //Arrange
    final User user = genNewUser().build();
    final Login credentials = Login.valueOf(user);
    final Response createResponse = postNewUser(user);









}
```

Setup the data needed for the scenario. (Creating a user)

End-To-End/API Example

```
@Test
@DisplayName("valid credentials")
void validCredentialsTest() throws IOException {
    //Arrange
    final User user = genNewUser().build();
    final Login credentials = Login.valueOf(user);
    final Response createResponse = postNewUser(user);

    //Act
    final String json = mapper.writeValueAsString(credentials);
    final Response response = requestTo(AUTH_URL)
        .method(POST)
        .body()
        .set(json)
        .back()
        .header(CONTENT_TYPE, APPLICATION_JSON)
        .fetch();




}
```

Setup the data needed for the scenario. (Creating a user)

Perform an actual HTTP request

End-To-End/API Example

```java
@Test
@DisplayName("valid credentials")
void validCredentialsTest() throws IOException {
    //Arrange
    final User user = genNewUser().build();
    final Login credentials = Login.valueOf(user);
    final Response createResponse = postNewUser(user);

    //Act
    final String json = mapper.writeValueAsString(credentials);
    final Response response = requestTo(AUTH_URL)
        .method(POST)
        .body()
        .set(json)
        .back()
        .header(CONTENT_TYPE, APPLICATION_JSON)
        .fetch();

    //Assert
    assertThat(response.status()).isEqualTo(OK_200);

    final String token = response.as(JsonResponse.class)
    .json().readObject().getString("token");

    assertThat(token).isNotEmpty();
}
```

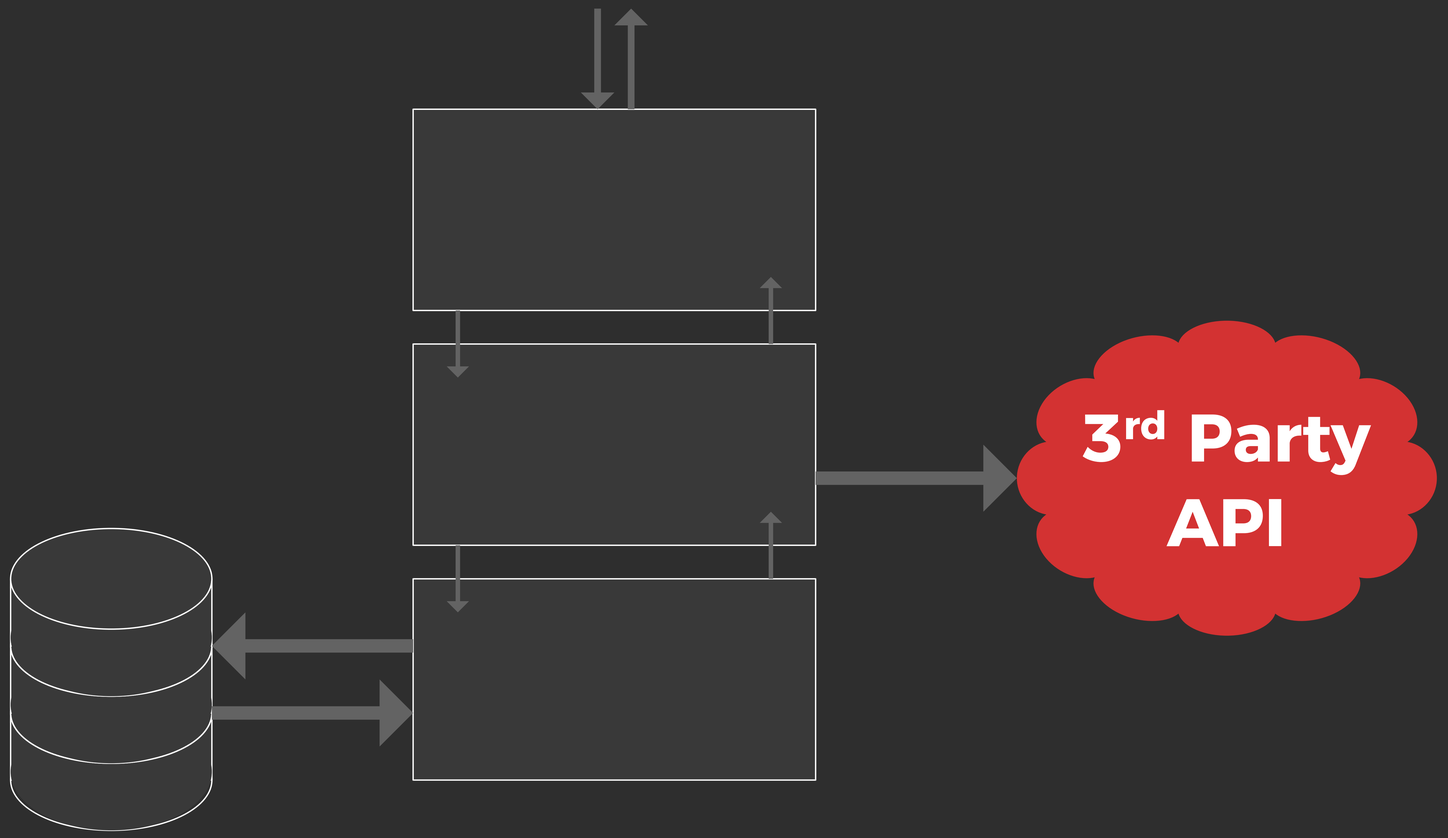
Setup the data needed for the scenario. (Creating a user)

Perform an actual HTTP request

Verify the HTTP result status code and body

Integration Testing

Integration Testing



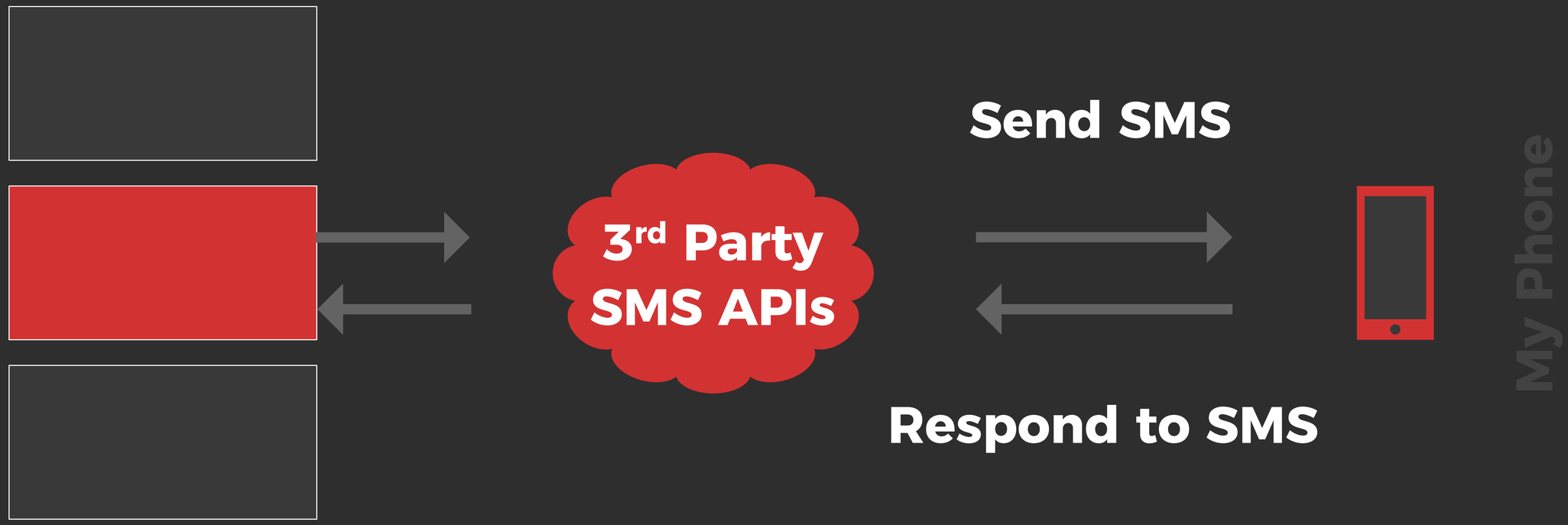
- Verify the 3rd Party credentials
- Verify requests and responses

Question

The JESI platform facilitates 2-way, conversational style, TXT messages to our users. As developers, how can we be confident that this feature is working before each deploy?

Integration Test

The JESI platform facilitates 2-way, conversational style, TXT messages to our users. As developers, how can we be confident that this feature is working before each deploy?



Project Orion

github.com/devnq/orion

Question

In project Orion, a registered user is able to create an event listing in the future and see it on the events page.

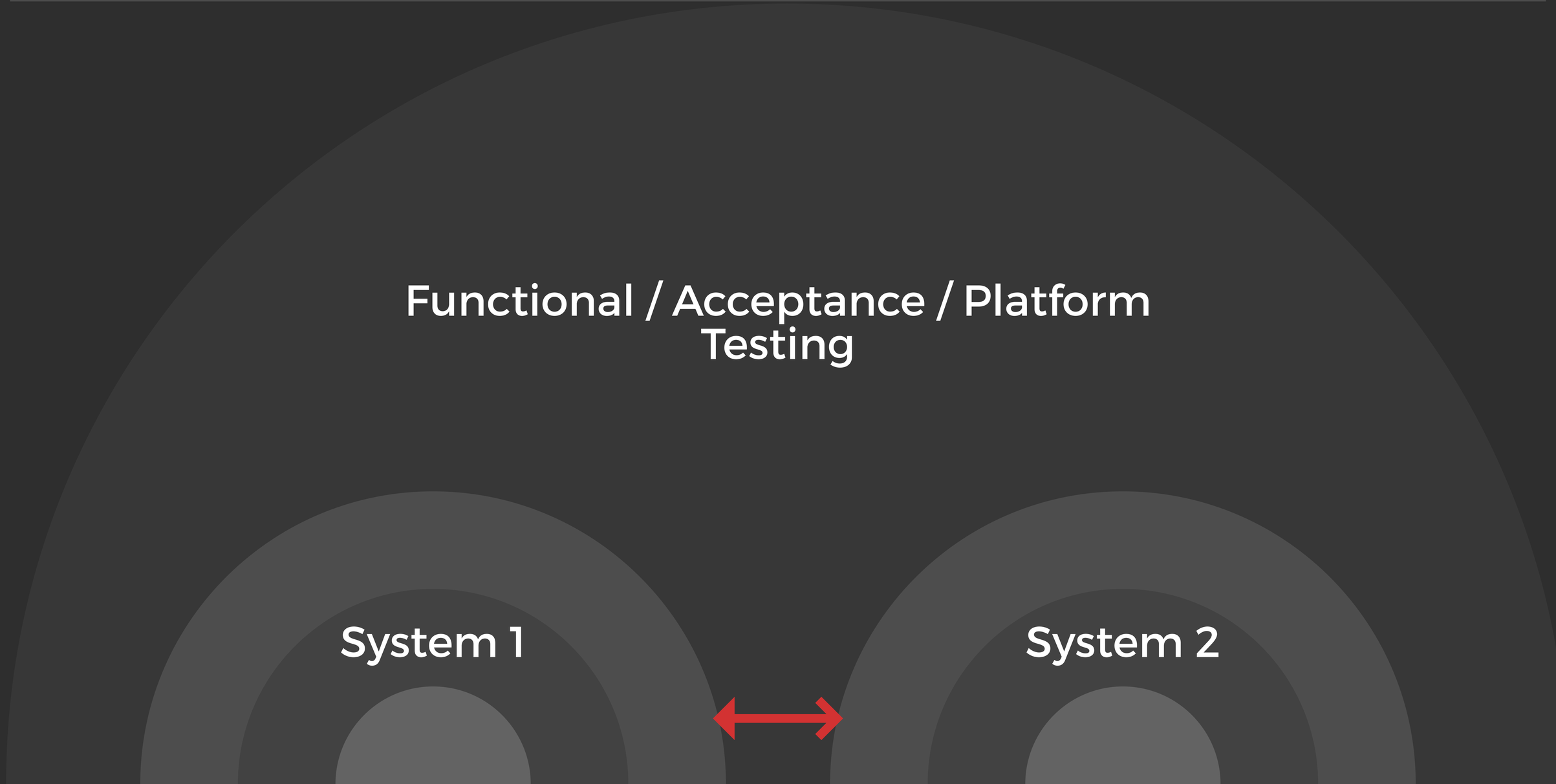
This scenario involves 2 code bases. The server, and web application.

How can we test that the web application communicates correctly with the server?

Functional / Acceptance / Platform

Testing

Functional / Acceptance / Platform Testing



Functional Test Example

```java
@Test
public void registerAndCreateEvent() {
    // Arrange
    final Faker faker = new Faker();
    final String name = faker.name().fullName();
    final String username = faker.name().username();
    final String password = faker.internet().password();
    final String eventTitle = faker.book().title();
    final String eventDescription = faker.lorem()
        .paragraphs(3)
        .toString();
    final LocalDateTime eventDate = now().plusDays(1);

    // Act
    goTo(loginPage)
        .registerAndLoginWith(username, name, password)
        .click(newEventPage.navLink);
    newEventPage
        .registerEvent(eventTitle, eventDescription, eventDate);

    // Assert
    eventsPage
        .waitForEvents()
        .assertSeeEvent(eventTitle, eventDescription);
}
```

Act from the very beginning of an account

Assert the Event was created and visible

Functional Test Example

```
@PageUrl("http://127.0.0.1:9000/#!/login")
class LoginPage extends FluentPage {

    @FindBy(css = "form.login")
    FluentWebElement loginForm;
...
    LoginPage typeIn(final FluentWebElement element,
                     final String value) {...}

    LoginPage click(final FluentWebElement element) {...}

    LoginPage waitFor(final FluentWebElement el) {...}

    LoginPage registerAndLoginWith(final String username,
                                   final String name,
                                   final String password) {
        return waitFor(loginForm)
            .click(signupButton)
            .waitFor(registerForm)
            .typeIn(usernameInput, username)
            .typeIn(nameInput, name)
            .typeIn(passwordInput, password)
            .click(registerButton)
            .waitFor(loginForm)
            .typeIn(passwordInput, password)
            .click(loginButton)
            .waitFor(logoutAnchor);
    }
}
```

Performs the same action a user would. Clicking and typing.

Are there other forms of Testing Software?

Availability

I18N (Translation)

System Load and Stress

Multi-Platform

Much More...

Chaos

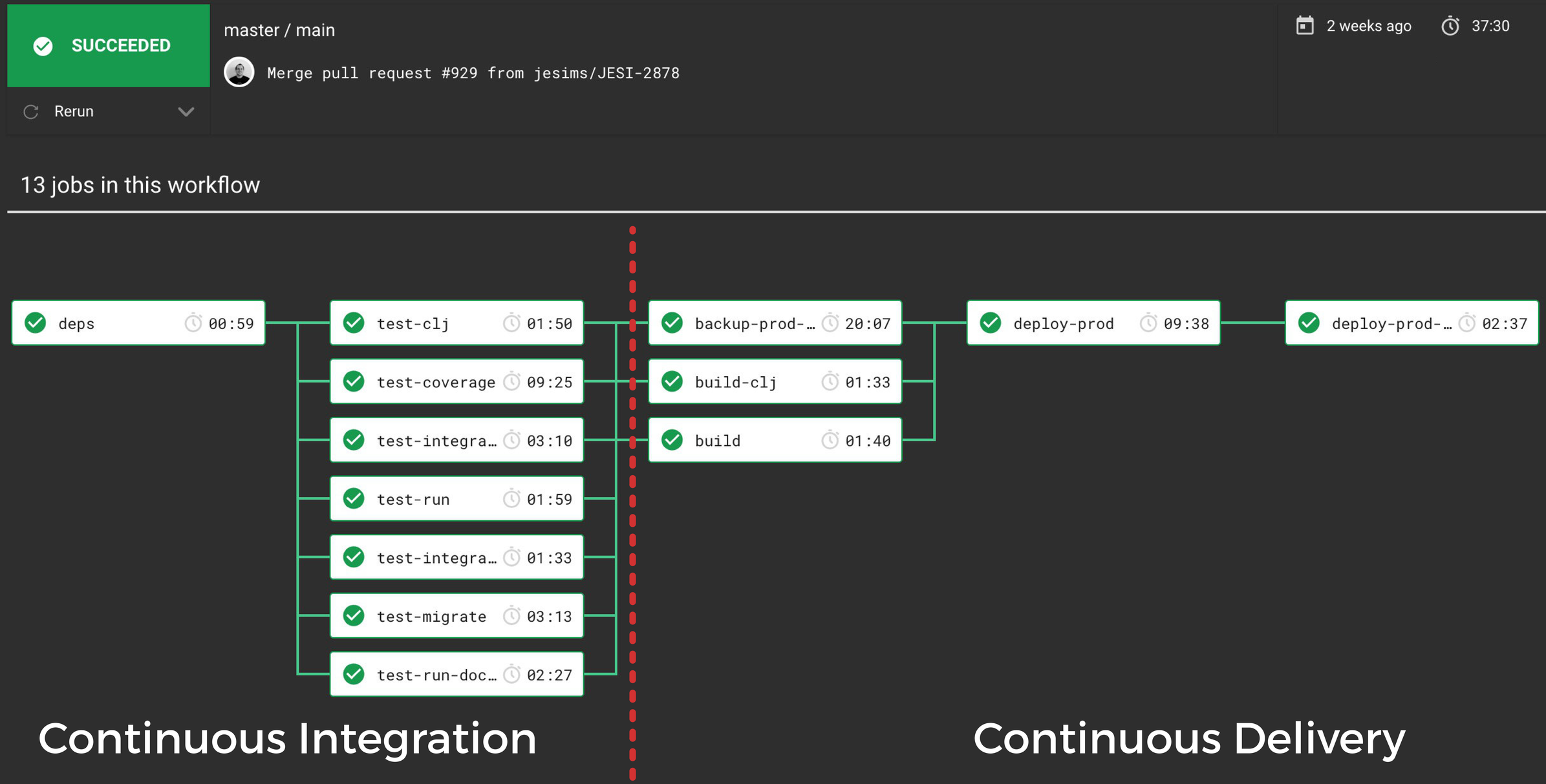
Visual

Penetration

Migration

Vulnerability

Continuous Integration (CI) &
Continuous Deployment (CD)


SUCCEEDED
master / main
Merge pull request #929 from jesims/JESI-2878
2 weeks ago
37:30
Rerun
13 jobs in this workflow
deps 00:59
test-clj 01:50
test-coverage 09:25
test-integra… 03:10
test-run 01:59
test-integra… 01:33
test-migrate 03:13
test-run-doc… 02:27
backup-prod-… 20:07
build-clj 01:33
build 01:40
deploy-prod 09:38
deploy-prod-… 02:37
Continuous Integration
Continuous Delivery

Questions?

Question

What do we test first? DAL? API? Functional?

Question

What do we test first? DAL? API? Functional?

What's better. TDD or BDD?

Question

What do we test first? DAL? API? Functional?

What's better. TDD or BDD?

What's my (and JESI) development cycle?

DevClub

Working and Learning Together

DevClub

- Play with new technology; languages; and paradigms
- Gain experience through collaborative projects
- Build skills used directly in the industry
- Learn from professional developers (project advisors)

Rudolph

Centrebull

PAASAAS

Your Project!

Experienced Mentors

New Tech

Discord Bots

Time Warp

NEXT

Dinner and Social